Cryptography Course Manuscript

Maceió — Winter 2019

Enno Nagel

July 2020

Notes for a course in cryptologoy that introduces

- the history,
- the applications,
- the implementations (in `Python`), and
- the (number) theory behind

cryptography.

Contents

Learning Objectives

In this course, you will learn about

1. the terminology, aims and history of cryptography,
2. the basic principles of symmetric algorithms (those with a single key),
3. the principles of asymmetric algorithms (those with a public and private key),
4. the methods of user authentication, the proof of the identity of a user,
5. the methods of cryptanalysis, the art of deciphering enciphered data without knowledge of the key,
6. the practical considerations of applied cryptography in a corporate environment,
7. its manifold implementations, for example, in Online banking, Blockchain and electronic voting.

The recommended reading will introduce you to the most important authors and articles in cryptology.

1 Basic concepts of cryptology

Study Goals

On completion of this chapter, you will have learned ...

- ... to distinguish between cryptology, cryptography and cryptanalysis.
- ... what cryptography is good for: IT-security for Confidentiality, Integrity and Availability.
- ... basic (historic) cryptographic algorithms.
- ... its historical impact.
- ... a key criterion (by Kerckhoff) for best cryptographical practice.
- ... The principal uses of Hash functions.

Cryptography serves to protect information by **encryption** (or **enciphering**), the shuffling of data (that is, the transformation of intelligible into indecipherable data) that only additional secret information, the **key**, can feasibly undo it (**decryption** or **deciphering**).

> **encrypt/encipher**: to shuffle data so that only additional secret information can feasibly undo it.
>
> **key**: the additional secret information that is practically indispensable to decrypt.
>
> **decrypt/decipher**: to invert encryption.

That is, the shuffled (enciphered) data can practically only be recovered (deciphered) by knowledge of the key. Since the original data is in principle still recoverable, it can be thought of as concealment.

Because historically only written messages were encrypted, the source data, though a string of `1`s and `0`s (the viewpoint adopted in symmetric cryptography) or a number (that adopted in asymmetric cryptography), is called *plaintext* and the encrypted data the *ciphertext.*

> **plaintext** respectively **ciphertext**: the data to be encrypted respectively the encrypted data.

Single and Public-Key Cryptography.

Historically, the key to reverse this transformation (of intelligible data into indecipherable data) was both necessary to decipher and to encipher, *symmetric encryption.* That is, in the past, the key used to encrypt and decrypt was always the same: *Symmetric* cryptography had been used by the Egyptians almost 2000 years before Christ, and was used, for example,

- during World War II on the `Engima` machine, and
- nowadays, in the encryption of a wireless network (for example, by the `AES` algorithm).

> **symmetric cryptography**: cryptography is *symmetric* when the same key is used to encrypt and decrypt.

In the 70s *asymmetric cryptography* was invented, in which the key to encipher (*the public key*) and the key to decipher (the *secret* or *private* key) are different.

> **asymmetric cryptography**: cryptography is *asymmetric* when different keys are used to encrypt and decrypt. The key to encipher is *public* and the key to decipher is *private* (or *secret*).

In fact, only the key to decipher is private, kept secret, while the key to encrypt is public, known to everyone. In comparison with symmetric cryptography, asymmetric encryption avoids the risk of compromising the key to decipher that is involved

- in exchanging the key with the cipherer, and
- in ownership of the cipher key (by the cipherer in addition to the decipherer).

On top, It is useful, that the keys exchange their roles, the private key enciphers, and the public one deciphers, a *digital signature*: While the encrypted message will no longer be secret, every owner of the public key can check whether the original message was encrypted by the private key.

Nowadays such asymmetric cryptography algorithms are ubiquitous on the Internet: Examples are

- RSA which is based on the difficulty of factoring in prime numbers, or
- ECC which is based on the difficulty of computing points in finite curves,

which protect (financial) transactions on secure sites (those indicated by a padlock in the browser's address bar).

Data Format. Up to the digital age, cryptography mainly studied the transformation of intelligle text into indecipherable text. Since then, cryptography studies the transformation of processible (digital) data into indecipherable (digital) data. This data is, for example, a digital file (text, image, sound, video, ...). It is considered a bit sequence (denoted by a string of 0s and 1s) or byte sequence (denoted by a string of hexadecimal pairs 00, 01, ..., FE, FF) or a number (denoted as usual by their decimal expansion 0, 1, 2, 3 ...). Let us recall that every 1011... bit sequence is a number n via its binary expansion

$$n = 1 + 0 \cdot 2 + 1 \cdot 2^2 + 1 \cdot 2^3 + \cdots$$

(and vice versa).

The point of view of a sequence of bits (or, more exactly, of hexadecimal digits whose sixteen symbols `0` – `9` and `A` – `F` correspond to a group of four bits) is preferred in symmetric cryptography whose algorithms transform them, for instance, by permutation and substitution of their digits. The point of view of a number is preferred in asymmetric cryptography whose algorithms operate on it by mathematical functions such as raising to a power (raising to a power) and exponentiation.

The *key*, the additional secret information, can take various form; which form is mainly a question of convenience, most common are:

- that of a number,
- that of a sequence of letters, for example,
  - a password, or
  - a secret phrase (with spaces).

For example, in the ancient *Scytale* algorithm (see Section 2) that uses a role of parchment wrapped around a stick, the key consists of the circumference (in letters) of the stick, a small number. Nowadays, `PIN` codes (= Personal Identification Number) or passwords are ubiquitous in day-to-day life; to facilitate memorization the memorization of complete secret sentences (= pass phrases) is encouraged.

Asymmetric encryption depends on larger keys and therefore stores them in files (of 64-letter texts, called `ASCII`-armor) of a couple of kilobytes. For example (where ... indicates tens of skipped lines):

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: SKS 1.1.6
Comment: Hostname: pgp.mit.edu

mQENBFcFAs8BCACrW3TP/ZiMRQJqWP0SEzXqm2cBZ+fyBUrvcu1fGU890pd4
3JdiWIreHx/sbJdW1wjABeW8xS1bM67nLW9VVHUPLi9QP3VGfmqmXqbWIB7O
...
-----END PGP PUBLIC KEY BLOCK-----
```

1.1 Terminology

The prefix *Crypto-*, comes from Greek kryptós, "hidden".

Cryptography. **Cryptography** (from the Greek *gráphein*, "to write") is the art of hidden writing: shuffling information so that it is indecipherable to all but the intended recipient.

> **Cryptography**: the art of transforming information so that it is indecipherable to all but the intended recipient.

That is, cryptography is the art of transforming information such that it is incomprehensible to all but the intended recipient. Useful, since Antiquity, for example to conceal military messages from the enemy. Since then, (electronic binary) data has replaced text, and what used to be concealing written messages exchanged by messengers or kept secret has become *symmetric cryptography*: securing data flowing between computers or stored on a computer.

Since the 70s, asymmetric cryptography makes it possible (by *digital signatures*) to verify the identities of participants and undeniably (non-repudiation) register their transactions in electronic commerce.

Cryptographic methods (or *Ciphers*) are generically classified

1. according to whether sender and recipient use the same key (*symmetric*, or *single-key*) cipher, such as AES, or different keys to encrypt and decrypt (*asymmetric*, or *two-key*, or *public-key*) cipher, such as RSA or ECC.

Among the symmetric ciphers, these are generically classified

2. according to whether they operate on blocks of bits of fixed length, say 128 bits, (*block cipher*, such as AES or RSA) or single bits (*stream ciphers* such as RC4): While stream ciphers typically are simpler, faster and predestined for real time transmissions, they tend to be less secure and are therefore less commonly used (for example, a Wi-Fi network is commonly secured by a block cipher such as AES).

Cryptanalysis. **Cryptanalysis** (from the Greek analýein, "to unravel") is the art of untying the hidden writing: the breaking of ciphers, that is, recovering or forging enciphered information without knowledge of the key.

> **Cryptanalysis**: the art of deciphering ciphertext without knowledge of the key.

Cryptanalysis (colloquially "code breaking") is the art of deciphering the enciphered information without knowledge of the secret information, the *key*, that is normally required to do so; usually by finding a secret key.

Cryptanalysis of public-key algorithms relies on the efficient computation of mathematical functions on the integers. For instance, cryptanalysis of the most famous public-key algorithm, RSA , requires the factorization of a number with > 500 decimal digits into its prime factors, which is computationally infeasible (without knowledge of the key).

Cryptanalysis of symmetric ciphers depends on the propagation of patterns in the plaintext to the ciphertext. For example, in a *monoalphabetic* substitution cipher (in which each letter is replaced by another letter, say A by Z), the numbers of occurrences with which letters occur in the plaintext alphabet and in the ciphertext alphabet are identical (if A occurred ten times, then so does Z). If the most frequent letters of the plaintext can be guessed, so those of the ciphertext.

A powerful technique is *Differential cryptanalysis* that studies how differences (between two plaintexts) in the input affect those at the output (of the corresponding ciphertexts). In the case of a block cipher, it refers to tracing the (probabilities of) differences through the network of transformations. Differential cryptanalysis attacks are usually *Chosen-plaintext attacks*, that is, the attacker can obtain the corresponding ciphertexts for some set of plaintexts of her choosing.

Cryptology. **Cryptology** (from the Greek *lógos*, "word", "reason", "teaching" or "meaning") is the science of hiding, the science of trusted communication which embraces cryptography and cryptanalysis; according to Webster (1913) it is "the scientific study of cryptography and cryptanalysis". Though cryptology is often considered a synonym for cryptography and occasionally for cryptanalysis, cryptology is the most general term.

> **Cryptology**: the science of trusted communication, including cryptography and in particular cryptanalysis.

Secrecy, though still important, is no longer the sole purpose of cryptology since the advent of public-key cryptography in the 80s. To replace by electronic devices what had historically been done by paperwork, digital signatures and authentication were introduced.

Adjectives often used synonymously are *secret*, *private*, and *confidential*. They all describe information which is not known about by other people or not meant to be known about. Something is

- *secret*, from Latin secretus "set apart", if it is only known by a particular person or group,
- *confidential*, from from Latin confidere, "to have full trust or reliance", if it is to be kept secret, that is, not to be told or shared with other people,
- *private*, from Latin privatus "set apart" (from the state), if it is meant to be secret, especially regarding an individual versus the state.

Frequently confused, and misused, terms in cryptology are code and cipher, often employed as though they were synonymous: A *code* is a rule for replacing one information symbol by another. A *cipher* likewise, but the rules governing the replacement (the key) are a secret known only to the transmitter and the legitimate recipient.

Code. A **codification** or an **encoding** is a rule for replacing one bit of information (for example, a letter) with another one, usually to prepare it for processing by a computer.

> **encoding**: a rule for replacing one bit of information (for example, a letter) with another one, usually to process it by a computer.

For example,

- Morse Code, that replaces alphanumeric characters with patterns of dots and dashes,
- the American Standard Code for Information Interchange (ASCII code) from 1963 that represents on computers 128 characters (and operations such as backspace and carriage return) by seven-bit numbers, that is, by sequences of seven 1s and 0s. For example, in ASCII a lowercase a is

always 1100001, a lowercase b is always 1100010, and so on (whereas an uppercase A is always 1000001, an uppercase B is always 1000010).
- more recently, UTF-8 (8-bit Unicode Transformation Format) is a variable-length character encoding by Ken Thompson and Rob Pike to represent any universal character in the Unicode standard (which possibly has up to $2^2 \approx 4$ billion characters, and includes the alphabets of many languages, such as English, Chinese, ..., as well as meaningful symbols such as emoticons) by a sequence of between 1 up to 4 bytes, and which is backwards compatible with ASCII , and is becoming the de facto standard.

Ciphers. A **cipher**, like an encoding, also replaces information (which may be anything from a single bit to an entire sequence of symbols) with another one. However, the replacement is made according to a rule defined by a key so that anyone without its knowledge cannot invert the replacement.

> **cipher**: a rule for replacing information (for example, a text) so that its inverse is only feasible by knowledge of the key.

Information is frequently both encoded and enciphered: For example, a text is encoded, for example, by ASCII, and then encrypted, for example, by the Advanced Encryption Standard (AES).

Self-Check Questions.

1. Please distinguish between cryptology, cryptography and cryptanalysis:
   - *cryptology is the **science** of secure storage and communication of information.*
   - *cryptography is the **art** of secure storage and communication of information.*
   - *cryptanalysis is the **art** of decryption of encrypted information without knowledge of the key.*
2. Please distinguish between encoding and encryption:

   While both encoding and encryption transform information into a computer readable format, only for encryption this transformation is not invertible without knowledge of the key.

1.2 IT security: threats and common attacks

A snappy acronym to resume the fundamental aims of information security is the **CIA**, which stands for: Confidentiality, Integrity and Availability. That is, confidentiality of information, integrity of information and availability of information.

> **CIA**: stands for Confidentiality, Integrity and Availability.

More comprehensive are "the **five pillars of Information Assurance**", that add authentication and non-repudiation: Confidentiality, integrity, availability, authentication and non-repudiation of information.

> The five *pillars of Information Assurance*: are formed by Confidentiality, integrity, availability, authentication and non-repudiation of information.

Cryptography helps to achieve all of these to good effect: Good encryption, as achieved by thoroughly tested standard algorithms such as AES or RSA, is practically impossible to break computationally; instead, keys are stolen or plaintext is stolen before encryption or after decryption. While cryptography provides high technical security, human failure, for example, arising out of convenience or undue trust, is the weakest point in information security.

Confidentiality. Information that is *confidential* is meant to be kept secret, that is, should not be disclosed to other people, for example information that is known only by someone's doctor or bank. In law, confidential is the relation existing between, for example, a client and her counsel or agent, regarding the trust placed in one by the other. In the information security standard ISO/IEC 27002 the International Organization for Standardization (ISO) defines *confidentiality* as "ensuring that information is accessible only to those authorized to have access". In IT, it means ensuring that sensitive information stored on computers is not disclosed to unauthorized persons, programs or devices. For example, avoiding that anyone with access to a network can use common tools to eavesdrop on traffic and intercept valuable private information.

Integrity. *Integrity* is the state of being whole, the condition of being unified or sound in construction.

(Data) Integrity is about the reliable, complete and error-free, transmission and reception or storage of data: that the original data had not been altered or corrupted; in particular, is valid in accordance with expectations.

When the data has been altered, either through electronic damage by software or physical damage to the disk, the data is unreadable. For example when we download a file we verify its integrity by calculating its hash and comparing with the hash published by source. Without such a check, someone could, for example, package a Trojan horse into an installer on Microsoft Windows (that, as a last resort, hopefully would already be known and detected by an antivirus program such as Microsoft Defender).

Availability. Though unrelated to cryptology, in IT security availability of information against threats such as DoS (Denial of Service) attacks (to deny users of the Website access to a Website by flooding it with requests) or accidents, such as power outages, or natural disasters such as earthquakes. To achieve it, it is best to have a safety margin and include redundancy, in particular, to have

- parallel redundant *failover hardware*, such as a server or network, which is always kept running so that at any moment, upon detected failure of the primary system, processing can be automatically shifted over.
- prevent intrusion by monitoring network traffic patterns for anomalies and block network traffic when necessary.

Authentication. Authentic (from Greek *authentes*, real or genuine) means according to Webster (1913)

- not false or copied, genuine, real.
- having the origin supported by unquestionable evidence, verified, or
- entitled to acceptance or belief because of agreement with known facts or experience; reliable; trustworthy.

Authentication thus is the verification of something (or someone) as "authentic". This might involve confirming the identity of a person or the origins of an object.

In IT, authentication means

- verification of the identity of a user and possibly her permission to access an object; convincing a computer that a person is who she claims to be after identification.
- verification that data is unchanged between two points of time; be it intentionally (*altered*) or accidentally (*corrupted*).

To verify her identity, a person proves that she is who she claims to be by showing some evidence. Devices that are used for authentication include passwords, personal identification numbers, smart cards, and biometric identification systems. For example, to login, she enters her user identification and password.

A common attack is that of the "man in the middle", where the attacker assumes to either correspondent the identity of the other correspondent. To solve this, certificates, digital signatures by third parties, are used. Either,

- as in the web of trust in `OpenPGP`, by signatures among persons known to each other over ends, or
- as on secure sites in a web browser, by a signature of an, unconditionally trusted, central certification authority, usually companies such as `VeriSign`.

Non-repudiation. Repudiation is a legal term for disavowal of a legal bind (such as an agreement or obligation); someone who repudiates:

- refuses to accept or be associated with a legal bind,
- refuses to recognize the validity of the legal bind (for example, her signature),
- refuses to fulfill the legal bind.

For example, a forged or forced signature is repudiable.

Non-repudiation is the assurance:

- that a contract cannot later be denied by either of the parties that agreed on it;
- of the identity of the claimed sender or recipient of a given message.

In computing, this means that authentication can hardly be refuted afterwards. This is achieved by a digital signature.

For example,

- an electronic receipt proves that a particular user has sent a message such as an instruction to buy an item in an online auction.
- if the e-mail says it was sent by Bob, then later on Bob can't claim that it was not originally sent by him.

In today's global economy, where face-to-face agreements are often impossible, non-repudiation is essential for safe commerce.

Self-Check Questions.

1. In practice, is sensitive information obtained by

   - ☐ *human*, or
   - ☐ cryptographic

   failure?

2. What does CIA in IT security stand for: *Confidentiality, Integrity and Availability.*

3. Please list the five pillars of information security: *Confidentiality, integrity, availability, authentication and non-repudiation of information.*

1.3 Aims of Cryptography in Child's terms

Alice wants to send Bob a locked box (with a message or a key) without sending the key, so that the box was never left unlocked throughout the process.

A castle. An open lock is the public key and locking its application. The key is the secret key:

1. Alice sends Bob a lock of hers.
2. Bob puts the keys in the box, locks it with Alice's lock and sends the box to Alice.
3. Alice opens the box.
4. Now Alice and Bob have a key to a common lock.

Two locks. The key or message in the box is the shared secret. The other two keys the mutual secrets. From two two-sided one can be constructed by the interchangeability of the order of the encryptions, commutativity.

0. Alice puts a message or key in a box.
1. Alice padlocks the box (and keeps the key) and sends the locked box to Bob.
2. Bob obviously can't open the box, but he secures the box with a second padlock (and also keeps the key) and sends the double-locked box back to Alice.
3. Alice opens her lock with her key and sends the box (still padlocked) to Bob.
4. Bob can now open the box with his own key.

1.4 Historical Overview

The history of cryptography dates back at least 4000 years. We distinguish three periods:

1. Till the 20th century, its methods were classic, mainly pen and paper.
2. In the early 20th century, they were replaced by more efficient and sophisticated methods by complex electromechanical machines, mainly rotor machines for polyalphabetic substitution, such as the Enigma rotor machine used by the axis powers during World War II.
3. Since then, digitalization, the replacement of analog devices by digital computers, allowed methods of ever greater complexity. Namely, the most tested algorithms are

- DES (or its threefold iteration 3DES) and its successor AES for symmetric cryptography,
- RSA and its successor ECC (Elliptic Curve Cryptography) for asymmetric cryptography.

Classic Cryptography. From antiquity till World War I, cryptography was carried out by hand and thus limited in complexity and extent to at most a few pages. The principles of cryptanalysis were known, but the security that could be practically achieved was limited without automatization. Therefore, given sufficient ciphertext and effort, cryptanalysis was practically always successful.

The principles of cryptanalysis were first understood by the Arabs. They used both substitution and transposition ciphers, and knew both letter frequency distributions and probable plaintext in cryptanalysis. Around 1412, al-Kalkashandī gave in his encyclopedia Subīal-aīshī a manual on how to cryptanalyze ciphertext using letter frequency counts with lengthy examples.

Scytale. A scytale (from Latin *scytala*) consists of a rod with a band of parchment wound around it on which is written a secret message. It was rolled spirally upon a rod, and then written upon. The secret writing on the strip wound around the rod is only readable if the parchment was to be wound on a rod of the same thickness; It is a *transposition cipher*, that is, shuffles, or transposes, the letters of the plaintext.

Caesar's Cipher. Caesar's Cipher is one of the simplest and most widely-known chiphers, named after Julius Caesar (100 – 44 BC), who used it to communicate with his generals. It is a **substitution cipher** that replaces, substitutes, each alphabetic letter of the plaintext by a fixed alphabetic letter. In Caesar's Cipher, each letter in the plaintext is shifted through the alphabet the same number of positions; that is, each letter in the plaintext is replaced by a letter some fixed number of positions further down the alphabet.

Bacon's Cipher. Francis Bacon's cipher from 1605 is an arrangement of the letters a and b in five-letter combinations (of which there are $2^5 = 32$) that each represent a letter of the alphabet (of which there are 26). Nowadays we would

call a code, but at the time it illustrated the important principle that only two different signs can be used to transmit any information.

Alberti's Cipher Disk. In 1470, Leon Battista Alberti described in Trattati in Cifra ("Treatise on Ciphers") the first cipher disk to shift the letters of the alphabet cyclically. He recommended changing the offset after every three or four words, thus conceiving a *polyalphabetic* cipher in which the same alphabetic letters are replaced by different ones. The same device was used more than four centuries later by the U.S. Army in World War I.

ADFGVX cipher. The best known cipher of World War I is the German ADFGVX cipher:

1. The 26 Latin letters and 10 arabic digits were replaced in a 6 x 6 matrix by pairs of the letters A, D, F, G, V, and X.
2. The resulting text was then written into a rectangle, and
3. then the columns read in the order indicated by the key.

Invented by Fritz Nebel, it was introduced in March 1918 for use by mobile units. The French Bureau du Chiffre, in particular, Georges Painvin, broke the cipher a month later — still too late as the German attacks had already ceded.

Electromechanical Cryptography by Rotor Machines. The mechanization of cryptography began after World War I by the development of so-called *rotor cipher machines*:

These rotors are stacked. The rotation of one rotor causes the next one to rotate 1/26 of a full revolution. (Just like in an odometer where after a wheel has completed a full revolution, the next one advances 1/10 of a full revolution.) In operation, there is an electrical path through all rotors. Closing the key contact of the plaintext letter on a typewriter-like keyboard

1. emits a current to one of the contacts on the initial rotor,
2. The current then passes through the cable salad of the stacked rotors (which depends on their rotational positions!), and
3. ends up at an indicator where it lights up the lamp of the ciphertext letter.

Figure 1: The Alberti substitution disk (Buonafalce (2014))

In the US, Edward H. Hebern made in 1917 the first patent claim to accomplish polyalphabetic substitution by cascading a collection of monoalphabetic substitution rotors, wiring the output of the first rotor to the input of the following rotor, and so on. In Europe, Already in 1915 such a rotor machine had been built by two Dutch naval officers, Lieut. R.P.C. Spengler and Lieut. Theo van Hengel, and independently by a Dutch mechanical engineer and wireless operator, Lieut. W.K. Maurits. Around the same time as Hebern, Arthur Scherbius from Germany (who filed his patent in February 1918) and Hugo A. Koch

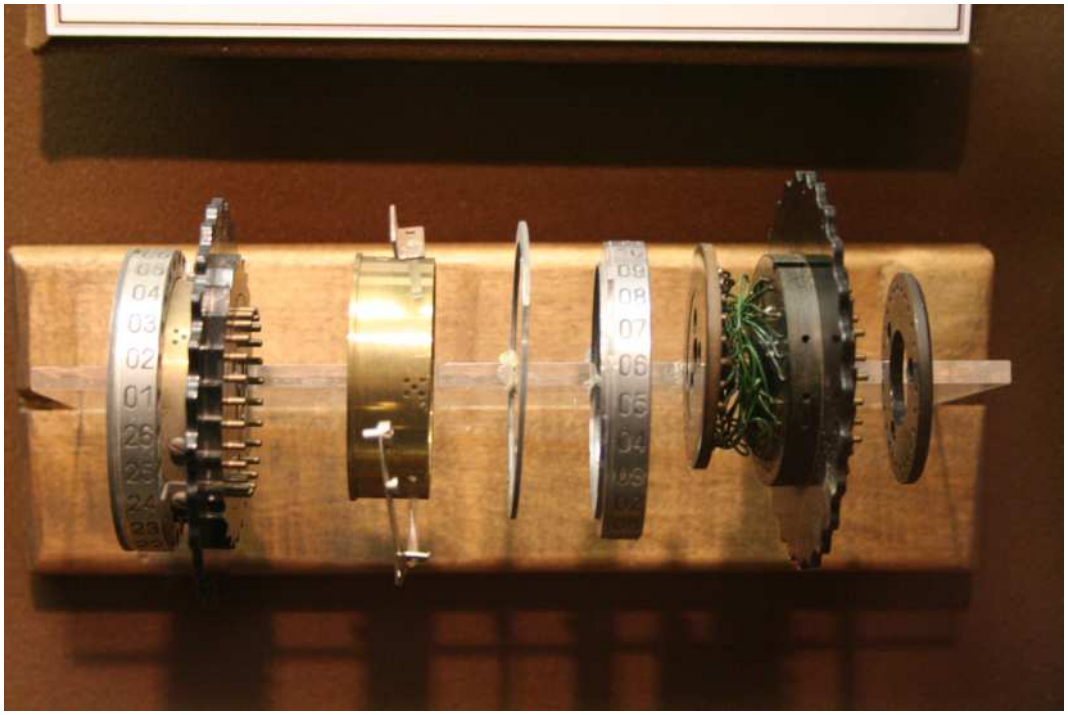


Figure 2: Enigma Rotor Wiring (CourtlyHades296 (2017))

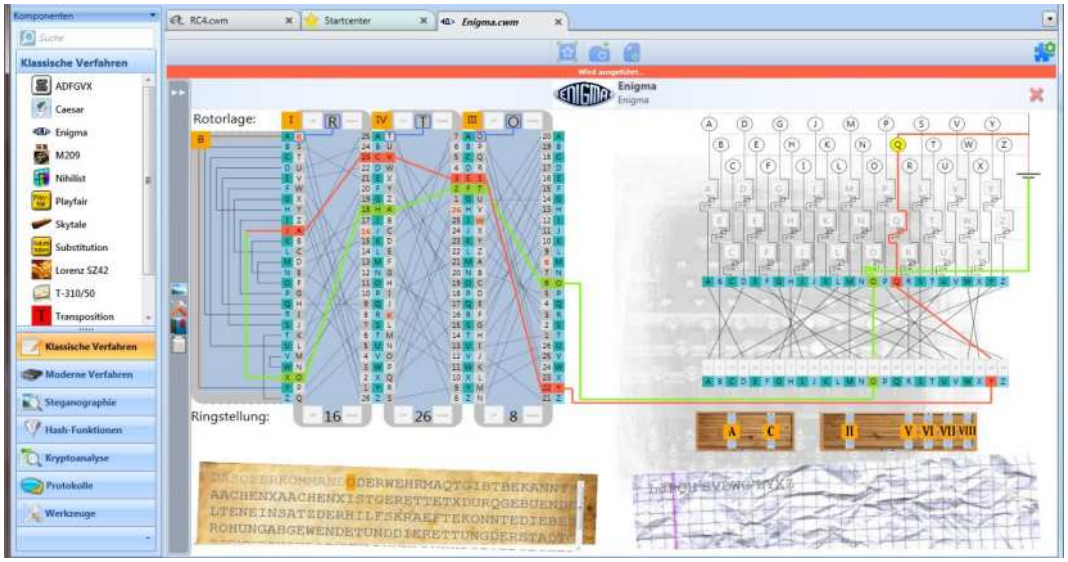


Figure 3: `CrypTool 2` has an animation of the encryption by `Enigma`; Esslinger et al. (2012)

from the Netherlands (a year later), also built rotor machines, which were commercialized and evolved into the German Enigma used in World War II.

In Japan, the Japanese Foreign Office put into service its first rotor machine in 1930, which was cryptanalyzed in 1936, using solely the ciphertexts, by the U.S. Army's Signal Intelligence Service (SIS). (In 1939 a new cipher machine was introduced, in which rotors were replaced by telephone stepping switches, but readily broken by the SIS again solely relying on ciphertext; even more so, their keys could be foreseen.)

The Invention of Engima. Arthur Scherbius was born in Frankfurt am Main on 20 October 1878 as son of a businessman. After studying at the Technical College in Munich, he completed his doctoral dissertation at the Technical College in Hanover in 1903, then worked for several major electrical companies in Germany and Switzerland. In 1918, he submitted a patent for a cipher machine based on rotating wired wheels and founded his own firm, Scherbius and Ritter. Since both the imperial navy and the Foreign Office declined interest, he entered the commercial market in 1923 and advertised the Enigma machine, as it was now called, in trade publications or at the congress of the International Postal Union. This sparked again the interest of the German navy in the need for a secure cipher, and a slightly changed version was in production by 1925. Still, the corporation continued to struggle for profitability because commercial as public demand was confined to a few hundred machines. While Scherbius fell victim to a fatal accident involving his horse-drawn carriage, and died in 1929, his corporation survived and by 1935 amply supplied the German forces under Hitler's rearmament program.

The Breakage of Enigma. Polish and British cryptanalysis solved the German Enigma cipher (as well as two telegraph ciphers, Lorenz-Schlüsselmaschine and Siemens & Halske T52). To this end

- the patents of the Enigma were filed in the United States,
- similar machines were commercially available, and
- the rotor wirings were known from a German code clerk:

Hans-Thilo Schmidt, decorated with an Iron Cross in World War I, worked as a clerk at a cipher office (previously lead by his brother). In June 1931, he contacted the intelligence officer at the French embassy and agreed with

Rodolphe Lemoine to reveal information about the Enigma machine, copies of the instruction manual, operating procedures and lists of the key settings. However, French cryptanalysts made little headway, and the material was passed on to Great Britain and Poland, whose specialists had more success:

The commercial version of the Enigma had a rotor at the entry and his wiring was unknown. However, the Polish cryptanalyst Marian Rejweski, guided by the German inclination for order, found out that it did not exist in the military version; what is more, he inferred the internal wirings of the cylinders by distance, that is, by mere cryptanalysis of the enciphered messages.

Figure 4: Marian Rejweski; Thuresson (2013)

The Britisch cryptanalysts in Bletchley Park (among them the mathematician Alan Turing, a founding father of theoretical computer science) could reduce by likely candidates the number of possible keys from `150` trillions to around a million, a number that allowed a work force of around `4200` (among them 80% women) an exhaustive key-search with the help of the Turing Bomb, an ingenious electromechanical code-breaking machine that imitated a simultaneous run of many Enigma machines and efficiently checked the likelihood of their results.

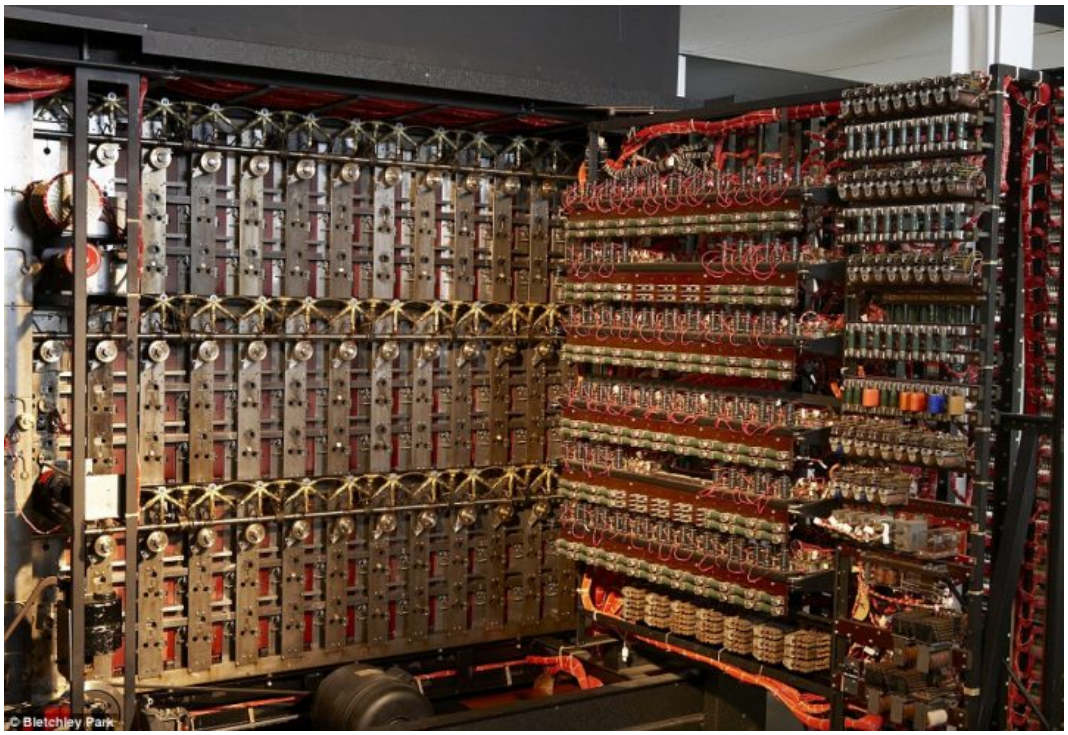


Figure 5: The inner workings of the Turing bomb; Petticrew (2018)

Schmidt continued to inform the Allies throughout war till the arrest (and confession) of Lemoine in Paris which lead to that of Schmidt by the Gestapo in Berlin on 1 April 1943 and his death in prison.

Digital Cryptography. After World War II, cryptographic machines stayed conceptually the same till the early 80s: faster rotor machines in which rotors had been replaced by electronic substitutions, but still merely concatenating shifted monoalphabetic substitutions to obtain a polyalphabetic substitution.

However, such letter per letter substitutions are still *linear over the letters*, so the ciphertext obtained from a plaintext will reveal how to decrypt all letters of a plaintext of (at most) the same length. That is, a letter per letter substitution *diffuses* little, that is, hardly spreads out changes; optimal diffusion is attained whenever the change of a single letter of the plaintext causes the change of half of the letters of the ciphertext. If the attacker has access to the ciphertexts of many plaintexts, possibly of his own choosing, then he can obtain the key by

the ciphertexts of two plaintexts that differ in a single position.

DES. Instead, computers made it possible to combine such substitutions (such as Caesar's Cipher) with transpositions (such as the Scytale), achieving far better diffusion, which lead to the creation of one of the most widely used ciphers in history, the Data Encryption Standard (DES), in 1976.

AES. In January 1997 the U.S. National Institute of Standards and Technology (NIST; former National Bureau of Standards, NBS) announced a public contest for a replacement of the aging DES, the Advanced Encryption Standard (AES). Among 15 viable candidates from 12 countries, in October 2000 Rijndael, created by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, was chosen and became the AES.

Since improvements in computing power allowed to find the fixed 56-bit DES key by exhaustive key-search (brute force), the NIST specifications for the AES demanded an increasing key length, if need be. Rijndael not only was shown immune to the most sophisticated known attacks such as differential cryptanalysis (in Daemen and Rijmen (1999) and Daemen and Rijmen (2002)) and of an elegant and simple design, but is also both small enough to be implemented on smart cards (at less than 10 000 bytes of code) and flexible enough to allow longer key lengths.

Public-key cryptography. Since the '80s, the advent of public-key cryptography in the information age made digital signatures and authentication possible; giving way to electronic information slowly replacing graspable documents.

Asymmetric encryption was first suggested publicly at Diffie and Hellman (1976).

Conceptually it relies on a *trap function* (more specifically, in op.cit. the modular exponential), an invertible function that is easily computable but whose inverse is hardly computable in the absence of additional information, the *secret key*.

To encrypt, the function is applied; to decrypt its inverse with the secret key. For example, in the approach according to Diffie and Hellman, this function is the exponential, however, over a different domain than the real numbers we are used to.

In fact, Diffie and Hellman (1976) introduced only a scheme for exchanging a secret key through an insecure channel. It was first put it into practice

- in Rivest, Shamir, and Adleman (1978), where the `RSA` cryptographic algorithm was introduced, or
- by the `ElGamal` algorithm, more recent, but the closest example of the original scheme.

Not only do these algorithms enable ciphering by a public key (thus removing the problem of its secret communication), but, by using the private key instead to encipher, made possible digital signatures, which might have been its commercial breakthrough. These algorithms still stand strong, but others, such as elliptic curve cryptography, are nowadays deemed more efficient at the same security.

Self-Check Questions.

1. Please list the major epochs of cryptography: *Classic (Pen and Paper), electromechanics (rotor machine) and digital age.*
2. How many possible keys has Caesar's Cipher? `26` *including the trivial one.*
3. Which one of Caesar's Cipher and the Scytale is a substitution cipher? *Caesar's Cipher is a substitution cipher and the Scytale is a transposition cipher.*
4. What deficiency was shared by all rotor machines? *As substitution is letter-wise, the frequency of the alphabetical letters was preserved.*

1.5 Security Criteria

Kerckhoff's Principle.

Kerckhoff principle postulates the independence of a cryptographic algorithm's security from its secrecy:

> **Kerckhoffs' principle**: The ciphertext should be secure even if everything about it, except the key, is public knowledge.

While knowledge of the key compromises a single encryption, knowledge of the algorithm will compromise all encryptions ever carried out. A public algorithm guarantees the difficulty of decryption depending only on the knowledge of the *key*, but not on the *algorithm*. The more it is used, the more likely it becomes that the algorithm will be eventually known. For the algorithm to be useful, it thus needs to be safe even though it is public.

Claude Shannon (1916 – 2001) paraphrased it as: "the enemy knows the system", Shannon's maxim. The opposite would be to rely on a potentially weak, but unknown algorithm, "security through obscurity"; ample historic evidence shows the futility of such a proposition (for example, the above ADFGVX cipher of Section 1.4 comes to mind).

Shannon's Criteria.

Shannon's principles of

- *Confusion* respectively *Diffusion*

give criteria for an uninferable relation between the *ciphertext* and

- the *key* respectively the *plaintext.*

Ideally, when one letter in the key respectively in the plaintext changes, then half of the ciphertext changes, that is, each letter in the ciphertext changes with a probability of *50%.* While the output of the cipher, the ciphertext, depends deterministically on the input, the plaintext, and the key, the algorithm aims to *obfuscate* this relationship to make it as *complicated*, intertwined, scrambled as possible: each letter of the output, of the ciphertext, depends on each letter of the input, of the plaintext, and of the key.

Self-Check Questions.

1. Name two algorithms that satisfy Kerckhoff's principle. DES *and* AES.

Summary

Cryptography protects information by shuffling data (that is, transforming it from intelligible into indecipherable data) so that only additional secret information, the *key*, can feasibly reverse it. Up to the end of the '70s, the key used to encrypt and decrypt was always the same: *symmetric* or (*single-key*) cryptography. In the 70s *asymmetric cryptography* was invented, in which the key to encipher (*the public key*) and the key to decipher (the *secret* or *private* key) are different. In fact, only the key to decipher is private, kept secret, while the key to encrypt is public, known to everyone. When the keys exchange their roles, the private key enciphers, and the public one deciphers, then the encryption is a *digital signature.* while the encrypted message will no longer be

secret, every owner of the public key can check whether the original message was encrypted by the private key. Because historically only written messages were encrypted, the source data, though a stream of 1s and 0s (the viewpoint adopted in symmetric cryptography) or a number (that adopted in asymmetric cryptography), is called *plaintext* and the encrypted data the *ciphertext.*

The security

- of public-key algorithm relies on the inefficient computation of mathematical functions on the integers. For example, the most famous public-key algorithm, RSA , requires the factorization of a number with > 500 decimal digits into its prime factors, which is computationally infeasible (without knowledge of the key);
- of symmetric cryptographic algorithms depends on the diffusion of small differences on the input and key to large differences in the output; ideally, if one bit of the input or key changes, then about half of the bits of the output changes.

Good encryption, as achieved by standard algorithms such as AES or RSA, is practically impossible to break computationally; instead, keys are stolen or plaintext is stolen before encryption or after decryption.

A hash function is an algorithm that generates an output of fixed (byte) size (usually around 16 to 64 bytes) from an input of variable (byte) size, for example, a text or image file, a compressed archive. The output string of fixed length that a cryptographic hash function produces from a string of any length (an important message, say) is a kind of inimitable "signature". A person who knows the "hash value" cannot know the original message, but only the person who knows the original message can prove that the "hash value" is produced from that message.

Questions

- How many keys has Caesar's Cipher (excluding the trivial one)? 12, 13, 25 , 26
- For which non-trivial key is Caesar's Cipher idempotent, that is, encrypting the ciphertext again yields the plaintext? 1, 2, 26, 13
- Which cryptographic algorithm comes closest to satisfying Kerckhoff's principle? Scytale, Enigma, ADFGVX, One-time pad

- Which one is an encryption algorithm? MD5, SHA-1, *AES*, CRC
- Which one is a hash function that is not cryptographic? MD5, SHA-1, *CRC*, AES

Required Reading

The article Simmons et al. (2016) gives a good summary of cryptology, in particular, historically; read its introduction and the section on history. As does the first chapter of Menezes, Oorschot, and Vanstone (1997), which focuses more on the techniques. The most recent work is Aumasson (2017), and a concise but demanding overflight of modern cryptography. Get started by reading its first chapter as well.

Further Reading

Some classics are Frederick) Friedman (1976) which is a manual for cryptanalysis for the U.S. military, originally not intended for publication.

The books Kahn (1996) and Singh (2000) trace out the history of cryptanalysis in an entertaining way.

The book Schneier (2007) is a classic for anyone interested in understanding and implementing modern cryptographic algorithms.

2 Symmetric ciphers

Study Goals

On completion of this chapter, you will have learned ...

- ... that the two fundamental symmetric cryptographic algorithms are
  - substitution that replaces the alphabet of the plaintext by an alphabet of the ciphertext (such as Caesar's cipher), and
  - transposition (or *permutation*) that transposes the letters of the plaintext (such as the Scytale).
- ... that the only cryptographically perfectly secure cipher is the *one-time* pad in which the key is as long as the plaintext
- ... that modern algorithms like DES and AES are *Substitution and Permutation Networks* that break the plaintext up into short blocks of the same size as the key and, on each block, iterate
  1. addition of the key,
  2. substitution, and
  3. permutation.

Introduction

Up to the end of the '70s, before the publication of Diffie and Hellman (1976) and Rivest, Shamir, and Adleman (1978), all (known) cryptographic algorithms were *symmetric* (or *single-key*), that is, used the same key to encipher and decipher. Thus every historic algorithm, as sophisticated as it may be, be it Caesar's Cipher, the Scytale or the Enigma, was symmetric.

While asymmetric algorithms depend on a computationally difficult problem, such as the factorization of a composed number into its prime factors, and regard the input as a natural number, symmetric ones operate on the input as a string (of bits or letters) by (iterated) substitutions and transpositions.

The only perfectly secure cipher is the *one-time pad* in which the key is as long as the plaintext and the ciphertext is obtained by adding, letter by letter, each letter of the key to the corresponding (that is, at the same position) letter of the plaintext.

However, such a large key is impractical for more complex messages, such as text, image or video files: In modern times, it means that to encrypt a hard drive, another hard drive that carries the key is needed.

To compensate the shorter key length, modern algorithms, ideally, create so much intertwining that they achieve almost *perfect diffusion*, that is, the change of a single bit of the input or key causes the change of around half of the output bits. Modern algorithms, such as `DES` or `AES`, are substitution and permutation network block ciphers, meaning that they encrypt one chunk of data at a time by iterated transpositions and substitutions.

2.1 Substitution and Transposition

The two basic operations to encrypt are *transposition* and *substitution*:

- A transposition changes the order (that is, transposes or permutes) of the symbols in the text but not the symbols themselves.
- A substitution replaces (that is, substitutes) every symbol in the text by another (group of) symbol, but not the order of the symbols.

The historical prototypical algorithms for these two operations are:

- the substitution cipher by Caesar, that advances every letter in the plaintext by the three positions, that is, encrypts `A` as `D`, `B` as `E`, and so forth, and
- the permutation of the plaintext by the scytale, or baton of Licurgo (Spartan lawgiver around the ninth century BC), where the parchment is wrapped around the baton and the text written on it horizontally.

We will see that even with many possible keys an algorithm, such as that given by any permutation of the alphabet which has almost 2^{80} keys, can be easily broken if it preserves regularities, like the frequency of the letters.

As a criterion for security, there is that of diffusion by Shannon: Ideally, if a letter in the plaintext changes, then half of the letters in the ciphertext changes. Section 2.2 will show how modern algorithms, called *substitution and permutation networks*, join and iterate these two complementary prototypical algorithms to reach this goal.

> **ideal diffusion** (according to Shannon): if a bit in the plaintext or key changes, then half of the bits in the ciphertext changes.

Substitution ciphers. In a **substitution cipher** the key determines substitutions of the plaintext alphabet (considered as a set of units of symbols such as single letters or pairs of letters) by the ciphertext alphabet. For example, if the units of the plaintext and ciphertext are both the letters of the Latin alphabet, then a substitution permutes the letters of the Latin alphabet. If the substitution cipher is *monoalphabetic* (such as Caesar's Cipher), then the same substitution is applied to every letter of the plaintext independent of its position. If the substitution cipher is *polyalphabetic* (such as the Enigma), then the substitution varies with the position of the letter in the plaintext. To encrypt, each alphabetical unit of the plaintext is replaced by the substituted alphabetical unit, and inversely to decrypt.

> **Substitution Cipher**: a cipher that replaces each alphabetical unit of the plaintext by a corresponding alphabetical unit.

Every **monoalphabetic** substitution cipher, that is, every plaintext symbol is always encrypted into the same ciphertext symbol, is insecure: the frequency distributions of symbols in the plaintext and in the ciphertext are identical, only

the symbols having been relabeled. Therefore, for example in English, around 25 letters of ciphertext suffice for cryptanalysis.

The main approach to reduce the preservation of the single-letter frequencies in the ciphertext is to use several cipher alphabets, that is, *polyalphabetic substitution.*

Shift by a fixed distance. The simplest substitution cipher is a cyclical shift of the plaintext alphabet; **Caesar's cipher**.

> **Caesar's Cipher** A substitution cipher that shifts the alphabetical position of every plaintext letter by the same distance.

This method was used by Roman emperors Caesar (100 – 44 B.C.) and Augustus (63 – 14 B.C.): fix a distance d between letters in alphabetical order, that is, a number between 0 and 25, and shift (forward) each letter of the (latin) alphabet by this distance d . We imagine that the alphabet is circular, that is, that the letters are arranged in a ring, so that the shift of a letter at the end of the alphabet results in a letter at the beginning of the alphabet.

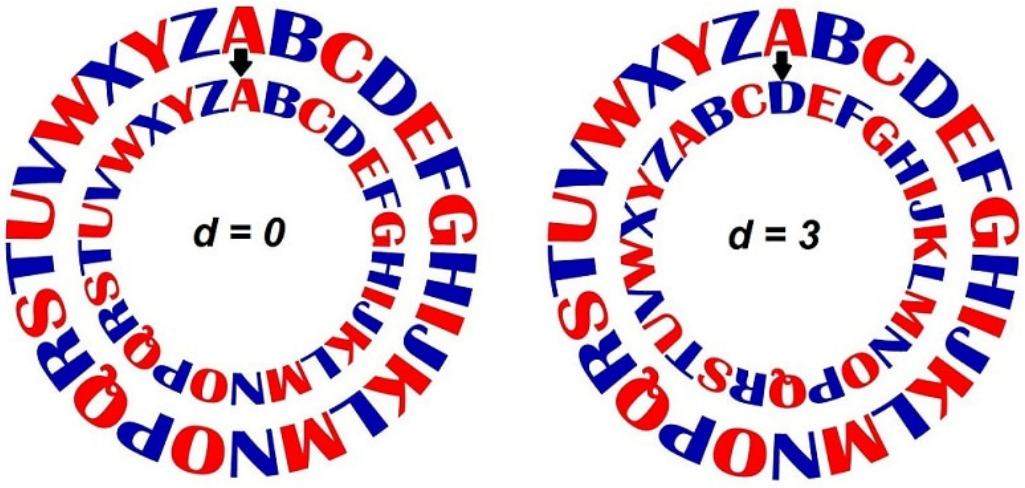


Figure 6: We imagine that the letters of the alphabet form a wheel (Simply-Science.ch (2014))

For example, if $d = 3$, then

$$A \mapsto D, B \mapsto E, \ldots, W \mapsto Z, X \mapsto A, \ldots, Z \mapsto C.$$

There are 26 keys (including the trivial key $d = 0$).

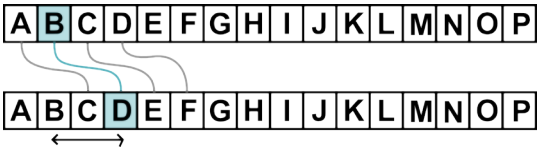


Figure 7: Caesar displaces each letter of the alphabet by the same distance

To decipher, each letter is shifted by the negative distance $-d$, that is, d positions backwards. If the letters of the alphabet form a wheel, then the letters are transferred

- clockwise during the encipherment, and
- counterclockwise during the decipherment.

By the cyclicity of the letter arrangement, we observe that a transfer of d positions in counterclockwise direction equals one of $26-d$ positions in clockwise direction.

Substitution by Permutation of the letters of the alphabet. Instead of replacing each letter by one shifted by the same distance d , let us replace each letter with some letter, for example:

A	B	...	Y	Z
↓	↓	...	↓	↓
E	Z	...	G	A

To revert the encipherment, never two letters be sent to the same letter! That is, we shuffle the letters among themselves. This way we obtain $26 \cdot 25 \cdot s1 = 26! > 10^6$ keys (which is around the number of passwords with 80 bits).

Transposition (or Permutation) ciphers. A **transposition** (or **permutation**) **cipher** encrypts the plaintext by permuting its units (and decrypts by the inverse permutation). Each alphabetical unit stays the same; the encryption depends only on the positions of the units.

Transposition Cipher: Transpose the alphabetical units of the plaintext.

The *Scytale* or *Licurgo's Baton* (= a Spartan legislator around 800 B.C.) is a cipher used by the Spartans, as follows:

1. wrap a stick into a narrow strip,
2. write on this strip horizontally, that is, along the larger edge, and
3. unroll the strip.

The letters thus transposed on the strip could only be deciphered by a stick with the same *circumference* (and being long enough) in the same way as the text was encrypted:

1. wrap the stick into the strip, and
2. read this strip horizontally, that is, along the larger edge.

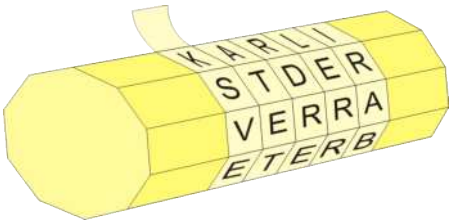


Figure 8: The skytale encrypting a military order in English (BeEsCommonsWiki (2015))

Here, the key is given by the stick's circumference, that is, the *number of letters that fit around the stick.*

For example, if the stick has a circumference of 2 letters (and a length of 3 letters), the two rows

B	I	G
S	U	M

become the three rows

B	S
I	U

G	M

which, once concatenated (to reveal neither the circumference nor the length), become

B	S	I	U	G	M

Security of Historical Examples. Let us apply the established security criteria to the substitution ciphers:

Caesar's Cipher. This simple substitution cipher violates all desirable qualities: For example, Kerckhoff's *principle* that the algorithm be public: Once the method is known, considering the small amount of *25* keys, the ciphertext gives way in short time to a **brute-force** attack:

> **brute-force** attack: an exhaustive key-search that checks each possible key.

Substitution by any permutation of the letters of the Alphabet. A substitution by any permutation of the letters of the alphabet, such as,

A	B	…	Y	Z
↓	↓	…	↓	↓
E	Z	…	G	A

has

$$26 \cdot 25 \cdots 1 = 26! > 10^{26}$$

keys, so a brute-force attack is computationally infeasible.

But it violates the goals of *diffusion and confusion.* If the key (= permutation of the alphabet) exchanges the letter α for the letter β , then there's

- bad *confusion* because the substitution of β in the key implies only the substitution of each letter β in the ciphertext,

- bad *diffusion* because the substitution of a letter α in the plaintext implies only the substitution of the corresponding letter β in the ciphertext.

In fact, the algorithm allows statistical attacks on the frequency of letters, bigrams (= pairs of letters) and trigrams (= triples of letters). See Section 12.1.

The Scytale. Also the scytale is weak in any sense given by the security principles. It violates

- the *Kerckhoff principle* that the algorithm be public.

In fact, the maximum value of the circumference of the stick in letters is $< n/2$ where n = the number of letters in the ciphertext. So a brute-force attack is feasible.

It has

- bad *diffusion* because the substitution of a letter α in the plaintext only implies that of the same letter α in the ciphertext.

In fact, the algorithm is prone to statistical attacks on the frequency of bigrams (= pairs of letters), trigrams (= triples of letters), and higher tuples. For example, a promising try would be the choice of circumference as number c that maximizes the frequency of the 'th' bigram between the letter strings at positions $1, 1+c, 1+2c, \ldots$, $2, 2+c, 2+2c, \ldots$. For example, if we look

TEHMHTUB

we notice that `T` and `H` are one letter apart, which leads us to the guess that the circumference is three letters, $c = 3$, yielding the decipherment

THE THUMB.

Product ciphers. A **product cipher** composes ciphers, that is, if the product is two-fold, then the output of one cipher is the input of the other.

> **product cipher**: a composition of ciphers where the output of one cipher serves as the input of the next.

The ciphertext of the product cipher is the ciphertext of the final cipher. Combining transpositions only with transpositions or substitutions only with substitutions, the obtained cipher is again a transposition or substitution, and hardly more secure. However, mixing them, a transposition with substitutions, indeed can make the cipher more secure.

A *fractionation cipher* is a product cipher that:

1. substitutes every symbol in the plaintext by a group of symbols (usually pairs),
2. transposes the obtained ciphertext.

The most famous fractionation cipher was the `ADFGVX` cipher used by the German forces during World War I:

	A	D	F	G	V	X
A	a	b	c	d	e	f
D	g	h	i	j	k	l
F	m	n	o	p	q	r
G	s	t	u	v	w	x
V	y	z	0	1	2	3
X	4	5	6	7	8	9

1. The 26 letters of the Latin Alphabet and 10 digits were arranged in a 6×6 -table and replaced by the pair of letters among `A` , `D` , `F` , `G` , `V` , and `X` that indicate the row and column of the letter or digit.
2. The resulting text was written as usual from left to right into the rows of a table and then each column read in the order indicated by a keyword.

However, it was cryptanalyzed within a month by the French cryptanalyst Georges J. Painvin in 1918 when the German army entered in Paris. We will see in Section 2.2 how modern ciphers refine this idea of a product cipher to obtain good diffusion.

Self-Check Questions.

1. For which distances d is Caesar's Cipher auto-inverse, that is, the output of the encipherment equals that of the decipherment? *For* $d = 0$ *and* 13 .

2. Does Caesar's Cipher satisfy Kerckhoff's principle? No, the number of possible keys is too small.
3. Why is a substitution cipher insecure? Because two identical letters in the plaintext are replaced by two identical letters in the ciphertext.

2.2 Block Ciphers

Classic ciphers usually replaced single letters, sometimes pairs of letters. Systems that operated on trigrams or larger groups of letters were regarded as too tedious and never widely used.

Instead, it is safer to substitute a whole block (of letters instead of a single letter, say) according to the key. However, the alphabet of this replacement would be gigantic, so this ideal is practically unattainable, especially on hardware as limited as a smart card with an 8 bit processor. For a block of, for example, 4 bytes, this substitution table would already have a 4 gigabytes ($= 2^{32} \cdot 4$ bytes). However, in modern single-key cryptography a block of information commonly has 16 bytes, about 27 alphabetic characters (whereas two-key cryptography based on the RSA algorithm commonly uses blocks of 256 bits, about 620 alphabetic characters).

Instead, for example, AES only replaces each byte, each entry in a block, a replacement table of $2^8 = 256$ entries of 1 byte (and afterwards transposes the entries.) We will see that these operations complement each other so well that they are practically as safe as a substitution of the whole block.

Block and Stream Ciphers.

A **block cipher** partitions the plaintext into blocks of the same size and enciphers each block by a common key: While a block could consist of a single symbol, normally it is larger. For example, in the Data Encryption Standard the block size is 64 bits and in the Advanced Encryption Standard 128 bits.

> **stream** cipher versus **block** cipher: a stream cipher operates on single characters (for example, single bytes) while a block cipher operates on groups of characters (for example, each 16 bytes large)

A **stream cipher** partitions the plaintext into units, normally of a single character, and then encrypts the i -th unit of the plaintext with the i -th unit of a key

stream. Examples are the one-time pad, rotor machines (such as the Enigma) and DES used in Triple DES (in which the output from one encryption is the input of the next encryption).

In a stream cipher, the same section of the key stream that was used to encipher must be used to decipher. Thus, the sender's and recipient's key stream must be synchronized initially and constantly thereafter.

Feistel Ciphers. A **Feistel Cipher** (after Horst Feistel, the inventor of DES) or a **substitution and permutation network** (SPN) groups the text (= byte sequence) into n -byte blocks (for example, $n = 16$ for AES and enciphers each block by iteration (for example, 10 times in AES , and 5 times in our prototypical model) of the following three steps, in given order:

1. add (XOR) the key,
2. substitute of the alphabet (which operates in sub-blocks of the block, for example, on each byte), and
3. permute of all the sub-blocks in a block.

> **Substitution and Permutation Network**: a cipher that iteratively substitutes and permutes each block after adding a key.

That is, after

1. the addition (by Or Exclusive) of the key as in the One-time pad,

are applied

2. the substitution of the alphabet, for example, in the AES algorithm (each byte, pair of hexadecimal letters, by another), and
3. the permutation of the text (from the current step, the *state*); for example, in AES , that groups the text into a 4×4 square (whose entries are pairs of hexadecimal letters), the entries in each row (and the columns) are permuted.

These two simple operations,

- the substitution of the alphabet, and
- the permutation of text

complement each other well, that is, they generate high confusion and diffusion after a few iterations. In the first and last round, the steps before respectively after the addition of the key are omitted because they do not increase the cryptographic security: Since the algorithm is public (according to Kerckhoff's principle), any attacker is capable of undoing all those steps that do not require knowledge of the key.

Though seemingly a Feistel Cipher differs from classical ciphers, it is after all a product cipher, made up of transpositions and substitutions.

Self-Check Questions.

1. What distinguishes a stream cipher from a block cipher? *a stream cipher operates on single characters while a block cipher operates on groups of characters*
2. What is a Substitution and Permutation Network (or Feistel Cipher)? *A block cipher that iteratively substitutes and permutes each block after adding a key.*

2.3 Data Encryption Standard (DES)

The Data Encryption Standard (**DES**), was made a public standard in 1977 after it won public competition announced by the U.S. National Bureau of Standards (NBS; now the National Institute of Standards and Technology, NIST). IBM (International Business Machines Corporation) submitted the patented Lucifer algorithm invented by one of the company's researchers, Horst Feistel, a few years earlier (after whom the substitution and permutation network was labelled Feistel Cipher). Its internal functions were slightly altered by the NSA (and National Security Agency) and the (effective) key size shortened from 112 bits to 56 bits, before it became officially the new Data Encryption Standard.

> **DES**: Block cipher with an effective key length of 56 bits conceived by Horst Feistel from IBM that won a U.S. National competition to become a cryptographic standard in 1977.

DES is a product block cipher of 16 iterations, or rounds, of substitution and transposition (permutation). Its block size and key size is 64 bits. However, only 56 of the key bits can be chosen; the remaining eight are redundant parity check bits.

As the name of its inventor Horst Feistel suggests, it is a Feistel Cipher, or substitution and permutation network, similar to the prototype discussed above. It groups the text (= byte sequence) into 32-bit blocks with sub-blocks of 4 bits and enciphers each block in 16 iterations of the following three steps, called the *Feistel function*, for short *F-function*, in given order:

1. add (XOR) the key,
2. substitution of each 4-bit sub-blocks of the block by the S-box (in hexadecimal notation), and

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

3. permutation of all the sub-blocks.

At each round i , the output from the preceding round is split into the 32 left-most bits, $L(i)$, and the 32 right-most bits, $R(i)$. $R(i)$ will become $L(i+1)$, whereas $R(i+1)$ is the output of a complex function, $L(i) + f(R(i), K(i+1))$ whose input is

- the $i+1$ -th block of the key bits, $K(i+1)$, and
- of the entire preceding intermediate cipher.

This process is repeated 16 times.

Essential for the security of DES is the non-linear S-box of the F -function f specified by the Bureau of Standards; it is not only non-linear, that is, $f(A) + f(B) \neq f(A+B)$ but maximizes confusion and diffusion as identified by Claude Shannon for a secure cipher in Section 2.1.

Key Size and the Birth of 3DES. The security of the DES like any algorithm is no greater than the effort to search 2^{56} keys. When introduced in 1977, this was considered an infeasible computational task, but already in 1999 a special-purpose computer achieved this in three days. A workaround, called "Triple DES" or **3DES**, was devised that effectively gave the DES a 112-bit key (using two normal DES keys).

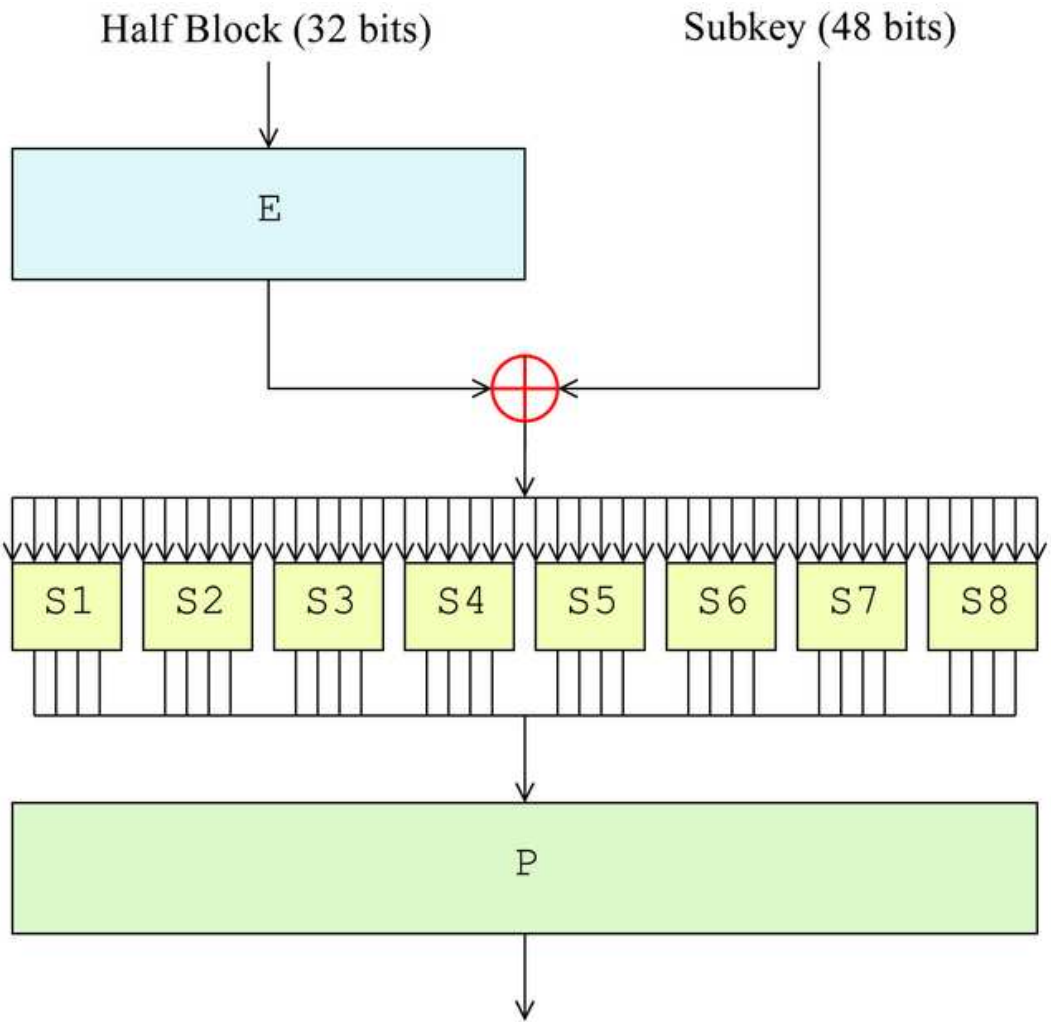


Figure 9: The DES F-function (Hellisp (2014))

3DES: Triple application of `DES` to double the key size of the DES algorithm.

(Which is after all the key size for the algorithm originally proposed by IBM for the Data Encryption Standard.) The encryption would be $E(1) \circ D(2) \circ E(1)$ while decryption would be $D(1) \circ E(2) \circ D(1)$, that is, the encryption steps are:

1. Encrypt by the first key,
2. Decrypt by the second key, and

3. Encrypt by the first key;

while the decryption steps are:

1. Decrypt by the first key,
2. Encrypt by the second key, and
3. Decrypt by the first key.

If the two keys coincide, then this cipher becomes an ordinary single-key DES; thus, triple DES is backward compatible with equipment implemented for (single) DES.

DES is the first cryptographic algorithm to fulfill Kerckhoff's principle of being public: every detail of its implementation is published. (Before, for example, the implementation records of the Japanese and German cipher machines in World War II were released only half a century after their cryptanalysis.)

Shortly after its introduction as a cryptographic standard, the use of the DES algorithm was made mandatory for all (electronic) financial transactions of the U.S. government and banks of the Federal Reserve. Standards organizations worldwide adopted the DES, turning it into an international standard for business data security. It only waned slowly after its successor AES was adopted around 2000 (after its shortcomings became more and more apparent and could only be worked around, by provisional means such as 3DES).

Self-Check Questions.

1. What key size does DES use?

    - ☐ 256 bits
    - ☐ 128 bits
    - ☐ 112 bits
    - ☐ *56 bits*

2. Name a cryptographic weakness of DES: *Short key length.*
3. What does 3DES stand for? *Tripe DES, that is, tripe application of DES.*
4. What key size does 3DES use?

    - ☐ 256 bits
    - ☐ 128 bits

- ☐ *112 bits*
- ☐ 56 bits

2.4 Advanced encryption standard (AES)

In January 1997 the U.S. National Institute of Standards and Technology (NIST; former National Bureau of Standards, NBS) announced a public contest (National Institute for Standards and Technology (2000)) for an *Advanced Encryption Standard* (**AES**) to replace the former symmetric encryption standard, the Data Encryption Standard (DES). Since improvements in computing power allowed to find the fixed 56-bit DES key by exhaustive key-search (brute force) in a matter of days, the NIST specifications for the AES demanded an ever increasable key length, if ever need be. The winner of this competition, the algorithm that became the AES, was Rijndael (named after its creators Vincent Rijmen and Joan Daemen):

- Rijndael: 86 positive votes, 10 negative votes.
- Serpent: 59 votes in favour, 7 against.
- Twofish: 31 positive, 21 negative votes
- RC6: 23 positive, 37 negative votes
- MARS: 13 votes in favour, 84 against.

> **AES**: Substitution and Permutation network with a (variable) key length of usually 128 bits conceived by Vincent Rijmen and Joan Daemen that won a U.S. National competition to become a cryptographic standard in 2000 and succeed DES.

Evaluation of Security. The creators of AES could demonstrate in Daemen and Rijmen (1999) that these two operations complement each other so well that, after several iterations, they almost compensate for the absence of a replacement of the entire block by another. For a more detailed source, see Daemen and Rijmen (2002).

As was the case with DES, the AES, decades after its introduction, still stands strong against any attacks of cryptanalysis, but foreseeably will not yield to developments in computing, as happened to the DES, also thanks to its adjustable key size.

Applications. Among the competitors of public contest by the NIST, none of them stood out for its greater security, but Rijndael for its simplicity, or clarity, and in particular computational economy in implementation. Since this algorithm is to be run everywhere, for example on 8-bit smart card processors (smartcards), the decision was made in favour of Rijndael. Rijndael not only was secure, but thanks to its elegant and simple design, also both small enough to be implemented on smart cards (at less than 10,000 bytes of code).

To this day, this algorithm remains unbroken and is considered the safest; there is no need for another standard symmetric cryptographic algorithm. And indeed, it runs everywhere: For example, to encrypt a wireless network, a single key is used, so the encryption algorithm is symmetrical. The safest option, and therefore most recommended, is AES.

Encipherment in Blocks. The AES algorithm is a block cipher, that is, it groups the plaintext (and the keys) into byte blocks of $4 \times$ B-byte rectangles where

$$\text{B} := \text{number of columns in the rectangle} = 4, 6 or 8.$$

Commonly, and for us from now on, B = 4 , that is, the rectangle is a 4×4-square (containing 16 bytes or, equivalently, 128 bits). Each entry of the block is a byte (= sequence of eight binary digits = eight-bit binary number).

On a hexadecimal basis (= whose numbers are 0 – 9, A = 10, B = 11, C = 12, D = 13, E = 14 and F = 15), such a square is for example

A1	13	B1	4A
A3	AF	04	1E
3D	13	C1	55
B1	92	83	72

Rounds. The AES algorithm enciphers each byte block B iteratively, in a number of rounds R which depends on the number of columns of B: there are R = 10 rounds for B = 4 columns, R = 12 rounds for B = 6 columns and R = 14 rounds for B = 8 columns. For us, as we assume B = 4 columns, R = 10.

The Substitution and Permutation cipher AES operates repeatedly as follows on each block:

1. Substitute each byte of the (square) block according to a substitution table (S-box) with $2^8 = 256$ entries of 1 byte each.
2. Permute the entries of each row, and
3. exchange the entries (= bytes = eight-digit binary numbers) of each column by sums of multiples of them.
4. Add the round key, which is a byte block of the same size, to the block (by XOR in each entry).

The first step is a substitution. The second and third step count as a horizontal permutation (of the entries in each row) respectively vertical permutation (of the entries in each column).

CrypTool 1 offers in Menu Individual Procedures -> Visualization of Algorithms -> AES

- an Animation entry to see the animation in Figure 10 of the rounds, and
- an Inspector entry in Figure 12 to experiment with the values of plaintext and key.

In these rounds, keys are generated, the plaintext replaced and transposed by the following operations:

1. Round $r = 0$:
    - AddRoundKey to add (by XOR) the key to the plaintext (square) block
2. Rounds $r = 1, \ldots, R-1$: to encrypt, apply the following functions:
    1. SubBytes to replace each entry (= byte = sequence of eight bits) with a better distributed sequence of bits,
    2. ShiftRows to permute the entries of each row of the block,
    3. MixColumn to exchange the entries (= bytes = eight-digit binary number) of each column of the block by sums of multiples of them,
    4. AddRoundKey to generate a key from the previous round's key and add it (by XOR) to the block.
3. Round $r = R$: to encrypt, apply the following functions:
    1. SubBytes
    2. ShiftRows
    3. AddRoundKey

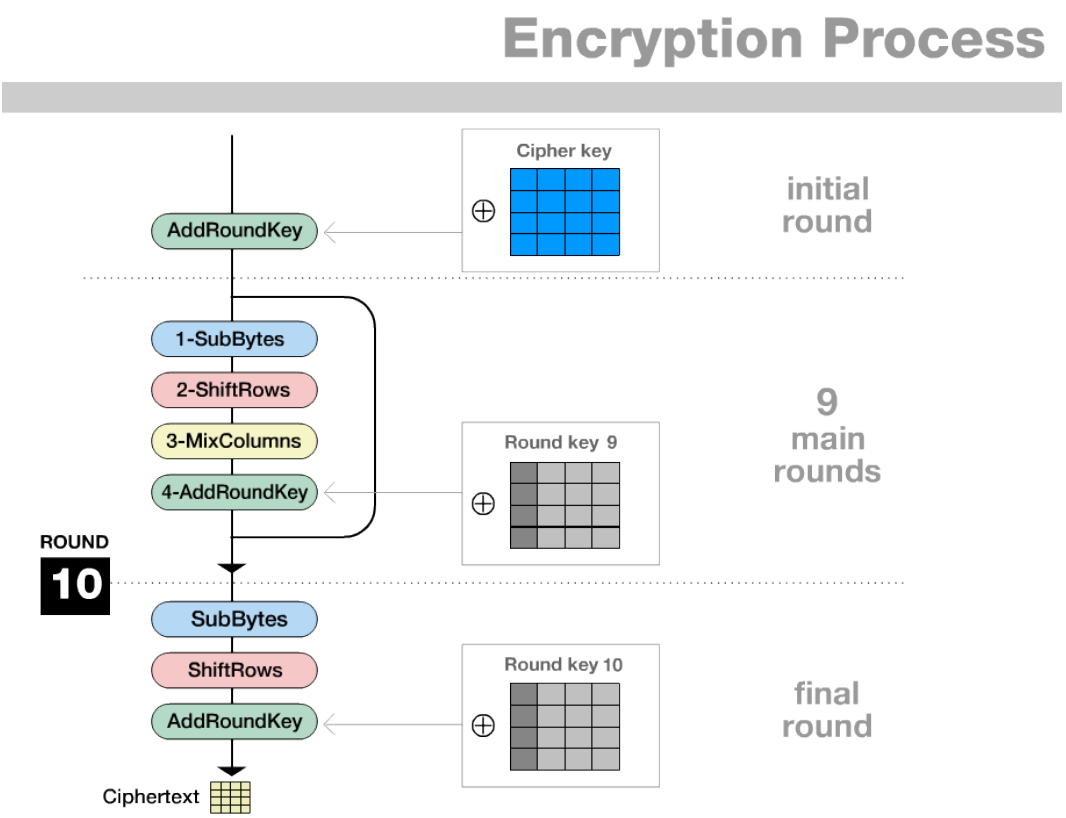


Figure 10: The AES rounds in CrypTool 1 (Zabala (2019a))

That is, compared to previous rounds, the MixColumn function is omitted: It turns out that MixColumn and AddRoundKey, after a slight change of AddRoundKey, can change the order without changing the end result of both operations. In this equivalent order, the operation MixColumn does not increase cryptographic security, as the last operation invertible without knowledge of the key. So it can be omitted.

The function MixColumn (and at its origin SubBytes) uses the multiplication of the so-called Rijndael field $\mathbb{F}_{2^8}$ to compute the multiple of (the eight-digit binary number given by) a byte; it will be presented at the end of this chapter. Briefly, the field defines, on all eight-digit binary numbers, an addition given by XOR and a multiplication given by a polynomial division with remainder: The eight-digit binary numbers $a = a_7 \ldots a_0$ and $b = b_7 \ldots b_0$ to be multiplied are

identified with polynomials

$$a(x) = a_7x^7 + \cdots + a_0 \quad \text{and} \quad b(x) = b_7 + \cdots + b_0$$

which are then multiplied as usual to give a polynomial $C(x) = a(x)b(x)$. To yield a polynomial with degree ≤ 7, the remainder $c(x) = c_7x^7 + \cdots + c_0$ of $C(x)$ by polynomial division with

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

is computed. The product $c = a \cdot b$ is then given by the coefficients $c_7 \ldots c_0$.

Let us describe all round functions in more detail:

SubBytes. `SubBytes` substitutes each byte of the block by another byte given by the `S-box` substitution table.

To calculate the value of the entry by which the `S-box` substitutes each byte:

1. Calculate its multiplicative inverse B in $\mathbb{F}_{2^8}$,
2. Calculate

$$a_i = b_i + b_{i+4} + b_{i+5} + b_{i+6} + b_{i+7} + c_i$$

where $i = 0, 1, \ldots, 7$ is the index of each bit of a byte, and

- B = $b_7b_6b_5b_4b_3b_2b_1b_0$ is the entry byte,
- A = $a_7a_6a_5a_4a_3a_2a_1a_0$ is the output byte of the operation and
- c is the constant byte 01100011.

In matrix form,

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

in hexadecimal notation (where the row number corresponds to the first hexadecimal digit and the column number to the second hexadecimal digit of the byte to be replaced):

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
A	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
B	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
C	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
D	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
E	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
F	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

ShiftRows. `ShiftRows` shifts the l-th row (counted starting from zero, that is, l runs through 0 , 1 , 2 and 3; in particular, the first row is *not* shifted) l positions to the left (where shift is cyclic). That is, the (square) block with entries

B_{00}	B_{01}	B_{02}	B_{03}
B_{10}	B_{11}	B_{12}	B_{13}
B_{20}	B_{21}	B_{22}	B_{23}
B_{30}	B_{31}	B_{32}	B_{33}

is transformed into one with entries

B_{00}	B_{01}	B_{02}	B_{03}
B_{11}	B_{12}	B_{13}	B_{10}

B_{22}	B_{23}	B_{20}	B_{21}
B_{33}	B_{30}	B_{31}	B_{32}

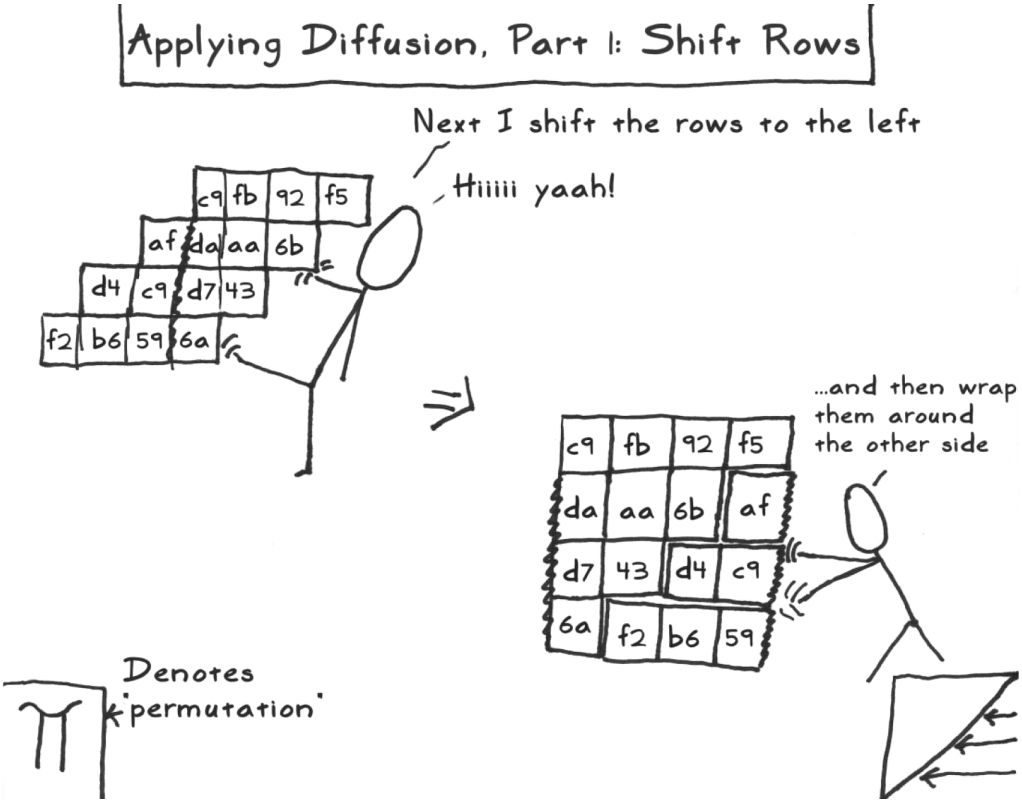


Figure 11: Transposition in AES algorithm (Moser (2009))

MixColumn. `MixColumn` exchanges all entries of each column of the block by a sum of multiples of them. This is done by multiplying each column by a fixed matrix. More exactly,

- if B_j (with coefficients $b_{0,j}$, $b_{1,j}$, $b_{2,j}$ and $b_{3,j}$) corresponds to the column j of the input block, and
- if A_j (with coefficients $a_{0,j}$, $a_{1,j}$, $a_{2,j}$ and $a_{3,j}$) corresponds to the column j of the output block of the operation,

then

$$\begin{pmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{pmatrix}$$

For example, the byte $a_{0,j}$ is computed by

$$a_{0,j} = 2 \cdot b_{0,j} + 3 \cdot b_{1,j} + b_{2,j} + b_{3,j}$$

AddRoundKey. `AddRoundKey` adds, by the `XOR` operation, the key $\mathrm{W}(r)$ of the current round r to the current block B of the ciphertext, the status, that is, B is transformed into

$$\mathrm{B} \oplus \mathrm{W}(r).$$

The key is generated column by column. We denote them by $\mathrm{W}(r)_0$, $\mathrm{W}(r)_1$, $\mathrm{W}(r)_2$ and $\mathrm{W}(r)_3$; that is,

$$\mathrm{W}(r) = \mathrm{W}(r)_0 \mid \mathrm{W}(r)_1 \mid \mathrm{W}(r)_2 \mid \mathrm{W}(r)_3.$$

Since the key has 16 bytes, each column has 4 bytes.

1. The first round key $\mathrm{W}(0)$ is given by the initial W key.
2. For $r = 1, \ldots,$ R (where R is the total number of rounds, R = 10 for us), the four columns $\mathrm{W}(r)_0$, $\mathrm{W}(r)_1$, $\mathrm{W}(r)_2$ and $\mathrm{W}(r)_3$ of the new key are generated from the columns of the old $\mathrm{W}(r-1)$ key as follows:
    1. The first column $\mathrm{W}(r)_0$ is given by

    $$\mathrm{W}(r)_0 = \mathrm{W}(r-1)_0 \oplus \mathrm{ScheduleCore}(\mathrm{W}(r-1)_3);$$

    that is, the last column of the previous round key plus the result of `ScheduleCore` applied to the first column of the previous round key (which we denote by T); here `ScheduleChore` is the composition of transformations:
        1. `SubWord` : Substitutes each of the 4 bytes of T according to the `S-box` of `SubBytes` .
        2. `RotWord` : Shift T one byte to the left (in a circular manner, that is, the last byte becomes the first).

3. `Rcon(r)`: Adds (by `XOR`) to T the constant value, in hexadecimal notation, $(02)^{r-1}000000$ (where the power, that is, the iterated product, is calculated in the `Rijndael field` $\mathbb{F}_{2^8}$). That is, the only byte that changes is the first one, by adding either the value 2^{r-1} (for $r \leq 8$) or the value 2^{r-1} in F_{2^8} for $r = 9, 10$.

2. The next columns $\mathrm{W}(r)_1$, $\mathrm{W}(r)_2$ and $\mathrm{W}(r)_3$ are given, for $i = 1, 2$ and 3 , by

$$\mathrm{W}(r)_i = \mathrm{W}(r)_{i-1} \oplus \mathrm{W}(r-1)_i;$$

that is, the previous column of the current round key plus the current column of the previous round key.

Diffusion. We note that the only transformation that *is not affine* (that is, the composition of a linear application and a constant shift) is the multiplicative inversion in the Rijndael field $\mathbb{F}_{2^8}$ in the `SubBytes` operation. In fact

1. In the operation `SubBytes` are applied, in this order,
    1. the inversion in $\mathbb{F}_{2^8}$,
    2. a linear application, and
    3. the translation by a constant vector.
2. `ShiftRows` is a permutation, in particular, linear.
3. `MixColumn` is an addition, in particular, linear.
4. `AddRoundKey` is the translation by the round key.

Regarding the goals of ideal *diffusion* and *confusion*, we can point out that in each step about half of the bits (in `SubBytes`) or bytes (in `MixColumn` and `ShiftRows`)is replaced and transposed. To convince oneself of the complementarity of the simple operations for high security, that is, that they generate in conjunction high confusion and diffusion after few iterations:

- the substitution of the alphabet, and
- the permutation of text, in particular,
    - of the permutation between the entries of every *row*, and
    - of the permutation between the *column*,

it is worth to experiment in `Individual Procedures -> Visualization of Algorithms -> AES -> Inspector` with some pathological values, for example:

- All key and plaintext entries equal `00`, and
- all key entries equals `00` and plaintext entries equals `00` except one entry equals `01`, that is, change a single bit.

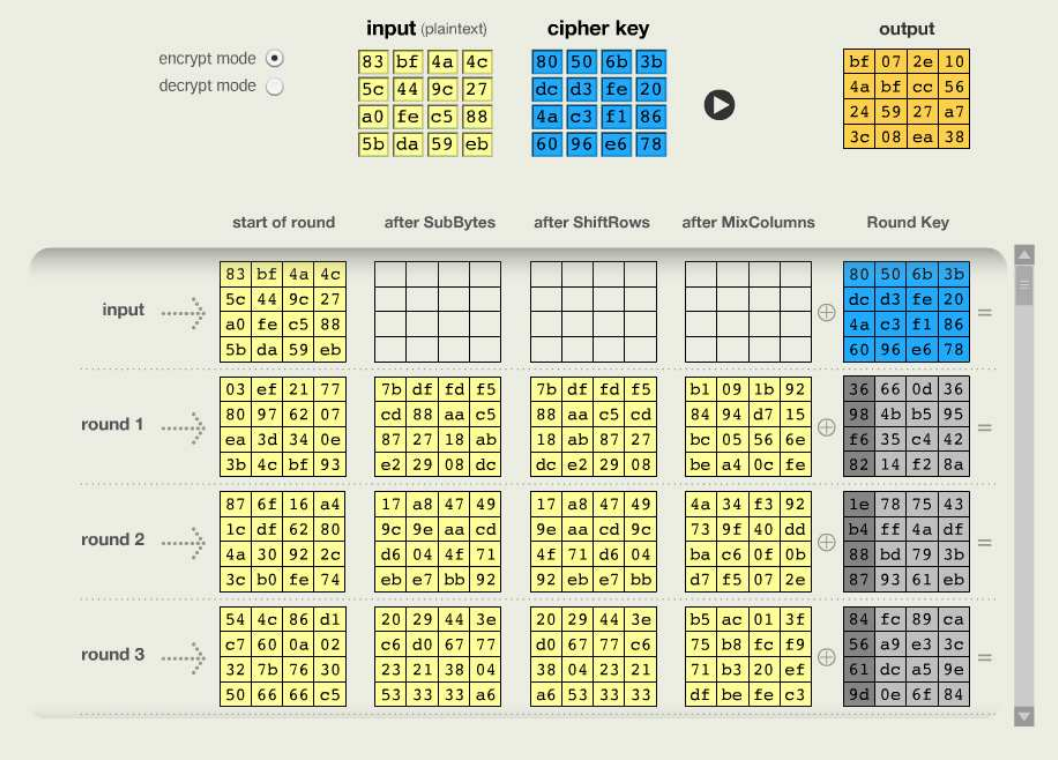


Figure 12: The values of the blocks along the rounds of `AES` in `CrypTool 1` (Zabala (2019b))

We see how this small initial difference spreads out, already generating totally different results after, say, four rounds! This makes plausible the immunity of `AES` against differential cryptanalysis.

In case all key and plaintext entries are equal to `00` , we also understand the impact of adding the `Rcon(r)` constant to the key in each round: that's where all the confusion comes from!

The Rijndael binary field. The function `MixColumn` (as well as the computation of the power of `Rcon` in `AddRoundKey`) uses the multiplication given by the so-called `Rijndael field`, denoted $\mathbb{F}_{2^8}$, to compute the multiple of (the number given by a) byte; let us quickly introduce it:

Groups and Fields. A **group** is a set G with

- an operation $\cdot : G \times G \to G$ that satisfies the associativity law,
- has a *neutral* element (that is, an element e such that $x \cdot e = x = e \cdot x$ for all x in G) and
- an *inverse* of every element (that is, for every x in G an element y such that $x \cdot y = e = y \cdot x$).

Generally,

- the operation is denoted by $\cdot$,
- the neutral element by 1 , and
- the inverse of x in G by x^{-1}.

Example. The set of nonzero rational numbers $\mathbb{Q}^*$ with the multiplication operation $\cdot$ is a group.

If the group G is commutative, that is, if the operation satisfies the commutativity law, then commonly

- the operation is denoted by $+$
- the neutral element by 0 , and
- the inverse element of x in G by $-x$.

Example. The set of rational numbers $\mathbb{Q}$ with the addition operation $+$ is a commutative group.

A **field** is a set F with an addition and multiplication operation $+$ and $\cdot$ such that

- the set F with $+$ is a commutative group,
- the set $F^* = F - \{0\}$ with $\cdot$ is a commutative group, and
- the distributivity law is satisfied.

Example. The set of rational numbers $\mathbb{Q}$ with addition $+$ and multiplication $\cdot$ is a field.

Bytes as Polynomials with Binary Coefficients of Degree 7. A byte, a sequence $b_7 \ldots b_1, b_0$ of eight bits in $\{0,1\}$ is considered a polynomial with binary coefficients by

$$b_7 \ldots b_1 b_0 \mapsto b_7 \mathrm{X}^7 + \cdots + b_1 \mathrm{X} + b_0$$

For example, the hexadecimal number $0x57$, or binary number 01010111 , corresponds to the polynomial

$$x^6 + x^4 + x^2 + x + 1.$$

All additions and multiplications in AES take place in the *binary field* $\mathbb{F}_{2^8}$ with $2^8 = 256$ elements, which is a set of numbers with addition and multiplication that satisfies the associativity, commutativity and distributivity law (like, for example, $\mathbb{Q}$) defined as follows: Let

$$\mathbb{F}_2 = \{0,1\}$$

be the *field of two elements* with

- *addition* $1 + 0 = 0 + 1 = 1$ and $0 + 0 = 1 + 1 = 0$ (which is the $\oplus$ addition given by XOR), and
- (the natural) *multiplication* $1 \cdot 0 = 0 \cdot 1 = 0 \cdot 0 = 0$ and $1 \cdot 1 = 1$.

Let

$$\mathbb{F}_2[\mathrm{X}] = \mathbb{Z}/2\mathbb{Z} = \text{the polynomials on } \mathbb{F}_2,$$

that is, the finite sums $a_0 + a_1 \mathrm{X} + a_2 \mathrm{X}^2 + \cdot s + a_n \mathrm{X}^n$ to a_0 , a_1 , …, a_n to $\mathbb{F}_2$ and be

$$\mathbb{F}_{2^8} := \mathbb{F}_2[\mathrm{X}]/(\mathrm{X}^8 + \mathrm{X}^4 + \mathrm{X}^3 + \mathrm{X} + 1).$$

That is, the result of both operations + and $\cdot$ in $\mathbb{F}_2\mathrm{X}$ is the remainder of the division by $\mathrm{X}^{8+\mathrm{X}^4+\mathrm{X}}3 + \mathrm{X} + 1$.

Addition. The + addition of two polynomials is the addition in $\mathbb{F}_2$ coefficient to coefficient. That is, as bytes, the + addition is given by the XOR addition.

Multiplication. The multiplication $\cdot$ is given by the natural multiplication followed by the division with rest by the polynomial

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

For example, in hexadecimal notation, $57 \cdot 83 = \mathrm{C1}$, because

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

and

$$x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1 = (x^5+x^3+1)(x^8+x^4+x^3+x+1)+x^7+x^6+1$$

The multiplication by the polynomial 1 does not change anything, it is the *neutral* element. For every polynomial $b(x)$, Euclid's extended algorithm, calculates polynomials $a(x)$ and $c(x)$ such that

$$b(x)a(x) + m(x)c(x) = 1.$$

That is, in the division with the remaining $a(x)b(x)$ for $m(x)$ left over 1 . This means that a is the *inverse multiplicative* in $\mathbb{F}_{2^8}$,

$$b^{-1}(x) = a(x) \text{ in } \mathbb{F}_{2^8}.$$

When we invert a byte b into $\mathbb{F}_{2^8}$, we mean byte $a = b^{-1}$.

Self-Check Questions.

1. How many rounds has AES for a 128 bit key?

   - ☐ 8
   - ☐ *10*
   - ☐ 12
   - ☐ 16

2. Which are the steps of each round? `SubBytes`, `ShiftRows`, `MixColumn` *and* `AddRoundKey`

3. Which one of the steps is non-linear?

   - ☐ `SubBytes`
   - ☐ `ShiftRows`
   - ☐ `MixColumn`
   - ☐ `AddRoundKey`

Summary

There are two basic operations in ciphering: transpositions and substitution.

- Transpositions rearrange the symbols in the text without changing the symbols themselves (such as Caesar' cipher that advances every letter in the plaintext by the same distance in the alphabet).
- Substitutions replace the symbols of the text (be it one by one or in groups, ...) by other symbols (or groups of symbols) without changing the order in which they occur (such as the scytale, where the parchment is wrapped around a stick and the text written on it horizontally).

Either of these ciphers is insecure because they preserve statistical data of the plaintext: For example, a mere (monoalphabetic) substitution cipher falls victim to the frequency distributions of symbols in the plaintext being carried over to the ciphertext. Instead, a modern cipher combines substitution and permutation ciphers, so called *substitution and permutation network* or *Feistel cipher*

While the only cipher proved to be *perfectly secure*, that is no method of cryptanalysis is faster than exhaustive key-search, modern ciphers such as DES from 1976 or AES from 2000 in practice achieve, up to (the k)now, the same, that is, no cryptanalytic method faster than exhaustive key-search is known. The key criterion for this feat is *high diffusion* as defined by Shannon, that is, if a bit in the plaintext or key changes, then half of the bits in the ciphertext changes. Compare it to that of ancient algorithms!

Questions

1. Which cipher is a substitution cipher? Scytale, *Enigma*, AES, DES
2. Which cipher is perfectly secure? RSA, *one-time pad*, AES, DES
3. What is the smallest key size used by AES? 56, 128 , 512, 2048

Required Reading

Read the section on symmetric cryptography in the article Simmons et al. (2016).

Read (at least the beginnings of) Chapter 9 on hash functions in Menezes, Oorschot, and Vanstone (1997), and (at least the beginnings of) that in the most recent work Aumasson (2017), Chapter 6.

Further Reading

Cryptanalyze a substitution cipher in Esslinger et al. (2008).

Follow the encryption process

- of AES by the AES inspector in Esslinger et al. (2008).
- of the Enigma by the animation in Esslinger et al. (2012).

See the book Sweigart (2013a) for implementing some simpler (symmetric) algorithms in Python, a readable beginner-friendly programming language.

Read the parts of the book Schneier (2007) on understanding and implementing modern symmetric cryptographic algorithms.

3 Hash Functions

Study Goals

On completion of this chapter, you will have learned about the manifold uses of (cryptographic) hash functions whose outputs serve as IDs of their input (for example, a large file).

Introduction

A **hash function** is an algorithm that generates an output of fixed (byte) size (usually around 16 to 64 bytes) from an input of variable (byte) size, for example, a text or image file, a compressed archive.

> **hash function**: algorithm that generates a fixed-size output from variable-size input

As a slogan it transforms a large amount of *data* into a small amount of *information.*

Concept. A hash function takes an input (or "message"), a variable-size string (of bytes), and returns a fixed-size string, the *hash value* (or, depending on its use, also (*message*) *digest*, *digital fingerprint* or *checksum*).

For example, the hash `md5` (of 16 bytes) of the word "key" in hexadecimal coding (that is, whose digits run through `0`, ...,`9`, `a`,`b`, `c`,`d`,`e` and `f`) is `146c07ef2479cedcd54c7c2af5cf3a80`.

One distinguishes between

- checksum (non-cryptographic) functions, and
- cryptographic (or one-way) hash functions

Checksum. A (simple) hash function, or **checksum function**, should satisfy:

- fast computation of a hash for any given data, and
- that it is highly unlikely for two (hardly) different messages to give the same hash.

That is, with respect to the second property, a hash function should behave as much as possible like a random function, while still being a fast and deterministic algorithm.

> **checksum function**: algorithm that quickly generates a fixed-size output from variable-size input without collisions.

For example, the most naive *checksum* would be the sum of the bits of the input, truncated to the fixed output size. It is almost a hash function: it is fast, and it is indeed unlikely that two different messages give the same hash. However, one easily obtains two almost identical messages with the same hash. Tiny alterations could therefore go undetected.

Cryptographic hash function. A **cryptographic** (or **one-way**) **hash function** should, moreover, satisfy:

- it is unfeasible to calculate an input that has a given hash.

Thus the output string of fixed length that a cryptographic hash function produces from a string of any length (an important message, say) is a kind of inimitable *signature*. A person who knows the *hash value* cannot know the original message; only the person who knows the original message can prove that the "hash value" is produced from that message.

> **cryptographic (or** one-way**) hash function**: a hash function such that, given an output, it is unfeasible to calculate a corresponding input.

More exactly:

- a hash function is **weakly collision resistant** if finding an *inverse* is computationally unfeasible, that is, given an output, finding a (yet unknown) corresponding input;
- a hash function is **strongly collision resistant** if finding a *collision* is computationally unfeasible, that is, finding two inputs with the same output.

> **weak collision resistance**: computationally unfeasible to find an unknown message for a given hash.
>
> **strong collision resistance**: computationally unfeasible to find two messages with the same hash.

Otherwise, an attacker could substitute an authorized message with an unauthorized one.

Applications. Hash functions are used for

- querying database entries,
- error detection and correction, for example,
  - for data integrity checks,
- in cryptography to identify data but conceal its content, for example,
  - for data authenticity checks,
  - for authentication, for example,
    * to store passwords and
    * for digital signatures.

Checksums. A checksum is a method of error detection in data transmission (where it also bears the name *message digest*) and storage. That is, a checksum detects whether the received or stored data was not *accidentally* or *intentionally* changed, that is, is free from errors or tampering. It is a hash of (a segment of) computer data that is calculated before and after transmission or storage.

That is, it is a function which, when applied to any data, generates a relatively short number, usually between 128 and 512 bits. This number is then sent with the text to a recipient who reapplies the function to the data and compares the result with the original number. If they coincide, then most probably the message has not been altered during transmission; if not, then it is practically certain that the message was altered.

Most naively, all the bits are added up, and the sum is transmitted or stored as part of the data to be compared with the sum of the bits after transmission or storage. Another possibility is a *parity bit* that counts whether the number of nonzero bits, for example, in a byte, is even or odd. (The sum over all bits for the exclusive-or operation instead of the usual addition operation.) Some errors — such as reordering the bytes in the message, adding or removing zero valued bytes, and multiple errors which increase and decrease the checksum so that they cancel each other out — cannot be detected using this checksum.

The simplest such hash function that avoids these shortfalls against *accidental* alterations is CRC, which will be discussed below. It is faster than *cryptographic* checksums, but does not protect against intentional modifications.

Hash Table. A hash table stores and sorts data by a table in which every entry is indexed by its hash (for a hash function that is fixed once and for all for the table). That is, its key is the hash of its value. This key has to be unique, and therefore the hash function ideally collision free. If not, then, given a key, first the address where several entries with this key are stored has to be looked up, and then the sought-for entry among them, causing a slow down.

Therefore, the hash size has to be chosen wisely before creating the table, just large enough to avoid hash collisions in the far future. If this can be achieved, then the hash table will always find information at the same speed, no matter how much data is put in. That is, hash tables often find information faster than other data structures, such as search trees, and frequently used; for example, for associative arrays, databases, and caches.

Examples. In practice, even for checksums, most hash functions are cryptographic. Though slower, they are still fast enough on most hardware. In fact, sometimes, for example to store passwords, they have to deliberately slow so that the passwords cannot be found quickly by their hash values through an exhaustive search among probable candidates (see rainbow tables in Section 12.6).

The most common cryptographic hash functions used to be MD5, usually with an output length of 128 bit, invented by Ron Rivest of the Massachusetts Institute of Technology in 1991. By 1996 methods were developed to create collisions for the MD5 algorithm, that is, two messages with the same MD5 hash. MD5CRK was a concerted effort in 2004 by Jean-Luc Cooke and his company, CertainKey Cryptosystems, to prove the MD5 algorithm insecure by finding a collision. The project started in March and ended in August 2004 after a collision for MD5 was found. In 2005, further security defects were detected. In 2007 the NIST (National Institute of Standards and Technology) opened a competition to design a new hash function and gave it the name SHA hash functions, that became a Federal Information Processing standard.

Cyclic Redundance Check (CRC). One exception is the Cyclic Redundance Check (CRC) which is a fast simple hash function to detect *noise*, expected accidental errors, for example, while reading a disc, such as a DVD, or in network traffic. The CRC uses a binary generating polynomial (a formal sum in an unknown whose only coefficients are 0 and 1 such as $X^2 + X$). The CRC is computed by:

1. a polynomial division of the binary polynomial obtained from the binary input (that is, 1001 corresponds to $X^3 + 1$) by the generating polynomial, and
2. taking the remainder from the division as output.

The choice of the generator polynomial is the most important one to be made when implementing the CRC algorithm; it should maximize the error-detection and minimize the chances of collision. The most important attribute of the polynomial is its length (or degree) as it determines the length of the output. Most commonly used polynomial lengths are 9 bits (CRC-8), 17 bits (CRC-16), 33 bits (CRC-32) and 65 bits (CRC-64).

In fact, the type of a CRC identifies the generating polynomial in hexadecimal format (whose 16 digests run through 0, ..., 9 and A, ..., F). A frequent CRC type is that used by Ethernet, PKZIP, WinZip, and PNG; the polynomial $0x04$.

Again, the CRC can only be relied on to confirm the integrity against accidental modification; through intentional modification, an attacker can cause changes in the data that remain undetected by a CRC. To prevent against this, cryptographic hash functions could be used to verify data integrity.

SHA. SHA stands for Secure Hash Algorithm. The SHA hash functions are cryptographic hash functions made by the National Security Agency (NSA) and the National Institute of Standards and Technology. SHA-1 is the successor to MD5 with a hash size of 160 bits, an earlier, widely-used hash function, that fell victim to more and more suspicious security shortcomings (even though it is not downright broken; for example, there is no known computationally feasible way to produce an input for a given hash). SHA-1 was notably used in the Digital Signature Algorithm (DSA) as prescribed by the Digital Signature Standard (DSS) by the Internet Engineering Task Force.

In the meanwhile, there are three SHA algorithms SHA-1, SHA-2 and SHA-3, released in 2015 of ever-increasing security, mitigating the shortcomings of each predecessor. “SHA-2” permits hashes of different bit sizes; to indicate the number of bits, it is appended to the prefix “SHA”, for example, “SHA-224”, “SHA-256”, “SHA-384”, and “SHA-512”.

3.1 Hash as ID

A hash function should be

- *onto*, that is, all possible fixed-length sequences are hash function value, and
- similar to a uniformly random *variable*, that is, the probability of each of the values of the function is the same.

So if, for example, the output has 256 bits, then ideally each value should have the same probability 2^{-256}. That is, the output identifies the input practically *uniquely* (with a collision chance of ideally 2^{-256}); So one might think of a data hash, for example, from a file, as its *ID card* (or more accurately, identity number); a hash identifies *much* data by *little* information.

Since the length of the hash sequence is limited (rarely ≥ 512 bits), while the length of the input sequence is unlimited, *there are collisions*, that is equal hashes from different files. However, the algorithm minimizes the probability of collisions by distributing their values as evenly as possible: Intuitively, make them as random as possible; more accurately, every possible fixed-length sequence is a value and the probability of each of the values is the same.

3.2 Cryptographic Hash Functions

It is **cryptographic** (or **one-way**)

- when reversion is computationally infeasible, that is, finding an input for a given output, and
- when similar inputs yield dissimilar outputs (in the sense of high diffusion, that is, ideally one different input bit implies about half of the output bits to be different).

  Cryptographic or **One-Way** hash function: a hash function such that it is computationally infeasible to find an input for a given output and similar inputs have dissimilar output.

More exactly, the algorithm should resist

- against the creation of *an inverse image* (that is the function is *unidirectional*): given an output, the quickest way to find an input with this output is by brute force, that is by proving all possible inputs,
- against the creation of *a second reverse image*: given an input, the quickest way to find *another* input with the same output is by brute force, that is by proving all possible inputs,
- against the creation of *collisions*: the fastest way to find two (arbitrary) entries with the same output is by brute force, that is, by proving all possible entries.

According to Kerckhoff's principle, the algorithm should also be public. In practice,

- the most important is resistance against the attack to create a second reverse image, and
- the least important is that against collisions attacks; there are several algorithms, for example, `MD4`, `MD5` and `SHA-1` that do not resist against collisions attacks, but are still in use.

For example, the `CRC` algorithm is a hash function (not cryptographic); Common cryptographic hash functions are, for example, `MD4`, `MD5`, `SHA-1`, `SHA-256` and `SHA-3`.

For example, the output of the hash function `SHA-256` of `ongel` is `bcaec91f56ef60299f60fbce80be31c49bdb36bc500525b8690cc68a6fb4b7f6`.
The output of a hash function, called *hash*, but also *message digest* or *digital fingerprint*, depending on the input, is used, for example, for message integrity and authentication.

3.3 Common Cryptographic Hash Functions

The most commonly used hash (cryptographic) algorithms even today are `16` bytes (`=128` bits) `MD4` and `MD5` or `SHA-1`, which uses `20` bytes (`= 160` bits). Although all of these, `MD4`, `MD5` and `SHA-1` do not withstand collision attacks, they remain popular for use. Their implementation details are described in `RFCs` (Request for Comments): an `RFC` publicly specifies in a text file the details of a proposed Internet standard or of a new version of an existing standard and are commonly drafted by university and corporate researchers to get feedback from others. An RFC is discussed on the Internet and in formal meetings of

the working group tasked by the Internet Engineering Task Force (IETF). For example, networking standards such as IP and Ethernet have been documented in RFCs.

- MD4 (Message-Digest algorithm):
  - Developed in 1991 by Ron Rivest, one of the three creators of the RSA algorithm (and the RSA Data Security company);
  - fast, but vulnerable to pre-image creation.
  - described in RFC 1320 .
- MD5:
  - Developed by RSA Data Security.
  - Described in RFC 132.
  - Vulnerable to collisions, but not to creating a second reverse image. Often used for
    * integrity check, by software with peer-to-peer protocol (P2P, or Peer-to-Peer, in English), and
    * password storage.
- SHA-1 (**Secure Hash Algorithm**):
  - Developed by NIST, the National Institute for Standards and Technology.
  - Vulnerable to collisions, but not to creating a second reverse image.

instead of MD4, MD5 or the ancient SHA-1, recommended are more recent versions like SHA-256 and SHA-3 of the Secure Hash Algorithm.

> **Secure Hash Algorithm**: Hash algorithm recommended by the National Institute for Standards and Technology.

3.4 Construction Scheme

Similar to modern symmetric ciphers that follow the design laid out by Feistel's Lucifer algorithm in the 70s, cryptographic hash functions follow the Merkle-Damgård construction:

The Merkle meta-method, or the Merkle-Damgård construction, builds from a lossless compression function (that is, whose inputs have the same length as

the output) a lossy compression function, that is, a hash function. This method allows us to reduce the immunity of the *entire hash function* against

- the creation of *an inverse image*, and
- the creation of *collisions*

to the corresponding immunity of the *compression function.*

Merkle Meta-Method Scheme. This construction needs

- an *initialization value* (IV = initialization value),
- a *compression function* C, and
- a *padding* to pad the entry string m of the hash function to a string $\bar{m}$ such that $\bar{m} = (m_1, m_2, \ldots)$ consists of a multiple of blocks m_1, m_2, ... which are accepted by the compression function.

With $h_0 = \mathrm{IV}$, one computes for $i = 1, 2, \ldots$

$$h_i = \mathrm{C}(m_i, h_{i-1}).$$

That is, for the computation of the hash of the current block, the value of the compression function of the last block enters jointly with the current block value.

The C compression function consists of a Feistel Cipher (or substitution and permutation network) c where

- or, in the *Mayer-Davis* scheme, m_i is the key and H_{i-1} the plaintext; then the ciphertext is added (by XOR) to H_{i-1} . This is

$$h_i = h_{i-1} \oplus c(h_{i-1}, m_i)$$

- or in the *Miyaguchi-Preneel* scheme, m_i is the plaintext and (a $g(\mathrm{H}_{i-1})$ change) H_{i-1} is the key; then, as in Mayer-Davis's scheme, the ciphertext is added (by XOR) to H_{i-1} . This is

$$h_i = h_{i-1} \oplus c(m_i, g(h_{i-1}))$$

The addition of h_{i-1} ensures that the C compression function is no longer invertible (unlike the c cipher for a fixed key); that is, for a given output, it is no longer possible to know the (unique) input. Otherwise, to create a collision, given an output h_i , one could decipher by different keys, m_i' and m_i'' to get different entries h_{i-1}' and h_{i-1}'' .

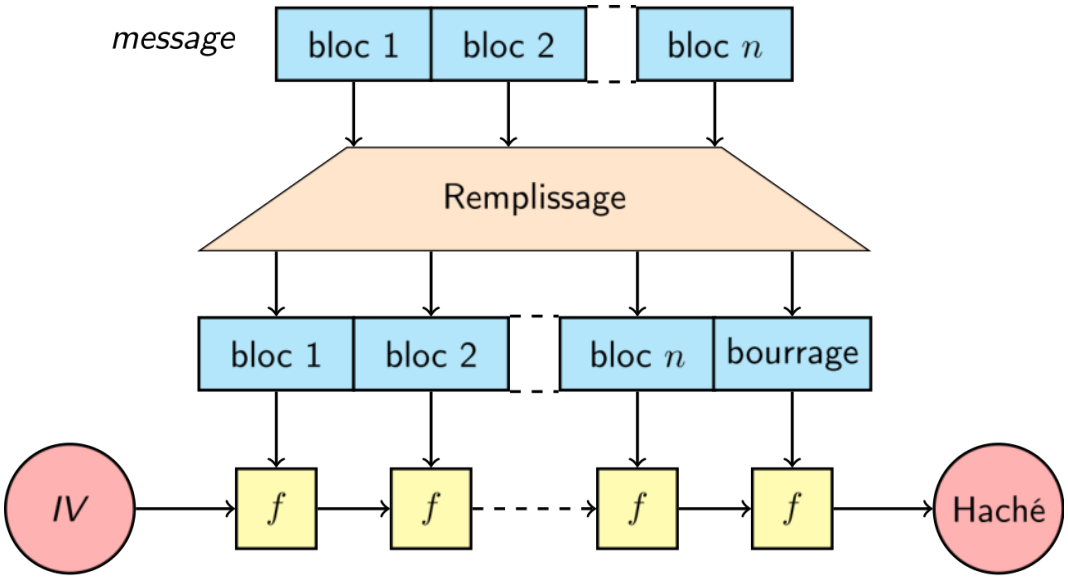


Figure 13: The Merkle meta-method (Chouhartem (2016)) where remplissage and bourrage = padding, and f is the compression function (denoted `C`)

Padding. To reduce the immunity from hash to compression function, the padding $m\overline{m}$ needs to meet sufficient conditions:

- m is a starting segment of $\overline{m}$, that is, the message is extended, but its initial segment not changed!
- two messages of the same length are extended by the same final segment.
- two messages of different lengths are extended differently, so that they differ in the last block.

The simplest pad that meets these conditions is the one that attaches the length $|m|$ to m and fills the segment between the end of m and $|m|$ by the number of `0` s prescribed by the block size, that is, the concatenation

$$\overline{m} = m \,||\, 0^k \,||\, |m|.$$

Observation: To avoid collisions, it is not enough for the padding to fill with zeros the rest of the message: This way, two messages that only differ in the number of final zeros at the last end would have the same padding!

Instead, the simplest way would be to attach a digit `1`, and then the rest with `0`s. However, we will see that this would allow collisions with the Merkle meta-method if the initial value `IV` was chosen in the following way:

Denote $B_1, \ldots, B_k$ the message blocks and `IV` the starting value. The hash is calculated by iterating the compression function $C : (X, B) \to C(X, B)$

$$H(M) = C(C(..C(C(IV, B_1), B_2)...B_{k-1}), B_k)$$

The clou of Merkle's meta-method is the reduction of collisions from the hash function to the compression function: a hash collision would imply a collision of the compression function, that is, different pairs of X', B' and X'', B'' blocks with $C(X', B') = C(X'', B'')$.

To see this, we note that

- if the m' and m'' collident messages have different lengths, then the last B'_k and B''_k blocks are different (because they contain the different lengths!), but they collide, because the value of the compression function of the last entry is the hash function.
- Otherwise, that is, colliding messages have the same length, so there's a block further to the right where the padded messages differ and we'll find a block collision after it.

Without the length in the padding, the collision of two messages with different lengths can ultimately only be reduced to a pre-image of the initial value `IV` under the compression function, that is, a value `B` such that

$$IV = C(IV, B).$$

If its choice were arbitrary, the authors of `MD5` and `SHA-256` could however have inserted the following back door: Both algorithms use Mayer-Davis's scheme, that is, a compression function

$$C(X, K) = X \oplus E(X, K)$$

for a Feistel cipher `E` with a key K ; in particular, with K fixed the decryption function $E_K = E(\cdot, K)$ is invertible! Now, if the authors wanted to, they could have chosen a key K and set

$$IV := E_K^{-1}(0)$$

exactly so that `IV = C(IV,K)`. Then `C(IV, B) = C(C(IV, K), B)` , that is, a collision between the hashes of B and K $\oplus$ B !

Since, for example, `MD5` and `SHA-256` choose as `IV` a value whose pre-image is supposedly not known (for example, the hexadecimal digits in ascending order in `MD5` or the decimal digits of the first eight primes in `SHA-256`) this problem is more theoretical than practical.

3.5 SHA-256

We won't study the inner workings of `SHA-256` in detail (in contrast to O'Connor (2022)), but a schematic look at its design shows that it follows the Merkle-Damgård construction:

It is then iterated in roughly `64` rounds:

On Lynn-Miller (2007), you can trace the bytes of an input of your choosing at each step of the `SHA-1` algorithm.

3.6 Uses

Uses of cryptographic hash functions abound:

Digital Signatures. If the roles of the public and private key are flipped, then the encryption is a **digital signature**: while the encrypted message will no longer be secret, every owner of the public key can check whether the original message was encrypted by the private key. However, in theory, signing a file (using the `RSA` algorithm) that was encrypted using the `RSA` algorithm would decrypt it. Therefore, in practice, since for a signature it suffices to unequivocally identify the signed file (but its content often secret, for example, when signing a secret key), usually a cryptographic *hash* is encrypted by the private key.

(H)MAC. A message authentication code algorithm uses a one-way hash function (such as MD5 or SHA-1) and a block cipher that accepts a secret key and a message as input to produce a MAC. The MAC value provides the intended receivers (who know the secret key) to detect any changes to the message content by:

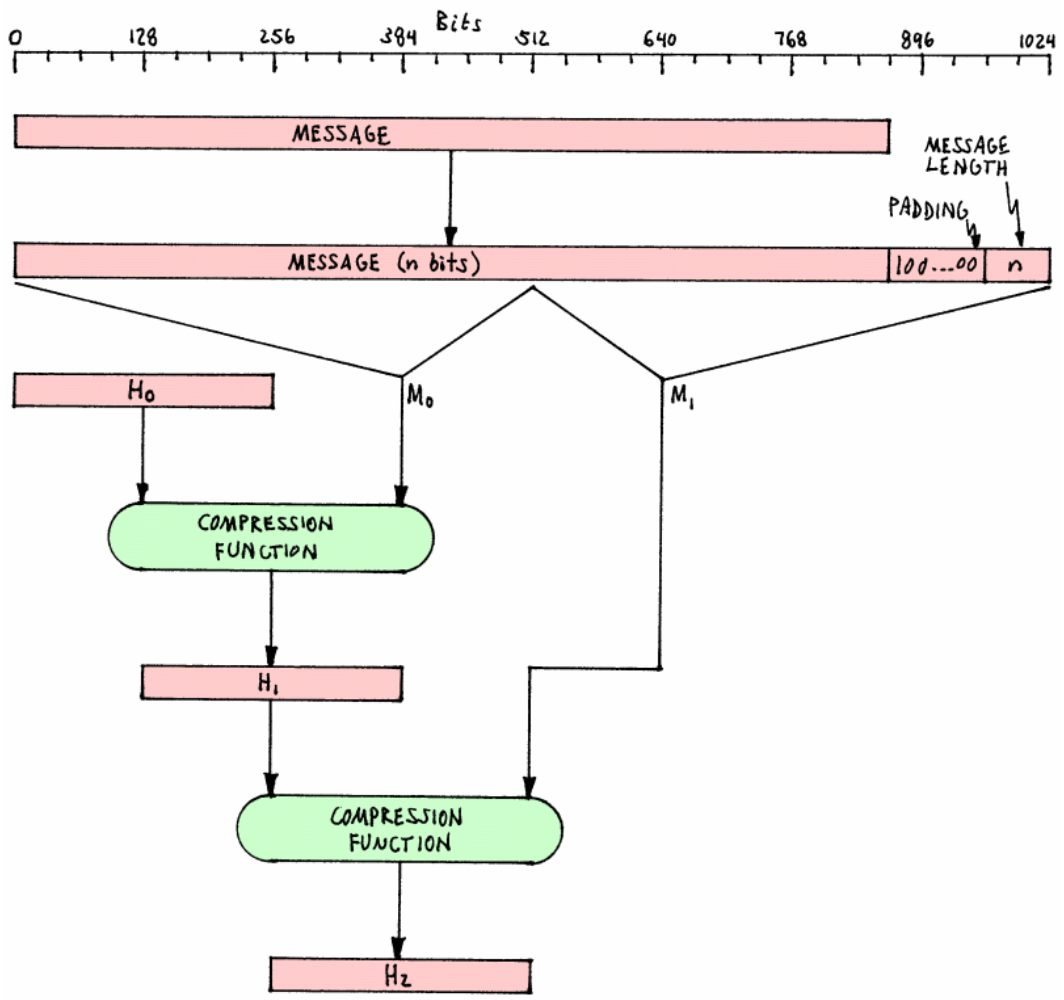


Figure 14: Compression function (Kaminsky (2004))

- an integrity check (by ensuring that a different MAC will result if the message has been altered) and
- an authenticity check (because only the person knowing the secret key could have produced a MAC).

Data Storage and Integrity. Hash functions (*not* necessarily cryptographic, like CRC) are used:

- for fast data query (that is, at fixed time, regardless of the number of entries, for example,

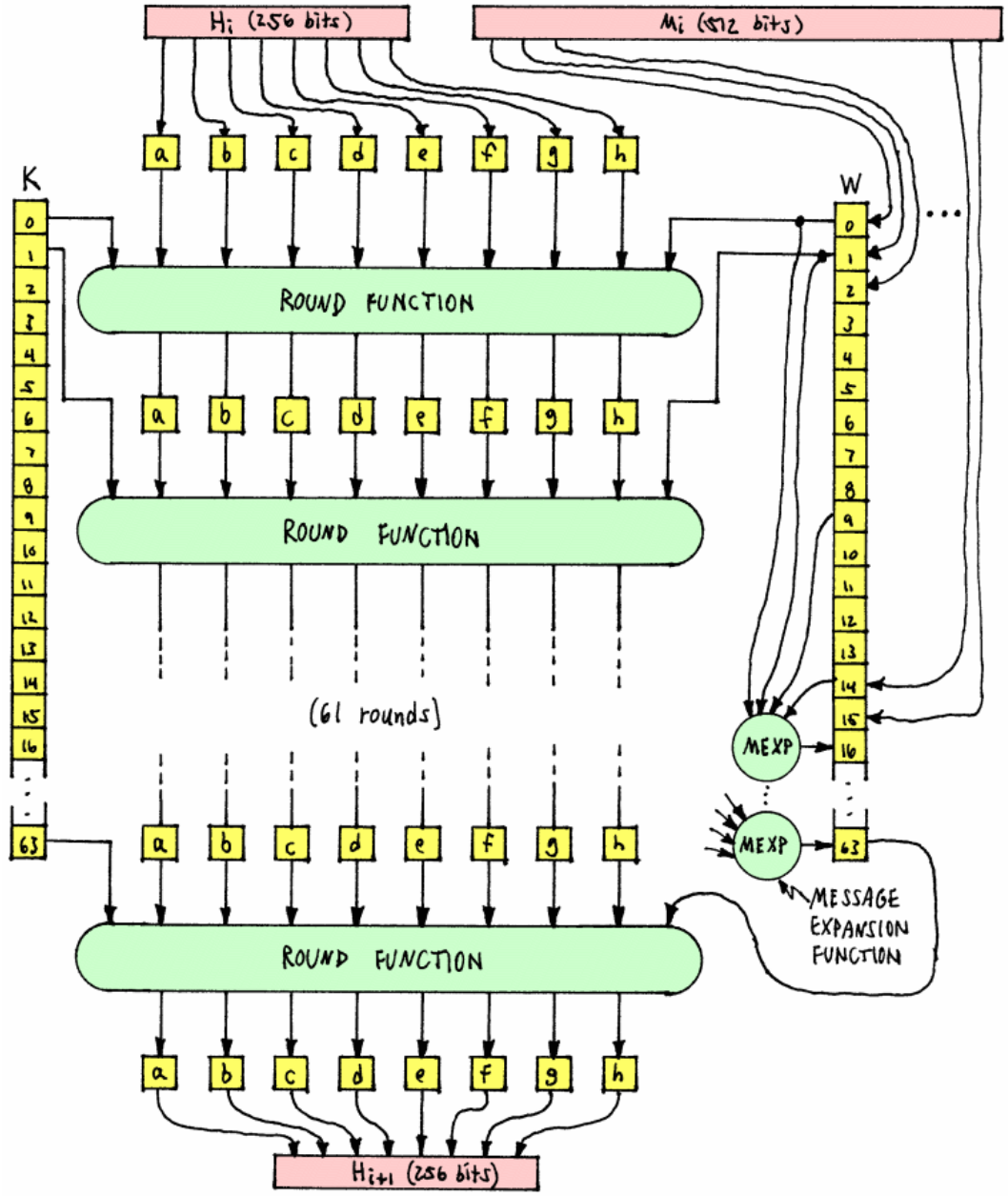


Figure 15: Iteration (Kaminsky (2004))

  - in a hash table, and
  - in a Merkle tree;

- to ensure the *integrity* of a file in case of *accidental* modifications, that is, to detect differences between the file and a reference version (typically the one before the file is transported).

Passwords. *Cryptographic* Hash functions are used:

- To distribute values evenly (`key stretching`), intuitively make them less predictable; that is, as `KDF` (= `Key Derivation Function`);
  - in particular, to *generate and store passwords*, that is, as `PBKDF` (= `Password Based Key Derivation Function`).
  - To ensure the *integrity* of a file against tampering, that is, to detect differences between the file and a reference version (typically the one before the file is transported);
  - in particular, to ensure the *authenticity* of a file: to detect differences between the file and a version that was under the control of a specific person.

Observation: Note the difference between *authenticity* and *authentication:* The former guarantees the equality of data received and sent from a person (for example, in the digital signature), the latter the identity of that person (for example, in a secure site access).

The cryptographic hash algorithms listed above, `MD4/5`, `SHA ...` distribute the values evenly, but are *fast*; so they are unsuitable for password creation because they are vulnerable to brute-force attacks. To prevent these, the `PBKDF` algorithm, for example, `PBKDF1`, `PBKDF2`, `bcrypt`, `scrypt`, `Argon2` (a new and more promising candidate), are

- deliberately *slow*, such as `bcrypt`,
- deliberately require a *lot of memory* to compute, such as `scrypt` algorithm;
- used only once for each entry (guaranteed by a `salt`, an additional, unique, usually random argument. Without `salt`, the algorithm is prone to attacks by so-called `Rainbow Tables`, tables of the hashes of the most common passwords.

3.7 Dispersion Table

A *h* table (or *scatter table*) uses a hash function to address the entries (= rows) of a *array*, that is, a data table.

Each entry has a name, that is, a unique identification (= *key*). For example, the key is the name of a person. The key data, for example, your telephone number, is stored in a table row. These rows are numbered from 0.

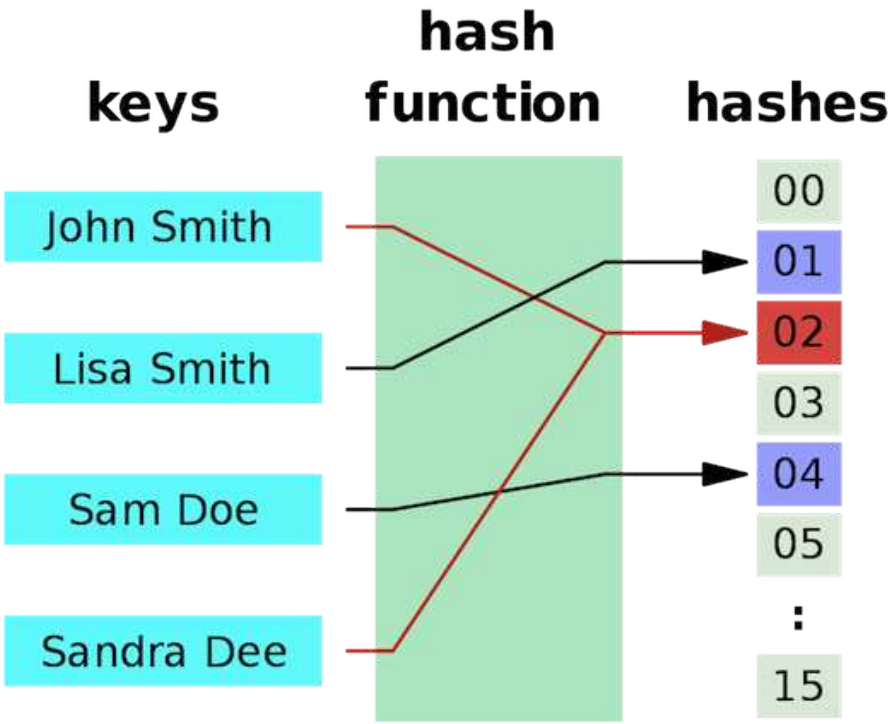


Figure 16: Hash Table

The row number, your *address*, of the key is determined by your hash. As an advantage, at a fixed time, the data can

- from the key being found, and
- added.

While,

- for a list of n entries, the search compares on average $n/2$ entries, and
- for a binary tree of n entries, $log_2 n$ entries.

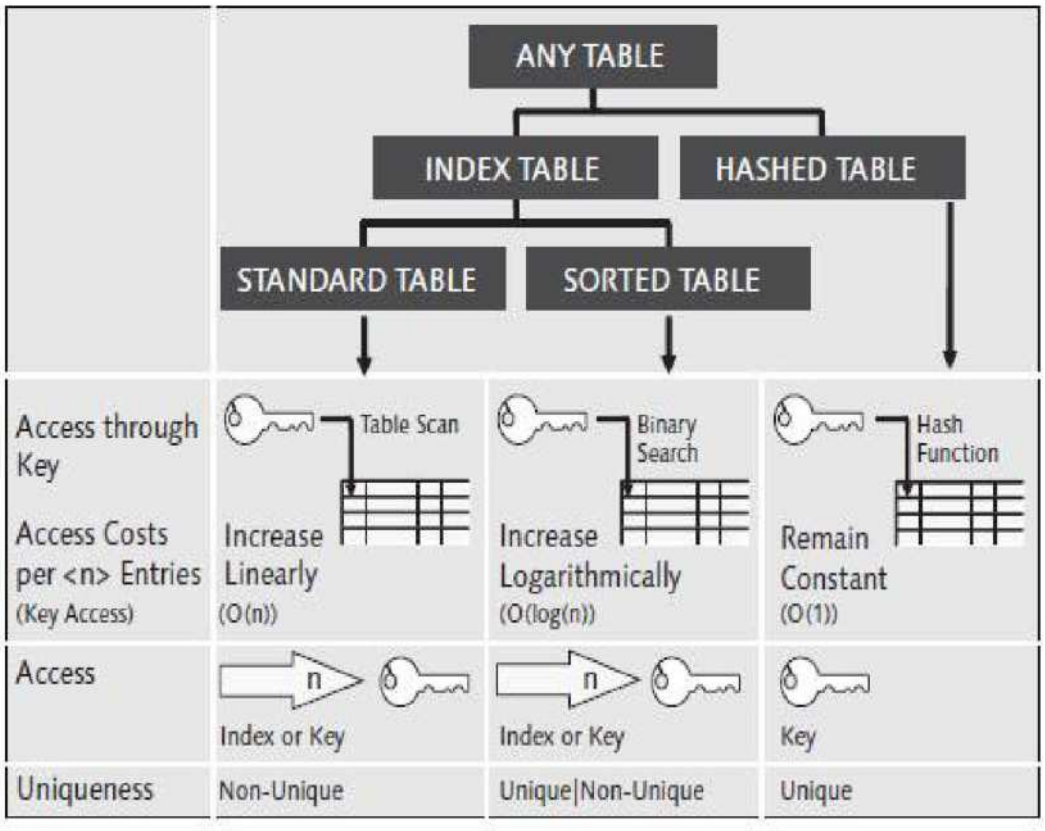


Figure 17: Comparison of data structures

Collisions, that is, two entrances with the same hash are more frequent than you might think; see the *Birthday Paradox*. To avoid collisions, it is necessary

- choose the number of addresses large enough, and
- the sufficiently *injector* hash function, that is, it rarely sends different arguments at equal values.

When collisions occur, a strategy is

- instead of the hash number address a single entry, use `Chaining`: address a *bucket*, a *bucket* of multiple entries, a list, or
- use “Open Hashing. put the entry in another free position, for example,

  - the next one, or
  - use another hash function to calculate it (`Double Hashing`).

Up to 80 of the table being filled, on average 3 collisions occur. That is, finding an entry takes 3 operations. To compare, with 1000 entries, it would take on average

- in a 500 operations table, and
- on a binary tree about 10 operations.

However, after this factor collisions occur so often that the use of another data structure, for example a binary tree, is recommended.

3.8 Merkle Tree

A **spreading** or **Merkle** tree (invented in 1979 by `Ralph Merkle`) groups data into blocks by a tree (binary), whose *vertices* (= nodes) are hashes and whose *sheets* (= end nodes) contain the data blocks, in order to be able to quickly check (in *linear* time) each data block by computing the root hash.

In a n (= 2^n sheets)deep scattering *tree* the data block of each sheet is verifiable

- by **knowledge**
  - of n hashes "antagonists" from a source **dubious**, and
  - from the hash of the top of a source **reliable**, and
- by **calculation**
  - from the hash of the data block on the sheet,
  - of the n hashes of his predecessors, and
  - the comparison of the calculated root hash with that obtained.

Use. The main use of Merkle trees is to ensure that blocks of data received from other pairs in a point-to-point network (P2P) are received intact and unchanged.

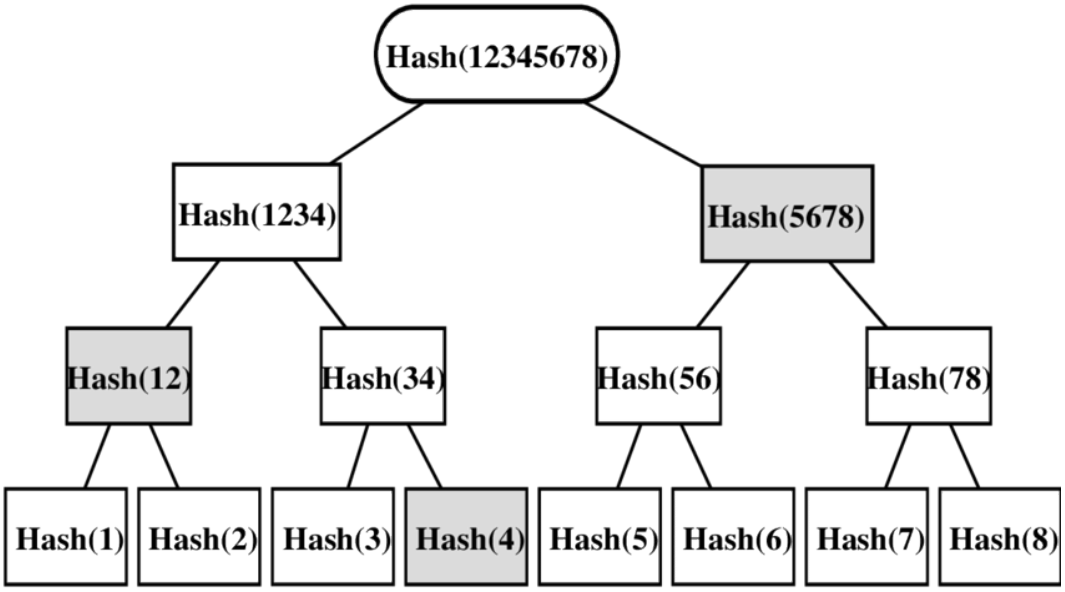


Figure 18: Checking the third block of a Merkle tree by computing the radical hash. The hash of each node (mother) is the hash (of the concatenation) of the two daughters. The hashes which are necessary and which have been informed to calculate the one of the top are grey.

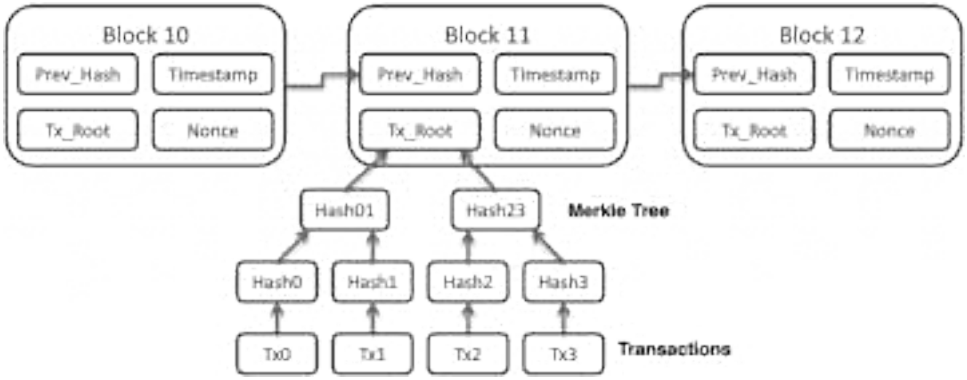


Figure 19: Transaction Tree

Operation. A Merkle tree is a (usually binary) hashes tree whose leaves are data blocks from a file (or set of files). Each node above the leaves on the tree is the hash of its two children. For example, in the image,

- Hash(34) is the hash of the concatenation of the hashes Hash(3) and Hash(4), that is, Hash(34) = Hash(3)||Hash(4)),
- Hash(1234) is the hash of the concatenation of the hashes Hash(12) and Hash(34), that is, Hash(1234) = Hash(12)||Hash(34)), and
- the radical hash Hash(12345678) is the hash of the concatenation of the hashes Hash(1234) and Hash(5678), that is,

$$\text{Hash}(12345678) = ||mathrm\text{H}ash(5678)).$$

.

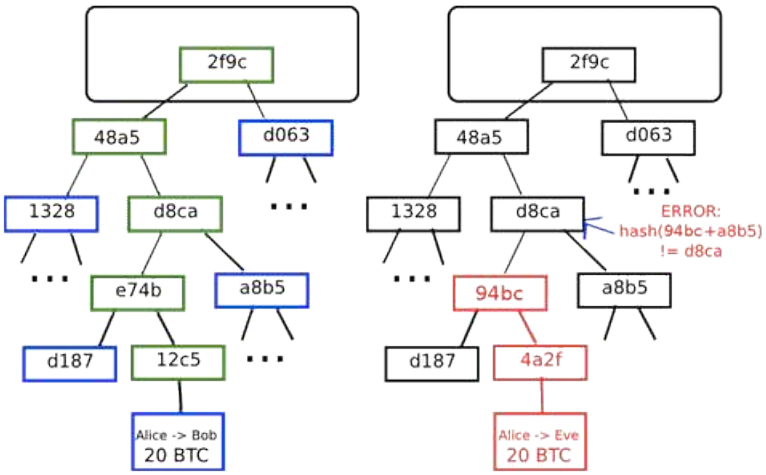


Figure 20: Merkle-tree defective

Normally a cryptographic shuffling function' is used, for example,SHA-1. However, if the Merkle tree only needs to protect against unintentional damage, "checkums" are not necessarily cryptographic, such asCRC's are used.

At the top of the Merkle tree resides the *root-dispersion* or *master-dispersion*. For example, on a P2P network, root dispersion is received from a trusted source, e.g. from a recognized website. The Merkle tree itself is received from any point on the P2P network (not particularly reliable). This (not particularly reliable)

Merkle tree is compared by calculating the leaf hashes with the reliable root dispersion to check the integrity of the Merkle tree.

The main advantage of a tree (deep n with 2^n leaves, that is, blocks of data), rather than a `dispersion` list, is that the integrity of each leaf can be verified by calculating n (rather than 2^n) hashes (and its comparison with the root hash).

For example, in Figure 18, the 3 integrity check requires only

- the knowledge of the hashes `Hash(4)` `Hash(12)`, and `Hash(5678)`, and
- the computation of the hashes `Hash(3)` `Hash(34)`, `Hash(1234)`, and `Hash(12345678)`
- the comparison between the calculated `Hash(12345678)` and the hash obtained from the trusted source.

Self-Check Questions

1. How do a checksum and cryptographic hash function differ? *Only for a cryptographic hash function it is unfeasible to create collisions.*
2. What are typical uses of hash functions? *Fast data queries, identification of files (for example, virus scanning), error correction and detection.*
3. What are typical uses of cryptographic hash functions? *Password storage, message authentication, integrity checks.*
4. What is the clou of the Merkle-Damgård construction? *Reduction of the resistance against collisions to that of the compression function.*

Summary

A *hash function* transforms any data (such as a file), in other words, a *variable-length* string, into a *fixed-length* string (which is usually `16` or `32` bytes long); as a slogan, it transforms a large amount of *data* into a small amount of *information.* Their output, a *hash*, can be thought of as an ID-card of their input (such as a file); to this end, the hash function should

- *diffuse* well, that is, the inversion of an input bit implies the inversion of about half of the output bits (the *avalanche effect*), that is, each output bit should depend on each input bit,

- be *onto*, that is all possible fixed-length sequences are hash function values, and
- be similar to a uniformly random *variable*, that is, the probability of each of the values of the function is the same.

So if, for example, the output has `256` bits, then ideally each value should have the same probability 2^{-256}. That is, the output identifies the input practically *uniquely* (with a collision chance of ideally 2^{-256});

It is *cryptographic* (or *one-way*)

- when reversion is computationally infeasible, that is, finding an input for a given output, and
- when similar data yield dissimilar hashes.

Recommended are, among others, the more recent versions `SHA-256` and `SHA-3` of the `Secure Hash Algorithm`.

Questions

1. How do a checksum and cryptographic hash function differ? *Only for a cryptographic hash function it is unfeasible to create collisions.*
2. What are typical uses for checksums? *(Accidental) error detection and correction, for example, noise on reading a Compact Disc or in network traffic, and many more.*
3. What are typical uses of cryptographic hash functions? *(Intentional) alteration correction in storage or network traffic, and many more.*
4. Name common cryptographic hash functions. *The MD and the SHA family.*
5. Which hash function is *not* cryptographic ? SHA-1, MD_4, *CRC*, WHIRLPOOL
6. Which cipher is a stream cipher? RSA, *Enigma*, AES, DES

Required Reading

Read the section on symmetric cryptography in the article Simmons et al. (2016).

Read (at least the beginnings of) Chapter 9 on hash functions in Menezes, Oorschot, and Vanstone (1997), and (at least the beginnings of) that in the most recent work Aumasson (2017), Chapter 6.

Further Reading

Observe the diffusion created by a cryptographic hash function in Lynn-Miller (2007).

4 Asymmetric Cryptography

Study Goals

On completion of this chapter, you will have learned ...

- the advantages of asymmetric cryptography, such as, key distribution over distance, and
- its limitations, principally the lack of trust in the distant owner of a public key.

Introduction

The big practical problem of single-key cryptography is *key distribution*, more exactly:

- to secretly pass the same key to all correspondents, usually far away from each other, before they can communicate securely. Even messengers as in diplomatic and military agencies have to be trusted unconditionally to neither intentionally nor accidentally disclose them.
- the high number of keys needed for a group of correspondents to communicate securely: Every pair of correspondents in a group needs a unique key to communicate securely. In a group of 10 correspondents, each user would need a different key for each of the other 9 correspondents; in total 45 different keys. The number of keys increases in proportion to the square of the number of correspondents: For example, in a group of 1000 correspondents, around half a million keys would be needed.

In 1976, Whitfield Diffie and Martin Hellman conceived that the key distribution problem could be solved by an algorithm that satisfied:

- (computationally) easy creation of a matched pair of keys for encryption and decryption,
- (computationally) easy encryption and decryption,
- (computationally) infeasible recovery of one of the keys despite knowledge of:
  - the algorithm,
  - the other key, and
  - any number of matching plaintext and ciphertext pairs.
- (computationally) infeasible recovery of the plaintext for almost all keys k and messages x .

Observation: This was the first public appearance of two-key cryptography. However, the British Government Communications Headquarters (GCHQ) knew it around a decade earlier.

If a user, say Alice, of such an algorithm keeps her decryption key secret but makes her encryption key public, for example, in a public directory, then:

- Everyone who wishes to communicate securely with Alice just needs to look up her public key to send her a ciphertext that only she can decrypt; that is, a message can be encrypted without any need for secrecy.
- a ciphertext encrypted with Alice's secret key can be deciphered by everyone who uses the corresponding public key; that is, a sender can be identified without any need for secrecy.

The security of two-key cryptographic algorithms relies on the computational difficulty of a mathematical problem, for example, factoring a number that is the product of two large primes; ideally, computing the secret key is equivalent to solving the hard problem, so that the algorithm is at least as secure as the underlying mathematical problem is difficult. This has not been proved for any of the standard algorithms, although it is believed to hold for each of them.

4.1 Asymmetric Cryptography

In comparison with symmetric cryptography, asymmetric encryption avoids the risk of compromising the key to decipher that is involved in exchanging the key with the cipherer. This *secure* communication with anyone via an *insecure* channel is a great advantage compared to symmetric cryptography. Let us recall the classic methods to exchange a symmetric key, before looking at its asymmetric counterpart: While asymmetric cryptography made it possible to exchange a secret key overtly, this convenience comes at the risk of the unknown identity of the key holder, prone to a man-in-the-middle attack which Public Key Infrastructure work around by the use of certificates, digital signatures by third parties of public keys.

Symmetric Ciphers.

A symmetric key must be passed secretly. Possible methods are:

- Derivation from a base key using a Key Derivation Function (KDF), a cryptographic hash function which derives a secret key from secret — and possibly other public — information, for instance, a unique number,
- Creating a key from key parts held by different persons, for example, as an analogue to the one-time pad: If s is the secret (binary number, then $s = s_1 \oplus s_2 \oplus ... \oplus s_n$ for the partial secrets s_1, s_2, ... Reconstruction of s is only possible if all s_1, s_2, ... are combined

- transmission via a different channel, for example:
  - personally,
  - a sealed letter,
  - by telephone, or
  - by quantum entanglement, where a change of state of one quantum particle (say from spin-up to spin-down) instantly implies that of the other and vice-versa. This information of state cannot be intercepted and lends itself to bit transmission; However, entanglement is fragile and can not be fully controlled. Quantum entanglement for key distribution was put into practice in Sasaki et al. (2011).

4.2 Man-in-the-middle Attack

Diffie and Hellman's achievement was to separate secrecy from authentication: Ciphertexts created with the secret key can be deciphered by everyone who uses the corresponding public key; but the secret-key holder has no information about the owner of the public key!

Thus the public keys in the directory must be authenticated. Otherwise A could be tricked into communicating with C when he thinks he is communicating with B by substituting C key for B in A's copy of the directory; the **man-in-the-middle** attack (MIM):

> **Man-in-the-middle attack**: an attacker intercepts the messages between the correspondents and assumes towards each of them either correspondent's identity.

Scenario. In an MITM, the attacker places himself between the correspondents, assuming towards each one of them the identity of the other to intercept their messages.

1. Bob sends his public key to Alice. Eve intercepts it, and sends Alice her *own* public key that claims Bob as its owner. If Alice sends a message to Bob, then she uses, without realizing it, the public key of *Eve*!
2. Alice enciphers a message with the public key of *Eve* and sends it to Bob.
3. Eve intercepts the message, deciphers it with her private key; she can read the message and alter it.

Figure 21: MITM Sweigart (2013b)

4. Eve enciphers the message with Bob's public key.
5. Bob deciphers the message with his private key and suspects nothing.

So both Alice and Bob are convinced to use each other's public key, but they are actually Eve's!

Example. In practical terms, this problem occurs for example in the 1982 'ARP' Address Resolution Protocol (in RFC 826) which standardizes the address resolution (conversion) of the Internet layer into link layer addresses. That is, ARP maps a network address (for example, an IPv4 address) to a physical address as an Ethernet address (or MAC address). (On Internet Protocol Version 6 (IPv6) networks, ARP has been replaced by NDP, the Neighbor Discovery Protocol).

The ARP poisoning attack proceeds as follows:

Maria Bonita wants to intercept Alice's messages to Bob, all three being part of the same physical network.

Maria Bonita sends a `arp who-has` packet to Alice which contains as the source IP address that of Bob whose identity we want to usurp (ARP spoofing) and the physical MAC address of Maria Bonita's network card.

0. Alice will create an entry that associates the MAC address of Maria Bonita with Bob's IP address.
1. So when Alice communicates with Bob at the IP level, it will be Maria Bonita who will receive Alice's packages!

4.3 Public and private key

Let us recall that there are two keys, a *public* key and a *private* key. Usually:

- The *public* key is used to encrypt, while the *private* key is used to decrypt.

Thus, a text can be transferred from the encipherer (Alice) to one person only, the decipherer (Bob).

The roles of the public and private keys can be reversed:

- The *private* key is used to encrypt, while the *public* key is used to decipher.

Thus, the encipherer can prove to all decipherers (those who have the public key) their ownership of the private key; the *digital signature.*

The (mathematical) algorithm behind the encryption either by the public key (for hiding the content of digital messages) or by the private key (for adding digital signatures) is in theory almost the same: only the roles of the arguments of the one-way function are exchanged (for example, in the RSA algorithm). In practice, however, usually:

- the public key encrypts *paddings* of the plaintext (to avoid a text so short that the private key can be easily computed), and
- the private key encrypts *hashes*, the value of a function that almost always sends different texts to different numbers. (This hash is usually a cryptographic *hash*, that is, given its output, it is practically impossible to deduce its input, so that the *integrity* of the signed message can be checked, that is, that whether it has been altered on the way).

That is, while

- for encryption by the public key, the preparatory transformation of the text (the padding) is easily invertible,
- for private key encryption, it (the hashing) is hardly invertible.

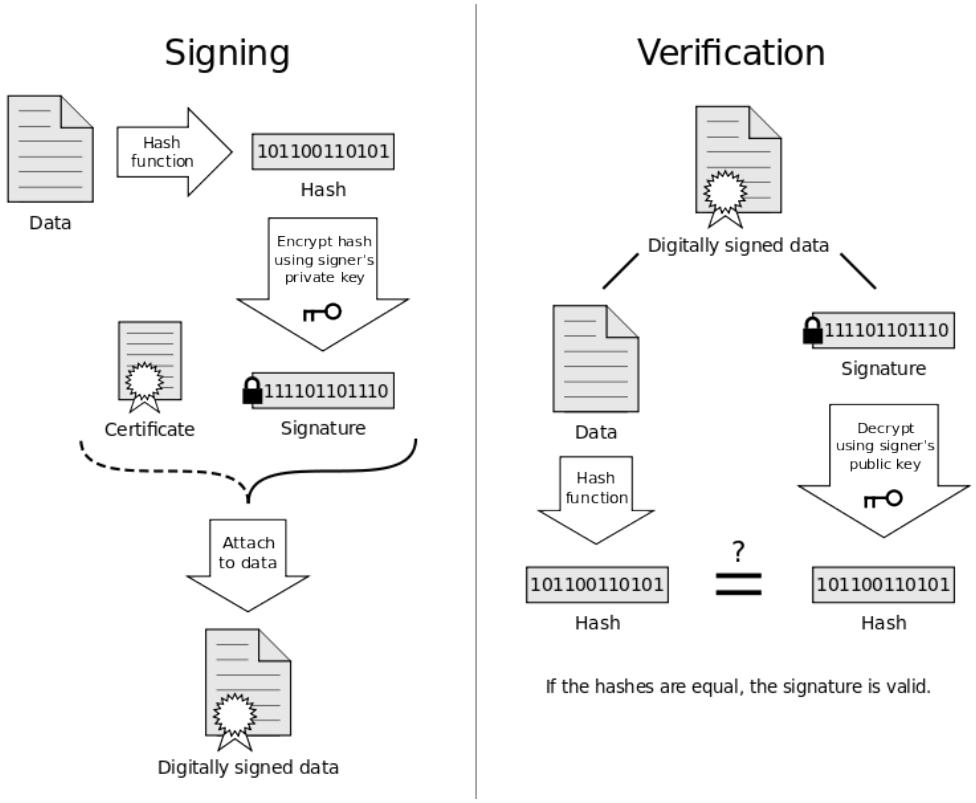


Figure 22: digital signature

Ephemeral Sub Keys. A **sub-key** of a master key is a key (whose cryptographic checksum is) signed by the master key. The owner often creates subkeys in order to use the *main key* (public and private) only

- to *sign*
  - someone else's key,
  - a personal sub key
- to *revoke* a key (that is, sign a revocation),
- to *change* the expiration date of a personal key.

The subkeys are used for all other *daily purposes* (signing and deciphering).

This way, if a sub key is eventually compromised (which is more likely that that of a main key due to its everyday use), then the main key will not be compromised. In this case,

- the owner revokes it (publishes a note invalidating the compromised public key which is digitally signed by its private main key), and
- creates another key.

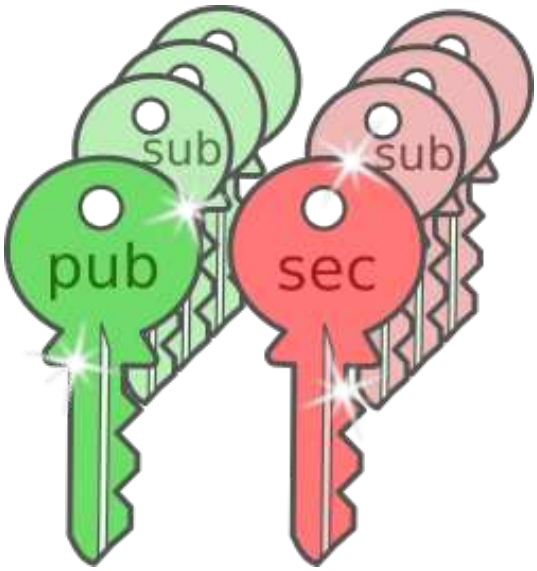


Figure 23: subkeys

We saw how *subkeys* work in practice in the common command-line program GPG discussed in Section 13.4. A good reference is https://wiki.debian.org/Subkeys.

Sub Keys for the Day-to-Day. For more security, you create (for example, in GPG)

- first a (public and private) **main key** and
- then several **sub keys** with expiry date, for everyday use:
  - a subkey to **decipher** in everyday life (for example, the encrypted emails received), and
  - a sub key to **sign** in everyday life (for example, the emails sent).

  Before their expiry, the keys are either extended, or revoked to create others.

As for using different keys to sign and encrypt,

```
 Empreinte de la clef = F43E 7A29 8FB9 77DB 7B2A  5003 992E 7026 125D FBE9
sub   rsa4096/0x7A7AF3571E1B5A03 2016-10-06 [S] [expire : 2019-10-06]
 Empreinte de la clef = 2D0E 8856 08CC B7C6 DAAD  2524 7A7A F357 1E1B 5A03
sub   rsa4096/0x039B4513D99ACDC5 2016-10-06 [E]
```

Figure 24: Fingerprints of sub keys in GPG for **S** subscribe and **E** encrypt

- it's necessary for some algorithms, for example,
  - Yes, in the ElGamal algorithm,
  - but not in "RSA".
- it is safer (but more inconvenient, which can lead to the user's sloppiness and then in practice it is less safe!) to have different keys to decipher and to sign,
  - because
    * it is useful to keep a copy of the private key to decipher (to be able, after its loss, to still read files enciphered by the corresponding public key)
    * it is useless to keep a copy of the private key to sign (because once lost another person can sign with it too).
  - because, for example in the RSA algorithm, the signature and the decipherment (by the private key) are equal (in theory, though in practice implemented differently) algorithms!

    Therefore, signing (by the private key) a document encrypted by the corresponding public key is equivalent to deciphering it! However, this possibility is theoretical, but it does not practice: All implementations of the RSA protect the user by the fact that always and exclusively
    * paddings of a document, and
    * sign a *cryptographic* check sum of a document (from which its original content cannot be deducted).

Best Practices for Key Management. Only the main key is immutably linked to the identity of the owner, and all others replaceable: While the

- **main key** is kept in a **safe** at home and only sees the light when keys need to be signed (be it from the owner or from someone else [for example, to establish the web of trust]).

  In practice,

  - is stored on a flash drive or memory card,
  - and to be more durable, it is even printed, for example:
    * By the `paperkey` program that extracts the secret part (of the file) of the private key (that is, it omits all public information like

      · the identity,
      · the algorithm, ...,

      and encodes this part in hexadecimal notation (stored in a text file). (So if the key has, for example, 4096 bits, the file generated by `paperkey` has ≥ 1024 bits). This command

      ```
      gpg --export-secret-key 0xAE46173C6C25A1A1! > ~/private.sec
      paperkey --secret-key ~/private.sec > ~/private.paperkey
      ```

      · exports only (indicated by the `!`) the main secret key (`0xAE46173C6C25A1A1`) of the private keys, and
      · extracts and converts it by `paperkey` into a text file.

    * By the `qrencoder` program (for keys whose file has < 2953 characters) which encodes it into a `QR` code.

- the **sub keys** are stored in a **smartcard** that is accessed by a USB reader with its own keyboard. Compared to using a digital file, it has the advantage that

  - reading the keys on a smart card is much more difficult than a file (stored on a USB stick or hard drive)
  - leaves fewer traces:
    * It never reveals the key, but only what is necessary to prove that it can access it, and
    * is immune to keyloggers that record keystrokes.

Figure 25: smartcard reader

(Perfect) Forward Secrecy. `(Perfect) Forward Secrecy` means that, after the correspondents exchanged their (permanent) public keys and established mutual trust,

1. *before* correspondence each correspondent *creates an ephemeral key (the* session key*) and signs it by the private (permanent) key to avoid a `MITM` attack (see Section 4.2),
2. *after* correspondence each correspondent *deletes* her (ephemeral) private key.

This way, even if the correspondence was eavesdropped and recorded, it cannot be deciphered later; in particular, it cannot be deciphered by obtaining a correspondent's private key.

For example, the `TLS` protocol, which encrypts communication of much of the Internet, has since version 1.2 support for Perfect Forward Secrecy: In the handshake between client and server in Section 13.3 : after the client has received (and trusted) the server certificate, the server and the client exchange a *ephemeral* public key which serves to encrypt the communication of only this correspondence. This ephemeral key is signed by the public (permanent) key of the server. (The creation of this asymmetric key in Perfect Forward

Secrecy makes the creation of a symmetric *preliminary* key by the client in the penultimate step in the handshake in the TLS protocol superfluous.)

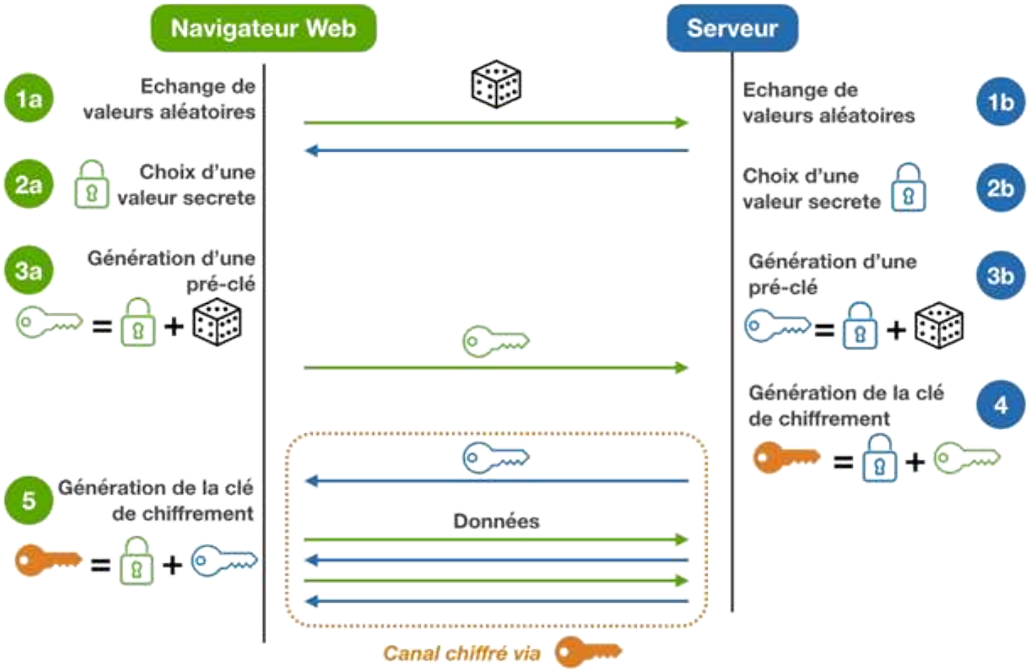


Figure 26: Perfect Forward Secrecy

Group Signature. Every granted signature shall first be thought through solemnly. The same goes for digital signatures:

A signature proves that the owner of the private key, say Alice, acknowledged the content. In e-mail communication, it avoids the risk that an attacker

- mimics the e-mail address of sender Alice, but
- enciphers with the key of the recipient Bob.

However, if the communication contains something Alice doesn't want to be seen by others (for example, to be read out loud by a prosecutor in court), better not prove her acknowledgement! Since this message may eavesdropped,

her correspondent may change his mind about the privacy of their conversation, or his account is hacked, ...

To give an analogy to our analog reality, an automatic digital signature by Alice compares to the recording of every private conversation of Alice.

In fact, usually Alice wants to prove only to Bob that she's the sender, but not to third parties! For this, in a group signature, an ephemeral key to sign is created and shared (that is, the public and *private* key) between Alice and Bob. This way, Alice and Bob are sure that the message was sent by the other correspondent, but a third party only that it was sent among the group members.

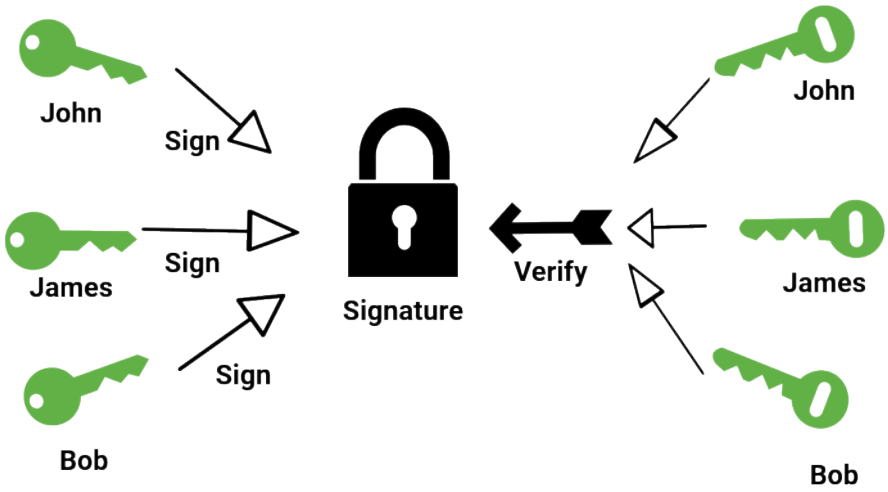


Figure 27: Group Signature

4.4 Public Key Infrastructures

A Public Key Infrastructure (**PKI**) of a network establishes trust among its spatially separated users by first authenticating them, then authorizing their public keys by signing them (referred to as digital certificates) and finally distributing them. In institutions and corporations, a PKI is often implemented as "a trust hierarchy" of Certification Authorities, whereas in looser communities it can be decentralized and trust mutually established by the users themselves.

A Public Key Infrastructure (**PKI**): establishes trust among its spatially separated users of a network by first authenticating them, then authorizing their public keys by signing them (referred to as digital certificates) and finally distributing them.

A PKI includes:

- (Digital) Certificates: public keys which are signed to authenticate their users. Other then the name and key, they contain additional personal data, such as an e-mail address, and usually an expiry date.
- Certificate Revocation List (CRL): A list of certificates that have been revoked before their validity expires, for example, because
  - the key has been compromised, or
  - the key owner is no longer trustable, for example, because of her departure.
- Directory Service: a searchable database of the emitted certificates; for example, in a trust hierarchy an LDAP server (Lightweight Directory Access Protocol; a standard used by large companies to administer access of users to files, printers, servers, and application data), and in the web of trust a server that hosts a database searchable by a web form.

Philosophy of Solutions. The identity of the key owner is confirmed by *third* parties, that is, other identities with private keys that confirm by their digital signatures that it is Alice who owns the private key.

However, the problem of the public key identity arises again: How can we ensure the identities of the private key owners? There are two solutions:

- *hierarchical* authorities, and
- the web of trust

For short: while in the Web-of-Trust the connections built by trust form a graph, in the approach by hierarchical authorities they form a tree.

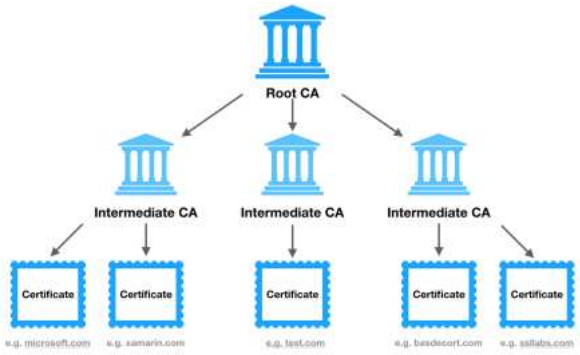


Figure 28: Hierarchical authorities, Cort (2018)

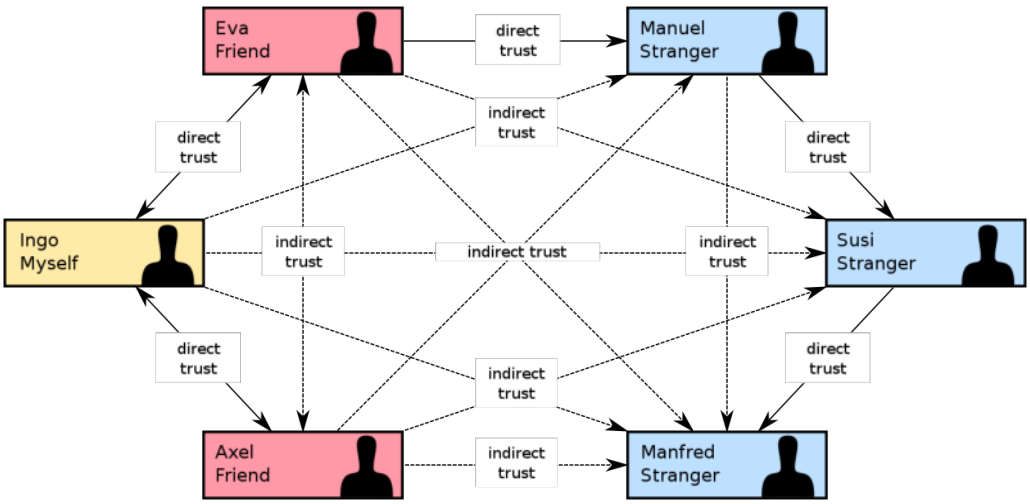


Figure 29: The web of trust, Kku (2019)

Hierarchical Authorities. In the approach via **hierarchical authorities**, private key owners are distinguished by hierarchical levels. At the highest level lie the *root authorities* on which one trusts unconditionally.

> **hierarchical authorities**: Key owners that confirm others' identities by digital signatures and are organized in a hierarchy, where trust passes from a higher to a lower level; total trust is placed in those at the highest level, the root authorities.

For example,

- VeriSign, GeoTrust, Commode, ... are major US certifying companies;
- as a recent addition, the (US intermediary) non-profit authority `Let us encrypt`;
- a look at the `/etc/ssl/certs` folder in the Linux distribution `openSUSE` reveals that there is, for example,
  - a German authority (`TeleSec` of `Deutsche Telekom AG`, the former national telecommunications operator),
  - three Spanish (`Firmaprofesional`, `ACCVRAIZ1` — Agencia de Tecnología y Certificación Electrónica, and `ACC RAIZ FNMT` — Fábrica Nacional de Moneda y Timbre), and
  - many U.S. authorities.

Web of Trust. In the **web of trust**, private key owners cannot be distinguished from each other.

> **web of trust**: Private key owners confirm other's identities by successively passing trust to each other among equals.

The absence of root authorities, unconditionally trusted entities, is compensated for by the

1. trust initially established by
   - having obtained the public key personally (for example, at *key-sign parties*, meetings where participants exchange and sign their public keys mutually), or
   - by
     1. having obtained the key through a different channel (Website, e-mail, ...) and
     2. having communicated your check sum via another channel (phone, `SMS`, instant messenger, ...);
2. then the trust is successively (transitively) passed from one to the other: If Alice trusts Bob, and Bob trusts Charles, then Alice trusts Charles.

Standardization of Philosophies on the Internet. On the Internet,

- the system of trust by hierarchical authorities has been standardized by the scheme `X.509`, principally used to encrypt the communication between a user and a (commercial) Website (but also between users in corporate environments, such as, `S/MIME` e-mail encryption), and
- The `OpenPGP` scheme (as implemented by the `GnuPG` program), with its main use of encrypting e-mails. This scheme radically rejects any hierarchy: the user can publish a public key with an e-mail address on a public key server such as that of the MIT without even confirming (by an activation e-mail) that he has access to the account of this e-mail address.

4.5 DANE

The IETF (Internet Engineering Task Force) proposed (in RFC 63941: DANE use cases and RFC 66982: DANE protocol) the DANE protocol that aims to cryptographically harden the TLS, DTLS, SMTP, and S/MIME protocols using DNSSEC. By DNSsec, a DNS resolver can authenticate a DNS resolution, that is, whether it is identical to that on the authoritative DNS server, by checking its signature (of the authoritative DNS server). Instead of relying, like these protocols, entirely on certificate authorities (CAs), domain holders

- can restrict the CAs that validate the domain's certificate, and
- can emit certificates for themselves, without reference to CAs.

Using CAs, there is no restriction on which CAs can issue certificates for which domains. If an attacker can gain control of a single CA among the many CAs that the client trusts, then she can emit fake certificates for every domain. DANE allows clients to ask the DNS servers, which certificate are trustworthy so that the domain holder can restrict the scope of a CA. When the user is passed a domain name certificate (as part of the initial TLS handshake), the client can check the certificate against a TLSA resource-record (TLSA-RR) published in the DNS for the service name which is authenticated by the authoritative DNS server.

The most common standard for a PKI is the hierarchy of `X.509` certificate authorities. X.509 was first published in 1998 and is defined by the International Telecommunications Union's Standardization sector (ITU-T), X.509 establishes in particular a standard format of electronic certificate and an algorithm for

the validation of certification path. The IETF developed the most important profile, PKIX Certificate and CRL Profile, or "PKIX" for short, as part of RFC 3280, currently RFC 5280. It is supported by all common web browsers, such as Chrome and Firefox, which come with a list of trustworthy `X.509` certificate authorities.

In more detail, the TLSA-RR contains the entry Certificate Usage, whose value (from `0` to `3`, the lower, the more restrictive) restricts the authority allowed to validate the certificate for the user:

0. PKIX-TA (CA constraint). The client's trust resides in a PKIX authority.
1. PKIX-EE (Service Certificate Constraint). The client's trust resides in a PKIX certificate.
2. DANE-TA (Trust Anchor assertion). The client's trust resides in an authority which, in contrast to PKIX-TA, does not have to be a PKIX certificate authority.
3. DANE-EE (Domain-issued certificate). The client's trust resides in an certificate which, in contrast to PKIX-EE, does not have to be a PKIX certificate.

The DANE check therefore serves to confirm certificates issued by public certification authorities. With DANE values (2 and 3), the domain holder has the option of creating his own, even self-signed certificates for his TLS-secured services, without having to involve a certification authority known to the client. By choosing between "Trust Anchor" (TA) and "End Entity" (EE), the domain owner can decide for himself whether to anchor DANE security to a CA or server certificate.

4.6 Hybrid Ciphers

> **hybrid encryption**: a two-key algorithm is used to authenticate the correspondents by digitally signing the messages or to exchange a key for single-key cryptography for efficient communication thereafter,

The most common key exchange method is to create a shared secret between the two parties by the Diffie-Hellman protocol and then hash it to create the encryption key. To avoid a man-in-the-middle attack, the exchange is authenticated by a certificate, that is, by signing the messages with a long term private

key to which the other party holds a public key to verify. Since single-key cryptographic algorithms are more efficient than two-key cryptographic algorithms by a considerable factor, the main use of two-key encryption is thus so-called **hybrid encryption** where the two-key algorithm is used

- to authenticate the correspondents by digitally signing the messages, or
- to exchange a key for single-key cryptography for efficient secure communication thereafter.

For example, in the `TLS` (Transport Layer Security; former SSL) protocol, which encrypts secure sites on the World Wide Web, a cryptographic package such as `TLS_RSA_WITH_3DES_EDE_CBC_SHA` (identification code `0x00 0x0a`) uses

- `RSA` to authenticate and exchange the keys,
- `3DES` in `CBC` mode to encrypt the connection, and
- `SHA` as a cryptographic hash.

Self-Check Questions.

1. What is a problem that Public Key cryptography solves? *Distributing a secret key over an open channel.*
2. What is a problem that Public Key cryptography does not solve? *The authentication of the private key owner.*
3. What are two common approaches to work around the Man-in-the-middle attack? *Certificate authorities and Web-of-trust*
4. What are common protocols to work around the Man-in-the-middle attack? `X.509` *(as used by* `S/MIME`*) and* `OpenPGP`

Summary

Single-key (or symmetric) cryptography suffers from the *key distribution problem*: to pass the same secret key to all, often distant, correspondents. Two-key (or asymmetric) cryptography solves this problem seemingly at once, by enabling the use of different keys to encrypt and decrypt. However, the identity of the key owner must be confirmed; if not personally, then by *third* parties, that is, identities with private keys that confirm by their digital signatures that it is Alice who owns the private key. However, the problem of the public key identity arises again: How can we ensure the identities of these private key owners?

There are two solutions: In the approach via hierarchical authorities, private key owners are distinguished by hierarchical levels. At the highest level lie the *root authorities* on which one trusts unconditionally. In the web of trust, trust is transferred from one to the other, that is, trust is transitive: If Alice trusts Bob, and Bob trusts Charles, then Alice trusts Charles.

Questions

Required Reading

Read the section on asymmetric cryptography in the article Simmons et al. (2016). Read in Menezes, Oorschot, and Vanstone (1997) Sections 3.1 and 3.3.

Further Reading

Read the parts of the book Schneier (2007) on understanding and implementing modern asymmetric cryptographic algorithms.

5 Modular Arithmetic

Study Goals

On completion of this chapter, you will have learned what a trapdoor function is, and:

1. why modular arithmetic is needed to define such a function, and
2. what modular arithmetic is by division with rest.

Introduction

The security of two-key cryptographic algorithms relies on the computational difficulty of a mathematical problem, for example, factoring a number that is the product of two large primes; ideally, computing the secret key is equivalent to solving the hard problem, so that the algorithm is at least as secure as the underlying mathematical problem is difficult. This has not been proved for any of the standard algorithms, although it is believed to hold for each of them.

To see the usefulness of (modular) arithmetic in cryptography, recall that asymmetric cryptography is based on a *trapdoor* function, which

- must be easily computable, but
- its inverse must be practically incomputable without knowledge of a shortcut, the key!

The ease of calculating the function corresponds to the ease of encryption, while the difficulty of calculating the inverse corresponds to the difficulty of decryption, that is, inverting the encryption. For example, RSA uses

- as an encryption function the raising to an n -th power, and
- as a decryption function its inverse, the root extraction.

While both, the function itself and even its inverse, are easily computed using the usual multiplication of numbers, instead cryptographic algorithms (such as RSA) use *modular arithmetic* to entangle the computation of the inverse function without knowledge of the key (which in RSA is root extraction).

We already know *modular* or *circular arithmetic* from the arithmetic of the clock, where $m = 12$ is considered equal to 0 : Because the indicator restarts counting from 0 after each turn, for example, 3 hours after 11 hours is 2 o'clock:

$$11 + 3 = 14 = 12 + 2 = 2.$$

Over these finite circular domains, called finite rings and defined in Section 5.3, (the graphs of) these functions become irregular and practically incomputable, at least without knowledge of a shortcut, the key.

In what follows, we

1. convince ourselves of the difficulty on this domain in contrast to that of the real numbers, and

2. introduce this domain.
3. we explain how to calculate the inverse function on it.

5.1 Modular Arithmetic as Randomization

The difficulty of calculating the inverse corresponds to the difficulty of decryption, that is, inverting the encryption. In an asymmetric cryptographic algorithm,

- the *ease* of encrypting (a number), and
- the *difficulty* of deciphering (a number)

are based on an invertible function such that

- it is *easily computable*, but
- its inverse function is *difficult* to compute.

For example, the inverses of the trap-door functions

- *raising to a power* $x \mapsto x^e$ (in the `RSA` algorithm), and
- *exponentiating* $x \mapsto g^x$ (in the `Diffie-Hellman` algorithm)

are given by

- the root extraction $x \mapsto x^{1/n}$, and
- the logarithm $\log_g$.

They are

- easily computable on the domain of real numbers $\mathbb{R}$ (for example, by the bisection method for continuous functions thanks to the connectedness of $\mathbb{R}$),
- but on their finite cryptographic domains they are almost incomputable.

Observation. Both functions are *algebraic*, that is, they are expressed by a formula of sums, products, and powers. *Analytical* functions, that is, infinite convergent sums, for example, sine, cosine, ..., are inconvenient by the rounding errors.

The domain of these functions is *not* the set of integers $\mathbb{Z}$ (or that of the real numbers $\mathbb{R}$ that includes them), because both functions, exponentiation and raising to a power, are *continuous* over $\mathbb{R}$:

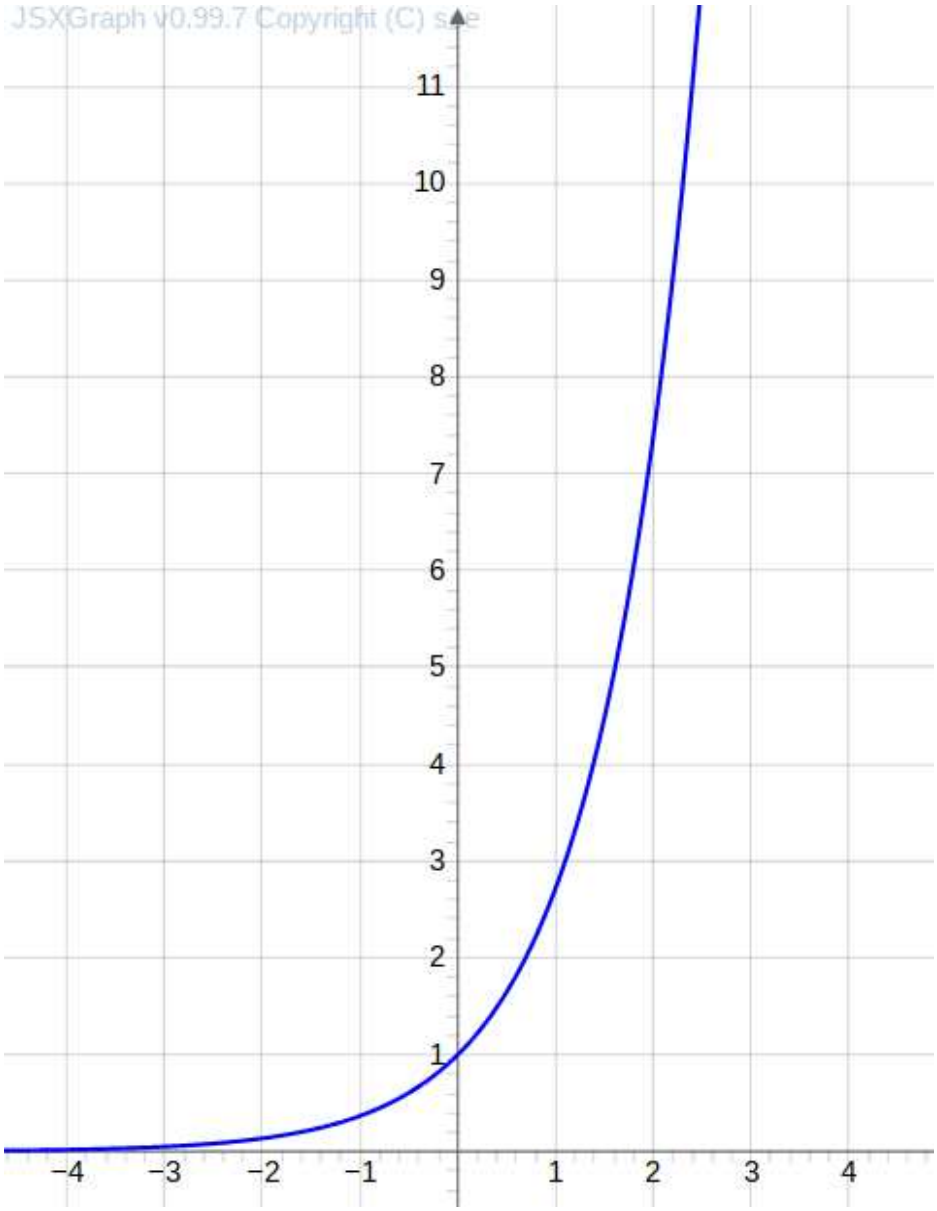


Figure 30: The function of the exponential exp (Wassermann (2020a))

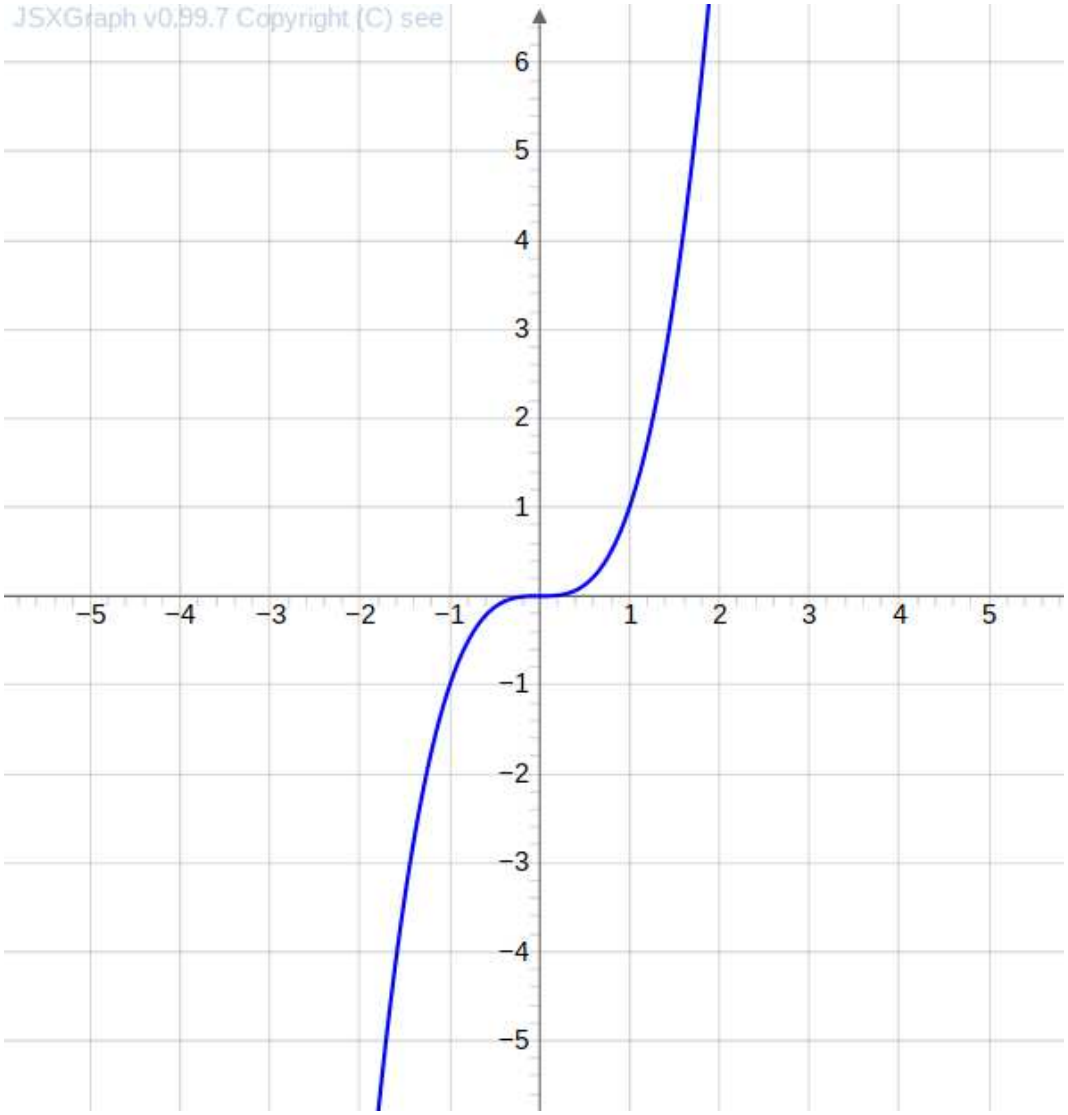


Figure 31: The parabola of the cubic function $x \mapsto x^3$ (Wassermann (2020b))

If the domain of these functions were $\mathbb{R}$, then their *inverses* could be *approximated* over $\mathbb{R}$, for example, by iterated bisection where the inverse point is besieged in intervals that are halved at every iteration: Given y_0, find an x_0 such that $f(x) = y_0$ is equivalent to finding a zero x_0 of the function

$$x \mapsto f(x) - y_0.$$

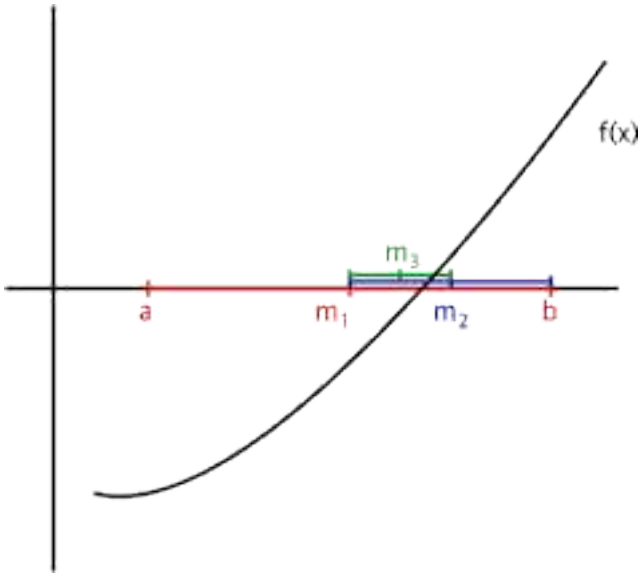


Figure 32: Bisection of a continuous function (Ziegler (2009))

1. (*Start*) Choose an interval $[a, b]$ such that

$$f(a) < 0 \text{ and } f(b) > 0.$$

2. (*Recalibration*) Calculate the midpoint $m := (a + b)/2$ of the interval $[a, b]$:

   - If $f(m) = 0$, then $m = x_0$, and we have found our zero.

   Otherwise:

   - either $f(m) < 0$, then replace the left edge a with m ,
   - or $f(m) > 0$, then replace the right edge b with m ,

   and recalibrate the newly obtained interval $[m, b]$ respectively $[a, m]$.

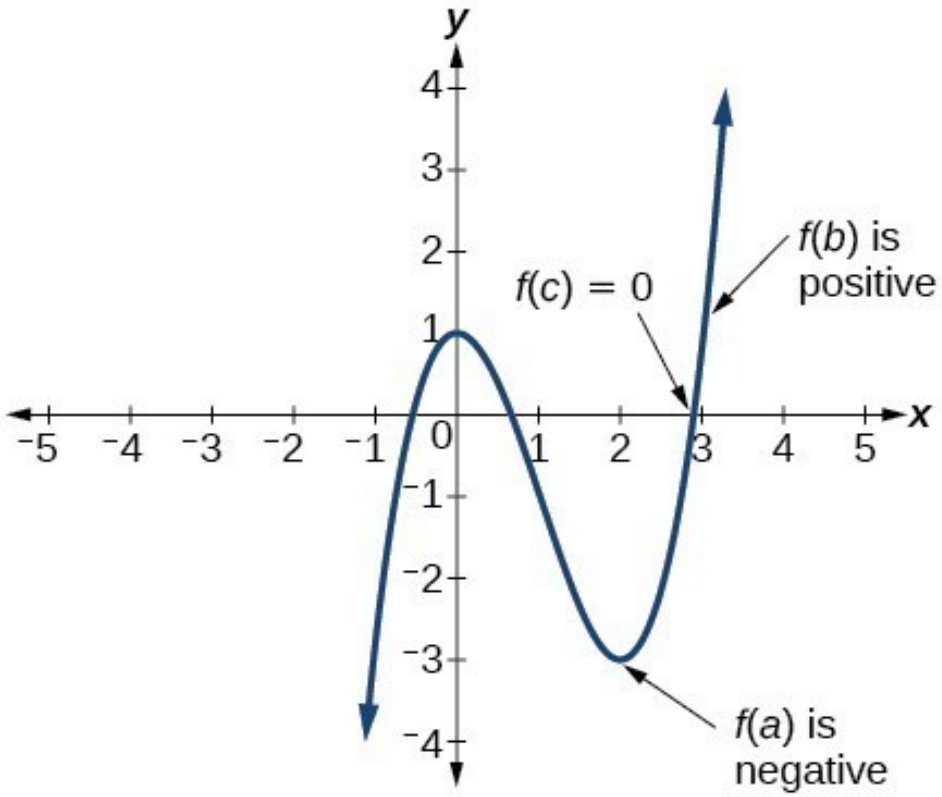


Figure 33: Intermediate Value Theorem (Abramson (2019))

By the *Intermediate Value Theorem*, the zero is guaranteed to be in the interval, which at each step decreases and converges to the intersection.

Finding the zero of the polynomial

$$\mathrm{F}(x) = x^3 + 3x^2 + 12x + 8$$

by bisection with starting points $x_1 = -5$ and $x_2 = 0$, yields in steps $i = 2, \ldots, 15$ the successive approximations

i	x_i	$\mathrm{F}(x)$	$x_i - x_{i-1}$
2	0	8	5
3	−2.5	−18.875	2.5
4	−1.25	−4.26563	1.25
5	−0.625	1.42773	0.625

i	x_i	$\mathrm{F}(x)$	$x_i - x_{i-1}$
6	−0.9375	−1.43726	0.3125
7	−0.7813	−0.02078	0.15625
8	−0.7031	0.69804	0.07813
9	−0.7422	0.33745	0.03906
10	−0.7617	0.15806	0.01953
11	−0.7715	0.06857	0.00977
12	−0.7764	0.02388	0.00488
13	−0.7783	0.00154	0.00244
14	−0.7800	−0.00962	0.00122

5.3 Finite Rings

To avoid the iterative approximation of the zero of a function and thus *complicate* the computation of the inverse function (besides facilitating the computation of the proper function), the domain of a trapdoor function is a **finite ring** denoted by

$$\mathbb{Z}/m\mathbb{Z} = \{0, 1, \ldots, m-1\}$$

for a natural number m.

> **finite ring**: A finite set that contains 0 and 1 and over which a sum + is explained that obeys the laws of associativity and commutativity.

In such a finite ring

$$m = 1 + \cdots + 1 = 0;$$

necessarily, *every addition* (and thus every multiplication and every raising to a power) *has result* $< m$. So the addition of $\mathbb{Z}/m\mathbb{Z}$ is different from that on $\mathbb{Z}$ (or $\mathbb{R}$). For example, for $m = 7$,

$$2^2 = 2 \cdot 2 = 4 \text{ and } 3^2 = 3 \cdot 3 = 7 + 2 = 0 + 2 = 2.$$

We will introduce these finite rings first by the examples $\mathbb{Z}/12\mathbb{Z}$ and $\mathbb{Z}/7\mathbb{Z}$, the rings given by the clock hours respectively weekdays), then for every m .

When we look at the graphs of the functions, which are so regular on $\mathbb{R}$, we note that over the finite ring $\mathbb{Z}/101\mathbb{Z}$, either graph, that of

- the exponentiation with base 2, and

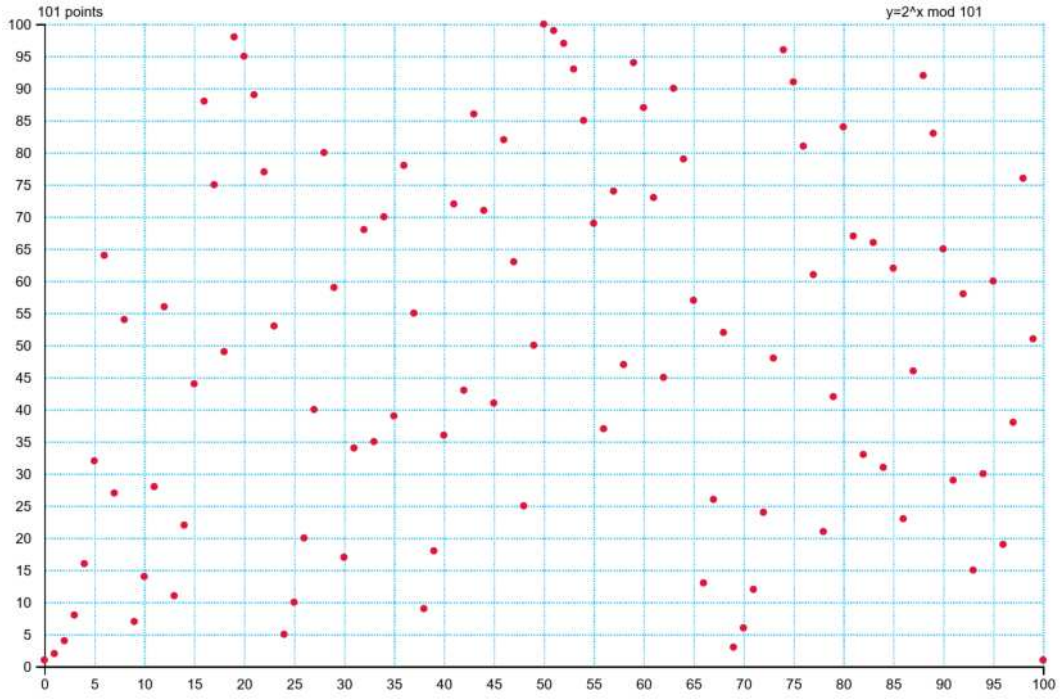


Figure 34: The graph of exponentiation with basis 2 over $\mathbb{Z}/101\mathbb{Z}$ (Grau (2018b))

- the parabola

is initially as regular over $\mathbb{Z}/101\mathbb{Z}$ as over $\mathbb{Z}$, but starting

- from $x = 7$ (because $2^7 = 128 > 100$), respectively
- from $x = 11$ (because $11^2 = 121 > 100$)

begins to behave erratically. (Except for the symmetry of the parabola on the central axis $x = 50.5$ due to $(-x)^2 = x^2$).

Task. Experiment with the function plotter Grau (2018d) to view the erratic behavior of other function graphs over finite domains.

5.4 Modular Arithmetic in Everyday Life

We apply modular arithmetic in everyday life when we add times in the daily, weekly and yearly cycle of clock hours, weekdays and months. It is this circularity that explains the naming “ring”.

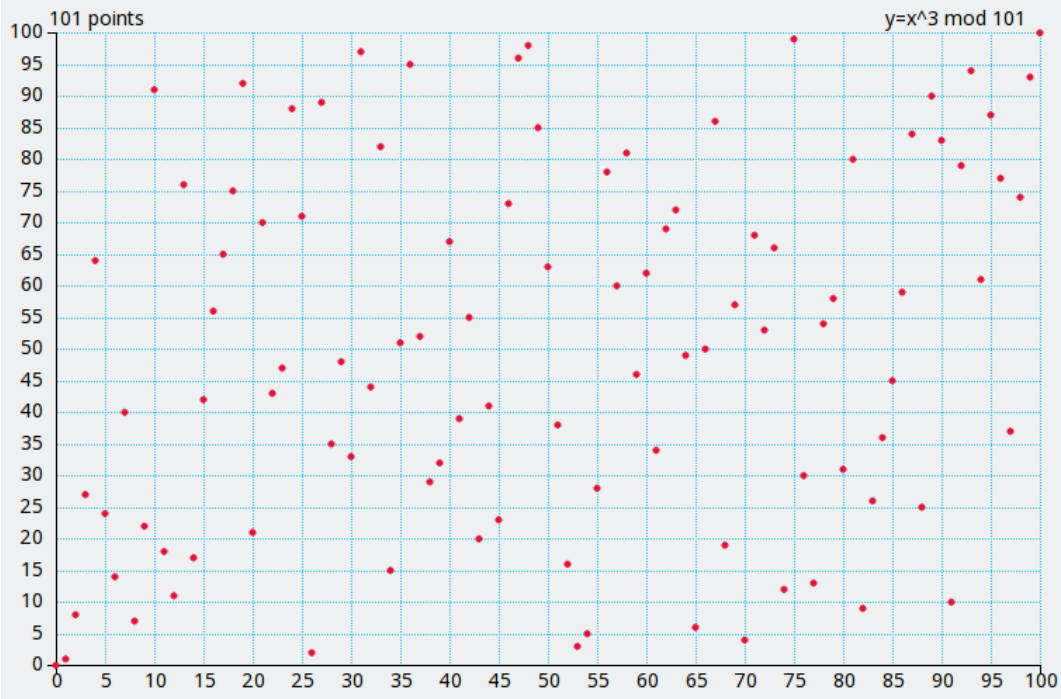


Figure 35: The graph of the cubic $Y = X^3$ over $\mathbb{Z}/101\mathbb{Z}$ (Grau (2018c))

Clock. The prototypical example of modular arithmetic is the arithmetic of the clock in which the pointer comes back to start after 12 hours; formally,

$$12 \equiv 0,$$

which implies, for example,

$$9 + 4 \equiv 1 \quad \text{and} \quad 1 - 2 \equiv 11. \tag{1}$$

That is, 4 hours after 9 hours is 1 o'clock, and 2 hours before 1 o'clock is 11 o'clock. We can go further: $9 + 24 \equiv 9$; that is, if it is 9 o'clock now, then in 24 hours (one day later) as well.

Days of the Week. Another example of modular arithmetic in everyday life are the days of the week: after 7 days, the days of the week start over: If we enumerate 'Saturday', 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday' and 'Friday' by $0, 1, 2, 3, 4, 5, 6$, then

$$7 \equiv 0,$$

Figure 36: The Clock as Ring of numbers 1 , 2 , . . . , 11 , 12 = 0; J. Smith (2009)

which implies, for example,

$$4 + 4 \equiv 1 \quad \text{and} \quad 1 - 2 \equiv 5.$$

Indeed, 4 days after Wednesday is Sunday, and 2 days before Sunday is Friday. We can go further: $5 + 14 \equiv 5$; that is if now it is Thursday, then in 14 days (two weeks later) as well.

Months. Another example of modular arithmetic in everyday life are the months of the year: after 12 months, the months of the year start over. If we number 'January', 'February', ... for 1, 2, ... then, as in the clock,

$$12 \equiv 0,$$

which implies, for example,

$$10 + 3 \equiv 1 \quad \text{and} \quad 1 - 2 \equiv 11;$$

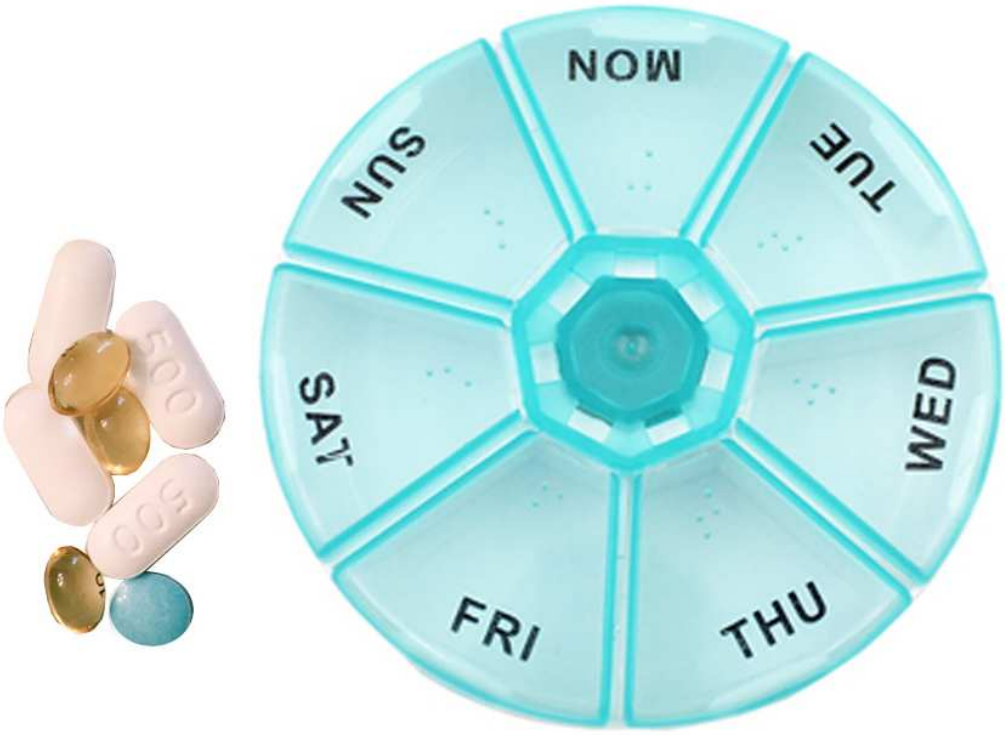


Figure 37: The weekly cycle of taking pills; Walmart (2019)

That is, a quarter after October the year starts over, and 2 months before January is November. We can go further: $5 + 24 \equiv 5$; that is, if it's 'May' now, then in 2 years as well.

5.5 Formalization

Formally, we derive the equation Equation 1 from the equalities

$$9 + 4 = 13 = 12 + 1 \equiv 0 + 1 = 1 \quad \text{and} \quad 1 - 2 = -1 = -1 + 0 \equiv -1 + 12 = 11.$$

and

$$9 + 24 = 9 + 2 \cdot 12 \equiv 9 + 2 \cdot 0 = 9.$$

In general, for every a and x in $\mathbb{Z}$,

$$a + 12 \cdot x \equiv a$$

or, equivalently, for all a and b in $\mathbb{Z}$,

$$a \equiv b \quad \text{if 12 divides } a - b.$$

Division with remainder. *Definition. Let a and b be positive integers. That a divided by b has **remainder** r means that there is such an integer q that*

$$a = b \cdot q + r \quad \text{with } 0 \leq r < b.$$

Example. For $a = 230$ and $b = 17$, we compute $230 = 17 \cdot 13 + 9$. That is, the remainder of 230 divided by 17 is 9 .

In other words, for every a and b in $\mathbb{Z}$,

$$a \equiv b$$

if and only if a and b leave the same remainder divided by 12.

There is nothing special about the number $m = 12$ (of clock hours). For example, analogous equalities would hold if the clock indicated $m = 16$ hours (as many as a day on the planet Neptune has):

Definition. Let m be a natural number. The integers a and b are **congruent** modulo m, formally,

$$a \equiv b \bmod m$$

if $m|a - b$, that is, if their difference $a - b$ is divisible by m. In other words, if a and b leave the same remainder divided by m.

The number m is called **modulus**.

Or, phrased differently, $a \equiv b \bmod m$ if a and b leave the same remainder divided by m .

> **Congruence**: Two integer a and b are congruent modulo m if they leave the same remainder after division by m .

Figure 38: A clock with 16 hours; Carleton (2011)

Construction. Let us finally define these finite domains $\mathbb{Z}/m\mathbb{Z}$ (*the ring of integers modulo m*) for a natural, usually prime, number m, on which the trap-door functions in asymmetric cryptography live; those that make a cryptanalyst's life so difficult when trying to compute their inverse (in contrast to the domains $\mathbb{R}$ or $\mathbb{Z}$).

Given an integer $m \geq 1$, we want to define the *ring* $\mathbb{Z}/m\mathbb{Z}$ (loosely, a set with an addition + and multiplication $\cdot$ governed by certain laws) such that

$$m = 1 + \cdots + 1 = 0.$$

More exactly, such a ring

- a set that contains 0 (= the neutral element of the addition) and 1 (= the neutral element of the multiplication),
- with two operations, the addition + (such that, for every x there is is an *inverse* $y = -x$, that is, $x + y = 0$) and the multiplication $\cdot$,
- that satisfy the associative, commutative and distributive law.

If this equality for + is to hold over $\mathbb{Z}/m\mathbb{Z}$, then the addition + over $\mathbb{Z}/m\mathbb{Z}$ has to be defined differently from that over $\mathbb{Z}$. We put

- as a set
$$\mathbb{Z}/m\mathbb{Z} := \{0, \ldots, m-1\},$$
- as neutral elements of addition and multiplication
$$0 \quad \text{and} \quad 1;$$
- As operations + and $\cdot$

  $x + y = \mathrm{r}(x + y)$ where $\mathrm{r}(x) =$ the remainder of $x + y$ divided by m

  and
  $$x \cdot y = \mathrm{r}(x \cdot y).$$
  The inverse $y = -x$ of x is given by $y = m - x$.

That is, to add and multiply in $\mathbb{Z}/m\mathbb{Z}$,

1. we calculate the sum or product as in $\mathbb{Z}$, and
2. We calculate its remainder divided by m.

For example, for $m = 4$ we get the addition and multiplication tables

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

and

*	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Exercise. Show that an integer is divisible by 3 (respectively 9) if, and only if, the sum of its decimal digits is divisible by 3 (respectively 9).

In `Python`, the modular operator is denoted by the percentage symbol `%`. For example, in the interactive shell, we get:

```
>>> 15 % 12
3
>>> 210 % 12
6
```

Conditions for Invertibility of Functions. The base g of $x \mapsto g^x$ and the exponent e of $x \mapsto x^e$ determine whether the function is invertible or not.

Exponential Function. The exponential function invertible if and only if it is *onto*, that is, every number, except 0 is a value of the function. For example, for $m = 101$, the values are contained in $\mathbb{Z}/101\mathbb{Z}^* := \mathbb{Z}/101\mathbb{Z} - \{0\}$:

- For example, for $g = 2$ over $\mathbb{Z}/101\mathbb{Z}$, its image is maximal, that is, it is $(\mathbb{Z}/101\mathbb{Z})^* := \mathbb{Z}/101\mathbb{Z} - \{0\}$; in other words, all the numbers (other than zero) are powers of g . (One says that *g generates* $\mathbb{Z}/101\mathbb{Z}^*$.)
- However, for example $g = 4 = 2^2$ generates on the same domain only half of $\mathbb{Z}/101\mathbb{Z}^*$, a set of 50 elements.

Theorem on the existence of a primitive root. A generator of $\mathbb{Z}/m\mathbb{Z}^*$ exists if, and only if,

- either $m = 1, 2, 4$,
- or $m = p^e$ respectively $m = 2p^e$ for a prime number $p > 2$.

Raising to a Power. A function (between a finite domain and counter-domain) is invertible if and only if it is *injective*, that is, sends different arguments to different values.

Task. Experiment with the function plotter Grau (2018d) to find examples of functions that are injective or not, that is, whose graph has two points at the same level.

If the exponent E is even, $E = 2e$ for an integer e , then raising to the power $x \mapsto x^E$ satisfies $(-x)^E = (-x)^e = ((-x)^2)^e = (x^2)^e = x^e = x^E$, that is, sends the arguments $-x$ and x to the same value. Thus, it is not injective. For example, for $m = 101$ and $e = 1$, we observe this symmetry in Figure 39 along the central axis $x = 50,5$ (but note that its restriction onto $0, \ldots, 50$ is injective).

Theorem. The raising to a power x^E is injective over $\mathbb{Z}/m\mathbb{Z}$

- for $m = p$ prime if and only if E has no common divisor with $p - 1$. For example, for $m = 101$, the exponent $E = 3$;
- for $m = pq$ with p and q prime (the kind of modulus used by RSA) if and only if E has no common divisor with neither $p - 1$ nor $q - 1$.

Example. For example, for $m = 21 = 3 \cdot 7$, the exponent $E = 5$ gives the invertible function $x \mapsto x^E$ on $\mathbb{Z}/m\mathbb{Z}$.

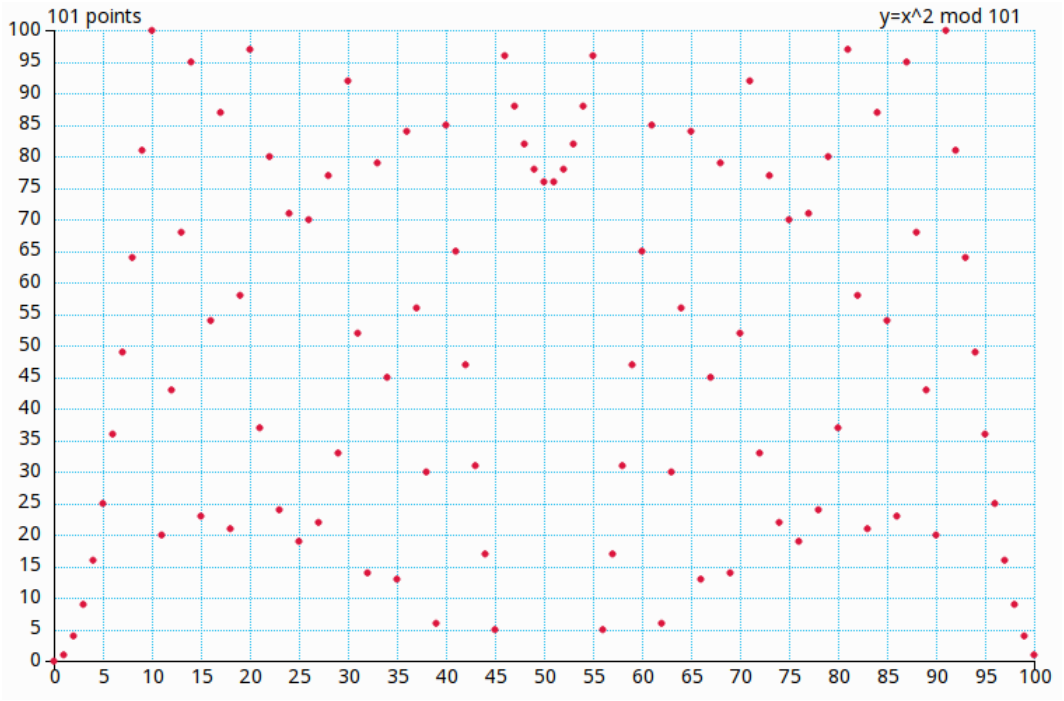


Figure 39: The graph of the quadratic $Y = X^2$ over $\mathbb{Z}/101\mathbb{Z}$ (Grau (2018e))

Self-Check Questions.

1. What is the remainder of 8^8 divided by 7 ?

   - ☐ *1*
   - ☐ 2
   - ☐ 4
   - ☐ 6

   Because $8 \equiv 1 \bmod 7$, *it is* $8^8 \equiv 1^8 = 1$.

2. Why is a number divisible by 3 if and only if the sum of its decimal digits is divisible by 3 ? *Because* $10 \equiv 1 \bmod 3$, *we have* $10^2, 10^3, \ldots \equiv 1 \bmod 3$. *Therefore, say* $a = a_2 \cdot 10^2 + a_1 \cdot 10 + a_0 \equiv a_2 + a_1 + a_0 \bmod 3$. *We have* $3|a$ *if and only if* $a \equiv 0 \bmod 3$.

5.6 Fast Raising to a Power

While the finite domains $\mathbb{Z}/m\mathbb{Z}$ for a natural number m complicate the computation of the inverse function of the trapdoor function, they actually *facilitate* the computation of the function itself given by the raising to a power:

- in the RSA algorithm, $x \mapsto x^{\mathrm{E}}$, and
- in the Diffie-Hellman key exchange, $x \mapsto g^x$.

Algorithm. Given a base b and an exponent e in $\mathbb{Z}$ to calculate

$$b^e \text{ in } \mathbb{Z}/\mathrm{M}\mathbb{Z},$$

1. expand the exponent in **binary** base, that is,

$$e = e_0 + e_1 2 + e_2 2^2 + \cdots + e_s 2^s \quad \text{with } e_0, e_1, \ldots, e_s \text{ in } \{0,1\},$$

2. compute

$$b^1, b^2, b^{2^2}, \ldots, b^{2^s} \bmod \mathrm{M}.$$

   Because $b^{2^{n+1}} = b^{2^n \cdot 2} = (b^{2^n})^2$, that is, each power is the *square of the previous one* (bounded by M), each power is, in turn, is easily computable.

3. raise to the power:

$$b^e = b^{e_0 + e_1 2 + e_2 2^2 + \cdots + e_s 2^s} = b^{e_0} (b^2)^{e_1} (b^{2^2})^{e_2} \cdots (b^{2^s})^{e_s}$$

   Only powers $e_0, e_1, \ldots, e_s$ equal to 1 matter, the others can be omitted.

This algorithm takes $2\log_2(e)$ module multiplications.

Examples. To calculate 3^5 module 7, expand

$$5 = 1 + 0 \cdot 2^1 + 1 \cdot 2^2$$

and calculate

$$3^1 = 3, 3^2 = 9 \equiv 2, 3^{2^2} = (3^2)^2 \equiv 2^2 = 4 \bmod 7,$$

yielding:

$$3^5 = 3^{1+2^2} = 3^1 \cdot 3^{2^2} = 3 \cdot 4 \equiv 5 \bmod 7.$$

To calculate 3^{11} module 5, expand

$$11 = 1 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3$$

and calculate

$$3^1 = 3, 3^2 = 9 \equiv 4, 3^{2^2} = (3^2)^2 \equiv 4^2 = 1 \text{ and } 3^{2^3} = 3^{2^2 \cdot 2} = (3^{2^2})^2 \equiv 1^2 = 1 \bmod 5,$$

yielding

$$3^{11} = 3^{1+2^1+2^3} = 3^1 \cdot 3^{2^1} \cdot 3^{2^3} = 3 \cdot 4 \cdot 1 = 12 \equiv 2 \bmod 5.$$

Summary

Asymmetric cryptography relies on a *trapdoor* function, which

- must be easily computable (for example, raising to the n-th power in RSA), but
- its inverse (for example, extraction of the n th root in RSA) must be practically incomputable without knowledge of a shortcut, the key!

This difficulty of calculating the inverse corresponds to the difficulty of decryption, that is, inverting the encryption. To *complicate* the computation of the inverse function (besides facilitating the computation of the proper function) is done using *modular* (or *circular*) arithmetic*, that we already know from the arithmetic of the clock, where $m = 12$ is considered equal to 0.

Questions

Required Reading

Read the section on asymmetric cryptography in the article Simmons et al. (2016). Read in Menezes, Oorschot, and Vanstone (1997), Sections 2.4, 2.5 and 2.6 on the basic notions of number theory behind public key cryptography. Use Grau (2018d) to get an intuition for the graphs over finite domains.

Further Reading

See the book Sweigart (2013a) for implementing some simpler (asymmetric) algorithms in Python, a readable beginner-friendly programming language.

Read the parts of the book Schneier (2007) on understanding and implementing modern asymmetric cryptographic algorithms.

6 Diffie-Hellman Key Exchange

Study Goals

On completion of this chapter, you will have learned the Diffie-Hellman key exchange using the (Discrete) Logarithm.

Introduction

In 1976, Whitfield Diffie and Martin Hellman conceived that the key distribution problem could be solved by an algorithm that satisfied:

- (computationally) easy creation of a matched pair of keys for encryption and decryption,
- (computationally) easy encryption and decryption,
- (computationally) infeasible recovery of one of the keys despite knowledge of:
  - the algorithm,
  - the other key, and
  - any number of matching plaintext and ciphertext pairs.
- (computationally) infeasible recovery of the plaintext for almost all keys k and messages x .

Observation: This was the first public appearance of two-key cryptography. However, the British Government Communications Headquarters (GCHQ) knew it around a decade earlier.

- James Ellis already had the idea a decade earlier but was unable to implement it,
- Clifford Christopher Cocks invented the widely-used encryption algorithm `RSA` about three years before it was independently developed by Rivest, Shamir, and Adleman, and
- Malcolm J. Williamson discovered what is now known as Diffie-Hellman key exchange while working at GCHQ.

The first published protocol to overtly agree on a mutual secret key is the **Diffie-Hellman key exchange protocol** published in Diffie and Hellman (1976).

This is not yet two-key cryptography, because the single secret key is known to *both correspondents* (in the following called Alice and Bob). The asymmetric cryptographic algorithms that build on this protocol (for example, `ElGamal` and `ECC`), generate a *unique* key for *every* message.

Diffie-Hellman Key exchange: overt agreement on a common secret key whose security relies on the infeasibility of computing the logarithm modulo a large number.

6.1 Key Exchange Protocol

Notation. Let us denote, in every asymmetric encryption algorithm,

- by an *upper case* letter exclusively a *public* number, and
- by a *lower case* letter preferably a *secret* number.

For both correspondent, say Alice and Bob, to overtly agree on a secret key, they first combine

- a *suitable* prime number p (the *modulus*), and
- a *suitable* natural number g (the *base*).

Then

1. Alice, to generate *one half* of the key, chooses a number a ,
   - calculates $\mathrm{A} \equiv g^a \bmod p$, and
   - transmits A to Bob.
2. Bob, to generate the *other half* of the key, chooses a number b ,
   - calculates $\mathrm{B} \equiv g^b \bmod p$, and
   - transmits B to Alice.
3. The secret *mutual* key between Alice and Bob is

$$c := \mathrm{A}^b \equiv (g^a)^b = g^{ab} = g^{ba} = (g^b)^a \equiv \mathrm{B}^a \mod p.$$

`CrypTool 1` offers in the menu entry `Individual Procedures -> Protocols` a dialogue to experiment with key values in the `Diffie-Hellman`.

Observation. This protocol shows how to overtly build a shared secret key. This key can then be used to encrypt all further communication, for example, by an asymmetric algorithm such as `AES`. However, the protocol shows

- neither how to encrypt a message (with a public key and decrypt it with the private key),

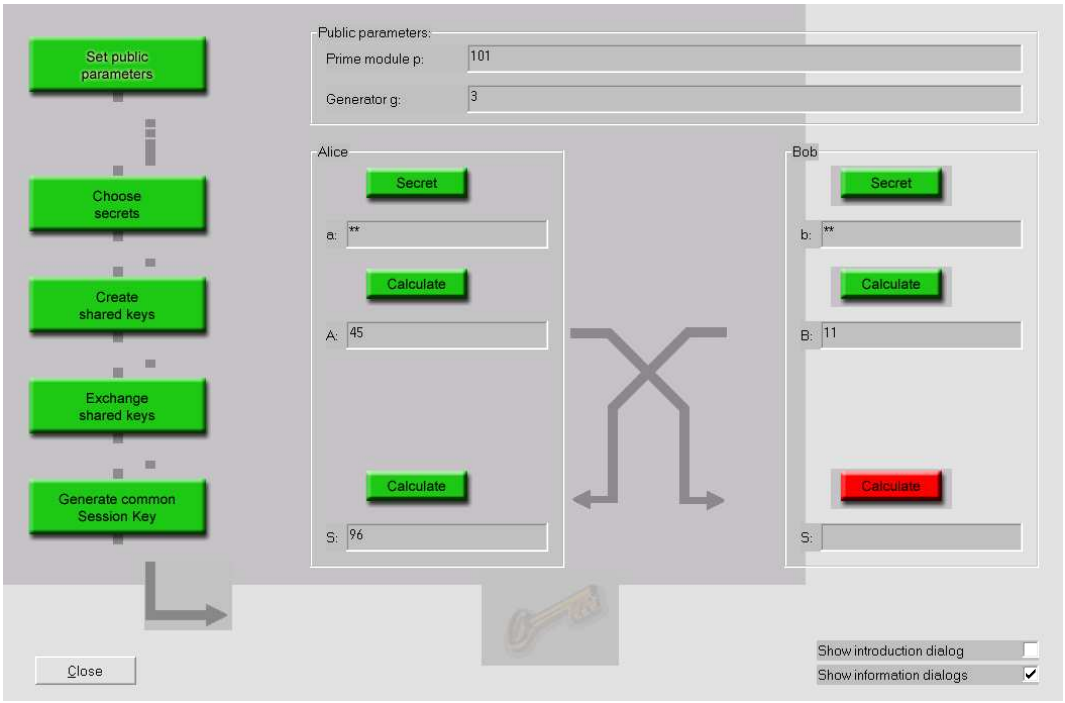


Figure 40: The steps of the `Diffie-Hellman` protocol in `CrypTool 1` (Esslinger et al. (2018a))

- nor how to sign one (with a private key and verify it with the public key).

ElGamal (1985) showed first how to build an encryption and signature algorithm on top of the Diffie-Hellman protocol. While its encryption algorithm is rarely employed (albeit, for example, the standard cryptography command-line tool `GnuPG` offers it as first alternative to `RSA`), its signature algorithm forms the basis of the Digital Signature Algorithm (DSA), which is used in the US government's Digital Signature Standard (DSS), issued in 1994 by the National Institute of Standards and Technology (NIST).

Elliptic Curve DSA (ECDSA) is a variant of the Digital Signature Algorithm (DSA) which uses points on finite (elliptic) curves instead of integers. The general number field sieve computes keys on DSA in subexponential time (whereas the ideal would be exponential time) . Elliptic curve groups are (yet) not vulnerable to a general number field sieve attack, so they can be (supposedly) securely implemented with smaller key sizes.

6.2 Security

The security of the Diffie-Hellman key exchange is based on the difficulty of computing the logarithm modulo p . An eavesdropper would obtain the secret key $\mathrm{A}^b = \mathrm{B}^a$ from A and B, if he could compute the *logarithm* $\log_g$ as inverse of the *exponentiation* $x \mapsto g^x = y$, that is

$$a = \log_g \mathrm{A} \quad \text{or } b = \log_g \mathrm{B} \mod p;$$

While a power is easily computable (even more so using the fast power algorithm in Section 5.6), even more so in modular arithmetic, its inverse, the *logarithm*, the exponent for a given power, *is practically incomputable* for p and g *appropriately chosen*, that is:

- the prime number p
  - is *large* and
  - there is a *large* prime number q that divides $p - 1$ (at best, p is a *safe prime*, that is, $p - 1 = 2q$ with q prime);
- the powers g , g^2, ... of the base g generate a *large* set (that is, its cardinality is a multiple of q).

Let us look for such appropriate numbers:

6.3 Appropriate Numbers

Euclid's Theorem. There are infinitely many prime numbers.

Demonstration: Otherwise, there are only finitely many prime numbers, say p_1, ..., p_n are all of them. Consider $q = p_1 \ldots p_n + 1$. Since q is greater than p_1, ..., p_n, it cannot be prime. Let p be a prime number that divides q . Because p_1, ..., p_n are prime, p divides at least one of p_1, ..., p_n. However, by its definition, q has remainder 1 divided by every prime p_1, ..., p_n. The last two statements are in contradiction! Therefore, there must be an infinite number of primes.

Euclid's Theorem. There are infinitely many prime numbers.

Euclid's Theorem guarantees that there are arbitrarily many *big* prime numbers (in particular, > 2048 bits).

Thank Heavens, for almost every prime number p there is a prime number q *large* (> 768 bits) that divides $p - 1$.

The Theorem on the existence of a *Primitive Root* ensures that (since the modulus is prime) there is always a number g in $\mathbb{F}_p^*$ such that

$$\{g, g^2, g^3, ..., g^{p-1}\} = \mathbf{F}_p^*$$

That is, every number 1 , 2 , 3 , $\ldots$, $p-1$ is a suitable power of g . In particular, the cardinality of 1 , g , g^2, $\ldots$, g^{p-1} is a multiple of any prime q that divides $p - 1$. In practice, the numbers p and g are taken from a reliable source, such as a standards committee.

6.4 Padding

Since initially (for $x < \log_g p$) the values g^x over $\mathbb{Z}/p\mathbb{Z}$ equal to the values g^x over $\mathbb{Z}$, the secret numbers a and b should be large enough, that is, $> \log_g p$. To ensure this, in practice these numbers are artificially increased, that is, the message is padded.

At present, the fastest algorithm to calculate the logarithm x from g^x , is an adaption of the *general number field sieve*, see Gordon (1993), that achieves subexponential runtime. That is, roughly, the number of operations to calculate the logarithm of an integer of n bits is exponential in

$$n^{1/3}.$$

Self-Check Questions

1. Does the Diffie-Hellman protocol explain how to encipher a message by a public key and decipher it by a secret key? *No, it only explains how to construct a mutual secret key publicly.*

Summary

The Diffie-Hellman Key exchange protocol shows how to build overtly a mutual secret key based on the difficulty of computing the logarithm modulo p . This key can then be used to encrypt all further communication, for example, by an asymmetric algorithm such as `AES`.

However, it shows

- neither how to encrypt a message (with a public key and decrypt it with the private key),
- nor how to sign one (with a private key and verify it with the public key).

ElGamal (1985) showed first how to build an encryption and signature algorithm on top of the Diffie-Hellman protocol; in particular, it gave rise to the Digital Signature Algorithm, DSA.

Questions

Required Reading

Read in Menezes, Oorschot, and Vanstone (1997) Sections 3.1, 3.3 and try to understand as much as possible in 3.2 and 3.6 on the number theoretic problems behind public key cryptography.

Further Reading

Use `CrypTool 1` to experiment with the `Diffie-Hellman` protocol.

See the book Sweigart (2013a) for implementing some simpler (asymmetric) algorithms in `Python`, a readable beginner-friendly programming language.

Read the parts of the book Schneier (2007) on understanding and implementing modern asymmetric cryptographic algorithms.

7 Euclid's Theorem

Study Goals

On completion of this chapter, you will have learned ...

- what a trapdoor function is, and:
    1. why modular arithmetic is needed to define such a function, and
    2. what modular arithmetic is by division with rest.

Introduction

We study multiplication in the finite rings $\mathbb{Z}/n\mathbb{Z}$. We take a particular interest in what numbers we can divide into them. It will be the Euclid Algorithm that computes the answer for us.

The private key

- in the `RSA` algorithm, or
- of the message encryption in the `ElGamal` algorithm (based on Diffie-Hellman key exchange),

defines a function that is the inverse of the function defined by the public key. This inverse is computed via the *greatest common divisor* between the two numbers. This, in turn, is computed by Euclid's *Algorithm*, an iterated division with rest.

We then introduce

- the notion of the *greatest common divisor* of two whole numbers, and
- how to calculate it by division with remainder, the so-called *algorithm of Euclid.*

7.1 Euclid's Algorithm

Definition. A common **divisor** of two whole numbers a and b is a natural number that divides both a and b. **The greatest common divisor** of two whole numbers a and b is the greatest natural number that divides both a and b. Denote by $\gcd(a,b)$ the greatest common divisor of a and b ,

$$\gcd(a,b) = \text{thegreatestnaturalnumberthatdivides} a \text{and} b.$$

Example. The greatest common divisor of 12 and 18 is 6 .

Iterated *Division with rest* yields an efficient algorithm to calculate the largest common divisor, *Euclid's Algorithm.*

Definition. *The integers a and b are* ***relatively prime*** *if* $\gcd(a,b) = 1$, that is, if no integer > 1 divides a and b.

For all integer numbers a and b, integer numbers a/g and b/g for $g = \gcd(a,b)$ are relatively prime.

Division with rest helps us build an efficient algorithm to calculate the largest common divisor. Let us look back on *Division with Rest*:

Definition. *Be a and b positive integer numbers. That a divided by b has **quotient** q and **rest** r means*

$$a = b \cdot q + r \quad \text{with } 0 \leq r < b. \tag{2}$$

Example. For $a = 19$ and $b = 5$, we get $19 = 5 \cdot 3 + 4$. That is, the remainder of 19 divided by 5 is 4.

Euclid's Algorithm. A **linear combination** (or **sum of multiples**) of two whole numbers a and b is a sum

$$s = \lambda a + \mu b$$

for whole numbers λ and μ.

Example. For $a = 15$ and $b = 9$, a sum of multiples of them is

$$s = 2 \cdot a + (-3) \cdot b = 2 \cdot 15 - 3 \cdot 9 = 3.$$

In particular, looking at the division with remainder in Equation 2, for an entire number d, we observe:

- if d divides a and b, then its linear combination $r = a - q \cdot b$, and, equally,
- if d divides b and r, then its linear combination a.

That is, d divides a and b if, and only if, d divides b and r. That is, the common dividers of a and b are the same as those of b and r. In particular,

$$\gcd(a,b) = \gcd(b,r).$$

By dividing the numbers b and r (which is $< b$), we obtain

$$b = r \cdot q' + r' \quad \text{with } 0 \leq r' < r$$

e

$$\gcd(b,r) = \gcd(r,r').$$

Iterating, and thus diminishing the remainder, we arrive at $s = r'\ldots'$ and $r'\ldots''$ with $r'\ldots'' = 0$, that is

$$\gcd(a,b) = \ldots = \gcd(s,0) = s.$$

That is, the highest common divisor is the *penultimate* remainder (or the last one other than 0).

Example. To calculate $\gcd(748,528)$, we get

$748 = 528 \cdot 1 + 220$

$528 = 220 \cdot 2 + 88$

$220 = 88 \cdot 2 + 44$

$88 = 44 \cdot 2 + 0$

thus $\gcd(528,220) = 44$.

`CrypTool 1`, in the entry `Indiv. Procedures -> Number Theory Interactive -> Learning Tool for Number Theory`, Section `1.3`, page 15, shows an animation of this algorithm:

Theorem. (Euclidean algorithm) Let a and b be positive whole numbers with $a \geq b$. The following algorithm calculates $\gcd(a,b)$ in a finite number of steps:

(start) Put $r_0 = a$ and $r_1 = b$, and $i = 1$.

(division) Divide r_{i-1} by r_i with rest to get

$$r_{i-1} = r_i q_i + r_{i+1} \quad \text{with } 0 \leq r_{i+1} < r_i.$$

Then

- either $r_{i+1} > 0$, then put $i := i + 1$ and continue with the step (division),
- or $r_{i+1} = 0$, then $r_i = \gcd(a,b)$ and the algorithm ends.

Demonstration: We need to demonstrate that the algorithm ends with the highest common divisor of a and b:

- Like $r_0 > r_1 > \ldots$, finally $r_I = 0$ for I large enough, and the algorithm ends.

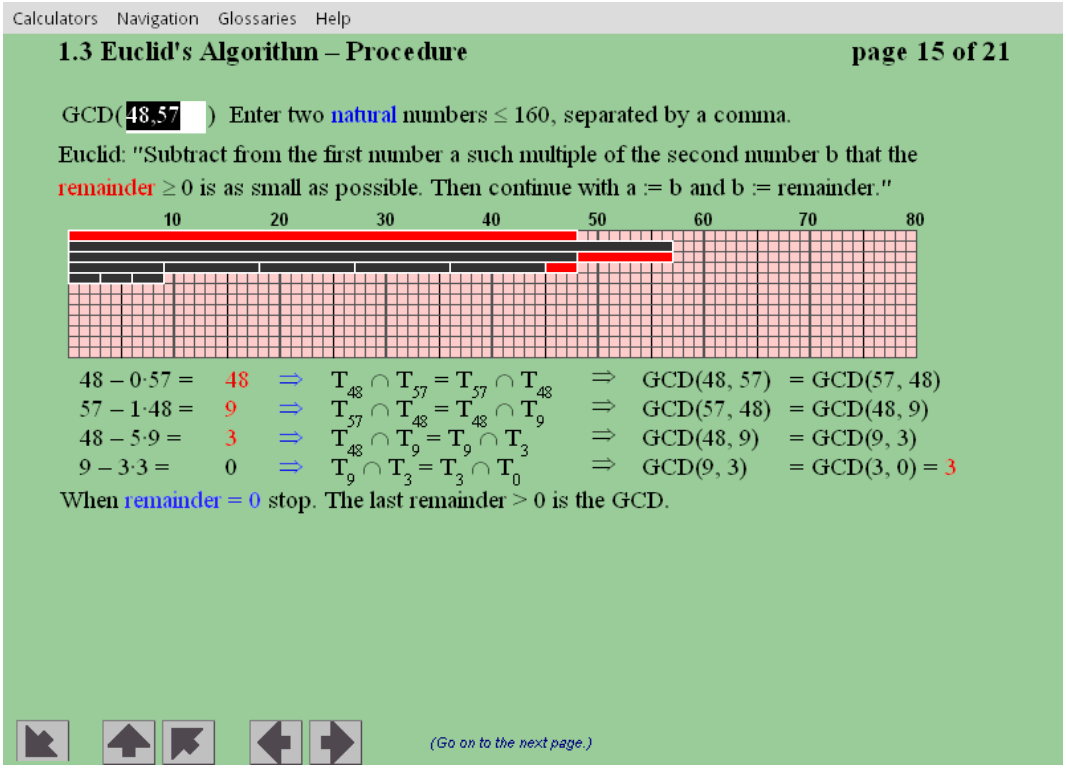


Figure 41: Euclid's algorithm in `CrypTool 1`, (Esslinger et al. (2008))

- For Equation 2, we have

$$\gcd(r_{i-1}, r_i) = \gcd(r_i, r_{i+1}) \quad \text{for all } i = 1, 2, \ldots$$

  As ultimately $r_{I+1} = 0$ for I big enough, we have

$$\gcd(a, b) = \gcd(r_0, r_1) = \ldots = \gcd(r_i, r_{i+1}). = \gcd(r_I, 0) = r_I$$

  That is, $r_I = \gcd(a, b)$.

Observation: All it takes is $2 \cdot \log_2 b + 1$ divisions with rest for the algorithm to finish.

Demonstration: We demonstrate

$$r_{i+2} < 1/2 \cdot r_i. \quad (\dagger)$$

We have

- or $r_{i+1} \leq 1/2 \cdot r_i$, and then $r_{i+2} < r_{i+1} \leq r_i$,
- or $r_{i+1} > 1/2 \cdot r_i$.

In the latter case, it follows

$$r_i = r_{i+1} \cdot 1 + r_{i+2}$$

and then

$$r_{i+2} = r_i - r_{i+1} < r_i - 1/2 \cdot r_i = 1/2 \cdot r_i.$$

For (†) we get iteratively that just $2 \cdot \log_2 b + 1$ divisions with rest for the algorithm finish.

In fact, it turns out that a factor of 1.45 is enough $\log_2(b) + 1.68$ with rest, and on average 0.85 is enough $\log_2(b) + 0.14$.

Euclid's Extended Algorithm. For the computation of the exponent of the decryption function, we need more information than the largest common divisor (calculated by the Euclid Algorithm). In fact, one observes (*Euclid's Extended Algorithm*) that in each step of the Euclid's Algorithm the largest common divisor $\gcd(x, m)$ of x and m is a *linear combination* (or *sum of multiples*) of x and m , that is,

$$\gcd(x, m) = \lambda x + \mu m \quad \text{for integers } x \text{ and } m.$$

The inverse of x modulo m is one of these multiples:

Example. For $a = 15$ and $b = 9$, a sum of multiples of them is

$$s = 2 \cdot a + (-3) \cdot b = 2 \cdot 15 - 3 \cdot 9 = 3.$$

Theorem. (Euclid's Extended Algorithm) For any positive integers a and b , their highest common divisor $\gcd(a, b)$ is a linear combination of a and b ; that is, there are integers u and v such that

$$\gcd(a, b) = au + bv.$$

CrypTool 1, in the `Indiv. Procedures -> Number Theory Interactive -> Learning Tool for Number Theory`, Section `1.3`, page 17, shows an animation of this algorithm:

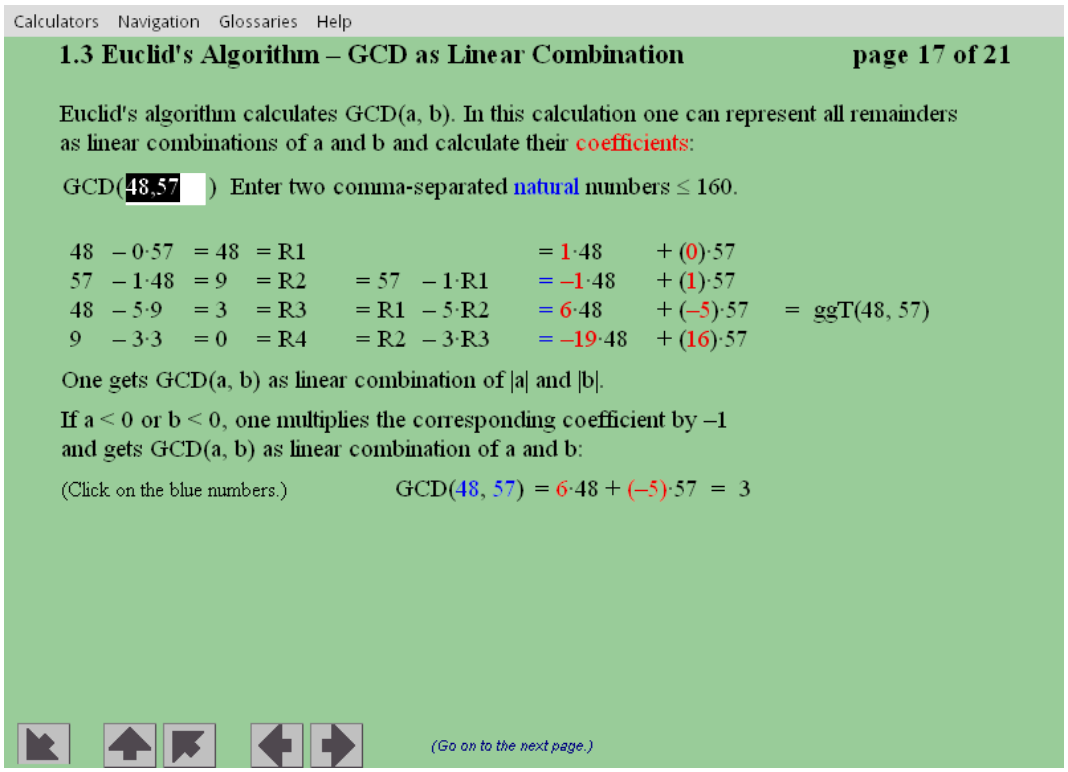


Figure 42: Euclid's extended algorithm in CrypTool 1 (Esslinger et al. (2008))

Example. We have $\gcd(528,220) = 44$ and, indeed,

$$44 = 5 \cdot 748 - 7 \cdot 528.$$

Demonstration: As $r_0 = a$, $r_1 = b$, and $r_2 = r_0 - q_1 r_1$, it follows that r_2 is a linear combination of a and b. In general, since r_{i-1} and r_i are linear combinations of a and b, first $q_i r_i$ is a linear combination of a and b, and so

$$r_{i+1} = r_{i-1} - q_i r_i$$

is a linear combination of a and b. In particular, if $r_{I+1} = 0$ then $r_I = \gcd(r_I, r_{I+1}) = \gcd(a,b)$ is a linear combination of a and b.

CrypTool 1, in the menu entry `Indiv. Procedures -> Number Theory Interactive -> Learning Tool for Number Theory`, Section `1.3`, page 17, shows an animation of this algorithm:

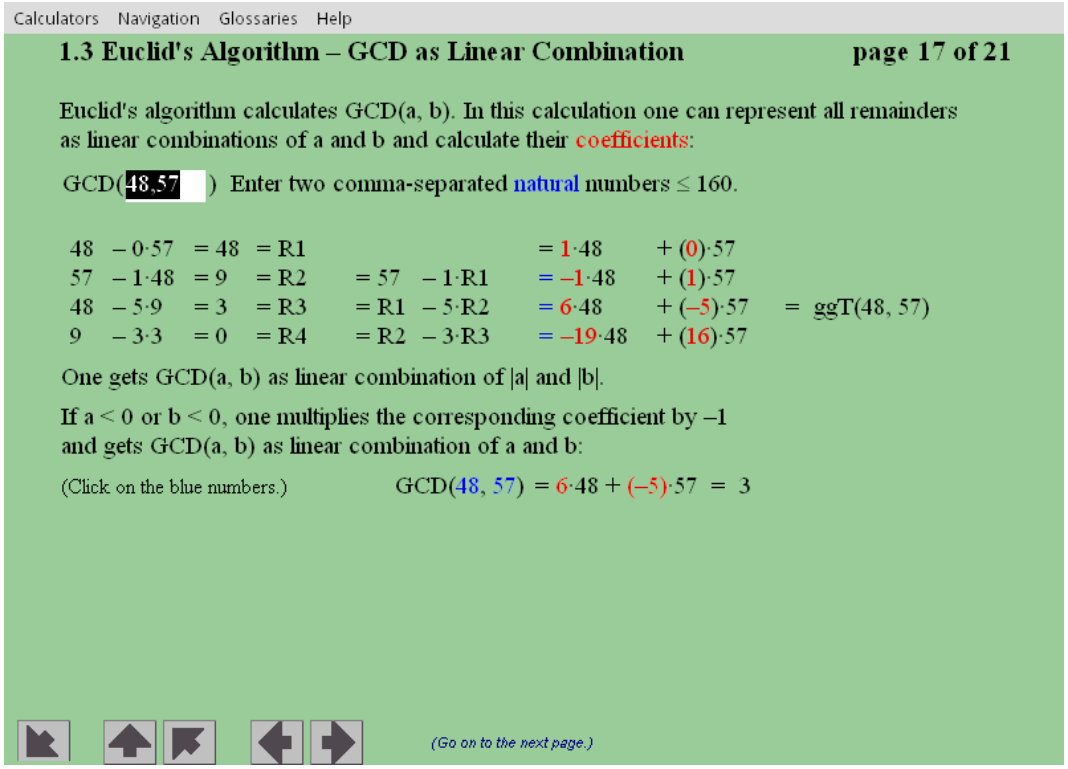


Figure 43: The Euclid algorithm extended in CrypTool 1

Example. Let's review the calculation of the largest common divisor of $a = 748$ and $b = 528$. Euclid's algorithm did:

$748 = 528 \cdot 1 + 220$

$528 = 220 \cdot 2 + 88$

$220 = 88 \cdot 2 + \mathbf{44}$

$88 = 44 \cdot 2 + 0,$

which provides the linear combinations

$220 = 748 - 528 \cdot 1 = a - b$

$88 = 528 - 220 \cdot 2 = b - (a - b) \cdot 2 = 3b - 2a$

$\mathbf{44} = 220 - 88 \cdot 2 = (a - b) - (3b - 2a) \cdot 2 = 5a - 7b.$

Indeed,

$$44 = 5 \cdot 748 - 7 \cdot 528.$$

Implementation in Python. To implement the Euclid algorithm, we will use multiple assignment in Python:

```
>>> spam, eggs = 42, 'Hello'
>>> spam
42
>>> eggs
Hello
```

The names of the variables and their values are listed to the left of = respectively.

Euclid's Algorithm. Here is a function that implements the Euclid algorithm in Python; it returns the largest common divisor $\gcd(a, b)$ of two whole a and b.

```
def gcd(a, b):
    while a != 0:
        a, b = b % a, a
    return b
```

For example, in the interactive shell:

```
>>> gcd(24, 30)
6
>>> gcd(409119243, 87780243)
6837
```

Extended Euclide's Algorithm. The // operator will figure in the implementation of the extended Euclid algorithm; it divides two numbers and rounds down. That is, it returns the greater integer equal to or less than the result of the division. For example, in the interactive shell:

```
>>> 41 // 7
5
>>> 41 / 7
5.857142857142857
>>> 10 // 5
2
>>> 10 / 5
2.0
```

We chose the following implementation of the extended Euclid algorithm:

```
def egcd(a, b):
    x,y, u,v = 0,1, 1,0
    while a != 0:
        q, r = b//a, b%a
        m, n = x-u*q, y-v*q
        b,a, x,y, u,v = a,r, u,v, m,n
    gcd = b
    return gcd, x, y
```

7.2 Modular Units

The private key is calculated by the *multiplicative reverse* in the modular arithmetic from the public key, both in the `RSA` algorithm and the `ElGamal` algorithm.

We have just learned how to calculate the largest common divisor by the Euclid Extended Algorithm; we now learn how it is used to calculate this *multiplicative reverse.*

While in $\mathbb{Q}$ we can divide by any number (except 0), in $\mathbb{Z}$ only by ± 1! The numbers you can divide by are called *invertible* or *units.* The amount of invertible numbers in $\mathbb{Z}/m\mathbb{Z}$ depends on the m module. Roughly speaking, the fewer prime factors in m, the more units in $\mathbb{Z}/m\mathbb{Z}$.

Definition. *The element x in $\mathbb{Z}/m\mathbb{Z}$ is a **unit** (or **invertible**) if there is y in $\mathbb{Z}/m\mathbb{Z}$ such that $yx = 1$. The element y is the **inverse** of x and denoted by x^{-1}. The set of units (where we can multiply and divide) is denoted by*

$$(\mathbb{Z}/m\mathbb{Z})^* := \text{the units in } \mathbb{Z}/m\mathbb{Z}.$$

Euler's totiente function # Φ is

$$m \mapsto \#\mathbb{Z}/m\mathbb{Z}^*;$$

that is, given m, it counts how many units $\mathbb{Z}/m\mathbb{Z}$ has.

- The neutral element 0 addition is never a unit. (But possibly, depending on m, all other elements).
- While in $\mathbb{Z}$ the only units are ± 1, in $\mathbb{Z}/m\mathbb{Z}$ possibly all of its elements, except 0, are.

Examples:

- On the clock, that is, for $\mathbb{Z}/12\mathbb{Z}$ the multiplication $v \cdot h$ of one hour h by v corresponds to iterate v times the path taken by the indicator in h (from $0 = 12$.) We note that for $h = 1, 5, 7$ and 11 there is a iteration of the path that leads the indicator to 1 (more exactly made 1, 5, 7 and 11 times), while for all other numbers this iteration leads the indicator to 0. These possibilities are mutually exclusive, and we conclude

  $$(\mathbb{Z}/12\mathbb{Z})^* = \{1, 5, 7, 11\}.$$

  That's $\Phi(12) = 11$.

- On days of the week, that is, for $\mathbb{Z}/7\mathbb{Z}$, we get

  $$(\mathbb{Z}/7\mathbb{Z})^* = \{1, 2, 3, 4, 5, 6\}.$$

  That is, the number of units is as large as possible, that is, P$hi(7) = 6$, all numbers except 0.

- For $\mathbb{Z}/4\mathbb{Z}$, the multiplication table

*	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

  shows

  $$(\mathbb{Z}/4\mathbb{Z})^* = \{1, 3\}$$

because $1 \cdot 1 = 1$ and $3 \cdot 3 = 1$ in $\mathbb{Z}/4\mathbb{Z}$. In contrast, $2 \cdot 2 = 0$ in $\mathbb{Z}/4\mathbb{Z}$, in particular 2 is not a unit. (But *a zero divisor*; in fact, each element in $\mathbb{Z}/m\mathbb{Z}$ is either a unit, or a zero divider). Thus $\Phi(4) = 2$.

- For $\mathbb{Z}/5\mathbb{Z}$, the multiplication table

*	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

  reveals the units $(\mathbb{Z}/5\mathbb{Z})^* = \{1,2,3,4\}$.

Proposition. *Let* $\mathbb{N}$ *and* x *in* $\mathbb{Z}/m\mathbb{Z}$*, that is,* x *in* $\{0,1,\ldots,m-1\}$*. The number* x *is a unit in* $\mathbb{Z}/m\mathbb{Z}$ *if, and only if,* $\gcd(x,m) = 1$.

Demonstration: We observe that each common divisor of x and m divides each sum of multiple $s = ux + vm$ of x and m; in particular, if $s = 1$, then the largest common divisor of $\bar{x}$ and m is 1.

By the Extended Euclid Algorithm, there is u and v in $\mathbb{Z}$ such that

$$ux + vm = \gcd(x,m).$$

So, from the above observation, $\gcd(x,m) = 1$ if, and only if, there is u in $\mathbb{Z}$ such that

$$ux \equiv 1 \bmod m.$$

That is, bx is a unit in $\mathbb{Z}/m\mathbb{Z}$ whose inverse is $bx^{-1} = bu$.

Observation. We concluded that for x in $\{0,\ldots,m-1\}$ with $\gcd(x,m) = 1$, we obtained by the Extended Euclid Algorithm u and v in $\mathbb{Z}$ such that

$$ux + vm = 1$$

The reverse x^{-1} of x in $\mathbb{Z}/m\mathbb{Z}$ is given by the remainder of u divided by m.

In `Python`, we can then calculate the inverse of a in $\mathbb{Z}/m\mathbb{Z}$ by

```
def ModInverse(a, m):
    if gcd(a, m) != 1:
        return None # no mod. inverse exists if a and m not rel. prime
    else
        gcd, x, y = egcd(a,m)
        return x % m
```

Self-Check Questions

1. What mathematical function is used to encrypt in RSA? Raising to a power modulo a composed integer.*
2. What mathematical function is used to decrypt in RSA? *Taking a root modulo a composed integer.*
3. What is a principal use of RSA on the Internet today? *The verification of certificates.*

Summary

Asymmetric cryptography relies on a *trapdoor* function, which

- must be easily computable (for example, raising to the n-th power in RSA), but
- its inverse (for example, extraction of the n th root in RSA) must be practically incomputable without knowledge of a shortcut, the key!

This difficulty of calculating the inverse corresponds to the difficulty of decryption, that is, inverting the encryption. To *complicate* the computation of the inverse function (besides facilitating the computation of the proper function) is done using *modular* (or *circular*) arithmetic*, that we already from the arithmetic of the clock, where $m = 12$ is considered equal to 0 .

Questions

Required Reading

Read the section on asymmetric cryptography in the article Simmons et al. (2016). Read in Menezes, Oorschot, and Vanstone (1997), Sections 3.1, 3.3 and try to understand as much as possible in 3.2 and 3.6 on the number theoretic problems behind public key cryptography

Further Reading

Use `CrypTool 1` to experiment with Euclid's algorithm.

See the book Sweigart (2013a) for implementing some simpler (asymmetric) algorithms in `Python`, a readable beginner-friendly programming language.

Read the parts of the book Schneier (2007) on understanding and implementing modern asymmetric cryptographic algorithms.

8 Classic Asymmetric Algorithms: RSA, ElGamal and DSA

Study Goals

On completion of this chapter, you will have learned ...

- what a trapdoor function is, and:
  1. why modular arithmetic is needed to define such a function, and
  2. what modular arithmetic is by division with rest.
- the most common asymmetric cryptographic algorithms and their underlying trapdoor functions:
  - the RSA Algorithm using the (discrete) Power Function.

Introduction

The best-known public-key algorithm is the Rivest-Shamir-Adleman (**RSA**) cryptoalgorithm from Rivest, Shamir, and Adleman (1978). A user secretly chooses a pair of prime numbers p and q so large that factoring the product $N = pq$ is beyond estimated computing powers for the lifetime of the cipher. The number N will be the modulus, that is, our trapdoor function will live on $\mathbb{Z}/N\mathbb{Z} = \{0, 1, ..., N-1\}$.

> **RSA**: algorithm that encrypts by raising to a power and whose security relies on the computational infeasibility of factoring a product of prime numbers.

N is public, but p and q are not. If the factors p and q of N were known, then the secret key can be easily computed. For `RSA` to be secure, the factoring must be computationally infeasible; nowadays `2048` bits. The difficulty of factoring roughly doubles for each additional three digits in N .

8.1 Algorithm

Having chosen p and q, the user selects any integer e less than n and relatively prime to $p-1$ and $q-1$, that is, so that 1 is the only factor in common between e and the product $(p-1)(q-1)$. This assures that there is another number d for which the product ed will leave a remainder of 1 when divided by the least common multiple of $p-1$ and $q-1$. With knowledge of p and q, the number d can easily be calculated using the Euclidean algorithm. If one does not know p and q, it is equally difficult to find either e or d given the other as to factor n, which is the basis for the cryptosecurity of the RSA algorithm.

The `RSA` algorithm creates

- a public key to encrypt, and
- a private key to decipher.

Compared to the `Diffie-Hellman` protocol, it has the advantage that it is completely asymmetric: there is no need to share a mutual secret key (and therefore the secret key is kept in a single place only, but not two). Instead a single correspondent has access to the secret key. However, in this case the

communication is encrypted only towards the owner of the secret key. To encrypt in both directions,

- either each correspondent creates an asymmetric RSA key,
- or the other correspondent enciphers and sends a symmetric key (which is so-called hybrid encryption, as it combines an asymmetric with a symmetric cipher).

8.2 Euler's Formula

The keys for encrypting, E and decryption, d , will be constructed via *Euler's Formula*, which in turn is based on *Fermat's Little Theorem.*

Fermat's Theorem. *Fermat's Little Theorem.* If p is a prime number, then for any integer a ,

- or $a^{p-1} \equiv 0 \bmod p$ if $p \mid a$,
- or $a^{p-1} \equiv 1 \bmod p$ if $p \nmid a$.

In particular, for every integer a ,

$$a^p = a^{(p-1)+1} = a^{p-1}a = a.$$

For example, if $m = \mathrm{E}d$, that is, E and d are such that $\mathrm{E}d \equiv 1 \bmod p-1$ (so to speak, $d \equiv 1/\mathrm{E} \bmod p-1$) then

$$a^{\mathrm{E}d} = (a^{\mathrm{E}})^d \equiv a \mod p,$$

that is,

$$a^d = a^{1/\mathrm{E}} = \sqrt[\mathrm{E}]{a}!$$

That is, the computation of the E -th root $\sqrt[\mathrm{E}]{\cdot}$ is equal to that of the d -th power $\cdot^d$, a great computational shortcut! The existence of such a shortcut d given E is assured by Euler's formula:

Euler's Formula. *Theorem. (Euler's formula)* Let p and q be different prime numbers. If a is divisible by neither p nor q , then

$$a^{(p-1)(q-1)} \equiv 1 \mod pq.$$

Proof: By Fermat's Little Theorem,

$$a^{(p-1)(q-1)} = (a^{p-1})^{q-1} \equiv 1^{q-1} = 1 \mod p$$

and

$$a^{(q-1)(p-1)} = (a^{q-1})^{p-1} \equiv 1^{p-1} = 1 \mod q$$

that is, p and q divide $a^{(p-1)(q-1)} - 1$. Since p and q are different prime numbers, pq divides $a^{(p-1)(q-1)-1}$, that is, $a^{(p-1)(q-1)} \equiv 1 \bmod pq$.

Taking Roots. *Corollary.* (Taking roots modulo N) Let p and q be different prime numbers, $\mathrm{N} = pq$ and $\phi(\mathrm{N}) = (p-1)(q-1)$. For every exponent n such that

$$n \equiv 1 \mod \phi(\mathrm{N})$$

we have

$$a^n \equiv a \mod \mathrm{N} \quad \text{for every integer } a.$$

Demonstration: If p or q divides a , then $a \equiv 0 \bmod \mathrm{N}$. Otherwise, we can apply Euler's Formula: Because $n \equiv 1 \bmod (p-1)(q-1)$, that is, there is ν such that $n - 1 = \nu(p-1)(q-1)$, by Euler's Formula,

$$a^n = a^{\nu(p-1)(q-1)+1} = (a^{(p-1)(q-1)})^{\nu} \cdot a \equiv 1^{\nu} \cdot a^1 = a \mod \mathrm{N}.$$

Observation (crucial for the RSA *algorithm).* If $m \equiv 1 \bmod \phi(\mathrm{N})$, then by *Euler's Formula* $a^m \equiv a \bmod \mathrm{N}$, that is, taking to the power is the identity function,

$$\cdot^m \equiv \mathrm{id} \mod \mathrm{N}.$$

In particular, if $m = \mathrm{E}d$ is the product of two whole numbers E and d , that is,

$$\mathrm{E}d \equiv 1 \mod \phi(\mathrm{N}),$$

then

$$a = a^m = a^{\mathrm{E}d} = (a^{\mathrm{E}})^d.$$

That is, $\cdot^d = \cdot^{1/\mathrm{E}} \bmod \mathrm{N}$. Calculating a power is *much easier* than a root!

Example. If $p = 3$ and $q = 11$ then

$$\mathrm{N} = pq = 33 \quad \text{and} \quad \phi(\mathrm{N}) = (p-1)(q-1) = 20.$$

If $\mathrm{E} = 7$ and $d = 3$, then $n = \mathrm{E}d = 21 \equiv 1 \bmod 20$. For example, for base 2 , we check

$$2^{\mathrm{E}} = 2^7 = 128 = 29 + 3 \cdot 33 \equiv 29 \mod \mathrm{N}$$

and

$$29^d = 29^3 = (-4)^3 \equiv -64 = 2 - 2 \cdot 33 \equiv 2 \mod \mathrm{N}.$$

That is,

$$\sqrt[\mathrm{E}]{29} = 2 = 29^d \mod \mathrm{N}.$$

8.3 Encryption Algorithm

(Recall that an *upper case* letter denotes a *public* number (and vice-versa), whereas a *lower case* letter denotes a *secret* number.) For Alice to secretly send the message m to Bob through an insecure channel:

1. Bob, to *generate* the key, chooses

   - two prime numbers p and q , and
   - an exponent E relatively prime to $\phi(\mathrm{N}) := (p-1)(q-1)$.

   Bob, to *transmit* the key, sends to Alice

   - the product $\mathrm{N} := pq$ (the *modulus*) and the exponent E (the *public key*).

2. Alice, to *cipher*,

   - calculates $\mathrm{M} = m^{\mathrm{E}} \bmod \mathrm{N}$, and
   - transmits M to Bob.

3. Bob, to *decipher*,

   - calculates (by Euclid's extended Algorithm) d such that $\mathrm{E}d \equiv 1 \bmod (p-1)(q-1)$ (and which exists because E is relatively prime to $\phi(\mathrm{N})$),

- calculates $M^d = m^d = m \bmod N$ (by *Euler's Formula*).

Computing d by knowing both prime factors of N is Bob's shortcut. `CrypTool 1` offers in the menu `Individual Procedures -> RSA Cryptosystem` the entry `RSA Demonstration` to experiment with different values of the key and message in RSA.

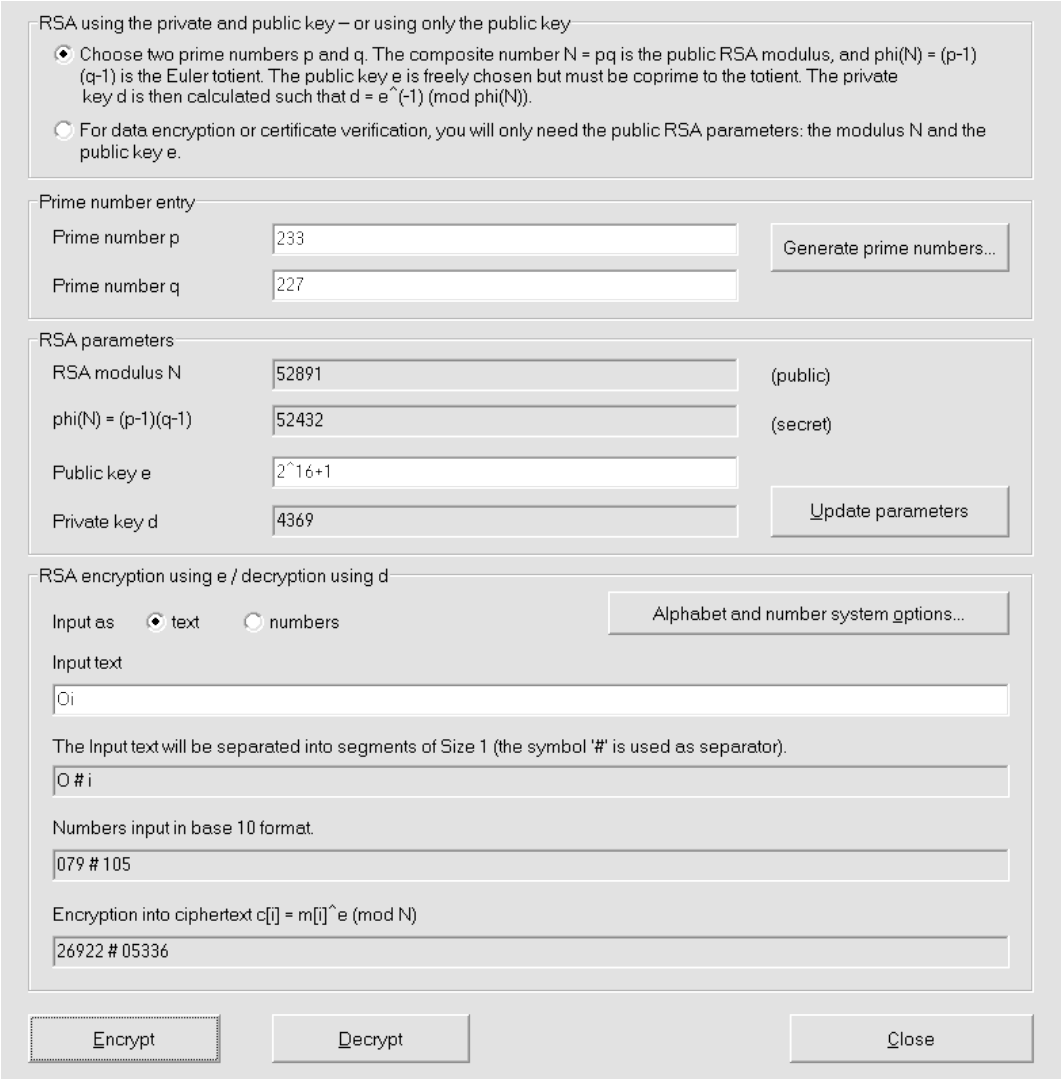


Figure 44: The encrpytion steps in RSA shown by `CrypTool 1` (Esslinger et al. (2018b))

In sum, raising to the power $y = x^E \bmod N$ encrypts where the exponent E

is the public key. Correspondingly, its inverse, taking the E-th root $x = y^{1/\mathrm{E}} \bmod \mathrm{N}$, decrypts. It is practically incomputable. However, modulo N, by Euler's formula, there is d such that

$$y^{1/\mathrm{E}} = y^d \mod \mathrm{N}$$

for a number d that Euclid's Algorithm calculates from E as well as p and q. Therefore, the secret key is d, or, sufficiently, the knowledge of the prime factors p and q of N.

8.4 Security

Since

- the modulus N,
- the exponent E and
- the encrypted message $\mathrm{M}\ (= m^{\mathrm{E}})$,

are all *public*, the computational security of RSA is solely based on the *difficulty of finding a root modulo a large number*

$$m \equiv \sqrt[\mathrm{E}]{\mathrm{M}}\ (= \mathrm{M}^{1/\mathrm{E}}) \mod \mathrm{N}.$$

An eavesdropper would obtain the secret message m from N, E and M only if he could compute

$$m \not\equiv \sqrt[\mathrm{E}]{\mathrm{M}}\ (= \mathrm{M}^{1/\mathrm{E}}) \mod \mathrm{N}.$$

The *shortcut* is the knowledge of the two prime factors p and q of $\mathrm{N} = pq$ that makes it possible to calculate

- the modulus $\phi(\mathrm{N}) := (p-1)(q-1)$, and
- the inverse multiplicative $d = 1/\mathrm{E}$ of E, that is, d such that

$$\mathrm{E}d \equiv 1 \mod (p-1)(q-1);$$

so that, by the *Euler Formula*, $\mathrm{M}^d = m^d \equiv m \bmod \mathrm{N}$.

Key Size Recommendations. Euclid's Theorem guarantees that there are arbitrarily *large* prime numbers (in particular, > 15360 bits). While taking powers is computationally easy, taking roots is computationally hard for *suitable* choices of N = pq and E , that is, for large enough prime numbers p and q (while the choice of the exponent E is free; for example, E = 2):

The fastest algorithm to calculate the prime factors p and q from N is the *general number sieve*, see A. K. Lenstra et al. (1993). The number of operations to factor an integer number of n bits is roughly

$$\exp(\log n^{1/3}).$$

Therefore, according to Barker (2016), the National Institute for Standards and Technology (NIST)

- recommends N to have 2048 bits, that is, p and q each have to have about 310 decimal digits.
- a key length of 3072 bits for security beyond 2030, and
- a key length of 15360 bits for security comparable to that of 256-bit symmetric keys.

Paddings. In practice, the plaintext number m needs to be *padded*, that is, the number m randomly increased. Otherwise, when the plaintext number m and the exponent E are both small (for example, E = 3), then possibly the enciphered message M = m^{E} satisfies M < N . In this case, the equality

$$m = \sqrt[\mathrm{E}]{\mathrm{M}} \quad \text{holds in } \mathbb{Z}!$$

So m is easily computable, for example, by the *Bisection method* already presented, (or, numerically more efficient, by Newton's method).

Attacks. A simple hypothetical attack is that by Wiener when

- d is too small, that is, $d < 1/3 \cdot n^{1/4}$, and
- p and q are too close, that is, $q < p < 2q$.

If the conditions of the theorem are met, then the secret d can be computed by a linear time algorithm as the denominator of a continuous fraction.

Observation. If E (but not d) is too small, then an attack is much more difficult; see Boneh et al. (1999).

8.5 Applications

RSA is still a standard of many asymmetric cryptographic methods. One principal use of RSA nowadays lies in the verification of (older) certificates emitted by certificate authorities. See Section 13.

Other uses are that by GPG for asymmetric cryptography, such as the OpenPGP protocol to encrypt e-mail messages, which creates RSA keys by default:

```
→ ~ LC_ALL=en gpg --full-gen-key
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
@What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
@Key is valid for? (0) 1y
Key expires at Mon Jan 11 23:03:23 2021 CET
@Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

@Real name: Fulano
@Email address: fulano@bar.org
@Comment:
You selected this USER-ID:
    "Fulano <fulano@bar.org>"

@Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
_
```

Figure 45: Default key generation with GPG (Koch et al. (2020))

The command line program GPG creates keys and makes it possible to (de)encrypt and sign/authenticate with them. Other (graphical) applications, for example, the Enigmail extension for the free e-mail client Thunderbird, use it for all these cryptographic operations.

Self-Check Questions.

1. What mathematical function is used to encrypt in RSA? *Raising to a power modulo a composed integer.*
2. What mathematical function is used to decrypt in RSA? *Taking a root modulo a composed integer.*
3. What is a principal use of RSA on the Internet today? *The verification of certificates.*

8.6 Signatures

Let us recall that in public-key cryptography there are two keys, a *public* key and a *private* key. Usually:

- The *public* key is used to encrypt, while the *private* key is used to decipher.

Thus, a text can be transferred from the encipherer (Alice) to one person only, the decipherer (Bob). The roles of the public and private keys can be reversed:

- The *private* key is used to encrypt, while the *public* key is used to decipher.

Thus, the encipherer can prove to all decipherers (those who have the public key) his ownership of the private key; the *digital signature.*

> **digital signature**: encryption of a message by the private key followed by decryption by the public key to check whether the original message was encrypted by the private key.

The theory (meaning the mathematics) behind the encryption by the public key (digital messages) or private key (digital signature) is almost the same; only the roles of the trap function arguments are reversed. (For example, in the RSA algorithm, this exchange of variables is indeed all that happens). In practice, however, usually encrypted by the private key are:

- *paddings* of the plaintext by the public key (to avoid pathologies that reveal the key when the text is too short), and
- a cryptographic *hash*

That is, while

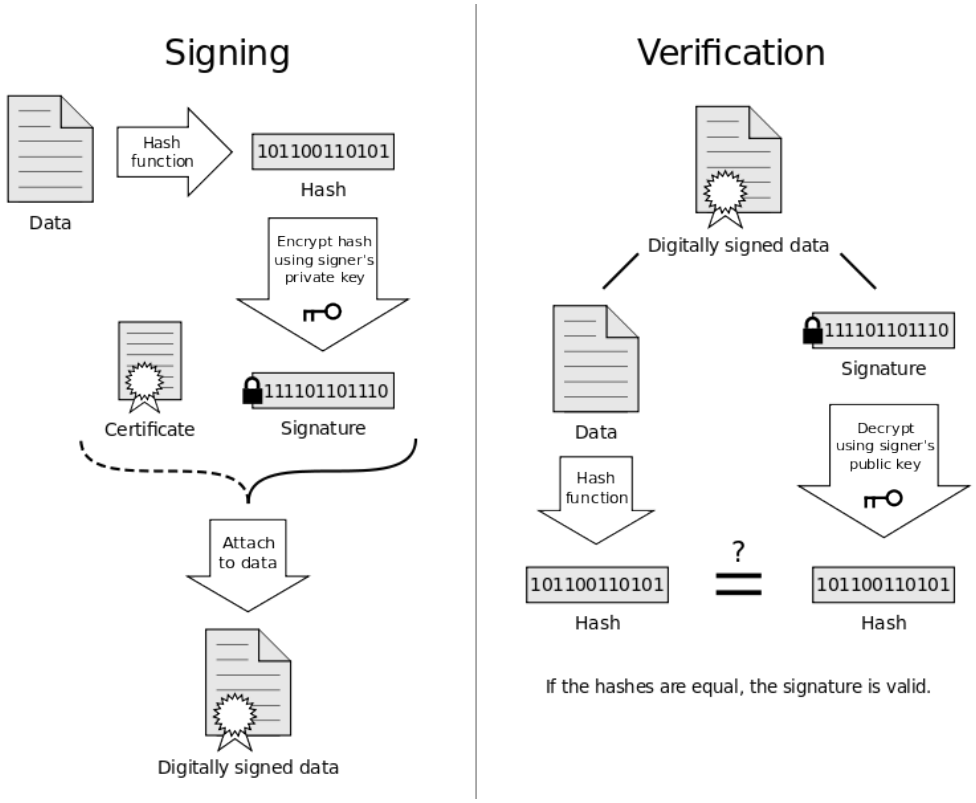


Figure 46: Digital Signature of a document and its Verification. (Acdx (2019))

- for encryption by the public key, the function used to first transform the text (the padding) is easily invertible,
- for private key encryption, the function used to first transform the text (the hash) is hardly invertible.

RSA Signature Algorithm. In the RSA Signature Algorithm, to sign (instead of encrypt), the only difference is that the exponents E and d exchange their roles. That is, the signed message is M = m^d (instead of m^{E}). For Samantha to sign the m message and Victor to verify it,

1. Samantha, to *generate* a signet, chooses
    1. two prime numbers p and q , and
    2. an exponent E relatively prime to $(p-1)(q-1)$, and

Samantha, to *transmit* the signet, sends to Victor

3. the product N = pq (the *modulus*) and the exponent E (the *public key*).

2. Samantha, to *sign*,

    1. calculates (by the Euclid's extended Algorithm) d such that $Ed \equiv 1 \mod (p-1)(q-1)$ (which exists because E is relatively prime to $(p-1)(q-1)$),
    2. calculates $M = m^d \mod N$, and
    3. transmits M to Victor.

3. Victor, to *verify*, calculates $M^E = m^{Ed} \equiv m \mod N$ (which holds by *Euler's Formula*).

Observation. Signing and the deciphering are both given by $\cdot^d$ for the private key d . So, signing an encrypted document (for the public key E that corresponds to d) is equivalent to deciphering it! Therefore, in practice,

- different key pairs are used
    - to encrypt / decrypt, and
    - to sign / verify, and
- a cryptographic *hash* $h(d)$ of the document d is signed, a small number that identifies the document.

`CrypTool 1` offers in the menu `Individual Procedures -> RSA Cryptosystem` the entry `Signature Generation` to experiment with different values of the signature and the message.

We note that instead of the original *message*, it signs:

- a *cryptographic* hash (for example, by the algorithm `MD5`) of the original message, and
- with additional information, such as
    - name of the signer, and
    - the algorithms used to encipher and calculate the hash.

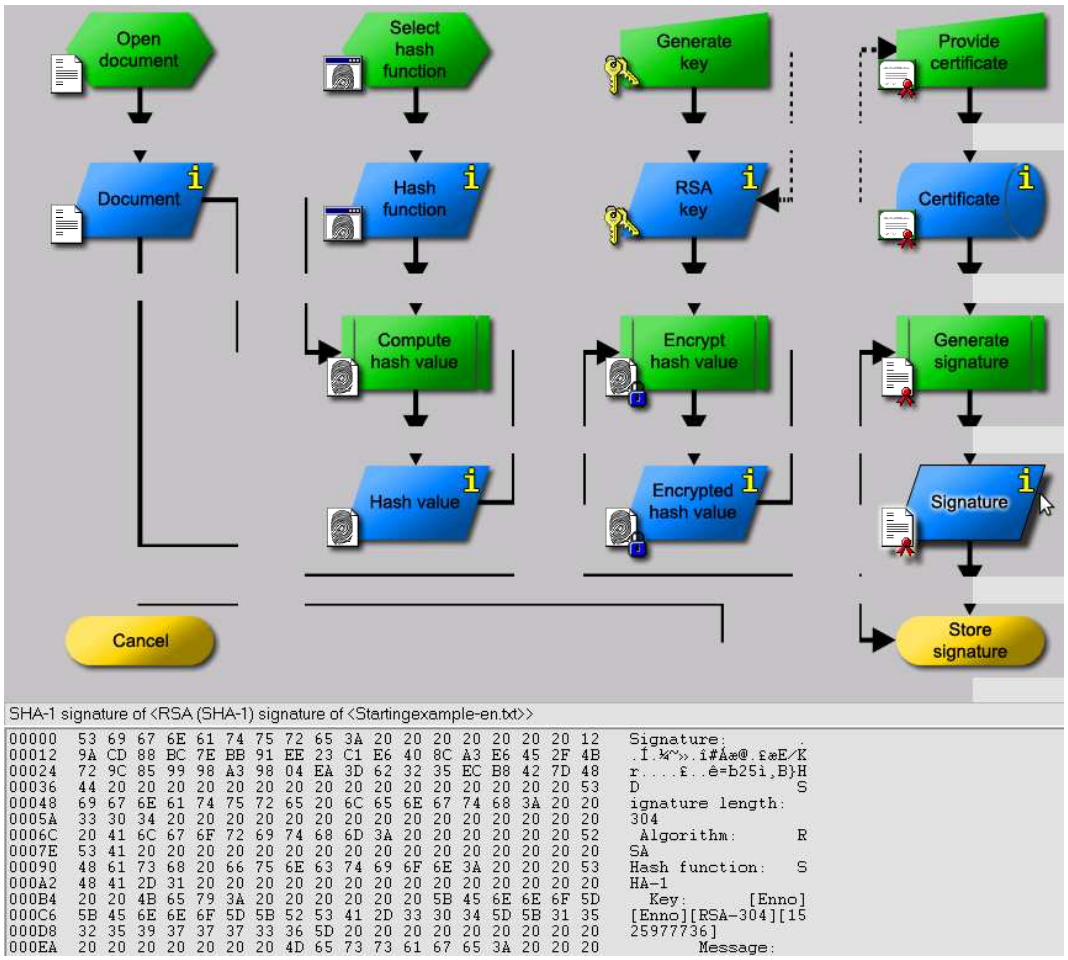


Figure 47: The signature steps by RSA in CrypTool 1 (Esslinger et al. (2008))

DSA (Digital Signature Algorithm). ElGamal (1985) showed first how to build an encryption and signature algorithm on top of the Diffie-Hellman protocol. While its encryption algorithm is rarely employed (albeit, for example, the standard cryptography command-line tool GnuPG offers it as a first alternative to RSA), its signature algorithm forms the basis of the Digital Signature Algorithm (DSA), which is used in the US government's Digital Signature Standard (DSS), issued in 1994 by the National Institute of Standards and Technology (NIST). Elliptic Curve DSA (ECDSA) is a variant of the Digital Signature Algorithm (DSA) which uses points on finite (elliptic) curves instead of integers.

Self-Check Questions.

1. What is the private key used for in digital signature algorithm? *signing by encryption.*
2. What is the public key used for in digital signature algorithm? *verification by decryption.*

Summary

Single-key (or symmetric) cryptography suffers from the *key distribution problem*: to pass the same secret key to all, often distant, correspondents. Two-key (or asymmetric) cryptography solves this problem seemingly at once, by enabling the use of different keys to encrypt and decrypt. However, the identity of the key owner must be confirmed; if not personally, then by *third* parties, that is, identities with private keys that confirm by their digital signatures that it is Alice who owns the private key. However, the problem of the public key identity arises again: How can we ensure the identities of these private key owners? There are two solutions: In the approach via hierarchical authorities, private key owners are distinguished by hierarchical levels. At the highest level lie the *root authorities* on which one trusts unconditionally. In the web of trust, trust is transferred from one to the other, that is, trust is transitive: If Alice trusts Bob, and Bob trusts Charles, then Alice trusts Charles.

Arithmetic.

Asymmetric cryptography relies on a *trapdoor* function, which

- must be easily computable (for example, raising to the n-th power in RSA), but
- its inverse (for example, extraction of the n th root in RSA) must be practically incomputable without knowledge of a shortcut, the key!

This difficulty of calculating the inverse corresponds to the difficulty of decryption, that is, inverting the encryption. To *complicate* the computation of the inverse function (besides facilitating the computation of the proper function) is done using *modular* (or *circular*) arithmetic*, that we already from the arithmetic of the clock, where $m = 12$ is considered equal to 0 .

The Diffie-Hellman Key exchange. The Diffie-Hellman Key exchange protocol shows how to build overtly a mutual secret key based on the difficulty of computing the logarithm modulo p . This key can then be used to encrypt all further communication, for example, by an asymmetric algorithm such as AES.

However, it shows

- neither how to encrypt a message (with a public key and decrypt it with the private key),
- nor how to sign one (with a private key and verify it with the public key).

ElGamal (1985) showed first how to build an encryption and signature algorithm on top of the Diffie-Hellman protocol; in particular, it gave rise to the Digital Signature Algorithm, DSA.

The RSA algorithm. The best-known public-key algorithm is the Rivest-Shamir-Adleman (RSA) cryptoalgorithm. A user secretly chooses a pair of prime numbers p and q so large that factoring the product $N = pq$ is beyond estimated computing powers during the lifetime of the cipher; The number N will be the modulus, that is, our trapdoor function will be defined on $\mathbb{Z}/N\mathbb{Z} = \{0, 1, \ldots, N-1\}$.

N is public, but p and q are not. If the factors p and q of N were known, then the secret key can be easily computed. For RSA to be secure, the factoring must be computationally infeasible; nowadays 2048 bits. The difficulty of factoring roughly doubles for each additional three digits in N .

To sign (instead of encrypt), the only difference is that the exponents E and d exchange their roles. That is, the signed message is $M = m^d$ (instead of m^E):

The trapdoor function of of RSA is raising to a power, $m \mapsto M = M^E$ for a message m , and its computational security relies upon the difficulty of finding a root modulo a large number

$$m \equiv \sqrt[E]{M}\ (= M^{1/E}) \mod N.$$

The *shortcut* is the knowledge of the two prime factors p and q of $N = pq$ that makes it possible to calculate the inverse multiplicative $d = 1/E$ of E , that is, d such that

$$Ed \equiv 1 \mod (p-1)(q-1).$$

Then

$$\sqrt[E]{M} \equiv M^d \mod N;$$

computing a power is a lot faster than a root.

Questions

1. What is the inverse of the trapdoor function used in RSA?

   - ☐ $x \mapsto x^{1/e}$
   - ☐ log
   - ☐ exp
   - ☐ $x \mapsto x^2$

2. What is the fastest known algorithm to attack RSA?

   - ☐ *General Number field sieve*
   - ☐ Pollard's ρ
   - ☐ Smart Attack
   - ☐ Baby-Step-Giant-Step

3. What is the minimum key size of RSA to be currently considered secure, for example, by the NIST?

   - ☐ 1024
   - ☐ 2048
   - ☐ 3072
   - ☐ 4096

Required Reading

Read the section on asymmetric cryptography in the article Simmons et al. (2016). Read in Menezes, Oorschot, and Vanstone (1997),

- Sections 3.1, 3.3 and try to understand as much as possible in 3.2 and 3.6 on the number theoretic problems behind public key cryptography
- Sections 8.1 and 8.2 on the algorithm RSA

Use Grau (2018d) to get an intuition for the graphs over finite domains.

Read Chapter 10 on RSA.

Further Reading

Use `CrypTool 1` to experiment with `RSA`.

See the book Sweigart (2013a) for implementing some simpler (asymmetric) algorithms in `Python`, a readable beginner-friendly programming language.

Read the parts of the book Schneier (2007) on understanding and implementing modern asymmetric cryptographic algorithms.

9 Primes

Study Goals

Introduction

9.1 Detection of Primes

Let us recall Euclid's Theorem which asserts that there are prime numbers **arbitrarily large** (for example, with ≥ 2048 binary digits for the RSA):

Euclides' theorem. There are **infinitely many** prime numbers.

Demonstration: Let's suppose otherwise, that there's only a finite number $p_1, ..., p_n$ of prime numbers. Consider

$$q = p_1 ... p_n + 1.$$

Since q is greater than $p_1, \ldots, p_n$, it is not prime. So let p be a prime number that divides q. Therefore, p must be one of $p_1, \ldots, p_n$. But, by its definition, q leaves rest 1 for any $p_1, \ldots, p_n$.

Contradiction! So there's no greatest prime number. q.e.d.

Examples of Greater Prime Numbers. Marin Mersenne (Oizé, 1588 — Paris, 1648) was a French Franciscan friar who tried to find, without success, a formula to give all prime numbers. Motivated by a letter from Fermat in which he suggested that all the numbers $2^{2^p}+1$, Fermat's *Numbers* be primes, Mersenne studied the numbers of the form

$$2^p - 1 \quad \text{for } p \text{ prime}.$$

In 1644 he published the work *Cogita physico-mathematica* which states that these numbers are primes for

$$p = 2, 3, 5, 7, 13, 17, 19, 31 \text{ and } 127.$$

(and mistakenly included $p = 63$ and $p = 257$). (Only a computer could show in 1932 that $2^{257} - 1$ is composed.)

Mersenne's prime numbers, in the form of $2^p - 1$ to p prime, are known to be

2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209, 44497, 86243, 110503, 132049, 216091, 756839, 859433, 1257787, 1398269, 2976221, 3021377, 6972593,

13466917, 20996011, 24036583, 25964951, 30402457, 32582657, 37156667, 42643801, 43112609, 57885161, 74207281, 77232917 e 82589933

The prime number

$$2^{82\,589\,933} - 1$$

has 24,862,048 digits. It was found on December 8,2018 and is to this day the **biggest known prime number**.

"CrypTool 1", in the menu entry "Indiv. Procedures -> Number Theory Interactive -> Compute Mersenne Numbers' allows you to calculate some of Mersenne's prime numbers.

Tests. A quick test if the natural number n is compound is the **Small Fermat Theorem** (formulated as its contraposition): If there is a natural number a such that

$$a^n \not\equiv a \bmod n$$

then n is compound.

But the reverse implication doesn't hold: There are n numbers (which are called **Carmichael numbers**) that are compound but for every natural number a,

$$a^n \equiv a \bmod n.$$

The lowest such number n is 561 (which is divisible by 3).

The Eratosthenes sieve. The simplest algorithm to verify whether a number is prime or not is the Eratosthenes sieve (285 – 194 B.C.).

To illustrate this, let's determine the prime numbers between 1 and 30.

Initially, we'll determine the largest number by what we'll divide to see if the number is composed; is the square root of the upper coordinate rounded down. In this case, the 30 root, rounded down, is 5.

1. Create a list of all integers from 2 to the value of the quota: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29 e 30.
2. Find the first number on the list. He's a prime number, 2.

Figure 48: Eratosthenes, the third head librarian of the Alexandria Library

3. Remove all multiples from 2 to 30 from the list: 2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27 e 29.

The next number on the list is prime. Repeat the procedure:

- In this case, the next number on the list is 3. By removing your multiples, the list stays: 2, 3, 5, 7, 11, 13, 17, 19, 23, 25 e 29.
- The next number, 5, is also prime 2, 3, 5, 7, 11, 13, 17, 19, 23 e 29.

As initially determined, 5 is the last number we divide by. The final list 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 contains only prime numbers.

Here's an implementation in Python:

```
def primeSieve(sieveSize):
    # Returns a list of prime numbers calculated using
    # the Sieve of Eratosthenes algorithm.
```

```
    sieve = [True] * sieveSize
    sieve[0] = False # zero and one are not prime numbers
    sieve[1] = False
    # create the sieve
    for i in range(2, int(math.sqrt(sieveSize)) + 1):
        pointer = i * 2
        while pointer < sieveSize:
            sieve[pointer] = False
            pointer += i
    # compile the list of primes
    primes = []
    for i in range(sieveSize):
        if sieve[i] == True:
            primes.append(i)
    return primes
```

The Deterministic 'AKS' Test. The **test of AKS** determines in polynomial time whether n is compound or prime (more exactly, in time $O(d)^6$ where d = the number of digits d [binary] of n). In practice, the **Miller-Rabin test** is usually enough to guarantee much more *witnesses* (= a numbers that prove whether n is composed or not) than Fermat's Little Theorem.

In fact, when we compare the duration between the two algorithms to check if a number is prime on a computer with a 2GHz `Intel Pentium-4` processor, we get

prime number	Miller-Rabin	AKS
7309	0.01	12.34
9004097	0.01	23 : 22.55
$2^31 - 1$	0.01	6 : 03 : 14.36

The `CrypTool 1` offers in the `Individual Procedures -> RSA` Menu an entry to experiment with different algorithms to detect prime numbers.

The Probabilist Test `Miller-Rabin`. The simplistic tests, to know if a n number is prime or not, are inefficient because they calculate the n factors.

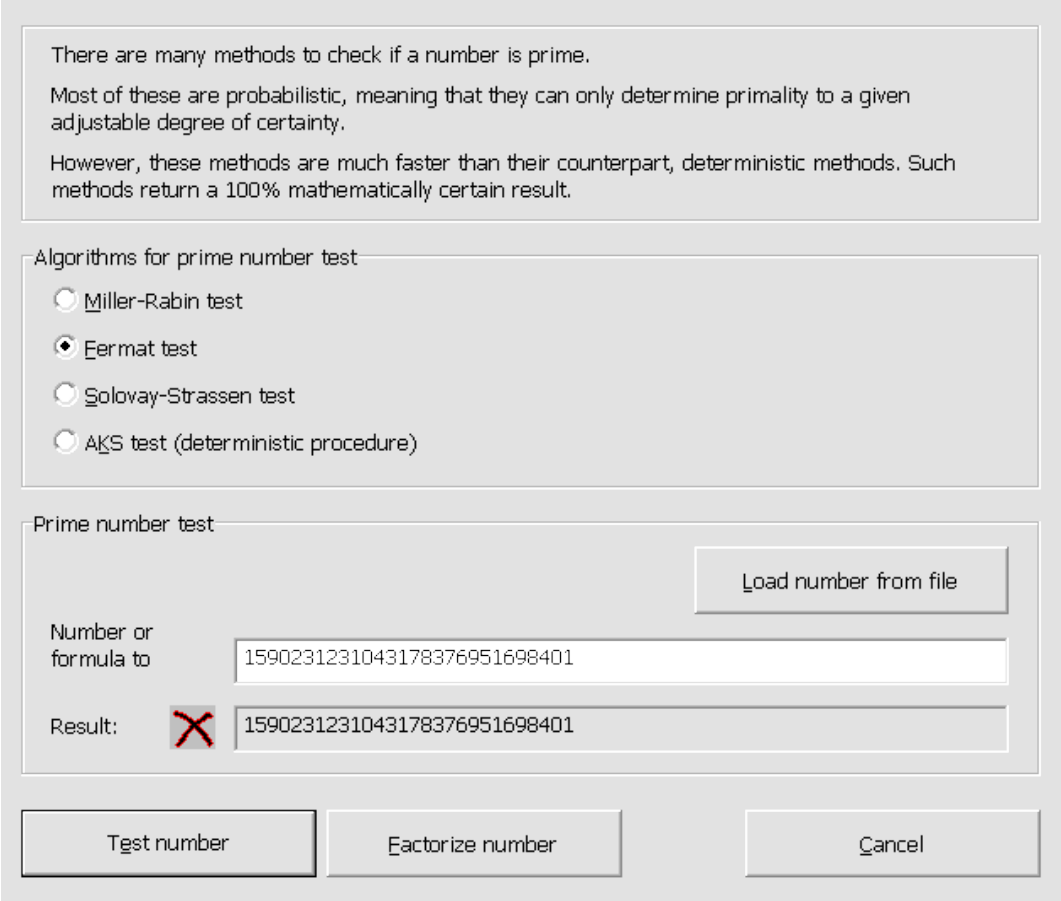


Figure 49: The algorithms to check if a number is prime in CrypTool 1

Instead of them, to know only if it is prime or not, there is the Miller-Rabin Test. After your demonstration, we'll give you your opposition; it's in this formulation that it's applied.

The Miller-Rabin Test. Be $p > 2$ a prime number, be $n - 1 = 2^k q$ for numbers k and q (with q odd). So, for any whole number a indivisible for p is worth

- or $a^q \equiv 1$,
- or there's r in $0, 1, \ldots, k-1$ such that $a^{2^r q} \equiv -1 \bmod n$

Demonstration: By the Little Theorem of Fermat

$$a^{p-1} = (a^d)^{2^k} \equiv 1 \bmod p$$

By iteratively extracting the square root, we obtain

- or $(a^q)^{2^r} \equiv 1 \bmod p$ for all $r = 1, \ldots, k-1$; in particular $a^q \equiv 1 \bmod p$,
- or there's r in $\{1, \ldots, k-1\}$ such that $(a^q)^{2^r} \equiv -1 \bmod p$. *q.e.d.*

If for an odd number, a possibly prime number, we write $n - 1 = 2^k q$, then by Fermat's Test n is not prime if there is a whole a such that $a^{2^k q} \equiv 1 \bmod n$. The Miller-Rabin Test explains the condition $a^{q2^k} = (a^q)^{2^k} = ((a^q)^2) \cdots)^2 \not\equiv 1$:

The Miller-Rabin Test. *(Contraposition) It's n odd and $n - 1 = 2^k q$ for numbers k and q (with q odd). An integer a relatively prime to n is a **Miller-Rabin Witness** (for divisibility) of n*, if

- $a^q \equiv 1 \bmod n$ e
- $a^{2q} a^{2^2 q}, \ldots, a^{2^{k-1} q}$ such that $a^{2^r q} \equiv -1 \bmod n$

Question: What are the chances that we declare by the Miller-Rabin Test accidentally a prime number, that is, a number that is actually compound?

Theorem. *(About the frequency of witnesses)* Be n odd and compound. So at least 75 of the numbers in $1, \ldots, n-1$ are Miller-Rabin Witnesses for n.

So, already after 5 attempts a_1, a_2, $\ldots$, a_5 without witness we know with a chance $1/4^5 = 1/1024 < 0.1\%$, that the number is prime!

Python Implementation. Let's implement

1. the Miller-Rabin algorithm,
2. a test for a prime number, and
3. a function to generate (large) prime numbers.

```
# Primality Testing with the Rabin-Miller Algorithm
# http://inventwithpython.com/hacking (BSD Licensed)

random import


def rabinMiller(num):
    # Returns True if num is a prime number.

    s = num - 1
    t = 0
```

```
    while s % 2 == 0:
        # keep halving s while it is even (and use t
        # to count how many times we halve s)
        s = s // 2
        t += 1

    for trials in range(5): # try to falsify num's primality 5 teams
        a = random.randrange(2, num - 1)
        v = pow(a, s, num)
        if v != 1: # this test does not apply if v is 1.
            i = 0
            while v != (num - 1):
                if i == t - 1:
                    return False
                else:
                    i = i + 1
                    v = (v ** 2) % in a
    return True

def isPrime(num):
    # Return True if num is a prime number. This function does a
    # quicker prime number check before calling rabinMiller().

    if (num < 2):
        return False # 0, 1, and negative numbers are not prime

    # About 1/3 of the time we can quickly determine if num is not
    # prime by dividing by the first few dozen prime numbers.
    # This is quicker # than rabinMiller(), but unlike rabinMiller()
    # is not guaranteed to prove that a number is prime.
    lowPrimes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    47, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107,
    109, 113, 127, 131, 137, 149, 151, 157, 163, 167, 173, 179, 181,
    191, 193, 197, 199, 211, 223, 227, 233, 239, 241, 251, 257, 263,
    269, 271, 277, 281, 283, 293, 307, 311, 313, 331, 337, 347, 349,
    353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 431, 433,
    439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509,
    523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607,
```

```
    613, 617, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691,
    701, 709, 719, 727, 739, 743, 751, 757, 761, 769, 773, 787, 797,
    809, 811, 821, 823, 827, 829, 853, 857, 859, 863, 877, 881, 883,
    887, 907, 911, 919, 929, 937, 941, 947, 967, 971, 977, 983, 991,
    997]

    if in a lowPrimes:
        return True

    # See if any of the low prime numbers can divide into one
    for prime in lowPrimes:
        if (num % prime == 0):
            return False

    # If all else fails, call rabinMiller() to check if num is a prime.
    return rabinMiller(num)

def generateLargePrime(keysize = 1024):
    # Return a random prime number of keysize bits in size.
    while True:
        num = random.randrange(2**(keysize-1), 2**(keysize))
        if isPrime(num):
            return num
```

9.2 Other Moduli

Let's observe for modules that are not primes, that is, a product of primes factors, that the difficulty increases *linearly* in the number of factors, unlike it increases *exponentially* in the number of bits of each factor:

Product of Different Primes. If the $m = pq$ module is the product of two factors p and q without common factor, then the modular logarithm

$$\log_g \bmod m$$

can be computed, by the Chinese Theorem of the Remains, by the logarithms

$$\log_g \bmod p \quad \text{and} \quad \log_g \bmod q$$

More exactly, there are integers a and b, computed (in linear time in the number of p and q bits) by the Euclid Algorithm (extended), such that $ap + bq = 1$ and

$$\log_g \bmod m = a(\log_g \bmod p) + b(\log_g \bmod q).$$

Power of a Prime. If the modulus $m = p^e$ is a power of a prime p, then Bach (1984) shows how the modular logarithm module m for a g base

$$\log_g : \mathbb{Z}/m\mathbb{Z}^* \to \mathbb{Z}/\phi(m)\mathbb{Z}$$

can be computed in polynomial time from the p module logarithm. Let's expose the steps to a prime number $p > 2$:

1. Let us recall the Section 7.2 about the existence of the primitive root for every module for which $\mathbb{Z}/p^e\mathbb{Z}^*$ is cyclical in order $(p-1)p^{e-1}$. Therefore, there is a multiplicative application

   $$\mathbb{Z}/p^e\mathbb{Z}^* \to \mu_{p-1} \times U_1$$

   given by

   $$x \mapsto x^{p^{e-1}}, x/x^{p^{e-1}} \quad (*)$$

   where

   $$\mu_{p-1} = \{\zeta \in \mathbb{Z}/p^e\mathbb{Z}^* : \zeta^{p-1} = 1\}$$

   denote the group of $(p-1)$-highest roots of the unit and

   $$U_1 = 1 + p\mathbb{Z}/p^e\mathbb{Z}$$

   The unitary units.
2. We have the isomorphism

   $$\mu_{p-1} \to \mathbb{F}_p^*$$

   given for $x \mapsto x \bmod p$ and its inverse for $X \mapsto X^{p^{e-1}}$ for every X in $\mathbb{Z}/p^e\mathbb{Z}$ such that $X \equiv x \bmod p$. (Note that the restriction of homomorphism

   $$\mathbb{Z}/p^e\mathbb{Z}^* \to U_1$$

   given by $x^{1-p^{e-1}}$ to U_1 is the identity because the order of U_1 is p^{e-1}).

3. We have the logarithm to the g base

$$\log_g \colon \mathbb{F}_p^* \to \mathbb{Z}/(p-1)\mathbb{Z}$$

and we have the natural logarithm

$$\log \colon \mathrm{U}_1 \to p(\mathbb{Z}/p^e\mathbb{Z})$$

which is calculated in polynomial time by the formula

$$u \mapsto [x^{p^e} - 1]/p^e; \tag{3}$$

and which provides the logarithm $\log_g \colon \mathrm{U}_1 \to p(\mathbb{Z}/p^e\mathbb{Z})$ for the base g by the scaling

$$\log_g = \log \cdot / \log g.$$

4. By the Chinese Remainder Theorem, we have the isomorphism

$$\mathbb{Z}/(p-1)\mathbb{Z} \times \mathbb{Z}/p^{e-1}\mathbb{Z} \to \mathbb{Z}/(p-1)p^{e-1}\mathbb{Z}$$

given by the product and its inverse given by $y \mapsto (ay \bmod p, by \bmod p^{e-1})$ where a and b satisfy $a(p-1) + b(p^{e-1}) = 1$ and were obtained by the Euclid Algorithm (extended).

We conclude that, given

- the number y in $\mathbb{Z}/p^e\mathbb{Z}$ and
- its value $\log_g(y)$ under $\log_g \colon \mathbb{F}_p^* \to \mathbb{Z}/(p-1)\mathbb{Z}$,

the value of $\log_g(y)$ of $\log_g \colon (\mathbb{Z}/p^e\mathbb{Z})^* \to \mathbb{Z}/(p-1)p^{e-1}\mathbb{Z}$ is computed in polynomial time.

Observation. To facilitate computing, instead of projection

$$\mathbb{Z}/p^e\mathbb{Z}^* \to \mathrm{U}_1$$

given in $(*)$ for $x \mapsto x^{1-p^{e-1}}$, it's faster to use that given in $(*)$ by $\pi \colon x \mapsto x^{p-1}$. However, its U_1 restriction is not identity. So you need to use it instead of

$$\log g \colon \mathrm{U}_1 \to p\mathbb{Z}/p^e\mathbb{Z}$$

the scaled logarithm

$$(p-1)^{-1} \log_g$$

in order to obtain

$$\log_g = (p-1)^{-1} \log_g \circ \pi = (\log(g)p - 1)^{-1} \log \circ \pi \colon \mathrm{U}_1 \to p\mathbb{Z}/p^e\mathbb{Z}.$$

Discreet Logarithm To Prime Power. Let's explain Equation 3 which defines the logarithm $\log\colon U_1 \to p(\mathbb{Z}/p^e\mathbb{Z})$: let us remember the definition of the exponential on $\mathbb{R}$ for compound interest

$$\exp(x) = \lim\left(1 + \frac{1}{n}\right)^n,$$

which leads to the definition of the inverse function

$$\log(x) = \lim n(x^{1/n} - 1) = x^\epsilon - 1$$

where $\epsilon = 1/n \to 0$.

Now, at $\mathbb{Z}/p^e\mathbb{Z}$, we have $1, p, p, 2, \ldots, p = 0$, that is, p^n, which may motivate the idea of considering p as small. So, the good analog about U_1 is

$$\log(x) = \lim \frac{1}{p^{e-1}}(x^{p^{e-1}} - 1).$$

In fact, over U_1,

$$\log(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \ldots$$

is a well defined value in $p\mathbb{Z}/p^e\mathbb{Z}$, because if p divides x, then no denominator of the fraction cut is divisible by p and all indivisible numbers by p are invertible by $\mathbb{Z}/p^e\mathbb{Z}$. Likewise, over $p\mathbb{Z}/p^e\mathbb{Z}$,

$$\exp(x) = \sum_{n\geq 0} \frac{x^n}{n!}$$

is a well defined value at $1 + p\mathbb{Z}/p^e\mathbb{Z}$, because if p divides x, then no cut denominator is divisible by p and all indivisible numbers by p are invertible by $\mathbb{Z}/p^e\mathbb{Z}$.

Of particular interest is the base e^p of the natural logarithm at $1 + p\mathbb{Z}/p^e\mathbb{Z}$, that is, the argument y such that $\log y = 1$. For example, for $p = 7$ and $e = 4$, we calculate

$$\exp(p) = \sum_{n\geq 0} \frac{p^n}{n!} = 1 + p + \frac{p^2}{2} + \frac{p^3}{3!} = 1 + 7 \cdot 127 = 1 + 1 \cdot 7 + 4 \cdot 7^2 + 2 \cdot 7^3.$$

Self-Check Questions

Summary

Asymmetric cryptography relies on a *trapdoor* function, which

- must be easily computable (for example, raising to the n-th power in RSA), but
- its inverse (for example, extraction of the n th root in RSA) must be practically incomputable without knowledge of a shortcut, the key!

This difficulty of calculating the inverse corresponds to the difficulty of decryption, that is, inverting the encryption. To *complicate* the computation of the inverse function (besides facilitating the computation of the proper function) is done using *modular* (or *circular*) arithmetic*, that we already from the arithmetic of the clock, where $m = 12$ is considered equal to 0 .

Questions

Required Reading

Further Reading

Use CrypTool 1 to experiment with various prime detection algorithms.

10 Finite Elliptic Curves

Study Goals

On completion of this chapter, you will have learned ...

- the Diffie-Hellman key exchange using finite elliptic curves.

Introduction

Denote $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$. Among all curves, the clou of the *elliptic* curves (given by an equation $y^2 = x^3 + ax + b$) is that one can *add* points on them: $p + q + r = 0$ if a line passes for p, q and r . By restricting the solutions to points (x, y) in ($\mathbb{F}_p \times \mathbb{F}_p$ for a large prime number p and fixing a point P on the curve,

- while it is *easy* to compute the *exponential*, that is, for n , compute

$$\mathrm{Q} = n\mathrm{P} = \mathrm{P} + \ldots + \mathrm{P},$$

- in contrast, for a point $\mathrm{Q} = \mathrm{P} + \ldots + \mathrm{P}$, it is *difficult* to compute the *logarithm*: that is, how many times P has been added, the number n such that $\mathrm{Q} = n\mathrm{P}$.

  Diffie-Hellman over Elliptic Curves: (an analog of) the Diffie-Hellman protocol, in which iterated multiplication of a number modulo p is replaced by iterated addition of a point on a finite elliptic curve.

The `Diffie-Hellman` protocol (over $\mathbb{F}_p$) has an analog over Elliptic Curves:

- Instead of *multiplying* repeatedly (n times) the base g in $\mathbb{F}_p^*$, that is, computing

$$g^n = g \cdots g,$$

- *add* repeatedly (n times) a point G , that is, compute

$$n \cdot \mathrm{G} = \mathrm{G} + \cdots + \mathrm{G}.$$

The advantage of using

- the *logarithm* over a *finite elliptic curve* (that is, the function that for a given point G and Y determines the scalar x in $\mathbb{N}$ such that $\mathrm{Y} = x\mathrm{G}$)
- instead of the logarithm over $\mathbb{F}_p$ (that is, the function that given numbers g and y determines the exponent x such that $y \equiv g^x \bmod p$),

is that depending on the number of bits n of p (regarding the fastest *presently* known algorithms):

- the time to compute the logarithm over an elliptic curve increases *linearly* and takes about $n/2$ operations, while

- the time to compute the multiplicative logarithm increases *sublinearly* and takes about $n^{1/3}$ operations.

For example, the security obtained by a `2048` bits key for the multiplicative logarithm equals approximately that of a `224` bits key for the logarithm over an elliptic curve. To a length of `512` bits of a key for an elliptic curve, corresponds a length of `15360` bits of an `RSA` key.

In the next sections, we:

1. introduce these general finite fields (because common finite elliptic curves are defined over more general finite fields than those of the form $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ for a prime number p that we know so far).
2. introduce elliptic curves.
3. study the addition of points of an elliptic curve.
4. present the Diffie-Hellman Key Exchange over elliptic curves, and
5. look at the cryptographic problem behind the curves and the algorithms that solve it.

10.1 General Finite Fields

We realized that we already use the modular arithmetic in everyday life, for example for the modulus $m = 12$, the arithmetic of the clock, and for $m = 7$, the days of the week. More generally, we defined, for any integer m the *finite ring* $\mathbb{Z}/m\mathbb{Z}$ (= a finite set where we can add and multiply), roughly,

- as a set for $0, 1, \ldots, m-1$, and
- the sum (respectively product) of x and y in

$$\mathbb{Z}/m\mathbb{Z} = \{0, 1, ..., m-1\}$$

  is defined by the remainder of the sum $x + y$ (respectively product) in $\mathbb{Z}$ divided by m.

If $m = p$ is prime, then it can be shown that $\mathbb{Z}/p\mathbb{Z}$ is a *field* denoted by $\mathbb{F}_p$: for every a in $\mathbb{F}_p$, except 0, there is always a^{-1} in $\mathbb{F}_p$, the inverse *multiplicative* of a defined by satisfying $aa^{-1} = 1$. In other words, in a field we can divide by every number except 0. (The most common examples are the infinite fields $\mathbb{Q}$ and $\mathbb{R}$.)

$\mathbb{F}_q$ for $q = p^n$: A ring of polynomials with coefficients in $\mathbb{F}_p$ of degree n

In cryptography, elliptic curves are defined over fields $\mathbb{F}_q$ whose cardinality is a power $q = p^n$ of a prime number p (and not just $\mathbb{F}_p$ till now); for example, $q = 2^n$ for a large number n . The case $p = 2$ is particularly suitable for computing (cryptographic). The fields of the form $\mathbb{F}_{2^n}$ are called *binary*.

In Section 2.4, we already made acquaintance with the Rijndael field, which was defined by polynomials of degree 7 with binary coefficient. More generally, the field $\mathbb{F}_q$ to $q = p^n$ is defined by polynomials of degree n over $\mathbb{F}_p$,

$$\mathbb{F}_q[\mathrm{X}] = \{a_n\mathrm{X}^n + a_{n-1}\mathrm{X}^{n-1} + \cdots + a_0 \text{ with } a_n, a_{n-1}, \ldots, a_0 \text{ in } \mathbb{F}_q\}.$$

- The + addition of two polynomials is the addition of polynomials, that is, the coefficient to coefficient addition in $\mathbb{F}_p$, and
- to multiply two polynomials,
  1. multiply the polynomials, and
  2. take the remainder of division by a polynomial $m(\mathrm{X})$ to be defined.

The Rijndael Field $\mathbb{F}_{2^8}$. For example, for $q = p^n$ with $p = 2$ and $n = 8$, we get the field $\mathbb{F}_{2^8}$ used in AES . As a set

$$\mathbb{F}_q = \{a_7\mathrm{X}^7 + \cdots + a_0 \text{ with } a_7, \ldots, a_0 \text{ in } \mathbb{F}_p\}.$$

that is, the finite sums

$$a_0 + a_1\mathrm{X} + a_2\mathrm{X}^2 + \cdots + a_7\mathrm{X}^7$$

for a_0, a_1, …, a_7 in $\{0,1\}$.

- The addition + of two polynomials is the addition of polynomials, that is, the coefficient to coefficient addition in $\mathbb{F}_p$, and
- Multiplication is first the usual multiplication of polynomials and then the remainder of division by the polynomial

$$m(\mathrm{X}) = \mathrm{X}^8 + \mathrm{X}^4 + \mathrm{X}^3 + \mathrm{X} + 1.$$

10.2 Elliptic Curves

An *elliptic* curve E over a finite field (in which $0 \neq 2, 3$) is an equation

$$y^2 = x^3 + ax + b$$

for coefficients a and b such that the curve is not *singular*, that is, its *discriminant* is nonzero, $4a^3 + 27b^2 \neq 0$.

Note.

- The equation $y^2 = x^3 + ax + b$ is the *form of Weierstraß*, but there are several others that have proved to be computationally more efficient, such as that of *Montgomery*

$$\mathrm{B}y^2 = x^3 + \mathrm{A}x^2 + x \qquad \text{where } \mathrm{B}(\mathrm{A}^2 - 4) \neq 0.$$

- If the characteristic is 2 , that is, $\mathbb{F}_q$ with $q = 2^n$, then the equation is $y^2 + cxy + dy = x^3 + ax + b$.

After choosing a domain (for example, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ or $\mathbb{F}_p$ for a prime number p) the points (x, y) that solve this equation, $\mathrm{E}(x, y) = 0$, form a curve in the plane. This plane,

- for $\mathbb{R}$ is the usual Cartesian plane,
- for $\mathbb{Z}$ is a lattice of points, and
- for $\mathbb{Z}/m\mathbb{Z}$ is the finite lattice of points inside the square of length m whose bottom left corner is at the origin.

In addition to the points in the plane, there is also the point at infinity (or *ideal* point) that is denoted 0 . Thus, the points of the elliptic curve are given by

$$\mathrm{E} \coloneqq \{ \text{ all points } (x, y) \text{ such that } \mathrm{E}(x, y) = 0\} \cup \{0\}$$

where the notion of point depends on the domain: On a finite field $\mathbb{F}_q$, the number of points #E is limited by $q + 1 - t$ where $t \leq 2\sqrt{q}$, that is, asymptotically equal to $\#\mathbb{F}_q^* = q - 1$. It can be computed by Schoof's algorithm Schoof (1995) in about n^5 operations for $n = \log_2 q$ the number of binary digits of q .

Just as the coefficients a and b , this plane can be a rational, real, complex or finite plan, that is, a finite field $\mathbb{F}_q$ for a power $q = p^n$ of a prime number p .

Continuous and Discrete Finite Curves. For the domain $\mathbb{R}$, the curves take the following forms in the real plane for different a and b parameters:

Figure 50: Real elliptic curves (Corbellini (2015a))

While on finite fields, we obtain a discrete set of points (symmetrical around the middle horizontal axis).

Curves used in Cryptography. For the cryptographic algorithm on this curve to be *safe*, that is, the computation of the logarithm on it takes time, there are restrictions on the choices of $q = p^n$ and the elliptic curve (that is, on its defining coefficients a and b). For example,

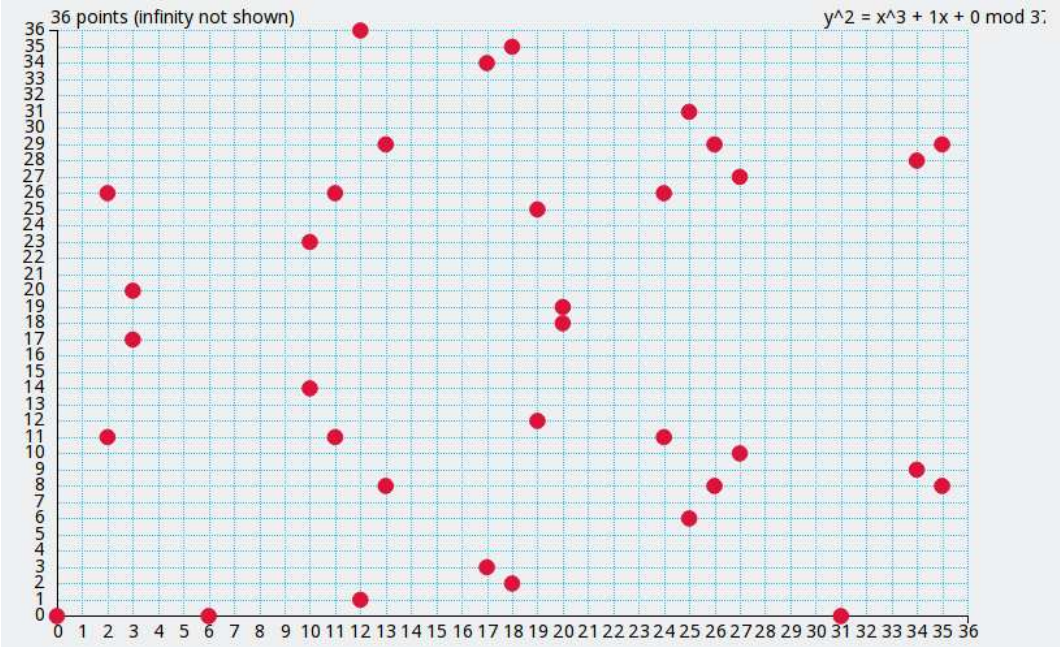


Figure 51: Finite elliptic curve (Grau (2018a))

- that the coefficients a and b are such that
  - the number of $\{\text{pointsover}\mathbb{F}_q\} = q$ (due to vulnerability to Smart's attack), and
  - the curve not be *supersingular* (due to vulnerability to Menezes, Okamoto and Vanstone's attack); that is, that $\#\{\text{pointsover}\mathbb{F}_q\} = q+1$ for $p > 3$ (while for $p = 2, 3$ there are exactly three respectively four equations that define supersingular curves).

The probability that a randomly generated curve (that is, one whose coefficients a and b in the equation $by^2 = x^3 + ax^2 + x$ are randomly generated) is vulnerable to one of these attacks is negligible.

- that the point G has a high *order*.

Ideally, these choices are publicly justified.

A safe choice is for example: The `Curve25519` given by

$$y^2 = x^3 + 486662x^2 + x$$

over $\mathbb{F}_q$ with $q = p^2$ where $p = 2^{255} - 19$ (which explains its name); its number of points is $\#E = 2^{252} + 27742317777358519379088364849 3$. (This curve became popular as an unbiased alternative to the recommended, and soon distrusted, curves by the National Institute for Standards and Technology, NIST).

- Edwards' curves (from 2007) given by

$$x^2 + xy = 1 + dx^2y^2$$

  for an integer $d \neq 0, 1$.
- Koblitz and binary curves over binary fields given by

$$x^2 + xy = x^3 + ax^2 + 1 \quad \text{or} \quad x^2 + xy = x^3 + x^2 + b$$

  for a and b in $\{0, 1\}$, which allow for a particularly efficient addition (and multiplication). Standardized examples are, nistk163, nistk283, nistk571 and nistb163, nistb283, nistb571 defined over the binary field with 163, 283 and 571 bits.
- The finite elliptic curve Brainpool P256r as specified in the RFC7027 used to encrypt the data on the German microchip ID card.

To ensure that the coefficients are not chosen to intentionally compromise cryptographic security, they are often obtained at random, that is,

1. obtained by a randomly generated number (a *seed*), and
2. transformed by a cryptographic hash such as SHA-256.

10.3 Addition (and Multiplication by an integer)

How is the *sum* of two points on an elliptic curve defined? Geometrically the sum of three points p , q and r on an elliptic curve in the Euclidean plane (that is, in $\mathbb{R} \times \mathbb{R}$) is defined by the equality $p + q + r = 0$ if a line passes p, q and r . However, over finite fields, this geometric intuition no longer applies, and we need an algebraic definition (which is also the form that the computer expects).

Geometric Addition (over $\mathbb{R}$). If we look at the *real* points of the curve E, that is, at all the points (x, y) in $\mathbb{R} \times \mathbb{R}$ such that $E(x, y) = 0$, the addition has a *geometric meaning:* We have $P + Q + R = 0$ if P, Q and R are on the same line. More exactly:

- If P and Q are different, then the line connecting P and Q intersects the curve at another point $-R$;
- If P and Q are the same, then we use the tangent at the point $P = Q$.

The reflection of $-R$ along the x-axis is the point $R = P + Q$.

`CrypTool 1` demonstrates this geometric addition in an animation accessible from the `Point Addition on Elliptic Curves` entry in the menu `Procedures -> Number Theory - Interactive`.

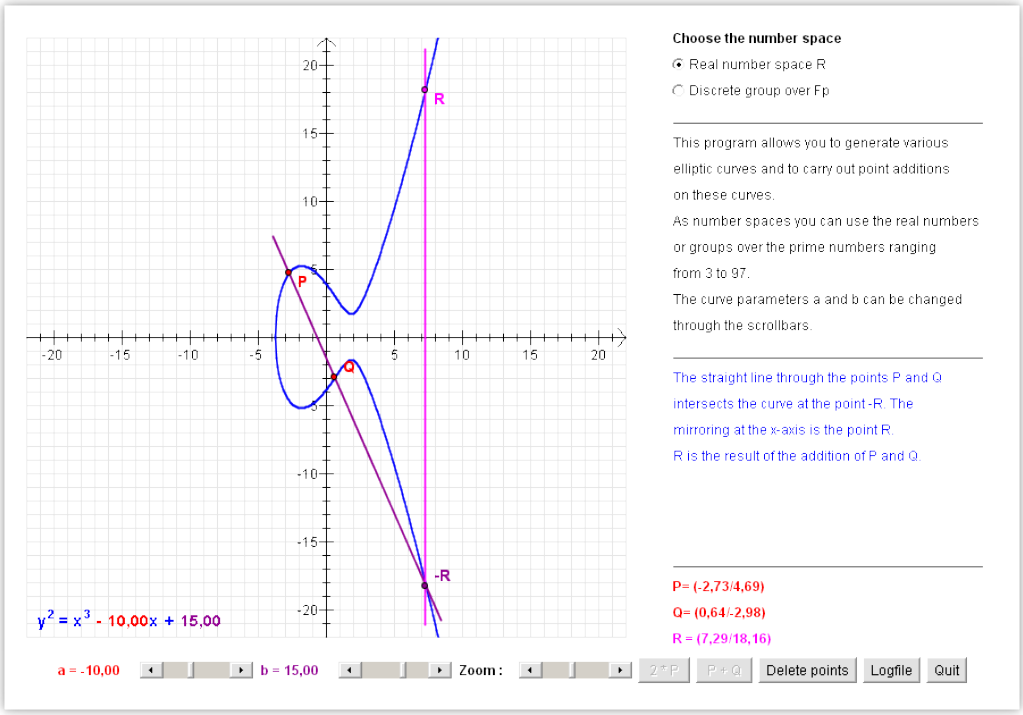


Figure 52: Two-point addition on real numbers in `CrypTool 1` (Esslinger et al. (2008))

Algebraic Addition (over a finite field $\mathbb{F}_p$). This geometric description of the addition leads us to the following algebraic description: Expressed by Cartesian coordinates, the addition of two points of an elliptic curve is given by an *algebraic* formula, that is, it involves only the basic operations of addition, multiplication (and raising to a power). (Thus, we can replace the unknowns by values in any domain, be it $\mathbb{Q}$, $\mathbb{R}$, or $\mathbb{F}_q$.)

Proposition: Denote P + Q = R by

$$(x_P, y_P) + (x_Q, y_Q) = (x_R, y_R).$$

If the curve E is given by $Y^2 = X^3 + aX + b$ and the points P , Q and R are non-zero, then

$$x_R = s^2 - x_P - x_Q \quad \text{and} \quad y_R = s(x_P - x_R) - y_P \quad (*)$$

where s is the degree of inclination of the line that passes through P and Q given by

$$s = \frac{y_Q - y_P}{x_Q - x_P} \quad \text{if } x_Q \neq x_P, \quad \text{and} \quad s = \frac{3x_P^2 + a}{2y_P} \quad \text{if } x_Q = x_P.$$

Demonstration: For a cubic curve not in the normal Weierstrass shape, we can still define a group structure by designating one of its nine inflection points as the O identity. On the projective plane, each line will cross a cubic at three points when considering multiplicity. For a P point, −P is defined as the third exclusive point in the line passing O and P . So for all P and Q , P + Q is defined as −R where R is the third exclusive point in the line containing P and Q .

Be K a field on which the curve is defined (that is, the coefficients of the equation or defining equations of the curve are in K) and denotes the curve as E . So, the rational K points are the E points whose coordinates are in K , including the point at infinity. The − -rational points set is indicated by E (K). It also forms a group, because the properties of the polynomial equations show that if P is in E (K), then −P is also in E (K), and if two of P , Q and R are in E (K), then it is the third. Also, if K is a subfield of L , then E (K) is a subgroup of E (L). Given the curve $Y^2 = X^3 + aX + b$ on the field K (such that $0 \neq 2, 3$), and the points P = (x_P, y_P) and Q = (x_Q, y_Q) on the curve.

1. If $x_P \neq x_Q$, then $y = sx + d$ the Cartesian equation of the line intersecting P and Q with the slope
$$s = \frac{y_P - y_Q}{x_P - x_Q}.$$
For the values x_P, x_Q and x_R the equation of the line is equal to the curve
$$(sx + d)^2 = x^3 + ax + b,$$
or, equivalently,
$$0 = x^3 - s^2x^2 - 2xd + ax + b - d^2.$$
The roots of this equation are exactly x_P, x_Q and x_R; thence
$$(x-x_P)(x-x_Q)(x-x_R) = x^3+x^2(-x_P-x_Q-x_R)+x(x_Px_Q+x_R+x_Qx_R)-x_Px_Qx_R.$$
Therefore, the coefficient of x^2 gives the value x_R; the value of y_R follows by replacing x_R in the Cartesian equation of the line. We conclude that the coordinates (x_R, y_R) = R = -(P + Q) are
$$x_R = s^2 - x_P - x_Q \quad \text{and} \quad y_R = y_P + s(x_R - x_P).$$

2. If $x_P = x_Q$, then
    1. either $y_P = -y_Q$, including the case where $y_P = y_Q = 0$, then the sum is set to 0 ; so the inverse of each point on the curve is found reflecting it in the x axis,
    2. or $y_P = y_Q$, then Q = P and R = (x_R, y_R) = -(P + P) = $-2P = -2Q$ is given by
$$x_R = s^2 - 2x_P \quad \text{and} \quad y_R = y_P + s(x_R - x_P) \quad \text{with} \quad s = \frac{3{x_P}^2 + a}{2y_P}.$$

Observation. For certain specific curves, these formulas can be simplified: For example, in an Edwards' Curve of the shape
$$x^2 + xy = 1 + dx^2y^2$$
for $d \neq 0,1$ (with neutral element 0 the point $(0,1)$), the sum of the points $p = (x_P, y_P)$ and $q = (x_Q, y_Q)$ is given by the formula
$$(x_P, y_P) + (x_Q, y_Q) = \left(\frac{x_Py_Q + x_Qy_P}{1 + dx_Px_Qy_Py_Q}, \frac{y_Py_Q - x_Px_Q}{1 - dx_Px_Qy_Py_Q}\right)$$

(and the inverse of a point (x,y) is $(-x,y)$). If d is not a square in $\mathbb{F}_q$, then there are no *exceptional* points: The denominators $1 + dx_P x_Q y_P y_Q$ and $1 - dx_P x_Q y_P y_Q$ are always different from zero.

If, instead of $\mathbb{R}$, we look at the points with entries in a finite field $\mathbb{F}_q$ from the curve E , that is, all points (x,y) in $\mathbb{F}_q$ such that $E(x,y) = 0$, the addition is uniquely defined by the formula $(*)$.

`CrypTool 1` demonstrates this addition in the entry `Point Addition on Elliptic Curves` of the menu `Indiv. Procedures -> Number Theory - Interactive`.

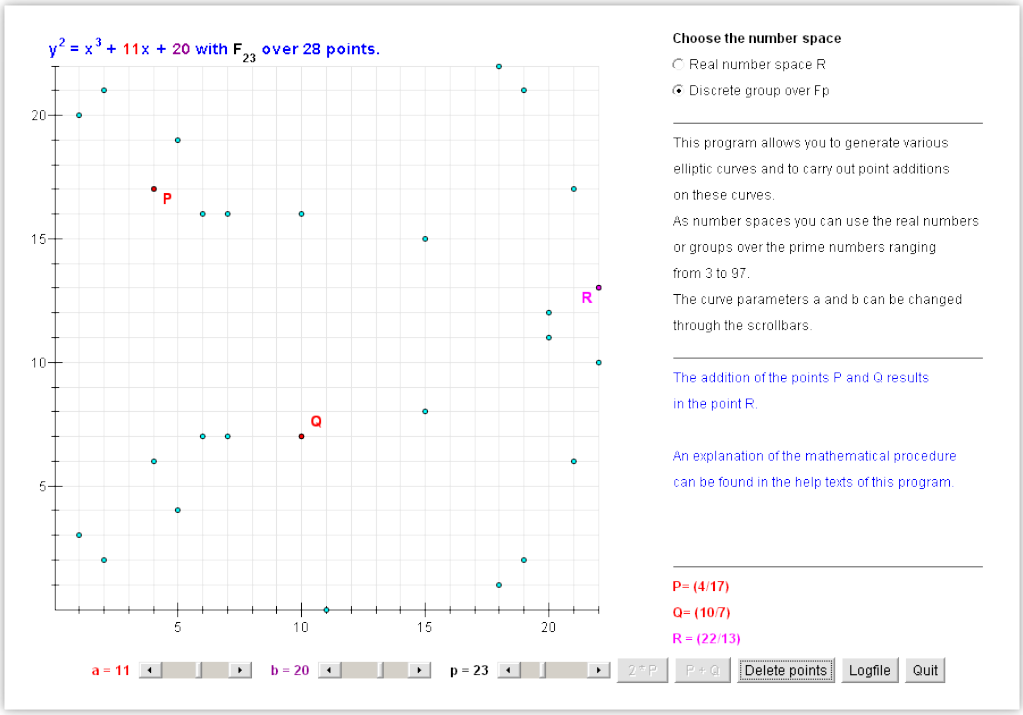


Figure 53: Two-point addition on a finite field in `CrypTool 1` (Esslinger et al. (2008))

Scalar Multiplication. The addition leads to *scaling* by iterated addition. That is,

- to the *exponential*, that, for a fixed point P on the elliptic curve, given x in $\mathbb{N}$, returns
$$x \cdot \mathrm{P} := \mathrm{P} + \cdots + \mathrm{P}$$
(iterated d times) for a natural number d ; and
- to the *logarithm*: given Y and P points on the elliptic curve, what is the x in $\mathbb{N}$ such that $\mathrm{Y} = x \cdot \mathrm{P}$?

Base Point. As the group of points on a finite field $\mathbb{F}_q$ is finite (of cardinality approximately q), necessarily for any point P the set $\langle \mathrm{P} \rangle := \{\mathrm{P}, 2\mathrm{P}, \ldots\}$ is finite. That is, there is n and $n+m$ in $\{0, 1, \ldots, q-1\}$ such that $n\mathrm{P} = (n+m)\mathrm{P}$, that is, there is a whole $m < q$ such that $m\mathrm{P} = 0$.

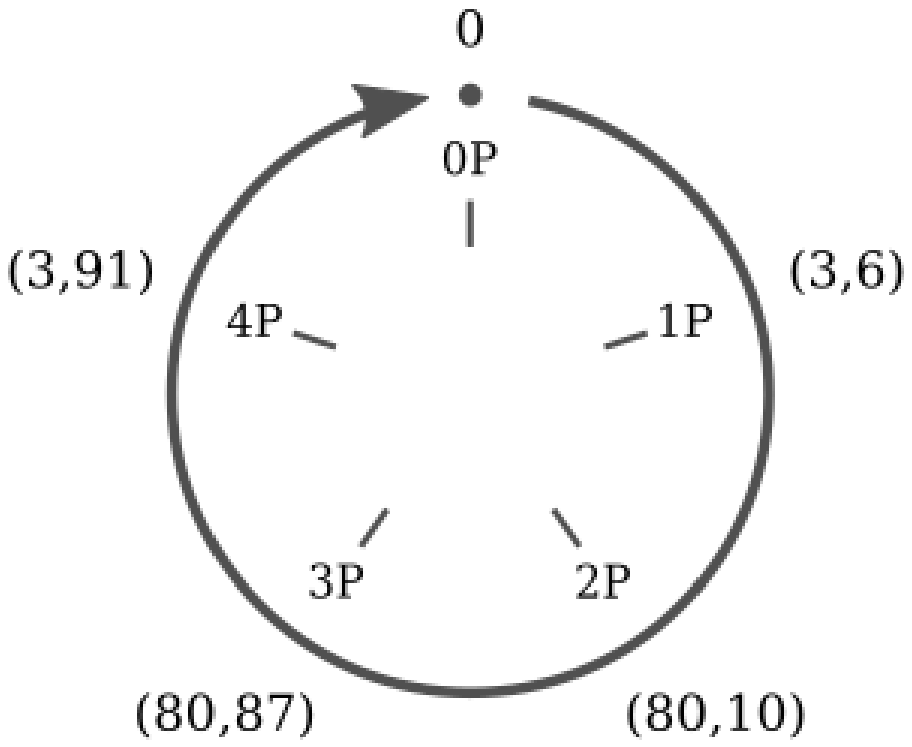


Figure 54: Cyclicity of a point on a finite elliptic curve (Corbellini (2015a))

Grau (2018a) shows for an elliptic curve over a finite field $\mathbb{F}_p$ the addition table between the points, and for each point P the finite cyclic group $\langle \mathrm{P} \rangle = \{\mathrm{P}, 2\mathrm{P}, \ldots\}$ generated by it. The cardinality $m = \#\langle \mathrm{P} \rangle$ is the smallest m such that $m\mathrm{P} = 0$ and is called the *order* of the point P .

10.4 Key Exchange using Elliptic Curves

Elliptic Curve Cryptography uses Diffie-Hellman Key Exchange to

1. build a secret key,
2. turn it into a cryptographic hash, and
3. use it to encrypt communication by a symmetric cryptographic algorithm.

Encryption by the ECC is standardized by the ECIES (Elliptic Curve Integrated Encryption Scheme), a *hybrid* procedure (asymmetric cryptography with symmetric cryptography).

Once the mutually secret c key (= a point on the finite elliptic curve) is agreed on, Alice and Bob derive from it a key to a symmetric cipher like AES or 3DES . The derivation function that transforms a secret information into the appropriate format is called a Key Derivation Function, KDF. Such a standardized function is ANSI-X9.63-KDF with the SHA-1 option. For example, the TLS protocol

- uses the x coordinate of the point c ,
- concatenates to it numbers relating to the connection (such additional specific data is called a *salt*), and
- calculates a cryptographic hash of this concatenated number.

Let us transfer the Diffie-Hellman protocol from multiplication in a finite field to addition on a finite elliptic curve: Denote G a point on the curve, and

$$x\mathrm{G} = \mathrm{G} + \cdots + \mathrm{G}$$

the x -fold iterated addition over the finite elliptic curve (instead of g and $g^x = g \cdot sg$ for a finite field).

Setup. one chooses first

1. a prime number p that defines the field $\mathbb{F}_p$, and
2. the coefficients a and b that define the points E in the plane on $\mathbb{F}_p$ (subject to condition $4a^3 + 27b^2 \neq uiv0 \bmod p$ to not be singular).

To resist

- against Smart's attack $\#\{\text{pointson}\mathbb{F}_q\} \neq q$, and

- against Menezes, Okamoto and Vanstone's attack, must not be *supersingular*; that is, that $\#\{\text{pointsover}\mathbb{F}_q\} = q+1$ for $p > 3$ (while for $p = 2,3$ there are exactly three respectively four equations that define supersingular curves).

Then one chooses

3. One chooses a *base point* G in E .

The critical cryptographic number is the order n of the base point G that should be big enough.

To resist

- against the Pohlig-Hellman attack (which reduces the problem to its prime numbers) should be prime,
- against the baby-step-giant-step (or Pollard's ρ) attack must be $\geq 2^{224}$ (to make it computationally unviable).

To find a base point G whose order n is big enough, proceed as follows:

1. Randomly select coefficients a and b in $\mathbb{F}_q$.
2. Compute the number of points $N = \#E(\mathbb{F}_q)$ of E in $\mathbb{F}_q$ by Schoof's algorithm.
3. Verify that $N \neq q, q+1$ (to avoid the attacks of Smart and Menezes, Okamoto and Vanstone). Otherwise, go back to step one.
4. Check if there is a *prime* factor of N such that
   - $n > 2^{224}$, and
   - $n > 4\sqrt{q}$ (to prevent the attack of Pohlig-Hellman). Otherwise, go back to the first step.
5. Randomly pick points g in E up to $G = hG \neq 0$ to $h := N/n$.

That the point thus found has order n can be shown by *Langrange's Theorem* which asserts that $\#H|\#G$ where

- G is a *group*, that is, a set with an addition + that satisfies the associative and commutative law, has a 0 element and an inverse $-x$ for every x in G ,
- and H is a subgroup, that is, a subset H such that if x and y in H , then $x + y$ in H .

For example, $\mathrm{G} = \mathbb{Z}/10\mathbb{Z}$ is a group and $2 \cdot \mathrm{H} = \{0,2,4,6,8\}$ a subgroup.

Demonstration: For every point P we have $\#\mathrm{EP} = n h \mathrm{P} = 0$ by Lagrange's Theorem. That is, $n\mathrm{G} = 0$ for $\mathrm{G} = h\mathrm{P}$, and for the Lagrange Theorem $\#\langle \mathrm{G} \rangle$ divides n . Since n is prime, either $\langle \mathrm{G} \rangle = 1$, or $\langle \mathrm{G} \rangle = p$. Since $\langle \mathrm{G} \rangle \neq 1$, or $\#\langle \mathrm{G} \rangle > 1$, so $\#\langle \mathrm{G} \rangle = n$.

Example (of a base point).

The elliptic curve `Curve25519` with

$$y^2 = x^3 + 486662x^2 + x$$

over $\mathbb{F}_q$ with $q = p^2$ where $p = 2^{255} - 19$, uses as base point $\mathrm{G} = (x_\mathrm{G}, y_\mathrm{G})$ uniquely determined by

- $x_\mathrm{G} = 9$, and
- $y_\mathrm{G} < q/2$, that is, $y_\mathrm{G} =$ 14781619 44758954 47910205 93568409 98688726 46061346 16475288 96488183 77555862 37401).

Steps. In the `ECDH` (Elliptic Curve Diffie-Hellman) protocol, for Alice and Bob to overtly build a secret key, they first combine

- a q power of a *suitable* prime number p ,
- a *suitable* elliptic curve E over $\mathbb{F}_q$, and
- a *suitable* point G in E .

and then

1. Alice, to generate *one half* of the key, chooses a number a ,
   - calculates $\mathrm{A} = a\mathrm{G}$, and
   - transmits A to Bob.
2. Bob, to generate *another half* of the key, chooses a number b ,
   - calculates $\mathrm{B} \equiv b\mathrm{G}$, and
   - transmits B to Alice.
3. The secret *mutual* key between Alice and Bob is

$$c := b\mathrm{A} = ba\mathrm{G} = ab\mathrm{G} = a\mathrm{B}.$$

We note that for both to compute the same key c , the addition must satisfy the associative and commutative law; that is, it is indispensable that E be a group.

The ECDHE protocol, where the additional final E stands for 'Ephemeral', is, regarding the key exchange, the same as the ECDH protocol, but discards the keys (which are necessarily signed by permanent keys to testify the identity) after the session. Corbellini (2015b) is an implementation in Python of ECDH.

Self-Check Questions

1. How much older is RSA compared to ECC? *Around 20 years.*
2. How does the ECC Diffie-Hellman key exchange compare to the original Diffie-Hellman key exchange? *The* ECC *Diffie-Hellman key exchange uses points on an elliptic curve given by pairs of numbers whereas the original Diffie-Hellman key exchange uses simple numbers.*
3. How do the key sizes between RSA and ECC compare? *Usual are* 2048 *bits for* RSA *and* 224 *bits for* ECC.
4. How much faster is ECC compared to RSA?

   - ☐ 2
   - ☐ 3
   - ☐ *5*
   - ☐ 8

Summary

Asymmetric cryptography relies on a *trapdoor* function, which

- must be easily computable (for example, raising to the n-th power in RSA), but
- its inverse (for example, extraction of the n th root in RSA) must be practically incomputable without knowledge of a shortcut, the key!

This difficulty of calculating the inverse corresponds to the difficulty of decryption, that is, inverting the encryption. To *complicate* the computation of the inverse function (besides facilitating the computation of the proper function) is done using *modular* (or *circular*) arithmetic*, that we already from the arithmetic of the clock, where $m = 12$ is considered equal to 0 .

Let us denote $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$.

The most current cryptography widely in use uses *elliptic* curves (given by an equation $y^2 = x^3 + ax + b$) where one can *add* points on them: $p + q + r = 0$ if a line passes for p, q and r . By restricting the solutions to points (x, y) in $(\mathbb{F}_p \times \mathbb{F}_p$ for a large prime number p and fixing a point P on the curve,

- while it is *easy* to compute the *exponential*, that is, for n , compute

$$\mathrm{Q} = n\mathrm{P} = \mathrm{P} + \ldots + \mathrm{P},$$

- in contrast, given a point $\mathrm{Q} = \mathrm{P} + \ldots + \mathrm{P}$, it is *difficult* to compute the *logarithm*: that is, how many times P has been added, the number n such that $\mathrm{Q} = n\mathrm{P}$. By virtue of this point addition, the `Diffie-Hellman` protocol (over $\mathbb{F}_p$) has an analog over Elliptic Curves.
- Instead of *multiplying* repeatedly (n times) the base g in $\mathbb{F}_p^*$, that is, computing

$$g^n = g \cdots g,$$

- *add* repeatedly (n times) a point G , that is, compute

$$n \cdot \mathrm{G} = \mathrm{G} + \cdots + \mathrm{G}.$$

Security.

- instead of the logarithm over $\mathbb{F}_p$ (that is, the function that given numbers g and y determines the exponent x such that $y \equiv g^x \bmod p$),
- the *logarithm* over a *finite elliptic curve* (that is, the function that for a given point G and Y determines the scalar x in $\mathbb{N}$ such that $\mathrm{Y} = x\mathrm{G}$)

The advantage of using elliptic curves are shorter key sizes: Because, depending on the number of bits n of the used modulus p , regarding the fastest *presently* known algorithms:

- the time to compute the logarithm over an elliptic curve increases *linearly* and takes about $n/2$ operations, while
- the time to compute the multiplicative logarithm increases *sublinearly* and takes about $n^{1/3}$ operations.

For example, the security of a 2048 bit key for the multiplicative logarithm equals approximately that of a 224 bit key for the logarithm over an elliptic curve. To a length of 512 bits of a key for an elliptic curve, corresponds a length of 15360 bits of an RSA key.

ECC, elliptic curve cryptography, is becoming the new standard because its cryptographic problem (the logarithm over a finite elliptic curve) is currently computationally more difficult than that of RSA (the factoring in prime numbers) or DH (the logarithm over the multiplicative group of a finite field). small keys for the ECC achieve the same level of security as large keys for the RSA or DH. As an example, the security of a 224 bits key from the ECC corresponds to that of a 2048 bits key from the RSA or DH. This factor in reducing key sizes corresponds to a similar factor in reducing computational costs. Regarding usability,

- an ECC public key can be shared by spelling it out (it has 56 letters in hexadecimal notation,
- while a public key for RSA or DH has to be shared in a file (which is for convenience referred to by a fingerprint).

Questions

1. What key size in ECC is as secure as a 256 bit key in AES ?

   - ☐ 256
   - ☐ 512
   - ☐ 1024
   - ☐ 2048

2. What key size in ECC is as secure as a 256 bit key in AES?

   - ☐ 256
   - ☐ 512
   - ☐ 1024
   - ☐ 2048

3. What certificate size in ECC is as secure as a 256 bit key in AES?

   - ☐ 256
   - ☐ 512
   - ☐ 1024
   - ☐ 2048

Required Reading

Use Grau (2018d) to get an intuition for the graphs over finite domains.

Read Chapter 12 on elliptic curves of Aumasson (2017).

Further Reading

Use `CrypTool 1` to observe the addition of points on an elliptic curve.

See the book Sweigart (2013a) for implementing some simpler (asymmetric) algorithms in Python, a readable beginner-friendly programming language.

Read the parts of the book Schneier (2007) on understanding and implementing modern asymmetric cryptographic algorithms.

11 Authentication

Study Goals

On completion of this chapter, you will have learned ...

1. ... How a user can be authenticated by:
    1. a password or personal identification number (PIN),
    2. smart card (that has a microprocessor that stores a private key and runs cryptographic algorithms), or
    3. biometric identifiers (recognition of the users' signature, facial features, fingerprint, ...).
2. ... How authentication is most securely achieved over distance.
3. ... How the secret is never revealed by challenge and response protocols.
4. ... How no information other than the knowledge of the secret is leaked by zero-knowledge proofs.
5. ... How Kerberos mediates between users and servers without either one revealing her password to the other.

Introduction

Authentication is the identification of a person or of data; the confirmation

- that the user is who she claims to be, for example, when logging into a server by entering user name and password, respectively
- that the message (for example, an instruction sent to a bank by e-mail) is authentic, that is, unchanged between the time when the message was under someone's sight and its time of arrival.

This chapter treats exclusively the former type, the identification of a person, the user; particularly important on the Internet where the person is far away. In this sense *identification* is telling a computer or a network who the user is, usually by her user (or account) name. This is followed by *authentication*, the verification of the identity of a user, that is, convincing a computer that a person is who he or she claims to be.

To authenticate, the user can use as a proof information that

- only she *knows*, such as a password or personal identification number (PIN),
- only she *has*, such as
  - software certificate (containing public or private keys),
  - hardware certificate, such as a token or smart card (that has a microprocessor that stores a key and runs cryptographic algorithms),
  - a device, such as a phone or an e-mail account, to receive a code,
- only she is described by (what she *is*) such as biometric identifiers (recognition of facial features, fingerprint, ...).

Authentication should not be confused with *authorization*, the final confirmation of authentication that determines the user's eligibility to access certain contents, that is, what a user is allowed to do or see.

The authentication protocol can be simple or two-factor, that is:

- simple : a single proof suffices, for example, a password;
- two-factor : more than one proof is necessary, for example, a PIN and a smart card.

and one-way or mutual, that is,

- one-way: party A, such as the user, authenticates herself to party B, and
- mutual: likewise party B, such as the server, authenticates himself to party A.

Most operating systems (such as Linux) and applications store a hash of the authentication data rather than the authentication data itself. During authentication, it is verified whether the hash of the authentication data entered matches the hash stored. Even if the hashes are known to an intruder, it is practically impossible to determine an authentication data that matches a given hash.

11.1 Passwords

A password is a secret sequence of letters attached to a user identity that grants access to a system (such as a computer) after deriving it from the data of authorized users stored on the system.

> **password**: a secret sequence of letters attached to a user identity that grants access to a system (such as a computer).

It is the most common approach for authentication: gratis, convenient and private. It should be easy to memorize but difficult to guess.

Conveniences. Comparing authentication by what one knows (such as a password) to

- what one is (biometric data such as a fingerprint), the advantages are:
  - does not require specific (rather sophisticated) hardware,
  - it is securely stored, and
  - cannot be forged.
- what one has (such as a smart-card), the advantages are:
  - does not have to be carried around,
  - it is transparently stored, and
  - cannot be lost, stolen or extorted.

Criticisms.

- The user's limitation in memorizing sequences of symbols. The more meaningful, the more easily guessed (for example, a word of the user's tongue), but the more patternless, the harder to remember. A compromise is a passphrase, that is, a complete sentence instead of a single word; though longer, its content is more meaningful and thus more easily remembered than a patternless sequence of symbols. To shorten it, the first letter of each word is taken. For example, "Better to light one candle than to curse the darkness." can become "B2l1ct2ctd.".

  Another workaround is a password manager, a program that stores all passwords in a file encrypted by a master password.

  - therefore the passwords need no longer to be remembered,
  - thus may be arbitrarily complex,

  However,

  - one has to unconditionally trust this program,
  - the master password still is a password whose exposure when used for an insignificant account entails that of all other accounts, including the most critical ones.

- Can be acquired over distance; as a workaround, two-factor authentication requires a second ingredient to authenticate, which usually has to be spatially close to the user, such as a hardware token.

Attacks. Attacks during the entry of another person's password are:

- spying during the entry of the password. Workarounds are inconvenient, for example,

  - masking the typed letters,
  - covering the keyboard,
  - using a screen keyboard.

- keylogging

- login spoofing where one user's account a login entry form is faked so that the next user's entered login data, instead of granting access, are stored, display an error message and log the first user out.
- asking a user for the password, either through e-mail or on the phone as a purported system administrator.
- asking a system administrator for the password by posing as a purported user who has forgotten her password;
- asking a user to change her password on a purported entry form.

Attacks on another person's stored password exploit principally the following deficiency: Since passwords have to be memorized, they tend to be memorizable, that is, follow patterns. For example, are built from (birth)dates, counts and words, in particular names. Common are (reversing, capitalizing, ...):

- the user name
- the first, middle, or last name,
- the spouse's, children's, friend's, or pet's name,
- the date of birth, telephone number, house address,
- a word contained a (language) dictionary.

These more likely candidates can then be guessed first. Or, instead of building likely passwords, the attacker uses those already leaked to begin with.

Self-Check Questions.

1. Which conditions must a secure but practical password fulfill? *Must be easy to memorize but difficult to guess.*
2. List at least three common attacks during another person's password entry! *spying, keylogging and login spoofing*

11.2 Challenge-Response Protocols and Zero-knowledge

A **challenge-response** protocol poses a task that can be solved only by a user with additional authentication data; usually:

- either a cryptographic hash function or an enciphering function of a symmetric cryptographic algorithm is used whose secret key is shared among the claimant and verifier; the verifier generates a random number, and the claimant responds with the result of applying the hash function on that number.
- or a digital signature algorithm where the claimant signs with his private key a message generated by the verifier, which the verifier checks with the public key.

challenge-response protocol: poses a task that can be solved only by a user who has the authentication data.

A **Zero-knowledge Protocol** goes, in theory, further, as it shows how a claimant can prove knowledge of a secret to a verifier such that no other information (than this proof of knowledge) is disclosed; however:

- Proof is meant probabilistically, that is, the probability of the claim being true is so high as beyond any reasonable doubt (though the probability can be increased as much as desired by repeating the protocol), and
- the impossibility to gather information on the secret from that exchanged relies on the computational difficulty to solve a mathematical problem.

Zero-Knowledge Protocol: a protocol (first presented in Goldwasser, Micali, and Rackoff (1989)) to prove knowledge of a secret but disclose no other information.

Challenge-Response. A challenge-response protocol poses a task that can be solved only by a user with additional authentication data. For example,

1. as challenge some (randomly) generated value is encrypted using the password for the encryption key, and
2. as response a similarly encrypted value that depends on the original value,

thus proving that the user could decrypt the original value. For example, smartcards commonly use such a protocol. Such a (randomly) generated value is a **nonce** (number used once) and avoids a replay attack where the exchanged data is recorded and sent again later on.

nonce: stands for (a randomly generated) *number used once* used in a cryptographic protocol (such as one for authentication).

For example, in `CRAM-MD5` or `DIGEST-MD5`,

- the challenge is the (iterated) hash of the password and a (randomly) generated value, and
- the response is the hash of the password, and a value that depends on the original value:

1. Server sends a unique challenge value `challenge` to the client.
2. Client computes `response = hash(challenge + secret)` and sends it to the server.
3. Server calculates the expected value of `response` and verifies that it coincides with the client's response.

Such encrypted or hashed information does not reveal the password itself, but may supply enough information to deduce it by a dictionary (or *rainbow table*) attack, that is, by probing many probable values. Therefore information that is (randomly) generated anew enters on each exchange, a **salt** such as the current time.

> **salt**: a (randomly) generated number that enters additionally into the input of a hash function to make its output unique; principally if the additional input is a secret information such as a password.

Observation. Nonce, Salt and IV (Initialization Vector) are all numbers that are (usually) randomly generated, disclosed, used once in a cryptographic process to improve its security by making it unique. The name is given according to its use:

- a nonce is a number used once in a cryptographic protocol to make the exchange unique,
- a salt is used as additional input to a hash function to make its input unique (so that the same original input hashed gives different output), and
- an initialization vector is a number used as additional input to an enciphering (of a block cipher) to make its input unique (so that the same original input encrypted with the same key gives different output).

Challenge-Response protocols such as those presented below are used, for example, in object-relational databases such as `PostgreSQL`, or e-mail clients such as `Mozilla Thunderbird`.

Digest-MD5. `Digest-MD5` was a common challenge-response protocol that uses the `MD5` hash function and specified in `RFC2831`. It is based on the HTTP Digest Authentication (as specified in `RFC2617`) and was obsoleted by Melnikov (2011)

Challenge-Response Authentication Mechanism (CRAM). CRAM-MD5 is a challenge-response protocol based on HMAC-MD5, Hashing for Message Authentication as specified in Krawczyk, Bellare, and Canetti (1997), that uses the `MD5` hash function. The RFC draft Zeilenga (2008) recommends obsoleting it by `SCRAM`.

Steps.

1. The server sends the client a `nonce`.
2. The client is supposed to respond with `HMAC(secret, nonce)`.
3. The server calculates `HMAC(secret, nonce)` and checks whether it coincides with the client's response to be convinced that the client knew `secret`.

Weaknesses.

- No mutual authentication, that is, the server's identity is not verified.
- The used hash function `MD5` is quickly computed, and thus facilitates dictionary attacks. Instead, key stretching, that is, using a hash function that is deliberately computationally expensive is preferable.
- Weak password storage:
  - Some implementations store the user's plain password, while
  - others (such as `Dovecot`) store an intermediate hash value of the password: While this prevents storage of the plain password, for authenticating with CRAM-MD5, knowledge of the hash value is equivalent to that of the password itself.

Salted Challenge-Response Authentication Mechanism (SCRAM). Salted Challenge-Response Authentication Mechanism (SCRAM) is a challenge-response protocol for mutual authentication specified in Menon-Sen et al. (2010) (that should supersede CRAM-MD5 as proposed in Zeilenga (2008)).

While in CRAM the client password is stored as hash on the server, now knowledge of the hash (instead of the password) suffices to impersonate the client on further authentications: The burden has just been shifted from protecting the password to its hash. SCRAM prevents this by demanding additional information (on top of the authentication information StoredKey stored on the server) that was initially derived from the client's password (ClientKey). Advantages of SCRAM in comparison to older challenge-response protocols, according to loc.cit., are:

- The authentication information stored on the server is insufficient to impersonate the client. In particular,
  - a dictionary attack (so-called rainbow tables) after an authentication-database leakage is prevented by salting the information,
  - the server cannot impersonate the client to other servers because it stores only partial authentication information,
  - password reuse after data breach is prevented by binding the hash to a single server: only the salted and hashed version of a password is used during login and the salt on the server is immutable.
- supports mutual authentication (by the client and server).

Key Creation, Transmission and Storage. When the client creates a password Password, the server stores derived keys StoredKey and ServerKey together with the parameters used for its derivation, as follows:

1. The client:
   1. computes SaltedPassword by applying the password hashing function PBKDF2 (which is variable; by default it is PBKDF2, but nowadays, for example, bcrypt is recommended) IterationCount many times on the input given by the password Password and the salt Salt, that is, `SaltedPassword := PBKDF2(Password, Salt, IterationCount)`

2. computes `ClientKey` respectively `ServerKey` by applying the HMAC function on `SaltedPassword` with the public constant strings "Client Key" respectively "Server Key", that is, ServerKey := HMAC(SaltedPassword,′ ServerKey′) and ClientKey := HMAC(SaltedPassword,′ ClientKey′)
3. computes `StoredKey` by hashing `ClientKey`, that is, `StoredKey := H(ClientKey)` and sends `ServerKey` and `StoredKey` to the server (but *not* `ClientKey`).

2. The server stores `StoredKey`, `ServerKey`, `Salt` and `IterationCount` to later check proofs from clients and issue proofs to clients: `ClientKey` is used first to authenticate the client against the server and `ServerKey` is used later by the server to authenticate against the client.

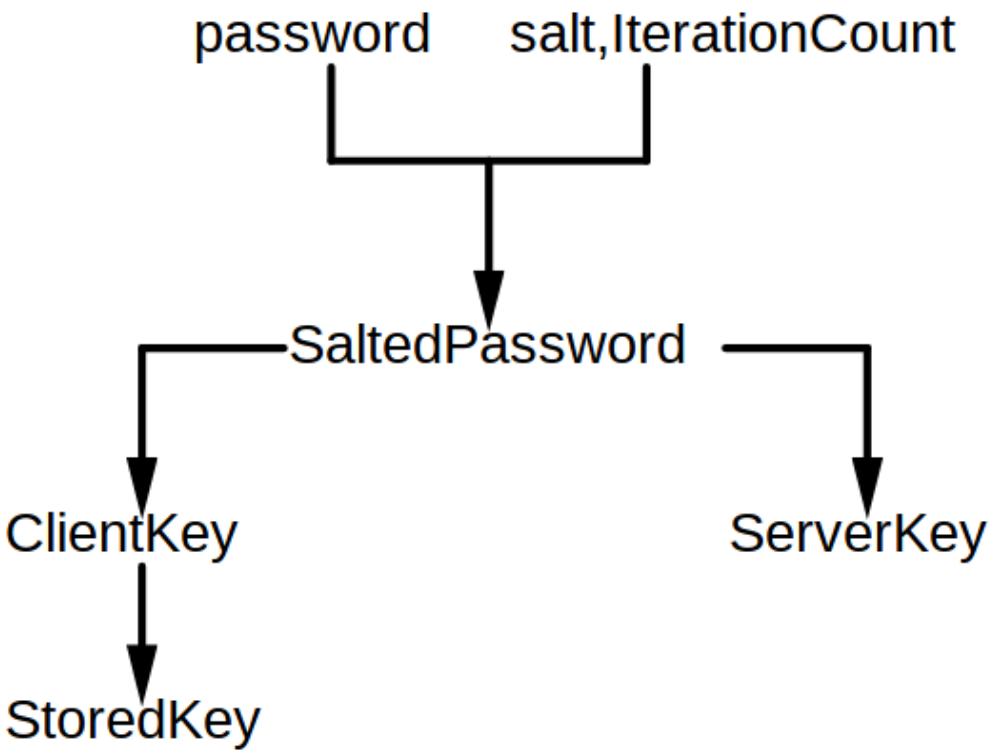


Figure 55: Dependency graph in which each oriented edge is the output of a one-way function, Cesar (2020)

The server only stores the public part of the root (Salt and `IterationCount`)

and the leafs (StoredKey and ServerKey) of this tree. That is, the password is never sent to the server, but only:

- the salt,
- the iteration count,
- ServerKey and StoredKey, that is,
  - HMAC(SaltedPassword,′ ServerKey′), and
  - H(HMAC(SaltedPassword,′ ClientKey′)).

Thus, after a database breach, that is, after an attacker has stolen a ServerKey, a client's password does not need to be replaced, but only the salt and iteration count changed and ClientKey and ServerKey replaced.

> **iteration count**: Given an initial input, apply a hash function that many times (–1) to the output.

Client Authentication to the Server. For the server to authenticate the client:

1. The client sends to the server an authenticator (containing her client-name and a client-nonce).
2. The server sends to the client a salt salt, iteration count ic and a server-nonce.

Therefore, both, the client and server know AuthMessage := client-name, client-nonce, salt, ic, server-nonce.

3. The client
   1. creates proof of her knowledge of StoredKey by computing

      ```
      ClientSignature := HMAC(StoredKey, AuthMessage)
      ClientProof     := ClientKey XOR ClientSignature
      ```

   2. and sends ClientProof to the server.
4. The server
   1. recovers ClientKey by

    1. computing `ClientSignature` (by knowing `StoredKey` from storage and `AuthMessage` from this exchange), and
    2. deciphering `ClientKey'` from the one-time pad `ClientProof` by computing

       ```
       ClientKey' = ClientProof XOR ClientSignature
       ```

2. computes `StoredKey'` by `H(ClientKey)`, and
3. checks whether the computed `StoredKey'` coincides with the stored `StoredKey`; if so the client is successfully authenticated.

If just `ClientSignature` were sent, then an attacker who knows `StoredKey` could impersonate the client. Instead, `ClientProof` additionally requires the client to know `ClientKey`. Therefore, the value of `ClientKey'` that is calculated on the server should be immediately and irreversibly (say, by zeroing) deleted after verification.

Client Authentication to the Server.

1. The client initiates the protocol by sending the Client's First Message to the server that contains:
    - the client's user name, and
    - a client nonce `ClientNonce` randomly generated by the client.
2. In response to the reception of a valid message from the client, the server sends the Server's First Message to the client that contains:
    - a server nonce `ServerNonce` randomly generated by the server.
    - a salt `salt` randomly generated by the server that enters with the password as input of a hash function.
    - an **iteration count** `IterationCount` generated by the server that indicates how many times the hash function is applied to the salt and the password to obtain its output.

The concatenation of the client's and server's message is `AuthMessage`

```
AuthMessage = username, ClientNonce, ServerNonce, salt, IterationCount
```

3. The client creates the proof for the server by computing:

    1. `ClientSignature = HMAC(StoredKey, AuthMessage)`, and
    2. ClientProof = ClientKey $\oplus$ ClientSignature

where `StoredKey` can be recomputed by `StoredKey = H(ClientKey)` and ClientKey = HMAC(SaltedPassword,′ ClientKey′) using the `salt` and `IterationCount` from `AuthMessage` and the user's password.

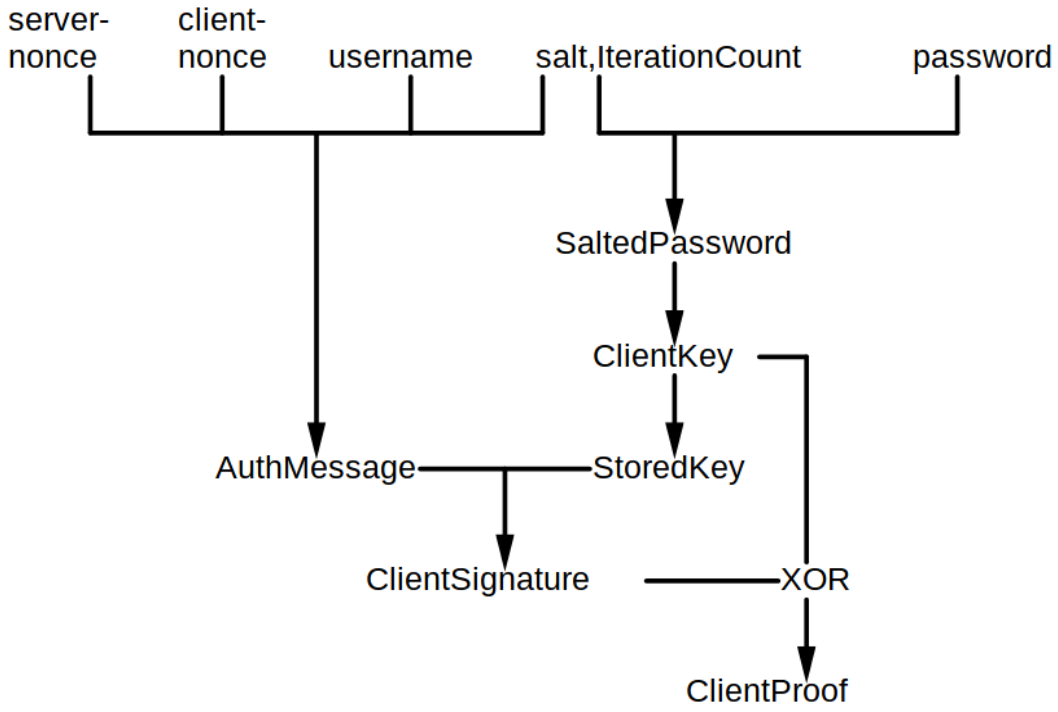


Figure 56: Dependency graph in which each oriented edge is the output of a one-way function, Cesar (2020)

4. The client sends `ClientNonce` and `ClientProof` to the server
5. The server checks the proof `ClientProof` from the client by

    1. recalculating
        1. `ClientSignature = HMAC(StoredKey, AuthMessage)`,
        2. `ClientKey` by ClientKey′ = ClientProof $\oplus$ ClientSignature,
        3. `StoredKey' = H(ClientKey')`, and
    2. checking whether StoredKey′ = StoredKey.

If just `ClientSignature` were sent, then an attacker that knows `StoredKey` could impersonate the client. Instead, `ClientProof` additionally requires the client to know `ClientKey`. Therefore, the value of `ClientKey'` that is calculated on the server should be immediately and irreversibly (say, by zeroing) deleted after verification.

We conclude that instead of the client's password, sent were

- the client nonce `ClientNonce` and server nonce `ServerNonce`,
- the salt `salt`,
- the iteration count `IterationCount`,
- ClientProof = ClientKey $\oplus$ ClientSignature where ClientSignature = HMAC($Stored Key, Auth Message$) ,

6. The server computes `ServerSignature = HMAC(ServerKey, AuthMessage)`
7. The server sends `ServerSignature`
8. The client checks `ServerSignature` from the server by
    1. recalculating
        1. ServerKey := HMAC(SaltedPassword,′ ServerKey′)
        2. ServerSignature′ = HMAC(ServerKey, AuthMessage)
    2. checking whether ServerSignature′ = ServerSignature.

We conclude that instead of the server's key, sent was

- `ServerSignature = HMAC(ServerKey, AuthMessage)`.

Caveat. If an attacker knows

- the `StoredKey` from the server, *and*
- the `AuthMessage` and `ClientProof` from an authentication exchange,

then he can calculate `ClientSignature`, thus `ClientKey` and can impersonate the client to the server.

Zero-knowledge Proofs. A **Zero-Knowledge Protocol** shows how a claimant can prove knowledge of a secret to a verifier such that no other information is disclosed; that is, **not a single bit of information**, other than the validity of the claimant's claim, is disclosed (to anyone, including the verifier).

Proof is meant probabilistically, that is, the claim is true beyond any reasonable doubt. More so, because the proofs are independent of each other, the probability can be increased as much as desired by increasing their number. The impossibility to gather information on the secret from that exchanged relies on the computational difficulty to solve a mathematical problem.

That no information whatsoever is leaked cites the following benefits:

- The verifier cannot obtain any information even if she does not adhere to the protocol; every proof is *independent* of each other.
- The verifier cannot impersonate the claimant to a third party: A recording of the proof does not help in convincing a third party, because the sequence could have been mutually fixed in advance.

Comparison to Classic Protocols. While claiming to know the secret alone is unconvincing, the leakage of information during classical protocols in which a claimant `C` proves knowledge of a secret to verifier `V`, still compromises the secret as follows:

- If `C` transmits his password to `V`, then `V` and everyone who eavesdropped this transmission obtains all data to impersonate `C` from this moment on.
- In a challenge-response protocol, a new challenge is used in every occurrence of the protocol. Therefore, `V` and an eavesdropper can during every occurrence accumulate new information on the secret so that it eventually yields to it. For example, if the challenge is the encryption of a plaintext, and the attacker can choose this plaintext, then this is a chosen-plaintext attack.

Public Key. To weigh up the advantages and inconveniences between a zero-knowledge proof and a digital signature by a private key (verified by its corresponding public key):

- Like every challenge-response protocol, every signature, that is, encryption of a document by the user's private key, leaks information. In the extreme case that the attacker can choose this plaintext, this is a chosen-plaintext attack.
- However, zero-knowledge protocols require less computation than public-key protocols. Since digital signatures are practically secure and many devices, such as smart cards, have little computing power, in practice digital signatures are more common.

The security of both, most zero-knowledge protocols and public-key protocols, depends on the unproved assumption that cryptanalysis is computationally as difficult as a mathematical problem (such as the computation of quadratic residuosity, the decomposition of an integer into its prime factors, discrete logarithm, ...).

History. The concept of

- Interactive zero-knowledge proofs was introduced in Goldwasser, Micali, and Rackoff (1989).
- Non-interactive zero-knowledge proofs in Blum, Feldman, and Micali (2019).

An interactive protocol is turned into a non-interactive one by taking the hash of the initial statement and then send the transcript to the verifier (who also checks that the hash was computed correctly).

Practical protocols were introduced in Fiat and Shamir (1987) and Feige, Fiat, and Shamir (1988).

The latter introduced so-called *sigma protocols* in three steps:

1. the claimant commits to a value by sending it to the verifier,
2. the verifier chooses a random challenge,
3. the claimant "opens" a combination of the original value and the challenge.

IEEE 1363.2 defines zero-knowledge password proof as "an interactive zero-knowledge proof of knowledge of password-derived data shared between a prover and the corresponding verifier".

Ali Baba's Cave. Ali Baba's cave illustrates the principles behind a zero-knowledge proof: In a circular cave there is a door invisible from the entrance that bars any passage if not opened by a password (such as "Sesame"). For C to prove to V that he knows the password without disclosing it:

1. C enters the cave unobserved from V till standing in front of the door's left or right.
2. V demands C to return to the entrance coming from the left or right.

Because the probability that C entered the cave on the same side as V asked for to leave on is one half and all proofs are independent, for example, after 10 proofs the chance C does not know the password is 2^{-10}, less than a thousandth.

While V is convinced that C knows the password, she cannot convince anybody else. Even if V recorded the sequence, then it could have been mutually fixed in advance.

Schnorr's Sigma Protocol. Schnorr (1991) presented a zero-knowledge protocol simple enough to run on smart cards. Knowledge of discrete logarithms is proved, that is,

- the prover P knows an integer x , and
- the verifier V knows $g^x \bmod p$

where p is a prime number. For P to prove knowledge of x without revealing it:

1. P chooses some integer a and sends g^a to V .
2. V tosses a coin and sends the result c in $0,1$ to P .
3. P sends to V
    - either a , whenever $c = 0$,
    - or $a + x$, whenever $c = 1$.

This is a zero-knowledge protocol, because

- if $c = 0$, then nothing about x is revealed (but only a)
- if $c = 1$, then the verifier learns $a+x \bmod p$, but again, as long as nothing about a is revealed (where we count on the difficulty of computing log mod p), neither anything about x is revealed.

If $c = 0$, then no knowledge of x is needed. If $c = 1$ however, then V can verify whether P knows $a + x$ by $g^{a+x} = g^a g^x$ where both values on the right-hand side are known. Since the probability that $c = 1$ is $1/2$ and all proofs are independent, after, say, 10 proofs the chance V does not know x is $2^{-10} < 1/1000$.

Fiege-Fiat-Shamir protocol. The security of the Fiege-Fiat-Shamir protocol rests on the assumed computational difficulty of extracting square roots modulo large composite integers whose prime factors are unknown (similar to RSA):

1. An arbiter (a trusted, independent entity) generates a product n of two large random primes; in practice a number of 1024 bits.
2. The arbiter then generates a public and private key pair for the prover P, as follows:
    1. choose a number v , which is a quadratic residue modulo n ; that is, $x^2 \equiv v \bmod n$ has a solution, and $v^{-1} \bmod n$ exists.
    2. let the public key be v , and
    3. let the private key be the smallest s such that $s^2 \equiv v \bmod n$, that is, $s \equiv v^{-1/2} \bmod n$.
3. P, the prover,
    1. chooses a random number r that shares no divisor with n .
    2. computes $x \equiv r^2 \bmod n$, and
    3. sends x to the verifier V .
4. V, the verifier,
    1. tosses a coin, and
    2. sends the result c in $\{0,1\}$ to P .
5. P, the prover,
    - If $c = 0$, then sends her r .
    - If $c = 1$, then sends her $y \equiv r \cdot s \bmod n$.
6. V, the verifier,
    - if $c = 0$, then verifies that $x \equiv r^2 \bmod n$, proving that P knows $x^{1/2} = r$.
    - if $c = 1$, then verifies that $x \equiv y^2 \cdot v \bmod n$, proving that P knows $(x/v)^{1/2}$.

If $c = 0$, then no knowledge of the private key s is needed. If $c = 1$ however, then V can verify whether P can compute $(x/v)^{1/2}$, thus presumably knows s . Because the probability that $c = 1$ is one half and all proofs are independent, for example, after 10 proofs the chance V does not know x is 2^{-10}, less than a thousandth.

This is nearly a zero-knowledge protocol, however, care must be taken:

- P should always choose a new r in each round. Otherwise, V could gather information by manipulating the "random" bits.
- This protocol leaks information: If the answer is $y = r \cdot s$, then this backs up that v is indeed a square modulo n ; because this is a sound protocol, after a certain number of iterations, we can conclude that this is true.

Self-Check Questions.

1. How does a challenge-response protocol authenticate a client? *By posing a task to the user that she can only solve if having authentication data.*
2. What is a salt? *A (randomly) generated number that enters additionally into the input of a hash function to make its output unique.*
3. How does storage of the identification data differ between SCRAM and CRAM? *In* CRAM *a hash of the client password is stored as is on the server, while in* SCRAM *additional data (salt) enters.*
4. Name an advantage and inconvenience of zero-knowledge proof in comparions to a digital signature. *The proof of knowledge does not leak any information on the user's private key, while a digital signature requires less computation.*
5. On the computational difficulty of which mathematical function does the zero-knowledge property of Schnorr's protocol rely? *On the discrete logarithm.*

11.3 Biometric Authentication

Biometric authentication identifies the user of a computer by

- either physical human characteristic, such as:
  - fingerprints (which have been around for a long time),

  - face characteristics, such as the relative positions of facial features such as eyes, nose, lips, jaw, and
  - eye characteristics, such as the pattern of blood vessels in the iris.

- or behavioral human characteristic, such as:

  - typing characteristics such as the speed of keystrokes or the occurrence of typos (particularly useful to supplement a log-in dialogue);

  - handwriting characteristics; either *static* where an image is used, or *dynamic* where the traces on a tablet are evaluated by the functions (of time):

    * x and y -coordinates,
    * pressure, and
    * inclination

  - voice properties (speaker recognition; particularly useful to verify the identity of telephone customers). Each spoken word is decomposed into its formants, the dominant frequencies, and then physiological and behavioral characteristics identified:

    * the physiological characteristics describe the shape of the person's vocal tract, that is, of her nose, mouth, jaw, tongue, ...
    * the behavioral characteristics describe the movement of the nose, mouth, jaw, tongue, ... that change the accent, tone, pitch, pace, ...

Advantages. Comparing authentication by what one is (for example, a fingerprint) to

- what one knows (for example, a password), the advantages are:

  - that even when known, it requires some effort to replicate, and
  - it cannot be forgotten.

- what one has (for example, a token), the advantages are:

  - does not have to be carried around, and
  - it cannot be lost or stolen.

Limitations.

- it requires sophisticated hardware, and
- is exposed and thus imitable:

For example, the German Chaos Computer Club (CCC) and its collaborators have demonstrated time and again the ease of forging fingerprints, for example, using wood glue and sprayable graphene. The many successful forgery attacks against biometric identification, such as fingerprints and photo ID, leads to the conclusion to treat them, instead of a replacement for passwords or smart-cards, rather

- either as a complement to them, a second factor,
- or even merely as a replacement for a username: publicly known information that must be complemented by secret information, such as a password, to authenticate.

Self-Check Questions.

1. List advantages of authentication by what one knows over what one is:

    - *stored securely,*
    - *cannot be forged,*
    - *does not require specific hardware.*

11.4 Authentication in a Distributed System

In a (distributed) system, only if

- the identity of the system against which to authenticate is guaranteed, and
- nobody is possibly eavesdropping,

then the secret itself for authentication could be responsibly sent in the clear. Otherwise, to ensure that the secret for authentication is seen by none other than possibly the intended recipient, the secret itself must never be sent in the clear. Instead, both the user and the system convince each other they know the shared secret (usually a password). That is, the identification data itself is never sent, but only proof she has access to it beyond any doubt. In practice,

for a system that is password-based, the most popular approach are challenge-response systems. A **challenge-response** protocol poses a task that can be solved only by a user with additional authentication data: Usually,

- either a cryptographic hash function or the enciphering function of a symmetric cryptographic algorithm is used whose secret key is shared among the claimant and verifier: the verifier generates a random number, and the claimant responds with the result of applying the hash function on that number;
- or a digital signature algorithm where the claimant signs with her private key a message generated by the verifier, which the verifier checks with the public key.

challenge-response protocol: poses a task that can be solved only by a user who has the authentication data.

As best practice, even though the secret itself is never sent, it is still advisable to encrypt all communication to establish authentication, for example, by public-key encryption.

Other approaches are:

- to reveal the secret only partially, for example in a **single-use** system uses the identification data only once. For example, TANs (TransAction Numbers) in banking. However, if the identification data can be eavesdropped and its use for authentication, and thus invalidation, be prevented (for example, by logging into a forged copy of the bank's Website), then it can be used later.
- to send additional secret information via a **second-channel**. For example, the sending of an SMS in the mobile TAN (mTAN) system.

single-use system: the identification data is only used once.

second-channel system: the identification data is sent via a second channel.

FIDO 2. The FIDO (Fast IDentity Online) Alliance was officially founded in February 2013 to develop open and license-free industry standards for authentication on the Internet in arrangement with many companies such as `Google` or `Microsoft`. On December 9, 2014, the first standard FIDO v1.0 was released that specifies:

- The standard U2F (Universal Second Factor) for hardware and software for two-factor authentication.
- The standard UAF (Universal Authentication Framework) for the associated network protocol for password-less authentication.

These standards aim to facilitate authentication on the Internet by using a user's

- belongings (what she has), such as security tokens, or
- properties (what she is), such as fingerprint,

instead of

- knowledge (what she knows), such as passwords or personal identification numbers

That is, no longer needs a user to memorize numerous secure passwords (though at the cost of the drawbacks discussed in Section 11.1). In comparison with previous methods of two-factor authentication such as SMS verification codes, FIDO2 requires the key, such as the smart phone, to be physically near your computer. FIDO2 ("Moving the World Beyond Passwords") consists of the

- W3C Web Authentication Standard (WebAuthn) that allows users to log into Internet accounts using biometrics, mobile devices and/or FIDO security keys.
- the Client to Authenticator Protocol (CTAP) of the FIDO Alliance that is based on U2F. (While, with the release of FIDO2, U2F was renamed to CTAP1.) CTAP is, among other use cases, for authentication in desktop applications and web services.

Personal data and private keys are always and exclusively in the hands of the user and are not stored on public servers. To register via FIDO2 an account on a server:

1. The server sends a request.

2. The FIDO2-key generates a public and a secret key from a secret initial key and the server address.
3. The public key is transmitted to the server, which stores it, and thus can uniquely identify the FIDO2-key in the future.

This way, the FIDO2-key can identify itself with an individual key at each server without the server obtaining information on the key pairs for the same FIDO2-key on other servers. That is, the FIDO2-key generates a separate key pair for each service, based on the domain of the other party. For example, Ebay and Google, cannot determine which of their users use the same FIDO2-key. In practice, this is an advantage (on the server-side!) to authentication by a password where a user often uses similar passwords on different servers.

Key. A FIDO2 key (or authenticator or token) is the device by which to authenticate oneself to a service. It can be

- either an external device to connect to the user's PC or smart phone via USB, NFC or Bluetooth; such as
  - a security token to be inserted into a USB port
  - a smart card to be inserted into a card reader,
  - an NFC token (Near Field Communication; a wireless communication technology to exchange data between closely located devices, that is, about a decimeter apart; jointly developed by Sony and Philips and approved an ISO standard in December 2003),
  - Smartphones with Bluetooth-Interface IEEE 802.15.1
  - Bluetooth Tokens (Bluetooth V4.0 Low Energy 2,45 GHz)
- or an internal authenticator. That is, software that uses the crypto chip of your PC, smart phone or tablet for FIDO2, supported by Windows 10 and Android 7 and above.

Against misuse of the FIDO2 key, it can be additionally secured biometrically or with a password/PIN. If the stick is lost, then either a registered backup key is available or one has to identify oneself again by, say, the mobile phone number in combination with an e-mail address or alike.

Adaption. A FIDO2 key can either be used instead of a password or in addition to it, as a second factor. Depending on how a service has implemented FIDO2, the key suffices for logging in (one-factor authentication) or additionally entering a password (two-factor authentication) is necessary. FIDO2 one-factor authentication already is available for Microsoft.com, Outlook.com, Office 365 and OneDrive in the Edge browser. FIDO2 two-factor authentication works, for example, with Google, GitHub, Dropbox and Twitter.

Kerberos. One solution to the problem of *key distribution*, that is,

- to secretly pass the same key to all correspondents,
- the many keys needed for a group of correspondents to communicate securely to each other,

is a *central authority* who is

- unconditionally trusted by all correspondents, and
- from whom each user can securely obtain a session key for each correspondence,

so that each correspondent has only to protect one key, while the responsibility to protect all the keys among the correspondents is shifted to the central authority.

Kerberos (pronounced "kur-ber-uhs") is named after Cerberus, the three-headed watchdog at the gate to Hades; while Cerberus authenticates dead souls, Kerberos mutually authenticates users over a network:

Authentication and Ticket-Emission Components. **Kerberos** is a network protocol (as specified in RFC 1510 — The Kerberos Network Authentication Service V5) which permits users to securely authenticate to each other over an insecure network by a trusted third party, the Key Distribution Center (**KDC**) . Once a user is authenticated (Kerberos), she is authorized by access protocols such as LDAP (Lightweight Directory Access Protocol).

> **Kerberos**: a network protocol to securely authenticate clients to servers over an insecure network by a trusted third party.

Key Distribution Center (**KDC**): stores the symmetrical keys of all registered users and servers to securely authenticate them.

The Key Distribution Center (**KDC**) stores the symmetrical keys of all registered users (be it client or server) to authenticate them as an intermediary third-party. Due to its critical role, it is good practice to have a secondary KDC as a fallback.

Kerberos groups users into clients and (service) servers (**SS**s) that host services for the clients. The authentication protocol authenticates a client only once so that she is trusted by all SSs for the rest of her session. This is achieved by

- a **ticket**, a one-use credential emitted by the KDC to authenticate a client to a server from which she is requesting a service, encrypted using the server's key; it contains the server's and the client's ID, the client's network address, a timestamp, a lifetime, and a session key encrypted using the client's key.
- an **authenticator**, a credential, encrypted using the session key shared between the client and the server, that accompanies the ticket to authenticate the client; it contains the client's ID, the client's network address and a timestamp.

The ticket, to further obstruct man-in-the-middle attacks (in comparison to a mere key authentication),

- (cryptographically) links the granted access, in addition to the client's ID, also to her *network address*, and
- has a *timestamp* and *lifetime*.

The KDC is split up into :

- an Authentication Server (**AS**): To authenticate each user in the network, the AS stores a symmetric key for each user, be it client or service server (**SS**), and which is only known to itself, the AS, and the user.

  1. After the AS has authenticated a client by the client's key,
  2. the AS sends her a Ticket-Granting Ticket (**TGT**) and a session key encrypted using the client's key.

- a Ticket-Granting Server (**TGS**):

1. After a client has sent her TGT,
2. has been authenticated by the TGS, and
3. requests a service of an SS,
4. the TGS emits a ticket and two copies of a session key, one encrypted using the user's TGT session key and the other encrypted using the SS's key, to authenticate the client to the SS and secure their communication.

Service Server (SS): the server that hosts a service the user wants to access.

Authentication Server (AS): stores for each user (be it client or server) in the network a symmetric key known only to itself, the AS, and the user.

Ticket-Granting Server (TGS): generates a session key as part of a ticket between two users of the network after authentication.

Authentication Protocol. To allow a user (commonly referred to as *client*) to securely communicate with another user (commonly referred to as Service Server or Application Server (**SS** or **AP**), the server that hosts a service the client wants to access) via the KDC, the Kerberos protocol defines ten messages, among them:

Code	Meaning
KRB_AS_REQ	Kerberos Authentication Service Request
KRB_AS_REP	Kerberos Authentication Service Reply
KRB_TGS_REQ	Kerberos Ticket-Granting Service Request
KRB_TGS_REP	Kerberos Ticket-Granting Service Reply
KRB_AP_REQ	Kerberos Application Request
KRB_AP_REP	Kerberos Application Reply

1. The client, to authenticate to the AS,
    1. authenticates to the authentication server (AS) (KRB_AS_REQ) using a long-term shared secret (client's key), and

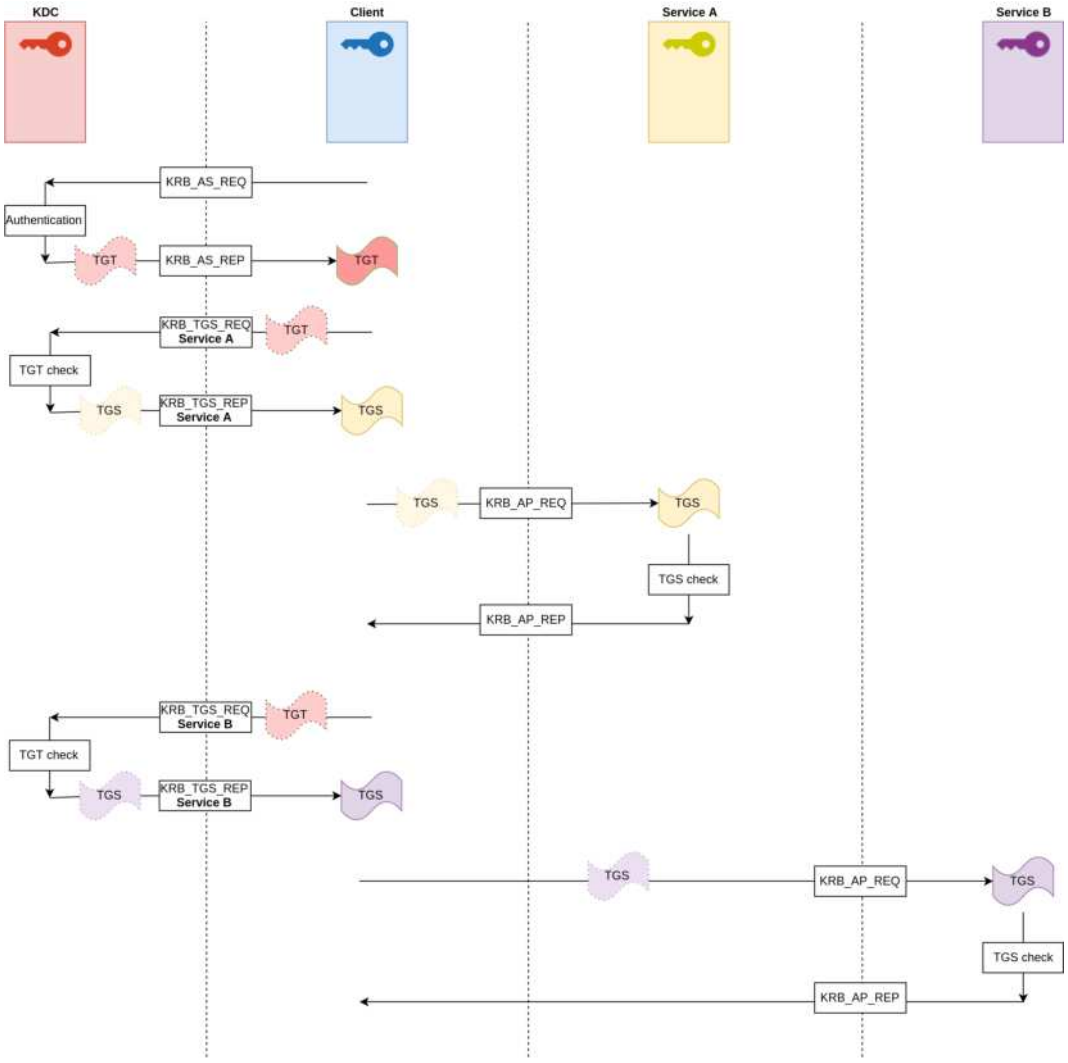


Figure 57: The authentication and subsequent ticket granting between a user and a server mediated by Kerberos server; Bentz (2019)

2. receives a short-term shared secret (session key) and a (ticket-granting) ticket from the authentication server (KRB_AS_REP).

2. The client, to authenticate to the SS via the AS,

    1. authenticates to the AS using her TGT, and
    2. requests (KRB_TGS_REQ) a ticket from the TGS (KRB_TGS_REP) that contains a session key between the client and the SS.

3. The TGS, to create the ticket,

    1. generates a client-to-server session key,
    2. encrypts, using the key of the client, the session key,
    3. encrypts, using the key of the SS, the client-to-server ticket that contains the client's ID, the client's network address, a timestamp, a lifetime, and the session key, and
    4. sends the results of both encryptions to the client.

4. The client, to authenticate directly to the SS:

    1. decrypts the client-to-server session key using her own key, and
    2. sends to the SS (KRB_AP_REQ and KRB_AP_REP)

        - the client-to-server ticket, encrypted using the SS's key, and
        - an authenticator that contains the client's ID and a timestamp N , encrypted using the client-to-server session key.

5. The SS

    1. retrieves the client-to-server session key by decrypting the client-to-server ticket using his own key,
    2. decrypts, using the session key, the authenticator and checks it. If the check succeeds, then the server can trust the client.
    3. The SS authenticates to the client by sending the client $N + 1$, the timestamp of client's authenticator incremented by 1 , encrypted using the client-to-server session key.

6. The client decrypts, using the client-to-server session key, the incremented timestamp and checks it. If the check succeeds, then the client can trust the server and can start issuing service requests to the server.

The server provides the requested services to the client.

History. The Kerberos protocol is based on the Needham-Schroeder authentication protocol, invented by Roger Needham and Michael Schroeder in 1978, and designed to securely establish a session key between two parties on an insecure network (the Internet, for example) by a symmetric-key algorithm.

MIT developed Kerberos to protect network services provided by Project Athena that aimed at establishing a computing environment with up 10,000 workstations on varying hardware, but where a user could on every workstation access all files or applications in the same user interface; similar to what a browser achieved today. Versions 1, 2 and 3 were only used at MIT itself. Version 4 employed the Data Encryption Standard encryption algorithm, which uses 56-bit keys, and was published in 1989.

In 2005, the Internet Engineering Task Force Kerberos working group introduced a new updated specifications for Kerberos Version 5, and in 2007, MIT formed the http://www.kerberos.org for continuation of development.

MIT Kerberos is the reference implementation that supports Unix, Linux, Solaris, Windows and macOS, but other commercial and non-commercial implementations exist: Most notably, Microsoft added a slightly altered version of Kerberos V5 authentication in Windows 2000.

Kerberos originally used the `DES` algorithm for encryption and the CRC-32, MD4, MD5 algorithms for hashing, but nowadays Kerberos implementations can use any algorithm for encryption and hashing.

Weak Spots.

- Single point of failure in the permanent availability of the key distribution center (KDC), which can only be alleviated by fall-back KDCs, referred to as "Kerberos slave servers" which synchronize their databases from the master Kerberos server.
- The tickets are managed locally on the client computer in the `/tmp` directory and only deleted after their expiration, thus, in multi-user system, risk to be stolen.
- To avoid *replay attacks* (where the attacker uses recorded data), the RFC requires synchronizing all clocks (no more than 5 minutes difference and preferably achieved by the Network Time Protocol; NTP) of the involved parties because of the period of validity

- The administration protocol is not standardized and differs between implementations, but password changes are described in RFC 3244.

Protocol Steps. Let us detail each step from logon and authentication to service authorization and request

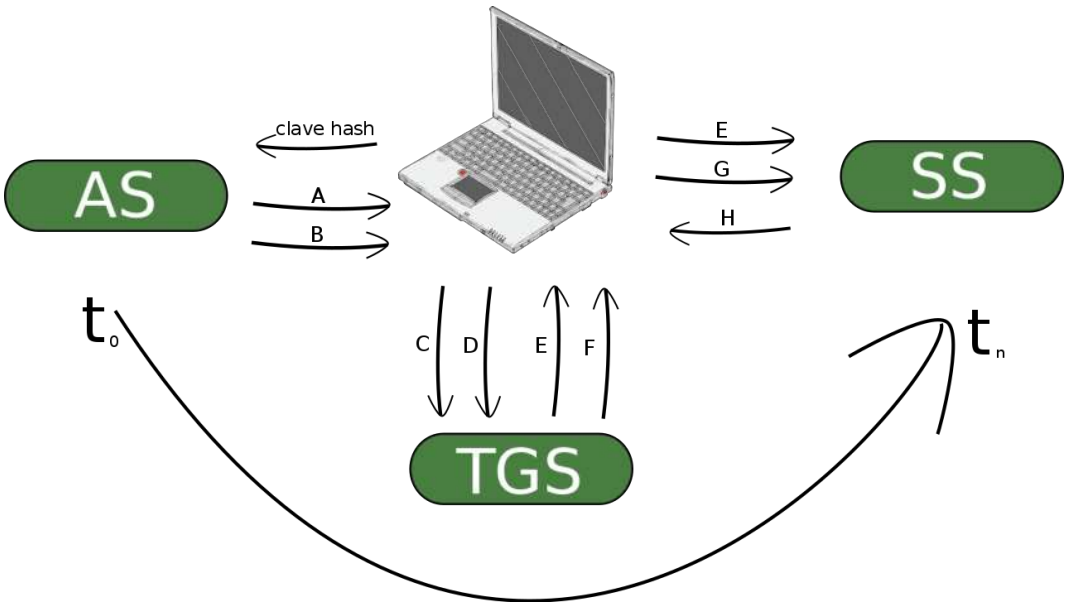


Figure 58: Kerberos Messages exchanged between the client and the AS = Authentication Server, TGS = Ticket-Granting Server and SS = Service Server (Omerta-ve (2012))

Client Logon.

1. A user enters a username and password on the client.
2. The client applies a one-way function (mostly a hash function) on the entered password, and this becomes the secret key of the client.

Client Authentication.

1. The client sends a cleartext message to the **Authentication Server** (AS) asking for services on behalf of the user (KRB_AS_REQ).
2. The AS checks whether the client is in his database.

3. If she is, then the AS sends back the following two messages to the client:

    a. Client/**Ticket-Granting Server** (TGS) *Session Key*, encrypted using the *secret key of the client.*

    b. **Ticket-granting Ticket** (TGT), encrypted using the *secret key of the TGS*, and which includes the

        - client ID,
        - client network address,
        - ticket validity duration, and
        - the *Client/TGS Session Key.*

4. Once the client receives messages `A` and `B`, she *retrieves the Client/TGS Session Key* by decrypting message `A` using her secret key.

The Client/TGS Session Key is used for further communication with TGS. At this point, the client has enough information to authenticate herself to the TGS.

We observe that neither the user nor any eavesdropper on the network can decrypt message `B`, since they do not know the TGS's secret key used for encryption.

Client Service Authorization.

1. To request services, the client sends the following two messages `C` and `D` to the TGS:

    c. Composed of the

        - TGT (message B), and
        - the identification number (`ID`) of the requested service.

    d. Authenticator, encrypted using the *Client/TGS Session Key*, and which is composed of the

        - client ID, and
        - timestamp

2. Once the TGS receives messages C and D, he retrieves the Client/TGS Session Key by

1. retrieving message B out of message C, and
2. decrypting message B using the TGS secret key.

3. The TGS

    1. retrieves the client ID and its timestamp by decrypting message D (Authenticator) using the Client/TGS Session Key, and
    2. sends the following two messages E and F to the client:

        e. Client/Server Session Key, encrypted using the Client/TGS Session Key.

        f. Client-to-Server ticket, encrypted using the **Service Server** 's (SS) secret key, which includes the

            - client ID,
            - client network address,
            - ticket validity duration, and
            - Client/Server Session Key.

At this point, after the client receives messages E and F from the TGS, the client has enough information to authenticate herself to the SS.

Client Service Request.

1. The client connects to the SS and sends the following two messages F and G:

    f. from the previous step (the Client-to-Server ticket, encrypted using the SS's s
    g. a new Authenticator, encrypted using the Client/Server Session Key, and which i

        - client ID, and
        - timestamp.

2. The SS

    1. retrieves the Client/Server Session Key by decrypting the ticket using his own secret key,
    2. retrieves the Authenticator by decrypting it using the session key, and

3. authenticates to the client by sending the following message H to the client :

    h. Timestamp found in the client's Authenticator plus 1 , encrypted using the Client/Server Session Key.

3. The client checks whether the timestamp is correctly updated by decrypting Message H using the Client/Server Session Key. If so, then the client can trust the server and can start issuing service requests to the server.
4. The server provides the requested services to the client.

Self-Check Questions.

1. What does FIDO2 achieve? *Standardizes authentication on the Internet by what one has, such as a USB key or smart phone, instead of what one knows, such as a password.*
2. Which parties are involved in establishing a Kerberos ticket granting ticket? *The client, the authentication server and the ticket granting server.*
3. What is the single point of failure in a Kerberos network? *The key distribution center.*

11.5 Smart cards

A smart card has the form of a credit card, but contains a microprocessor to securely store and process information. (In contrast, a magnetic-stripe card only stores little information (around < 100 bytes) but cannot process it.) Security algorithms are implemented on the card so that only properly authorized parties can access and change this data after they have authenticated themselves.

History. The first smart card was invented in 1973. While it initally consisted only of read and write memory, a microprocessor was added in 1976. In 1981, the French state telephone company introduced it to store a pre-paid credit, which was reduced when calls were made. As memory capacity, computing power, and data encryption capabilities of the microprocessor increased, smart cards are graudally replacing cash, credit or debit cards, health records, and keys.

Components.

- Random-Access Memory (RAM) reads and writes data, but only stores information as long as there is electricity and not permanently.
- Read-Only Memory (ROM): No information can be written, but it can be stored permanently. The Operating System and encryption algorithms are stored. For improved security, the ROM is buried in lower layers of silicon.
- Electrically Erasable Programmable Read Only Memory (EEPROM) stores information permanently, but is slow and one can only read/write to it a limited number of times (around 100 000 times). Typically, a smart card has 8K – 128 kByte of EEPROM. For improved security, the EEPROM is shielded in a metal coating.
- The Processor used to be an 8-bit microcontroller, but increasingly more powerful 16 and 32-bit chips are being used. A coprocessor is often included to improve the speed of encryption computations.

Most of the processing in smart cards is dedicated to cryptographic operations; in particular, encryption between on-chip components To function, a smart card needs external power and clock signal provided through contact with a smart card reader (that usually can write as well). The operating system on most smart cards implements a standard set of control commands such as those standardized in ISO 7816 or CEN 726.

There is a single Input/Output port controlled by small data packets called APDUs (Application Protocol Data Units). Because data flows only at around 9600 bits per second and half-duplex, that is, the data can either flow from the reader to the card or from the card to the reader, but not simultaneously in both directions, the smart card can only be read out slowly, thus complicating attacks. The reader sends a command to the smart card, the card executes the command and returns the result (if any) to the reader; then waits for the next command. The smart card and the reader authenticate each other by a challenge-response protocol using a symmetric key encryption:

1. The card generates a random number and sends it to the reader,
2. The reader encrypts the number with a shared encryption key and returns it to the card.
3. The card compares the returned result with its own encryption.

(and possibly with interchanged roles). Once mutually authenticated, each exchanged message is verified by a message authentication code (HMAC) which is calculated using as input the message, encryption key, and a random number.

SIM cards. For example, the UICC (Universal Integrated Circuit Card) or Universal Subscriber-Identity Module (USIM) is a smart card used in mobile phones in GSM and UMTS networks. It ensures the integrity and security of all kinds of personal data, and it typically holds a few hundred kilobytes. The Subscriber Identification Modules (SIM) is an application in the UICC that stores the subscriber's information to authenticate her with the network:

- mobile phone service subscriber's identifying key,
- the subscriber's subscription information, and
- subscriber's preferences, text messages, contacts, and last dialed numbers; therefore has a lager EEPROM than credit cards.

Advantages. Comparing authentication by what one has (for example, a smart card) to

- what one knows (for example, a password), the advantages are:
  - cryptographic keys do not need to be remembered and thus can be arbitrarily complex;
  - cryptographic keys cannot be acquired over distance;
- to what one is (for example, the fingerprint), the advantages are:
  - less sophisticated and expensive hardware (such as an iris scanner), and
  - cryptographic keys are stored securely on a device; thus avoids forgeries with fingerprints or vein scanners; In particular, storing **keys** on a **smartcard**, that is accessed by a USB reader with its own keyboard, instead of a digital file has the advantage that reading the keys from a smart card:
    * is much more difficult than from a file (stored, say, on a USB stick or hard disk), and

* leaves fewer traces: it never reveals the key, but provides only the information derived from it that was asked for, and it is immune to keyloggers that record keystrokes.

Self-Check Questions.

1. List advantages of authentication by what one knows over what one has:
   - *does not have to be carried around,*
   - *it is transparently stored,*
   - *cannot be lost, stolen or extorted,*
2. List disadvantages of authentication by what one knows over what one has or is:
   - *must not be forgotten, and therefore is limited in its complexity,*
   - *can be acquired over distance.*

11.6 Identity and Anonymity

Anonymity:. Anonymity comes from the Greek word *anonymía*, "name-less", and means "namelessness". Colloquially, it means that a person's identity is unknown. More abstractly, that an element (for example, a person or a computer) within a set (for example, a group or network) is unidentifiable (within this set).

Surfing. Protecting one's identity from being disclosed is not only necessary for someone who acts against the law, for example, when attempting to exploit a computer in a network, but also as a precaution to a possible abuse; for example, to amass data of the user's online habits to build a profile for targeted advertising.

Countermeasures are

- a proxy server between the user and the Internet, that, among other taks such as caching frequently used data and restricting access to some users, can hide IP addresses (which uniquely identifies a computer in a network),

- the Tor project which encrypts all the user's traffic and routes it through many relays, which do not know of each other,
- Virtual Private Networks, such as OpenVPN or IPsec, which encrypts all the user's traffic and routes it towards a central server.

However, these countermeasures involve inconveniences, such as an involved setup and slower data transfer. As a less compromising practical measure is to adapt best practices for protecting one's privacy on the world wide web, for example, by using the `Firefox` browser with add-ons that filter out

- trackers (for example, by cookies),
- referrers (the URL of the previous webpage from which a link was followed), and
- requests to centralized content delivery networks (CDNs),

such as

- `uBlock Origin`,
- `Privacy Badger`
- `Don't track me Google`, and
- `Decentraleyes`.

`Datasets.` Also, there is a need for anonymised datasets, for example:

- for hospitals to release patient information for medical research but withhold personal information.
- to verify whether a password has leaked, for example, by the `I Been Pwned` web form by Troy Hunt that contains over half a billion leaked passwords. This prevents Credential Stuffing where usernames and passwords from compromised websites are entered into website forms until compromised user accounts are found; an attack likely to succeed, because
  - users use the same password on different websites.
  - websites often limiting the number of login tries by a challenge login request, potentially enabling brute force attacks by using common passwords for a given user,
  - some websites do not hash passwords,

– others do, but their database could leak and the password hashes be reversed offline by GPUs or FPGAs using a dictionary of passwords (especially fast for the MD and SHA family of hash algorithms, less so for the intentionally slow ones like Argon2, PBKDF2 and BCrypt),

For example, the `I Been Pwned` web form transmits only the first five digits of the `SHA-1` of the password to compare, thus leaking little personal information. This achieves what is generally called *k-Anonymity* where a data set has for each record $k-1$ identical copies.

Identity theft:. Identity, from Late Latin identitas, from Latin idem, same, and entitas, entity, are the characteristics by which someone or something is uniquely recognizable, that is, held by no other person or thing. Identity theft is the assumption of another person's identity, usually another person's name or other personal information such as her PINs, social security numbers, driver's license number or banking details, to commit fraud, for example, to open a fraudulent bank or credit card account to receive loans. With the advent of digitalized records and (the anonymity on) the Internet, identity theft has become more and more common.

For example, in **SIM-card swapping**, the attacker obtains a victim's mobile-phone number to assume, temporarily, her online identity. The attacker initially obtains personal data about the victim, usually her name, mobile phone number and postal address. He then exploits that mobile operators usually offer their customers a new SIM card, for example, after the phone is lost, onto which the previous phone number is transferred. The attacker now pretends to be the actual customer to the mobile phone operator, for example, by phone in the customer service center (where often the date of birth and postal address suffice to identify oneself if no customer password was agreed on signing the contract). For example, it suffices to know the mobile phone number to reset the password of an Instagram account.

Self-Check Questions.

1. What does anonymity of a computer in a network mean? *That the identity of the computer in a network is unknown.*

2. How does the Tor network achieve anonymity? *By onion routing in which every node only knows its predecessor and successor, and in which all traffic between both endpoints is indecipherable to every node but the endpoints.*

Summary

Identification is telling a computer who the user is, usually by her user (or account) name. This is followed by *authentication*, the verification of the user's identity, that is, convincing a computer that a person is who he or she claims to be. To authenticate, the user prove her identity by information that

- only she *knows* such as a password,
- only she *has*, hardware such as a smart card,
- only she is described by (what she *is*), such as biometric identifiers (fingerprint, ...).

Each method with its proper advantages and disadvantages. In particular, passwords, the most common method, have to be memorizable, which in practice weakens them. The FIDO2 standard aims at replacing (or at least complementing) them by hardware and biometric authentication.

Instead of revealing the secret itself when authenticating, it is safer to prove only its knowledge. In a challenge-response authentication protocol, the successful response to the challenge requires its knowledge (such as encryption and decryption of random data by the secret key). In a zero-knowledge protocol, in contrast to a challenge-response protocol, no information whatsoever can be won on the secret key provided the computation of a mathematical function is assumed infeasible.

The Kerberos protocol mediates between users and servers by a central server that stores symmetric keys of all parties; then instead of the parties mutually proving the knowledge of their symmetric keys, it creates for each correspondence a session key of limited validity.

Questions

- Which protocol is not a challenge-response authentication protocol? Digest-MD5, CRAM, TAN, SCRAM

- On the computational difficulty of which discrete mathematical function does Schnorr's protocol rely on? logarithm, quadratic root, exponential, square?
- Which data does *not* enter the computation of the salted password in SCRAM? IterationCount, password, salt, current time
- How many minutes difference does Kerberos allow for between the different authenticating parties times? 1,2,5,10
- How many times does a client encrypt a TCP packet before sending it to the server on the Tor network? as many as there are nodes, twice as many, once, twice.

Required Reading

Read Bentz (2019) to get an idea how Kerberos works and Quisquater et al. (1989) for zero-knowledge proofs.

Further Reading

Read Schnorr (1991) to understand a basic zero-knowledge protocol.

Have a look at Menon-Sen et al. (2010) to get acquainted with the format of Request for Comments, in particular, thoroughly understand the SCRAM protocol.

12 Cryptanalysis — how to break encryption

Study Goals

On completion of this chapter, you will have learned ...

- ... against which cryptographic attacking scenarios to protect.
- ... which additional (physical) information leaks (such as computation time) can be exploited (side-channel attacks).
- ... how to estimate how long an exhaustive key-search (brute-force attack) takes.
- ... how to speed up such an attack by testing likely candidates first (rainbow tables).
- ... how to break monoalphabetic substitution ciphers by frequency analysis.
- ... how to break a block cipher with little diffusion by differential cryptanalysis.

Introduction

Let us recall that the prefix *Crypto-*, comes from Greek kryptós, "hidden" and *analysis* from the Greek analýein, "to untie": the breaking of ciphers, that is, recovering or forging enciphered information without knowledge of the key.

> **Cryptanalysis**: Recovering or forging enciphered information without knowledge of the key.

History delivers plenty cryptanalytic success-stories; for example, among many others, the decryption of the German rotor machine Enigma by the Polish and British forces.

- During World War I:
  - British cryptanalysis of a telegram from the German foreign minister, Arthur Zimmermann, to the German minister in Mexico City, Heinrich von Eckardt, enticing Mexico to ally with Germany, enkindling the U.S.'s entry into war on the side of the Allies.
  - French cryptanalysis of the `ADFGVX` cipher used by the German forces within a month just before the German army entered in Paris in 1918 (however, still too late to save much).
- During World War II:
  - Polish and British cryptanalysis of
    * the German rotor machine Enigma, and
    * the telegraphers Lorenz-Schlüsselmaschine and Siemens & Halske T52).
  - The U.S. Army's Signal Intelligence Service's (SIS) cryptanalysis of the Japanese rotor machines . The pivotal Battle of Midway for the naval war in the Pacific was won by awareness of the Japanese attack on the Aleutian Islands being a diversionary maneuver and the Japanese order of attack on Midway.
  - In World War II the Battle of Midway, which marked the turning point of the naval war in the Pacific, was won by the United States largely because cryptanalysis had provided Admiral Chester W. Nimitz with information about the Japanese diversionary attack on the Aleutian Islands and about the Japanese order of attack on Midway.

- During a debate over the Falkland Islands War of 1982, a member of Parliament, in a now-famous gaffe, revealed that the British were reading Argentine diplomatic ciphers with as much ease as Argentine code clerks.
- Cryptanalytic feats were achieved also by the defeated powers, but no success-stories to be told.

While we will present some established principles of cryptanalysis, in practice the cryptanalyst's intuition and ability to recognize subtle patterns in the ciphertext were paramount, but difficult to convey. Today, however, cryptanalysis is based on mathematics and put into practice by efficient use of extensive computing power.

12.1 Frequency Analysis

Cryptanalysis of single-key ciphers relies on patterns in the plaintext carrying over to the ciphertext. For example, in a monoalphabetic substitution cipher (that is, each alphabetic letter a, b, ... in the plaintext is replaced, independently of its position, by another alphabetic letter), the frequencies of the occurrences of the letters in the plaintext alphabet are the same as those in the ciphertext alphabet. This can be put to good cryptanalytic use by

1. recognizing that the cipher is a monoalphabetic substitution cipher and,
2. giving the likeliest candidates of plaintext letters.

Latin Alphabet. A substitution by any permutation of the letters of the alphabet, such as,

A	B	...	Y	Z
↓	↓	...	↓	↓
E	Z	...	G	A

has

$$26 \cdot 25 \cdots 1 = 26! > 10^{26}$$

keys, so a brute-force attack is computationally infeasible. But it violates the goals of *diffusion and confusion*: If the key (that is, the given permutation of the alphabet) exchanges the letter α for the letter β , then there's

- bad *confusion* because the substitution of β in the key implies only the substitution of each letter β in the ciphertext,
- bad *diffusion* because the substitution of a letter α in the plaintext implies only the substitution of the corresponding letter β in the ciphertext.

In fact, the algorithm allows statistical attacks on the frequency of

- letters,
- bigrams (= pairs of letters) and
- trigrams (= triples of letters).

in English For example,

- the most frequent letter in English is `e`,
- the most frequent bigram in English is `th`, and
- the most frequent trigram in English is `the`.

Thus substituting

- the most frequent letter from *ciphertext* to the most frequent letter *in English* (= `e`),
- the most frequent bigram from *coded text* to the most frequent bigram *in English* (= `th`), ...
- the most frequent trigram of the *ciphertext* to the most frequent trigram *in English* (= `the`), ...

is a good starting point to decipher the text: The more ciphertext, the more likely that this substitution coincides with that the text was enciphered by.

Example. For example, using these frequencies on the ciphertext

```
ACB ACBGA ACSBDQT
```

gives

```
THE THE** TH*E***
```

for yet to be deciphered letters marked by *. By the restrictions of English vocabulary and sentence structure, yielding

```
THE THEFT THREAPS
```

As an exercise, Dear reader, build a short English sentence with common letter frequencies, ask a friend to encrypt it and try your hands at cryptanalyzing it!

Homophones.

Homophones: Multiple cipher symbols for the same plaintext letter.

To hide the frequencies of the alphabetic letters, one approach is to use *homophones*, a number of ciphertext symbols for the same alphabetic plaintext letter, chosen in proportion to the frequency of the letter in the plaintext; for example, twice as many symbols for `E` as for `S` and so forth, so that each cipher symbol occurs on average equally often in the ciphertext. However, other frequencies in the plaintext (partially) still resist encryption (and ease cryptanalysis), such as digraphs: `TH` occurs most often, about `20` times as frequently as `HT`, and so forth.

Self-Check Questions.

1. What is a monoalphabetic substitution cipher? *Each alphabetic letter in the plaintext is replaced, independently of its position, by another alphabetic letter.*
2. Which patterns preserves a monoalphabetic substitution cipher? *Among many others, (single) letter, bigram and trigram frequencies.*
3. Which patterns preserves a substitution cipher using homophones? *The frequencies of pairs of letters (bigrams), triples (trigrams), ...*

12.2 Brute-force attacks

In practice the security of a cipher relies foremost

- on its resistance to the most efficient (known!) methods of cryptanalysis (the quest for an — overlooked?! — back door into the system), and

- the computational effort needed to check all keys (a brute-force attack). Given the time and computational resources, the right key will eventually be found: If only the ciphertext is available, say, that of a block cipher, then a brute-force attack would decrypt a block of the cipher with one key after another until a block of meaningful text was produced (although not necessarily of the original plaintext). In this case, proceed likewise for the next block.

In contrast to single-key cryptography whose cryptanalysis exploits statistical patterns, by its reliance on computationally difficult mathematical problems (that is, whose runtime grows exponentially in the bit-length of the input), the cryptanalysis of two-key cryptography is that of computational mathematics: to find an algorithm that quickly computes the solutions of the difficult mathematical problem. Ideally, one whose runtime is polynomial in the number of input bits. However, in practice, for example, for

- the prime factor decomposition used in RSA or
- the discrete logarithm in the Diffie-Hellman key exchange,

algorithms whose runtime grows slower than exponentially in the number of input bits (sub-exponential) are known, but none with polynomial runtime.

Comparison of Computational Efforts. ECC, elliptic curve cryptography, is becoming the new standard because its cryptographic problem (the logarithm over a finite elliptic curve) is (from what we know) computationally more difficult than that of RSA (the factoring in prime numbers) or DH (the logarithm over the multiplicative group of a finite field). Therefore, small keys for the ECC achieve the same level of security as large keys for the RSA or DH. As an example, the security of a 224 bits key from the ECC corresponds to that of a 2048 bits key from the RSA or DH. This factor in reducing key sizes corresponds to a similar factor in reducing computational costs. Regarding usability,

- an ECC public key can be shared by spelling it out (it has 56 letters in hexadecimal notation,
- while a public key for RSA or DH has to be shared in a file (which is for convenience referred to by a fingerprint).

Let us compare the cryptographic problem behind ECC to that of RSA and DH:

Exponential and Generic Logarithm. Both groups, the multiplicative group of a finite field and the group of a finite elliptic curve, are *finite cyclic* groups, that is, *generated* by an element g of *finite* order n . Mathematically speaking, they are equal to a group of type

$$\mathrm{G} = \langle g \rangle = \{g^0, g^1, g^2, \ldots, g^{n-1}\} \simeq \{0, 1, 2, \ldots, n-1\} = \mathbb{Z}/n\mathbb{Z}$$

However, one way of this identification, from $\mathbb{Z}/n\mathbb{Z}$ to G , is a lot faster than the other way around:

Exponential. Given a generator g in G , the identification of $\mathbb{Z}/n\mathbb{Z}$ with G for $x \mapsto g^x$, the *exponential*

$$\exp\colon \mathbb{Z}/n\mathbb{Z} \to \mathrm{G}$$

is quickly computed in *every* commutative group G : Given d in $1, \ldots, n-1$, to calculate g^d, expand d in binary base

$$d = d_0 + 2d_1 + 2^2 d_2 + \cdots + 2^m d_m$$

for d_0, …, d_m in $0, 1$ and calculates by multiplying by 2 successively g^2 , …, g^{2^m}}. So

$$g^d = g^{d_0 + 2d_1 + 2^2 d_2 + \cdots + 2^m d_m} = g^{d_0} g^{2d_1} g^{2d_2} \cdots g^{2^m d_m}.$$

That is, we calculate g^d in $\leq 2\log_2 n$ group operations.

Example. For G the group of points of an elliptic curve, P in G , and d in $\mathbb{N}$ we calculate successively 2P , $2^2 = 2 \cdot \mathrm{P}$, $2^3\mathrm{P} = 2 \cdot 2 \cdot 2\mathrm{P}$. Let i_0, i_1, … be the indices of the binary digits … that are different from 0 . So

$$d\mathrm{P} = 2^{i_0}\mathrm{P} + 2^{i_1}\mathrm{P} + \cdots$$

Generic Logarithm. Given a g generator in G , the computation of the reverse identification, the *logarithm*

$$\log\colon \mathrm{G} \to \mathbb{F}_p,$$

that is, given y in G , calculate x in $0, \ldots, n-1$ such that $y = g^x$ is usually hard: By Shoup's theorem in Shoup (1997), every *generic* algorithm, that is, using

only the operations of the group, takes at least $n^{1/2}$ operations (of the group) to calculate the logarithm.

A generic algorithm that achieves this speed (except a 2 factor) is the `Baby Step, Giant Step` (or Shanks algorithm): Given h in $\langle g \rangle$, to calculate d in $\{0,1,...,n-1\}$ such that $h = g^d$:

1. Put $m := \lceil \sqrt{n-1} \rceil$ (where $\lceil \cdot \rceil$ indicates the smallest integer above or equal to the real number $\cdot$).
2. Calculate $\alpha_0 := g^0$, $\alpha_1 := g^m$, $\alpha_2 := g^{2m}$, ..., $\alpha_{m-1} := g^{(m-1)m}\}$. (The giant step)
3. Calculate $\beta_0 := h$, $\beta_1 := hg^{-1}$, $\beta_2 := hg^{-2}$, ..., $hg^{-(m-1)}$ until you find some β_i that equals some α_j from the above list. (The baby step)
4. If $\beta_i = \alpha_j$, then the result is $d = jm + i$.

This algorithm works, because:

- Every d in $\{0,...,m^2-1\}$ is by division with remainder by m of the form $d = qm + r$ for $q, r < m$; so every h element in $\langle g \rangle$ is of the form $g^{qm+r} = h$;
- If $\beta_i = \alpha_j$, then $g^m = hg^{-i}$, hence $g^{jm+i} = h$.

Note. For elliptic curves, Pollard's ρ -algorithm is slightly faster.

Specific Logarithms. This estimation of the operations necessary to compute the logarithm of a group, that is, which uses only the operations of the group, applies to *generic* algorithms. However, *specific* algorithms, that is, those that use group-specific properties (such as the units of a finite field or that of an elliptic curve), may use less.

For example, for the multiplicative group of a finite field, there are indeed faster algorithms (called *sub-exponential*) to calculate the prime factors of a product and the logarithm. They are based on *Index Calculus* which makes use of the properties of this specific group. For example, for the cryptographic problems of `RSA` and `Diffie-Hellman` on the multiplicative group of a finite field, that is,

- or the factoring of an integer in its prime factors,
- or the logarithm of the multiplicative group of a finite field,

there are indeed faster algorithms (called *sub-exponential*). They are based on *Index Calculus* which makes use of the properties of this specific group. The fastest known algorithm is the *General Number Field Sieve*; see A. K. Lenstra et al. (1993) for the computation of the prime factors and Gordon (1993) for that of the logarithm.

Its *complexity* is *sub-exponential*, roughly for large n , the number of group operations is

$$2^{n^{1/3}(\mathrm{C}+o(1))(\log n)^{2/3}}$$

where $\mathrm{C} = 64/9$ and $o(1)$ means a function f over $\mathbb{N}$ such that $f(n) \to 0$ to $n\infty$.

In contrast, all known algorithms for the ECC cryptographic problem, that is, calculating the logarithm over a finite elliptic curve, are generic algorithms. For these generic algorithms, by Shoup's Theorem, the complexity $2^{n/2}$ in the number of bits n of input is as small as possible. The fastest algorithm at present is Pollard's ρ -algorithm, which has a roughly *exponential* complexity of

$$2^{n/2+\mathrm{C}}$$

where $\mathrm{C} = \log_2(\pi/4)^{1/2} \simeq -0.175$.

Minimal Key Length. World's fastest supercomputer, IBM's Summit (taking up 520 square meters in the Oak Ridge National Laboratory, Tennessee, USA) has around 150 petaflops, that is, $1.5 \cdot 10^{17}$ floating point operations per second. The number of flops needed to check a key depends for example, on whether the plaintext is known or not, but can be very optimistically assumed to be 1000. Therefore, Summit can check approximately $1.5 \cdot 10^3$ keys per second, and, a year having $365 \cdot 24 \cdot 60 \cdot 60 = 31536000 \approx 3 \cdot 10^8$ seconds, approximately $4.5 \cdot 10^{11}$ keys a year.

To counter the increasing computing power, one prudently applies a Moore's Law that stipulates that computing power doubles every other year. Therefore, every twenty years computing power increases by a factor $2^{10} = 1024 \approx 10^3$. Therefore, to ensure that in, say, sixty years, a key not surely be found during a yearlong search by world's fastest supercomputer at least $4.5 \cdot 10^{20}$ key combinations have to be used.

For a key of bit length n, the number of all possible keys is 2^n. If $n = 80$, then there are are $2^{80} \approx 1.2 \cdot 10^{24}$ possible key combinations. While this number is

sufficient for now, the probability for the key to be found during a yearlong search by world's fastest supercomputer being around $1/250$, the projected fastest super computer in twenty years will likely find it in half a year. Instead, to be safe in 40 years, a minimal key length of 112 is recommended.

Comparison of Key Sizes. This Table from A. Lenstra and Verheul (2001) compares the key sizes in bits to a security level comparable between

- a symmetrical algorithm like the AES,
- an asymmetric algorithm by elliptic curves, and
- an asymmetric algorithm such as Diffie-Hellman or RSA.

Symmetric Key	Asymmetric Elliptic Key	Classic Asymmetric Key
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

The numbers in the table are estimated by the fastest known algorithm to solve the cryptographic problem: Given an input key with n bits,

- for the symmetric algorithm AES , the fastest known algorithm is to try out all possible keys, whose complexity (= the number of operations) is 2^n,
- for the logarithm over a finite elliptic curve, the fastest algorithm today is the generic baby step, giant step algorithm (or, slightly faster, Pollard's ρ -algorithm) whose complexity is roughly $2^{\sqrt{n}}$, and
- for classic asymmetric algorithms, either RSA or Diffie-Hellman , on a finite field, the fastest algorithm is the *General number field sieve* whose complexity, roughly, for large n is $2^{2\sqrt[3]{n}\log(n)^{2/3}}$.

In practice, the smaller ECC keys speed up cryptographic operations by a factor of > 5 compared to RSA and Diffie-Hellman (in addition to facilitating their exchange among people and saving bandwidth). However, there are also disadvantages to ECC compared to RSA, for example: its signature algorithm depends on the generation of an additional ephemeral key pair by a random

number generator that, if its output is predictable or repetitive, reveals the private signature key!

- ECC is newer, so:
  - less proven than the RSA, and
  - some implementations are still patented (for example, by Certicom).

Self-Check Questions.

1. Which minimal key size is currently recommend as secure for RSA and Diffie-Hellman?
   - ☐ 512 bits
   - ☐ 1024 bits
   - ☐ *2048 bits*
   - ☐ 4096 bits
2. Which minimal key size is currently recommend as secure for Elliptic Curve Cryptography?
   - ☐ 128 bits
   - ☐ *256 bits*
   - ☐ 512 bits
   - ☐ 1024 bits
3. Which minimal key size is currently recommend as secure for AES?
   - ☐ *112*
   - ☐ 128 bits
   - ☐ 256 bits
   - ☐ 1024 bits

12.3 Rainbow Tables

Rainbow tables are a method of password cracking that compares a password hash to precomputed hashes of the most likely passwords; a time-memory trade-off: more memory for less computation.

Lookup Table as Time-Memory trade-off. A lookup table is a data structure, usually an array, used to replace a runtime computation by an array indexing operation. That is, a value is first computed and then looked up to save runtime, because retrieving a value from memory is usually faster than computing it. For example, values of a common mathematical function, say the logarithm, can be precomputed to look up a nearby value be from memory.

Rainbow Tables. A rainbow table is a table of cryptographic hashes of the most common passwords. Thus, more likely passwords are revealed sooner. The generation of the table depends on

- the character set used, the password length, the number of table entries and so forth.
- the cryptographic hash functions.

Common cryptographic hash algorithms such as `MD4/5`, `SHA` ... are *fast*; thus they are unsuitable for password creation because they are vulnerable to brute-force attacks. For example, `MD5` as a cryptographic hash function is designed to be fast and thus lends itself towards a rainbow table attack, while hash functions such as `PBKDF1`, `PBKDF2`, `bcrypt`, `scrypt` and the recent `Argon2` were designed to prevent this kind of attack by being:

- deliberately *slow*, such as `bcrypt`,
- deliberately *memory* hungry, such as `scrypt`.

A rainbow attack can however most effectively be prevented by making the used hash function unique for each password. This is achieved by adding a `salt`, an additional unique, usually random, argument.

Meet-in-the-Middle Attack on DES. The key size of the symmetric industry standard cryptographic algorithm `DES` was merely `56` bits, so little that it ceded to brute-force attacks shortly after its vetting. Therefore, a twofold encryption for two different keys was thought to effectively double the key size to `112` bits. However, the meet-in-the-middle attack by Diffie and Hellman trades off memory for time to find the key in only 2^{n+1} encryptions (using around 2^n stored keys) instead of the expected 2^{2n} encryptions: Assume the attacker knows a plaintext P and its ciphertext C , that is,

$$C = E_{K''}(E_{K'}(P)),$$

where E denotes the encryption using K′ respectively K″ . The attacker:

1. computes $E_K(P)$ for all possible keys K and stores the results in memory.
2. decrypts the ciphertext by computing $D_K(C)$ for every K , and
3. looks for matches between these two sets, whose keys likely match those used to encrypt P to C .

Therefore, triple encryption, 3DES was necessary to effectively double the key size and harden decryption for the computing power to come.

Self-Check Questions.

1. Why is encrypting twice for different keys less secure than encrypting thrice? *Because the meet-in-the-middle attack uses a memory-time trade-off to find the used key of n bits in only* 2^{n+1} *encryptions.*

12.4 Known/Chosen Plain/Ciphertexts

Asymmetric cryptography uses mathematical methods, more exactly modular arithmetic, to encipher. The security, the difficulty of deciphering, of asymmetric cryptography is based on computational mathematical problems that have been recognized as difficult for centuries. Symmetric cryptography (like hash functions) uses more artisanal methods of ciphering, which aim to maximize diffusion and confusion, mainly by iterated substitution and permutation. The security of symmetric cryptography is simply based on its resistance to years of ongoing attacks. That is, it is satisfactory from a practical standpoint, but less so regarding the struggle for eternal truths.

Perfect Security.

Plaintexts generally do not occur with the same probability. It depends, for example, on the language, jargon or protocol used. A cipher is **perfectly secure** if none of its ciphertext reveals anything about the corresponding plaintext. That is, the probability that a plaintext and a key resulted in a given ciphertext is the same for all plaintexts and all keys.

> **perfect security**: the probability that a plaintext and a key resulted in a given ciphertext is the same for all plaintexts and all keys.

More exactly: A cipher is called *perfectly secure* if, for every plaintext, its probability is (stochastically) independent of any ciphertext. Let p denote a plaintext and by $P(p)$ its probability. In formulas, for every plaintext p and every ciphertext c, we have $P(p|c) = P(p)$. In practice, this means that if an attacker intercepts a ciphertext c, then he has no advantage, that is, his probability of knowing the plaintext is the same as if he does not know c.

In 1949, Shannon proved the following theorem on the conditions for a cipher to be perfectly secure: Given a finite number of keys and plaintexts with positive probabilities, that is, $P(p) > 0$ for every plaintext p. The cipher is *perfectly secure*, if

- the probability distribution is *uniform*, that is, all probabilities are equal, and
- for each plaintext p and every ciphertext c, there is a unique key k to get c from p.

That is, statistical deviations tend to weaken the cipher. In particular, it is important to use a completely random number generator for the keys.

One-time pad. The only perfectly secure cipher is the one-time pad where a key (of the same size as the plaintext) is added (bit by bit) to the plaintext. Such a perfectly secure cipher is however impractical. For real-time applications, such as on the Internet, it is little used.

The **One-time pad** adds (by the XOR operation) each bit of the plaintext t with the (positionally) corresponding bit of a key c that

- is of the same length, and
- is discarded after use, that is, it will not be used to encrypt other plaintexts.

> **One-time pad**: The key is as long as the plaintext and they are added letter by letter (or bit by bit) to obtain the ciphertext.

That is, the ciphertext $T = (T_1, T_2, \ldots)$ is

$$T = t \oplus c = (t_1 \oplus c_1, t_2 \oplus c_2, \ldots).$$

This cipher is as safe as theoretically possible!

If the plaintext has a single block t , then this simple (XOR) addition of a key, the one-time pad , is a secure algorithm. However, it is often inconvenient or even close-to impossible to have a key as large as the plaintext: For example,

- to encrypt a hard drive, you need another one of the same size to store the key, and
- to encrypt communication over a (for example, wireless) network, you would need to know already *before* how much text will be transferred. to encrypt communication over a network, it must be known *beforehand* how much data will be transferred.

In practice, imagine an agent duplicating gigabytes of noise on two storage media, for example, a hard disk and a flash drive, and taking one of these media to encrypt his communication by the one-time pad.

Unfortunately, it is a bad idea (though natural) to use the same key for two different blocks: if, for example, the plaintext has *two* blocks b' and b'' , then, with this algorithm, the sum (XOR)

$$(b' \oplus c) \oplus (b'' \oplus c) = b' \oplus b''$$

of the two cipher blocks $b' \oplus c$ and $b'' \oplus c$ equals the sum $b' \oplus b''$ of the two clear blocks (because the addition XOR is by definition *auto-inverse*, that is $x \oplus x = 0$ regardless of whether the binary digit is $x = 0$ or $x = 1$)!

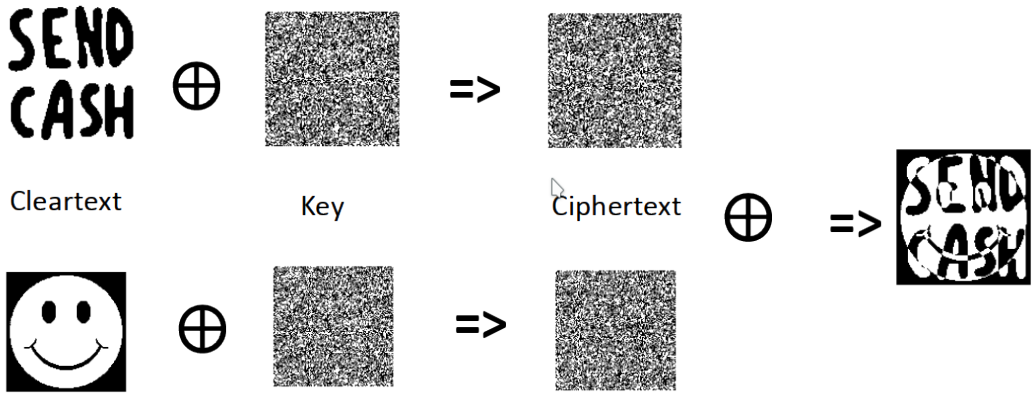


Figure 59: The sum of two plaintexts may reveal each one of them! (R. Smith (2008))

It can be seen as the ciphering of the first block by one-time pad whose key is the second block. Unfortunately, the second block is not a good key, because

far from being random; on the contrary, usually its content is similar to that of the first block, that is, the key is predictable.

Proven Security. As perfect security is unfeasible, security is demonstrated

- or by resistance against known attacks,
- or by reducing the computational difficulty to that of a (computational mathematical) problem recognized difficult (called *proven* security).

Although there are provenly secure symmetric ciphers the most efficient and widely used algorithms, such as AES, prove their resistance only against known attacks, such as those of differential or linear cryptanalysis.

Formally Proven Security. The mathematical problems on which the difficulty of the decryption in asymmetric cryptographic algorithms are based, are all NP-complete, that is, its solutions are verifiable in polynomial runtime (in the bit-length of the input) and all other such problems can be reduced to it. That is, every cryptographic algorithm (enciphers or) deciphers a message *with* the key in polynomial runtime in the bit-length of the key.

In contrast, all known algorithms for deciphering *without* the key take exponential time in the bit-length of the key. By the P-versus-NP conjecture, there is no algorithm that takes polynomial runtime (in the bit-length of the key). For now, the conjecture being unresolved, there may theoretically exist polynomial algorithms for deciphering *without* the key in polynomial runtime; however, after decades of continuous vain efforts by the community of cryptanalysts, it is assumed unlikely.

Example. The initial example of such a provenly secure cipher was semantic security (in Goldwasser and Micali (1984)) which reduces the difficulty of decipherment to that of the computation of the *Quadratic Residue*: Given x and N a product of two primes, it is difficult to determine whether x is quadratic modulo N (that is, whether there is y such that $x = y^2$ mod N or not) if, and only if, the so-called *Jacobi symbol* for x is +1 and the prime factors of N are unknown.

This (Goldwasser-Micali) cipher consists of:

- a *key generation* algorithm that produces
  - the private key as two primes p and q , and
  - the public key $N = pq$ and a number x which is quadratic neither modulo p nor modulo q (such that Jacobi's symbol of x for N is +1 , and for both his factors p and q is −1). For example, if $p, q \equiv 3 \mod 4$, then $x = N - 1$ will do.
- a *probabilistic encipherment* algorithm: If $m = (m_1, m_2, ...)$ are the bits of the plaintext, then numbers y_1 , y_2, ... that are indivisible by p and q are generated and the enciphered message is $M = (M_1, M_2, ...)$ with $M_1 = y_1^2 x^{m_1}$, $M_2 = y_2^2 x^{m_2}$, ...
- a deterministic *decipherment* algorithm: If $M = (M_1, M_2, ...)$ is the enciphered message, then $m_1 = 0$ if, and only if, M_1 is quadratic modulo N , ... which is quickly determined by knowledge of both factors p and q of N .

Paradox. *Caution:* Theoretical security remains an insufficient idealization for reality: For example, Ajtai and Dwork (1999) presented a cipher and proved it theoretically secure; however, it was broken a year later. *Proven* does not mean *true:* a provenly secure system is not necessarily truly secure, because the proof is made in a formal model which assumes

- certain operation principles, attackers and a set security objective and
- difficulty of the problem to which the proof is reduced.

For example,

- The implemented cipher differs from the formal cipher.
- A partial objective is already sufficient for the attacker: If, for example, the security objective is that the attacker does not derive the entire plaintext from the ciphertext, then it may already be enough for him to derive a passage of the plaintext.

Besides, the proof may be wrong! Despite this uncertainty, a proof of security is a useful criterion (though theoretically necessary, but practically insufficient) for the security of a cipher.

Attacking Scenarios. What does security mean? The criterion that the attacker cannot derive the plaintext from the ciphertext is insufficient, because he could acquire other useful (partial) information about the plaintext. But even the impossibility to derive useful information on the plaintext is insufficient in some circumstances: If

- the public-key encryption is deterministic (that is, if the same input always returns the same output, as is the case with RSA encryption as implemented in a textbook) and
- the attacker can limit the number of possible plaintexts (he knows, for example, that the ciphertext is "yes" or "no"),

then he can encrypt all these possible plaintexts with the public key and compare the ciphertexts to the encrypted texts. For an asymmetric algorithm, the attacker should be assumed to know the public key. Thus, he can encrypt any plaintext of his choice and compare it to the ciphertext; that is, he can mount a Chosen-Plaintext Attack (**CPA**).

Generally, the attacking scenarios are categorized by how much the cryptanalyst's knows about the ciphertext (ordered below from less to more knowledge):

- ciphertext only,
- probable or known ciphertext/plaintext pairs, and
- chosen plaintext or chosen ciphertext.

For example, to break a monoalphabetic cipher, the ciphertext alone usually suffices thanks to frequency analysis. But often the cryptanalyst either will know or can guess some of the plaintext, such as a preamble of a letter (like a formal greeting) or a computer file format (like an identifier). Lastly, most opportunely, he can ask the sender to encrypt a plaintext that he chose or the recipient to decrypt a ciphertext that he chose.

Ciphertext-Only Attack. The attacker has the ciphertexts of several messages that were encrypted by the same algorithm. His task is to recover as much plaintext as possible or, better, to recover the (algorithms and) keys that were used.

Probable-Plaintext Attack. The attacker has the ciphertext and suspects that the plaintext contains certain words (a *crib*) or even whole sentences. His task is to recover as much plaintexts as possible or, better, to recover the (algorithms and) keys that were used. For example, Enigma, the cryptographic electromechanical rotor-machine used by the power axes in World War II, was broken by the repetitiveness of the messages it enciphered: for example, the weather report was sent on a daily basis and announced as such (`Wetterbericht` in German) at the beginning of every such message.

crib: a text probably contained in the plaintext of a given ciphertext.

Known-Plaintext Attack KPO. The attacker has a ciphertext and the corresponding plaintext. His task is to recover the (algorithm and) key that was used. (For example, linear cryptanalysis falls into this scenario). For example, an attack from 2006 on the Wired Equivalent Privacy (`WEP`) protocol for encrypting a wireless local area network exploits the predictability of parts of the encrypted messages, namely the headers of the `802.11` protocol.

Chosen (or Adaptive) Plaintext Attack CPA. The attacker has the ciphertexts of the plaintexts that he can freely choose; so that the attacker can freely *adapt* the plaintext depending on the text obtained after each decipherment and analyze the resulting changes in the ciphertext. His task is to recover the (algorithm and) key that was used. (For example, differential cryptanalysis falls into this scenario).

This is the minimal attacking scenario to be prepared against for asymmetric cryptography! Since the encryption key is public, the attacker can encrypt messages at will. Therefore, if the attacker can reduce the number of possible plaintexts, for example, if he knows that they are either “Yes” or “No”, then he can encrypt all possible plaintexts by the public key and compare them with the intercepted ciphertext. For example, the `RSA` algorithm in its textbook form suffers from this attack. Therefore, to protect against this `CPA` attack, every implementation of this algorithm must pad the plaintext with random data before encryption.

Chosen (or Adaptive) Ciphertext Attack CCA. The attacker has a ciphertext c and the plaintexts of the ciphertexts (except c) that he can freely choose; so that the attacker can freely *adapt* the plaintext depending on the text obtained after each decipherment and analyze the resulting changes in the ciphertext. His task is to recover the (algorithm and) key that was used. For example, the attacker has to analyze a cipher machine black-box, that is, whose inner workings are unknown.

Few practical attacks fall into this scenario, but it is important for proofs of security: If resistance against the attacks of this scenario can be proven, then resistance against every realistic attack of chosen ciphertext is granted.

Semantic Security. If an asymmetric algorithm is used, then the attacker should be assumed to know the public key. Thus, he can encrypt any plaintext of his choice and compare it to the ciphertext. That is, he can mount a chosen-plaintext attack (CPA).

A cipher is secure against IND-CPA (*indistinguishability of the ciphertext for chosen plaintexts*), if no attacker can distinguish which one of two plaintexts, that he selected before, corresponds to the ciphertext that he receives afterwards. More exactly, the cipher is **indistinguishable under chosen-plaintext attack** if every probabilistic polynomial-time attacker has only an insignificant "advantage" over random guessing:

> **IND-CPA-secure**: no attacker has a probability significantly higher than $1/2$ to distinguish two ciphertexts.

The four steps of the game IND-CPA with polynomial-runtime restriction (in the bit-length of the key k) on the attacker's computations (carried out on creating the two plaintexts, step two, and on choosing the plaintext that corresponds to the ciphertext, step four):

1. A pair of keys is created, one secret and one public, both with k bits. The attacker receives the public key.
2. The attacker computes two plaintexts M_0 and M_1 of the same size.
3. The cipher machine
    1. randomly chooses a bit b in $\{0,1\}$
    2. enciphers M_b, and

3. passes the ciphertext to the attacker.

4. The attacker chooses a bit b' in $\{0,1\}$.

The attacker who chooses the bit b' in the fourth step randomly is right with a probability of $1/2$. A cipher is **IND-CPA-secure** if no attacker has a probability of success $\mathrm{P}(b = b')$ significantly higher than $1/2$: That is, if every attacker's difference $\mathrm{P}(b = b') - 1/2 = \epsilon(k)$ is *insignificant*, that is: for every (nonzero) polynomial function p there is k_0 such that $\epsilon(k) < 1/p(k)$ for every $k > k_0$.

An insignificant difference should be granted, because the attacker easily increases his probability of success above $1/2$ by guessing a secret key and trying to decipher the ciphertext with it.

Observation. Although the above game is formulated for an asymmetric cipher, it can be adapted to the symmetric case by replacing the public key cipher by a cryptographic *oracle*, a black-box function, that is, whose inner workings are unknown, that retains the secret key and encrypts arbitrary plaintexts at the attacker's request.

Semantically Secure Algorithms. Secure semantic encryption algorithms include *El Gamal* and *Goldwasser-Micali* because their semantic security can be reduced to solving some difficult mathematical problem, that is, irresolvable in polynomial runtime (in the number of input bits); in these cited examples, the *Decisory Diffie Hellman Problem* and the *Quadratic Residue Problem.* Other semantically insecure algorithms, such as RSA, can be made semantically secure by random cryptographic paddings such as OAEP (Optimal Asymmetric Encryption Padding).

Example. The Elgamal encryption method is IND-CPA-secure under the assumption that the *Decisory Diffie Hellman Problem* is difficult. To prove security, let us transform

- a *winner* A from the IND-CPA game for El Gamal , that is, given the ciphertext of one among two plaintexts under the public key, she identifies the corresponding plaintext (among the two) with probability $1/2 + \epsilon$),
- into a decision maker S for DDH , that is, given a base g and exponents x , y and z in $1,\ldots,p-1$, she decides in polynomial runtime whether $g^z \equiv g^{xy} \bmod p$ or not;

as follows: Given a base g and exponents x , y and z in $1,\ldots,p-1$.

1. S simulates the creation of a key pair by giving g^x as public key to A (but not knowing the corresponding secret key).
2. A produces two plaintexts m_0 and m_1.
3. S simulates the encipherment by randomly choosing a bit b and defining the ciphertext as $g^y, g^z m_b$.
4. A decides whether the plaintext is m_0 or m_1.

- If $g^z = g^y$, then the cipher is indistinguishable from a normal cipher and A wins with probability $1/2 + \epsilon$.
- If g^z is random, then A just guesses and wins with probability $1/2$.

S 's strategy is therefore to opt for $g^z = g^y$ if, and only if, A is correct. Thus, the probability that S is correct is $1/2 + \epsilon/2$.

Semantic Security IND-CCA. Let us recall that:

- A cipher is secure against IND-CPA (*indistinguishability of ciphertext for chosen plaintexts*), if no attacker can distinguish which one of two plaintexts, that he selected before, corresponds to the ciphertext that he receives afterwards.
- In the *chosen* (or, more exactly, *adaptive*) ciphertext attack CCA the attacker has a ciphertext c and the plaintexts of the ciphertexts (except c) that he can freely choose;

A cipher is secure against IND-CCA (indistinguishability of ciphertext for chosen *ciphertexts*), if the attacker, in the second and fourth steps of the IND-CPA game, can ask for any ciphertext to be deciphered (except the one in question c), and still cannot distinguish which one among two plaintexts corresponds to the ciphertext:

1. The oracle creates a secret key.
2. The attacker asks for the decryption of any ciphertext (except c), and creates two plaintexts M_0 and M_1 of equal size.
3. The oracle
    - randomly picks a bit b in $0,1$
    - encrypts M_b, and

- passes the ciphertext to the attacker.

4. The attacker asks for the decryption of any ciphertext (except c), and chooses a bit b' in $\{0,1\}$.

Example. Bleichenbacher's attack on PKCS#1 from 1998 is secure against the RSA variant of IND-CPA (but precisely not against IND-CCA!).

Bellare and Namprempre showed in 2000 for a symmetric cipher that if

- the cipher resists against IND-CPA, and
- the unidirectional function resists (is not *forgeable*) against an attack of chosen messages,

then the cipher with "Encrypt-then-MAC" resists an "IND-CCA" attack.

Self-Check Questions.

1. Is the one-time pad perfectly secure in theory?

   - ☐ *Yes*
   - ☐ No

2. What are all known perfectly secure cipher in theory? *one-time pad*

3. Is the one-time pad perfectly secure in practice?

   - ☐ Yes
   - ☐ *No*

4. What is a practical inconvenience of the one-time pad? *Its key must be as long as the plaintext.*

12.5 Side-channel attacks

A side-channel attack uses information from the physical implementation of a cipher machine. For example, measures of timing, power consumption, electromagnetic or sound emissions.

> **side-channel attack**: an attack that uses information on the physical implementation of a cipher machine.

We will restrict to **timing attacks** that measure the runtimes of cryptographic operations (of a specific software on a specific hardware) and compare them to estimated ones. A timing attack can be carried out remotely, however, the measurements often suffer from noise, that is, random disturbances from sources such as network latency, disk drive access times, and correction of transmission errors. Most timing attacks require that the attacker knows the implementation; however, inversely, these attacks can also be used to reverse-engineer.

> **timing attack**: an attack that measures the runtime of cryptographic operations and compares them to estimated ones.

We will restrict to the example of a timing attack that measures the time for computing integer powers. For this, we first have to understand how integer powers are computed:

Exponentiating by squaring.

Exponentiating by squaring (or square-and-multiply algorithm or binary exponentiation) is an algorithm to quickly compute large integer powers of a number by binary expansion of the exponent; especially useful in modular arithmetic. To compute b^n , instead of b multiplying b by itself n times, only $2 \cdot \log_2 n$ multiplications are needed:

> **Exponentiating by squaring** (or **square-and-multiply algorithm** or **binary exponentiation**): an algorithm to quickly compute integer powers of a number by binary expansion of the exponent.

Given a nonnegative integer base b and exponent e , to compute $b^e \bmod \mathrm{M}$:

1. Expand the exponent binarily, that is,

$$e = e_0 + e_1 2 + e_2 2^2 + \cdots + e_s 2^s \text{ with } e_0, e_1, \ldots, e_s \text{ in } \{0,1\},$$

2. Compute

$$b^1, b^2, b^{2^2}, ..., b^{2^s} \bmod M.$$

Because $b^{(2^{n+1})} = b^{2^n \cdot 2} = (b^{2^n})^2$, that is, each power is the *square of the previous one* (and at most `M`), each power, one after the other, is easily computable, yielding:

$$b^e = b^{e_0 + e_1 2 + e_2 2^2 + \cdots + e_s 2^s} = b^{e_0} (b^2)^{e_1} (b^{2^2})^{e_2} \cdots (b^{2^s})^{e_s}$$

(*In hindsight*, only powers with $e_0, e_1, ..., e_s$ equal to `1` count, the others can be omitted.)

Example. To calculate $3^5 \bmod 7$, expand

$$5 = 1 + 0 \cdot 2^1 + 1 \cdot 2^2$$

and calculate

$$3^1 = 3, 3^2 = 9 \equiv 2, 3^{2^2} = (3^2)^2 \equiv 2^2 = 4 \bmod 7;$$

yielding

$$3^5 = 3^{1+2^2} = 3^1 \cdot 3^{2^2} = 3 \cdot 4 \equiv 5 \bmod 7.$$

Vulnerable Algorithms. The execution time of binary exponentiation depends linearly on the number of bits equal to `1` in the exponent. While the number of these bits alone is insufficient information to find the key, statistical correlation analysis (and the Chinese remainder theorem) on exponentiations with different bases (but the same exponent) derives the exponent.

Crypto-algorithms that encipher using exponentiation modulo a large prime number, and as such are vulnerable to this attack, include `RSA`, `Diffie-Hellman` and `ElGamal` and the Digital Signature Algorithm (that derives from the former).

For example, in (textbook) `RSA` , the message is the base b and the key is the exponent e . Brumley and Boneh (2005) demonstrated a network-based timing attack on SSL-enabled web servers using RSA that successfully recovered the private key in a day; this lead to the widespread deployment of *concealment* techniques to conceal correlations between key and encryption time.

Example for the Exponential (as used in Diffie-Hellman). Kocher (1996) exposed a flaw in the following algorithm to compute modular exponentiation, that is, to compute $R(y) = y^x \bmod n$ for n public and y known, but x secret. The attacker, by computing $R(y)$ for several values of y and knowing n , y and the computation time, can derive x as follows:

```
Let w be the bit length of x and put s_0 = 1.

For k ranging from 0 to w-1:

    If the k-th bit of x is 1, then
        put R_k = (s_k * y) mod n;
    Otherwise,
        put R_k = s_k.

    Put s_{k+1} = R_kˆk mod n

End (of For loop)

Return (R_{w-1})
```

According to the value of the k -th bit of x , either $(s_k \times y) \bmod n$ or nothing is computed; therefore, the execution time of the algorithm, for different values of y will eventually yield the value of the k -th bit.

To prevent this attack, the algorithm can be changed so that all calculations, whatever the key bits, take the same time (slowing it down, but, security counts more than speed):

```
Let w be the bit length of x and put s_0 = 1.

For k ranging from 0 to w - 1:

    Put temp = (s_k y) mod n.

    If the k-th bit of x is 1, then
        put R_k = temp;
    Otherwise,
        put R_k = s_k.
```

```
    Put s_{k+1} = R^2_k mod n.

End (of For loop)

Return (R_{w-1})
```

Self-Check Questions.

1. Name examples of side-channel attacks: *measures of timing, power consumption, electromagnetic or sound emissions.*

12.6 Modern Cryptanalytic Algorithms

To understand the reasons behind the design choices of each step of a block cipher algorithm, such as AES, one must understand which attacks it defies. A powerful modern cryptanalytic algorithms is differential cryptanalysis, applicable to block ciphers. It assumes a chosen-plaintext attack: The attacker sends pairs of (slightly) differing plaintexts, whose ciphertexts he receives. He then studies how differences in the input propagate (on so-called differential trails) through the network of encipherment transformations to differences at output. The resistance of AES against this resistance by so-called wide trails was proved in the paper of proposal Daemen and Rijmen (1999).

Prototypical Feistel Cipher by Heys.

Let us demonstrate this technique in the toy model of a Feistel Cipher given in Heys (2002) that

- divides the plaintext into blocks of 16 bits, and
- subdivides each block into 4 blocks of 4 bits.

For each round, there is a corresponding (independent) key. In each one of the first three rounds 1, 2 and 3:

1. Add the round key C to the block B, in formulas: $B \mapsto B \oplus C$.
2. Substitute each of the 4 sub-blocks bits according to the table (in hexadecimal notation)

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

3. Swap bit i from the sub-block j with j from the sub-block i ;

In the penultimate 4th round:

1. Add the round key to the block, $\mathrm{B} \mapsto \mathrm{B} \oplus \mathrm{C}$.
2. Substitute each of the 4 sub-blocks of 4 bits with the table.

In the last 5th round

1. Add the round key to the block, $\mathrm{B} \mapsto \mathrm{B} \oplus \mathrm{C}$.

That is:

- In the last 5th round, the last two steps, substitution and permutation, are omitted, because, the algorithm being public (following Kerckhoff's principle), can be undone by any decipherer without knowledge of the key. That is, from a cryptographic point of view, they are superfluous.
- In the 4th round, the last step, the permutation, is omitted as it would only permute the last 5th round key. That is, from a cryptographic point of view, it is superfluous.

The substitution table originates from the DES algorithm and is commonly called the *S-box*, Substitution box.

Differential Cryptanalysis. A cryptanalyst's dream is to learn whether a part of the chosen key is *correct*, that is, whether it coincides with the corresponding part of the key used to encrypt the text: For example, in Heys's cipher the key has 16 bits: If one could learn whether, say, one half (8 bits) of the whole key tried out matches the corresponding half of the correct key, then the cryptanalyst

- instead of testing out all possible combinations, of which there are $2^{16} = 65536$,
- only, needs to test out all possible combinations of these 8 bits (of which there are $2^8 = 256$) and the remaining 8 bits (of which there are $2^8 = 256$).

That is, the number of combinations that need to be tested out has been reduced from $2^{16} = 65536$ to $2 \cdot 256 = 512$.

Criterion for Decipherment. In a brute-force attack, the cryptanalyst deciphers the enciphered text with each possible key. To know whether the key tried out is *correct*, that is, if it coincides with the key used to encrypt the text, he checks whether the content is *intelligible*; for example, by a criterion such as

1. counting the frequencies of the letters, pairs and triples of the deciphered text, and
2. comparing them to the frequencies of the likely tongue in which the plaintext was written: If they come close, then the plaintext is probably intelligible and the key tried out was that used by the encipherer.

If the cipher has a single round, then this criterion is applicable. However, if the cipher has two or more rounds, and the decipherer executes the last round of the decryption algorithm with a certain key, then this criterion is no longer applicable, because the text obtained is the output of the encryption algorithm (with the same key) from the penultimate round. Instead, the criterion of differential cryptanalysis for having found the correct key is probabilistic: the key tested out is probably correct if, for a certain "incoming" difference ΔX and a certain "outgoing" difference ΔY , plaintext pairs with difference ΔX result with a certain probability in cipher pairs with difference ΔY .

For differential cryptanalysis to be applicable, the cryptanalyst must be able

1. to encrypt by the same key any number of freely chosen plaintexts, and
2. to examine the encrypted texts.

Differential cryptanalysis exploits the high probability of a **difference** $\Delta X := X' \oplus X''$ between two plaintexts X' and X'' propagating to a difference $\Delta Y = Y' \oplus Y''$ between the two ciphertexts Y' and Y'' (for X' and X'') (in the penultimate round); here $X' \oplus X''$ is the addition XOR , bit per bit, where the output is 1 if, and only if, the two entries are different. (In particular, the addition XOR is *auto-inverse*, that is, the reverse operation to $\oplus$ is $\oplus$; in contrast to the + operation with its reverse − . Therefore, the *difference* is given by the addition XOR. That is, ΔX , indicates all the bits in which X' and X'' differ.) The pair $D = (\Delta X, \Delta Y)$ is the **differential**.

The **difference** of X' and X'' is $\Delta X := X' \oplus X''$

A **differential** is a pair $D = (\Delta X, \Delta Y)$ of input respectively output differences ΔX respectively ΔY .

For differential cryptanalysis to be efficient, there must be a differential D with *high probability* p_D (to be quantified in Equation 4); that is, among all incoming pairs with difference ΔX , the probability of an outgoing pair having difference ΔY (in the penultimate round) is p_D. More exactly, the encipherer will encipher a statistically significant number of pairs ($> 1/p_D$) of plaintexts with difference ΔX to count the number of the enciphered pairs of ciphertexts with difference ΔY .

Frequency of Differences for a Substitution Table. An **affine transformation** A is the composition

- of a *linear* application, that is, $A(x \oplus y) = A(x) \oplus A(y)$ for all x and y , and
- of a *translation*, that is, $A(x) = x \oplus x_0$ for some fixed x_0).

Observation. For an affine transformation A , the outgoing difference ΔY is independent of the incoming pair X' and X' (but only depends on ΔX): The transformation A :

- if it is linear, then always, that is, for *every* incoming pair with difference ΔX , the outgoing difference is $\Delta Y = A(\Delta X)$, and
- if it is a translation, then always $\Delta Y = \Delta X$.

Regarding the first and second function of each round of a Feistel cipher:

- the addition of the key is a translation,
- the permutation is linear,

that is, the outgoing difference is independent of the incoming pair. However, the outgoing difference ΔY of the substitution is *not* determined by the incoming difference ΔX alone, but it *depends* on X' and X'' ! We examine the substitution table to find a differential D of high probability p_D, that is, to find an incoming difference ΔX with a large number of pairs X' and X'' that yield an outgoing difference ΔY : Given ΔX , there are $2^4 = 16$ possible inputs X' (which determines

$X'' = X' \oplus \Delta X$), and we count the frequencies of the $2^4 = 16$ possible outgoing differences $\Delta Y = 0, 1, \ldots, F$:

Table 22: S-box in binary notation.

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111

This table lists the outgoing differences of each incoming pair whose difference is one of 1011, 1000 or 0100 :

Table 23: the outgoing differences ΔY for three incoming differences ΔX (listed horizontally) and all possible inputs X (listed vertically).

X/ΔX	1011	1000	0100
0000	0010	1101	1100
0001	0010	1110	1011
0010	0111	0101	0110
0011	0010	1011	1001
0100	0101	0111	1100
0101	1111	0110	1011
0110	0010	1011	0110
0111	1101	1111	1001
1000	0010	1101	0110
1001	0111	1110	0011
1010	0010	0101	0110
1011	0010	1011	1011
1100	1101	0111	0110
1101	0010	0110	0011
1110	1111	1011	0110
1111	0101	1111	1011

Let us count, for every incoming difference ΔX , how many times each outgoing difference ΔY appears among all the incoming pairs X' and X'' such that $X' \oplus X'' = \Delta X$.

Table 24: The frequency of outgoing differences ΔY (listed horizontally) for all incoming differences ΔX (listed vertically).

ΔX/ΔY	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	0	0	0	2	0	2	4	0	4	2	0	0
2	0	0	0	2	0	6	2	2	0	2	0	0	0	0	2	0
3	0	0	2	0	2	0	0	0	0	4	2	0	2	0	0	4
4	0	0	0	2	0	0	6	0	0	2	0	4	2	0	0	0
5	0	4	0	0	0	2	2	0	0	0	4	0	2	0	0	2
6	0	0	0	4	0	4	0	0	0	0	0	0	2	2	2	2
7	0	0	2	2	2	0	2	0	0	2	2	0	0	0	0	4
8	0	0	0	0	0	0	2	2	0	0	0	4	0	4	2	2
9	0	2	0	0	2	0	0	4	2	0	2	2	2	0	0	0
A	0	2	2	0	0	0	0	0	6	0	0	2	0	0	4	0
B	0	0	8	0	0	2	0	2	0	0	0	0	0	2	0	2
C	0	2	0	0	2	2	2	0	0	0	0	2	0	6	0	0
D	0	4	0	0	0	0	0	4	2	0	2	0	2	0	2	0
E	0	0	2	4	2	0	0	0	6	0	0	0	0	0	2	0
F	0	2	0	0	6	0	0	0	0	4	0	2	0	0	2	0

The entries for each row add up to 16 , the number of all possible pairs for a given difference. The first row confirms that two equal inputs result in two equal outputs. The highest number is 8 and reached for $\Delta X = B$ and $\Delta Y = 2$. In addition, the number 6 comes up five times.

We will choose our differentials among those with these high frequencies:

Example. In the frequency table

- of a translation, such as the addition of the secret key, all boxes are null except those in the first column which have value 16 ;
- for a linear operation, such as a permutation of bits, in each row all the entries are null except one of value 16 .

Differential Trails. Given a Feistel cipher, a **differential trail** is a finite sequence of differences

$$(\Delta U_1, \Delta U_2, \ldots)$$

so that every entry ΔU_i is the input of the S-box of the i -th cipher round. Given the outgoing difference ΔV_i of the S-box of round i , the incoming difference of the next round ΔU_{i+1} is the result of applying the permutation to ΔV_i . (The key addition, as a translation, does not change the difference.)

> **differential trail**: a tuple of differences ΔU_1, ΔU_2, ... so that every entry ΔU_i is the input of the S-box of the i -th cipher round.

We want to find the most probable differential trail D in the Heys cipher

$$D = (\Delta U_1, \Delta U_2, \Delta U_3, \Delta U_4)$$

or at least a trail in which each differential $(\Delta U_i, \Delta V_i)$ is among the most probable. Every differential consists of 4 sub-differentials, corresponding to the 4 sub-blocks of 4 bits that constitute block of 16 bits. To find such a probable differential trail, in each round:

- *maximize the frequency* of each sub-differential, that is, the number of times the S-box transforms the incoming difference (of the sub-differential) into the outgoing difference;
- in particular, *minimize the number of (so-called* active*) nonzero sub-differentials.*

An example of such a trail D is the following: Let the difference in the first round be

$$\Delta U_1 = [0000\ 1011\ 0000\ 0000],$$

which is by S-box 2 replaced by

$$\Delta V_1 = [0000\ 0010\ 0000].$$

By the subsequent permutation, we obtain as difference entering the second round

$$\Delta U_2 = [0000\ 0000\ 0100\ 0000]$$

which is by S-box 3 replaced by

$$\Delta V_2 = [0000\ 0000\ 0110\ 0000].$$

Because the bits number 2 and 3 are nonzero, we obtain by the subsequent permutation as incoming difference of the third round that with two active sub-differentials

$$\Delta U_3 = [0000\ 0010\ 0000]$$

which is by S-boxes 2 and 3 replaced by

$$\Delta V_3 = [0000\ 0101\ 0000].$$

Finally, by the subsequent permutation, the fourth round input is

$$\Delta U_4 = [0000\ 0110\ 0000\ 0110].$$

Let $S_{i,j}$ denote the substitution of the sub-block j by the S-box in the i -th round. On our differential trail, we enlist

- for each round $i = 1, 2, 3$ and
- for each sub-differential $j = 1, 2, 3, 4$ different from zero,

the probability of substitution $S_{i,j}$ transforming the incoming difference ΔX (in hexadecimal notation) into outgoing difference ΔY :

Substitution	In	Out	Probability
S_{12}	B	2	8/16
S_{23}	4	6	6/16
S_{32}	2	5	6/16
S_{33}	2	5	6/16

If we suppose that the differentials of one round are independent of the differentials of the previous round (which is a negligibly inaccurate simplification), then the probability p_D of the concatenated substitutions transforming

$$\Delta U_1 = [0000\ 1011\ 0000\ 0000]$$

into

$$\Delta U_4 = [0000\ 0110\ 0000\ 0110].$$

is the product of the probabilities of each substitution,

$$p_D = 8/16 \cdot 6/16 \cdot (6/16 \cdot 6/16) = 27/1024.$$

To find the key, for

- every possible combination of $K_{5,5}, \ldots, K_{5,8}$ and $K_{5,13}, \ldots, K_{5,16}$,

- an (integer) multiple m of $1/p_D \approx 38$ pairs of plaintexts U_1' and U_1'' with difference ΔU_1,

the cryptanalyst

1. enciphers the pair U_1' and U_1'',
2. reverses the cipher up to the `S-box` input in the fourth round by the round key
$$K_5 = [0000\ K_{5,5}, \ldots, K_{5,8}\ 0000\ K_{5,13}, \ldots, K_{5,16}]$$
to obtain the pair v_4' and v_4'' with difference v_4, and
3. compares the difference Δv_4 to ΔU_4; if they match, then he increments the count n by 1 .

If for a combination of sub-blocks $K_{5,5}, \ldots, K_{5,8}$ and $K_{5,13}, \ldots, K_{5,16}$ the count yields $n/m \approx p_D$, that is, the ratio between

- the number n of matched pairs and
- the number m of total pairs

is close to the probability p_D, then these sub-blocks are probably the sub-blocks `2` and `4` of the round key `5` used by the cipher.

Observation. To conclude that we found the correct sub-blocks, we use the hypotheses

1. that the differentials of a round are independent of the differentials of the previous round, and
2. that a probability of matching pairs close to that calculated indicates the correct key.

Both have no rigid mathematical foundation, but are only plausible, because, respectively:

1. Each round tries to diffuse as much as possible, that is, to make the value of each output bit practically independent of all input bits, and
2. there is unlikely a key which is different but reproduces the same probability.

Note that for this attack to be faster, that is, to be more effective, than the brute-force attack (which simply tries out all possible keys), it is necessary that

$$\#\{ \text{ active bits } \} - \log_2 p_D < \#\{ \text{ key bits } \} \tag{4}$$

where

- a bit is *active* if it belongs to an active sub-differential in the penultimate round (in our differential trail, these are the bits of the sub-differentials 2 and 4),
- the probability p_D is that of the incoming differential ΔU_1 from the first round leading to the incoming differential ΔU_4 from the penultimate round (on our trail, we have $\log_2 p_D = \log_2(27/1024) \approx -5$), and
- the key is that from the last round (which has 16 bits in this figure).

Therefore, it is necessary that the trail is *strict*, that is, has few active blocks, in order to be able to learn whether the tested key is correct, that is, coincides with the key used, only in these active blocks (which reduces the number of combinations logarithmically). In the example given, only 2 out of the 4 sub-differentials are active, which allowed the cryptanalyst to learn whether the key is correct in only these 2 blocks: the number of combinations that need to be proved was accordingly reduced from $2^{16} = 65536$ to $2 \cdot 256 = 512$.

Self-Check Questions.

1. What is a differential in a substitution and permutation network? A pair $D = (\Delta X, \Delta Y)$ of input respectively output differences ΔX respectively ΔY .
2. What is a differential trail in a substitution and permutation network? A tuple of differences ΔU_1, ΔU_2, ... so that every entry ΔU_i is the input of the S-box of the i -th cipher round.
3. How to find the most likely differential trail? By maximizing the number of times the S-box replaces the incoming difference with the outgoing difference of the sub-differential.

Summary

Cryptanalysis is the art of breaking ciphers, that is, recovering or forging enciphered information without knowledge of the key. Historically, the cryptanalyst's

intuition and ability to recognize subtle patterns in the ciphertext were paramount. Today, however, cryptanalysis is based on mathematics and put into practice by efficient use of extensive computing power.

Brute-Force Attacks. In practice, the security of a cipher, and thus the recommended key sizes, relies foremost

- on its resistance to the most efficient (known!) methods of cryptanalysis (using a back door), and
- the computational effort needed to check all keys (taking the front door) by checking the decrypted output for probable patterns of a plaintext.

Rainbow Tables. It the secret information, for example, passwords, was stored as cryptographic hashes, then a practically more efficient brute-force attacks uses a rainbow table, a table of the cryptographic hashes of the most common passwords to reveal more likely passwords sooner. This is particularly promising against quickly computed hash functions such as `MD4/5`, whereas hash functions used for hashing passwords, such as `bcrypt`, were designed to be deliberately slow. Such a rainbow attack is however most effectively prevented by making the used hash function unique for each password by adding a `salt`, an additional unique, usually random, argument.

Attacking Scenarios. The only perfectly secure cipher is the one-time pad where a key (of the same size as the plaintext) is added (bit by bit) to the plaintext; too unwieldy to be useful in practice.

Asymmetric cryptography uses mathematical methods, more exactly modular arithmetic, to encipher. The security, the difficulty of deciphering without knowledge of the key, of an asymmetric cryptographic algorithm is based on mathematical problems that have been established as computationally difficult. Symmetric cryptography (like hash functions) uses more artisanal methods of ciphering, which aim to maximize diffusion and confusion, mainly by iterated substitution and permutation. The security of symmetric cryptographic algorithms is simply based on its resistance against years of ongoing attacks.

Side-Channel Attack. A side-channel attack uses information from the physical implementation of a cipher machine. For example, measures of timing, power consumption, electromagnetic or sound emissions. In particular, a **timing attack** measures the runtimes of cryptographic operations, and compares them to the estimated ones. For example, one exploits that the runtime of the computation of a power depends on the number of nonzero bits of its exponent (which, in RSA and Diffie-Hellman is the key).

Differential Cryptanalysis. Among all modern cryptanalytic algorithms, one of the most powerful ones is differential cryptanalysis that applies to block ciphers and assumes a chosen-plaintext attack: The attacker sends pairs of (slightly) differing plaintexts whose ciphertexts he receives. He then studies how differences on input propagate (on so-called differential trails) through the network of encipherment transformations to differences at output. The resistance of AES against this resistance by so-called wide trails was proved in the paper of proposal Daemen and Rijmen (1999).

Questions

1. Which minimal key size is currently recommend as secure for `RSA` and `Diffie-Hellman`?

   - ☐ `512` bits
   - ☐ `1024` bits
   - ☐ *2048 bits*
   - ☐ `4096` bits

2. Which minimal key size is currently recommend as secure for Elliptic Curve Cryptography?

   - ☐ `128` bits
   - ☐ `160` bits
   - ☐ *256 bits*
   - ☐ `512` bits

3. Which minimal key size is currently recommend as secure for `AES`?

   - ☐ `80` bits

- ☐ *112 bits*
- ☐ 128 bits
- ☐ 256 bits

4. Which computationally difficult problem is the security of RSA based on?

   prime number decomposition, discrete logarithm, point counting, quadratic residue

5. Which computationally difficult problem is the security of the Diffie-Hellman key exchange based on?

   prime number decomposition, *discrete logarithm*, point counting, quadratic residue

Required Reading

Read Heys (2002) to understand the basic principles of differential cryptanalysis.

Unit 5. Read the submissions Daemen and Rijmen (1999) and Daemen and Rijmen (2002) that present the algorithm AES and show its security by its resistance against differential cryptanalysis.

13 Cryptology and the Internet

Study Goals

On completion of this chapter, you will have learned …

1. How the Internet and its protocols came about,
2. what a Virtual Private Network, in particular IPSec, achieves.
3. how the Transport Layer Security protocol encrypts and authenticates Internet connections,
4. how secure e-mails can be sent by protocols such as `TLS`, `S/MIME` or `PGP`, and
5. how domain names such as `ongel.de` are looked up and how this look can be authenticated and encrypted by Secure DNS.

Introduction

While before the Internet age it was unimaginable to use cryptography in everyday life, today everyday life on the Internet would be unimaginable without (public-key) cryptography; for example, for securely shopping online. Besides ciphering, cryptography establishes trust where previously paper documents were used, for example, for signing, identity authentication, granting authority, license, or ownership. And cryptography achieves this more securely so: While a written signature is imitable, a digital signature is uniquely linked to (the contents of) the signed document.

To secure transactions on the Internet, for example, in electronic banking, commerce or mailing, a cryptographic protocol (such as Transport Layer Security, TLS, formerly Secure Sockets Layer, SSL):

1. Establishes trust: When a client connects to a web server, then the server's identity has to be guaranteed to avoid a man-in-the-middle attack; to this end, central authorities issue digital `X.509` certificates.
2. Encrypts all traffic: For example, in an open wireless-network, such as those using log-in portals in public places, no traffic between the router (that connects to the Internet) and the client is being encrypted, neither by the router nor by the client.

13.1 The Internet Protocols

The various protocols that standardize the processing of exchanged data on the Internet can be grouped into layers, ordered according to how highly structured the processed data is:

- The lower layers format the raw data and serve as interfaces for the upper layers, whereas
- the upper layers are closer to the user's applications and handle more abstract data.

Among these protocols, the two most important protocols (and those that were defined first) are

- the Transmission Control Protocol (TCP), and
- the Internet Protocol (IP).

that specify how data should be formatted, addressed, transmitted, routed and received at the destination. Though the TCP/IP protocol reliably delivers data packets over the Internet, it

- neither guarantees security, neither confidentiality nor authenticity,
- nor manages sessions between a client and the server (for example, suspension, termination and restart of a session).

Most Internet applications rely on a higher (application) layer, such as the HTTP protocol for Web servers.

The best known stacks of such layers of Internet protocols are:

- The **Open Systems Interconnection (OSI) reference-model** with *seven* layers, created by the International Standards Organization (ISO) as the international standard for the architecture of computer networks, and
- the **Internet Protocol Suite** with *four* layers, specified in Section 1.1.3 of Braden (1989). It is sometimes called the DoD(-Layer) Model, because development began in the late 1960s with a study under the supervision of the United States Department of Defense (DoD).

  OSI model: model by the International Standards Organization (ISO) that stacks the various protocols of the Internet protocol suite into seven abstraction layers.

  Internet Protocol Suite: TCP/IP stacks the various protocols of the Internet protocol suite into *four* layers.

World Wide Web. First, the Internet is not to be confused with the **World Wide Web** (WWW, for short the Web) which is an application of the Internet, albeit the most popular serves linked documents. A web document is written in HyperText Markup Language (HTML), transmitted by the HyperText Transfer Protocol (HTTP) and retrievable at an online address called a Uniform Resource Locator (URL). The web does not include, for example, e-mail, instant messaging and file sharing.

> **World Wide Web**: an application of the Internet that serves on a graphical user interface linked documents written in HyperText Markup Language (HTML), transmitted by the HyperText Transfer

Protocol (HTTP) and retrievable at an online address called a Uniform Resource Locator (URL).

Originally, the Web was developed to let scientists around the world exchange information instantly. In 1989, English computer scientist Timothy Berners-Lee developed at the Conseil Européen pour la Recherche Nucléaire (CERN, the European Organization for Nuclear Research) the first Web server and client, a hypertext browser and editor, and specified the URL, HTTP and HTML formats on which the Web depends. The Web was made available within CERN in December 1990 and on the Internet at large in the summer of 1991. (Since 1994, Berners-Lee is Director of the W3 Consortium, which coordinates Web development worldwide.)

History of the Internet. The Internet itself evolved out of an early wide area network called **ARPAnet**: President Dwight D. Eisenhower saw the need for the Advanced Research Projects Agency (ARPA), after the Soviet Union's launch of Sputnik, world's first satellite, in 1957. The ARPA followed suite by developing the United States' first successful satellite in 18 months. Several years later ARPA developed computer network of funded research laboratories that eventually evolved into ARPAnet in the 70s. It was expanded in the United States to some laboratories and academic institutions, and then to Europe, where the European Organization for Nuclear Research (CERN) was one of the nodes.

ARPAnet: computer network developed by the Advanced Research Projects Agency (ARPA) between a number of laboratories and academic institutions

A driving force behind the creation of the ARPAnet was the desire to share computer resources more efficiently:

Time Sharing. Computer time in the 50s was so costly access time had to be limited and scheduled. Much time on the computer was used for input and output, but not computing, so the computing power often went unused, and the computer was practically idle. To use computer resources more efficiently, the idea of "Time-sharing" to run multiple programs (seemingly) "at the same time" was born: The computer switched from user to user while receiving input or returning output, giving them the impression of a live interaction with the computer instead of taking turns. The Ethernet protocol was created (by the

Xerox Corporation) to connected different computers into a single (so called *local-area*) network. These interactions then took place on a local area network, but the more users, the less responsive the computer.

Networking. The idea was born to share computational resources by connecting various local networks to a single larger network. In 1966 the Information Processing Techniques Office (IPTO) funded the creation of a high-speed network among the funded research laboratories, which eventually evolved into the ARPANET that differed from existing computer networks by:

- connecting a network of computers (among equals) instead of sharing a single computer (server) among many terminals (hosts).
- so-called *packet switching*, that is, messages had a designated destination and return address but no mandatory delivery route (as in the telephone system):

Such a decentralized network, without critical paths, sparked the interest of the military, since it could reroute messages even if part of the network was destroyed (while the destruction of a telephone operation center entails that of its entire dependent network).

ARPAnet. The American computer scientist Vinton Cerf at Stanford University wrote the first TCP protocol with Yogen Dalal and Carl Sunshine, called Specification of Internet Transmission Control Program (RFC 675), published in December 1974. From 1972-1976, Cerf co-designed the TCP/IP protocol-suite with Robert Kahn that would form the basis of the Internet. Cerf worked at the United States Defense Advanced Research Projects Agency (DARPA) from 1976 to 1982 and funded various groups to develop TCP/IP. When Robert Kahn became IPTO director in 1979, the DARPA had multiple incompatible packet-switching networks, which on 1 January 1983, adopted the suite of TCP/IP protocols. After some demonstrations of the network technology, such as linking the networks of SATNET, PRNET and ARPANET from Menlo Park, CA to University College London and back to USC/ISI in Marina del Rey, CA in November 1977, the Internet started in 1983.

Comparison between the OSI and TCP/IP reference-model. Since the Internet runs on the TCP/IP reference-model, which has only *four* layers, the standard ISO 7-layer stack is more of a theoretical abstraction than a practical standard. Unlike the standard ISO 7-layer stack, the TCP/IP 4-layer stack evolved by being used rather than drafted and thus is testified to work, even cross-platform. While the Internet Protocol Suite is descriptive, the OSI reference-model was intended to be prescriptive: the OSI model is a wonderful abstract construction; though the network, which exists and works, does not fully follow it. However, the iOS Model is of historical and conceptual interest, as it precedes the former, and the same principles apply.

When the models are (incorrectly) used interchangeably, both are referred to as the Internet reference-model.

Layers.

Table 26: approximative comparison table between the OSI and TCP-IP layers:

OSI Layer	TCP/IP Layer	Examples
Applications (7)	Applications (4)	HTTP(Hypertext Transfer Protocol FTP(File Transfer Protocol) SMTP(Simple Mail Transfer) IMAP(Instant Message Access Protocol) DHCP(Dynamic Host Configuration)
Presentation (6)		
Session (5)		SOCKS (Socket Secure)
Transport (4)	Transport (3)	TCP Transmission Control Protocol) UDP(User Datagram Protocol)
Network (3)	Internet (2)	IPv4/6(Internet Protocol) ICMP(Internet Control Message)
Data Link (2)	Network Access (1)	IEEE 802.3 (Ethernet LAN) IEEE 802.11,802.11a -- g (Wi-Fi)
Bit Transmission (1)		

1. The physical layer provides only the means to transmit raw bits. Electrical specifications such as Network hardware or physical cabling are specified.

2. The network access layer (*Link Layer*) does not contain protocols of the TCP/IP family, but subsumes those for data transmission from point-to-point, to connect different subnets, such as *Ethernet*, *Point-to-Point_Protocol* (PPP) or *802.11* (Wireless_LAN).
3. The Internet layer comprises all protocols for the forwarding and routing of packets, that is, determining the next intermediate destination for a received packet and to forward the packet there; also segmenting, error detection and error correction. The core of this layer is the [Internet Protocol] (IP) in version 4 or 6
4. The transport layer comprises all protocols for establishing, prioritizing, maintaining, and terminating the communication between two computers on a network and checking that data sent from one computer to another has correctly reached its destination. Most importantly, the Transmission Control Protocol (TCP) to reliably send data streams, but also unreliable protocols such as the User Datagram Protocol (UDP).
5. The session layer comprises protocols keeps track of the progress of data transfers, for session hibernation (checkpointing), suspension, termination and restart procedures, for example, after a transmission error or interruption.
6. Presentation Layer: For the formatting of messages, converting data into a format understandable by an application such as a Web Server, for example, encryption and decryption (or codeset conversions, say from ISO Latin-1 to UTF-8).
7. The application layer comprises all protocols to exchange application-specific data over the network, for example, the Mozilla HTML engine used by FireFox and Chrome, or the SMTP protocol used by e-mail programs

However, different application protocols (such as HTTP, FTP, IMAP, ...) implement the functions of Layer 5, 6 and 7 differently, and do not separate these layers strictly; hence, practitioners, such as network engineers, subsume all those layers as Layer 5+, the application layer, just like the Internet Protocol Suite does.

Self-Check Questions.

1. Please name the layers of the OSI model! *Applications , Presentation , Session , Transport , Network , Data Link , Bit Transmission*

2. Please name the layers of the DoD model! *Applications , Transport , Internet, Network Access*
3. Please provide example protocols for five layers of the OSI model! *HTTP, SOCKS, TCP, IP, Ethernet*
4. Please provide example protocols for each layer of the DoD model! *HTTP, TCP, IP, Ethernet*

13.2 IPsec

A private IP network is a network (commonly a local area networks (LANs) in residential and enterprise environments) whose computers have IP addresses which fall into the ranges specified by IPv4 (in RFC 1918; analogue ones exist in IPv6):

- from `10.0.0.0` to `10.255.255.255`,
- from `172.16.0.0` to `172.31.255.255`, and
- from `192.168.0.0` to `192.168.255.255`

These addresses can be used without approval from an Internet registry. (Where here and henceforth we often mean computer to mean an endpoint of the network; this includes tablet, smartphones and other network devices.)

> **Private Network**: network that uses private IP addresses without any need of approval from an Internet registry.

A Virtual Private Network (**VPN**) is a private network made up of (at least) two (spatially separate) closed networks connected via an open network (such as the Internet). For example, to connect

- networks between two companies (to form a so-called extranet),
- various networks within the same company (so-called intranets), and
- a single client via the Internet to an intranet (so-called remote access; the most common use of VPNs for the end-user).

> **VPN**: A private network made up of two or more (spatially separate) closed networks connected via an open network (such as the Internet).

To establish privacy, the connections between two closed networks use authentication and encryption: The parties authenticate mutually using a previously set shared secret (such as a password or certificate); then the exchanged data is encrypted and decrypted at the end points.

Gateway. Briefly, a gateway is a device that links two networks such as a router that distributes network traffic flow. An internet connection at home usually uses a router to deliver internet data packets to the devices at home, for example, a smartphone, tablet or laptop. This is the first router the device at home connects to for an Internet connection; also known as a default gateway. By convention the gateway has the lowest IP address in the subnet (a group of addresses).

A Firewall filters data packets to protect an inner (private) network from an outside (public) network. It is usually located on a gateway, or possibly as software on a user's computer, for example, as part of the operating system such as the "Microsoft Windows firewall".

Many (consumer) devices are both a router and a firewall. Therefore, these three terms, Gateway, Router and Firewall, are sometimes used interchangeably.

Router. Routing is the directing of data packets from their source toward their ultimate destination through intermediary nodes, called routers, over the network. A **router** is a computer networking device that distributes data packets across a network of (two or more) networks toward their destinations, through a process known as routing, that is,. Routing occurs at layer 3 (the Network layer) of the OSI seven-layer model. To compute the best routes to network destinations the routers use routing tables: a basic routing table stores details of every computer in the network and the connections between them; others also the current state of the network with respect to the amount of traffic.

The simplest routing is hop-by-hop routing: each routing table lists, for all reachable destinations, the address of the next device along the path to that destination; the next hop. If the routing tables are consistent, then the simple algorithm of relaying packets to their destination's next hop thus suffices to deliver data anywhere in a network. In practice, hop-by-hop routing is now replaced by the newer Multiprotocol Label Switching (MPLS), where a single

routing table entry can select the next several hops resulting in less table lookups and faster arrival.

Routing Protocols specify how routers (mutually) exchange the (changes in their) routing tables. In the basic Router Information Protocol (RIP) they are periodically exchanged *entirely*. Because this is a rather inefficient process, RIP is replaced by the newer Open Shortest Path First(OSPF) protocol (RFC2178 from 1998) which causes smaller, more frequent updates (but require more processing power and memory).

Firewall. A **Firewall** is a software that monitors network traffic, usually between two networks, one trusted (for example, a private local-area network at home or at a company), the other one not, (for example, a public wide-area network such as the Internet), to provide security by blocking access from the public network to certain services in the private network. It can run on a multi-purpose device (like a personal computer) or a dedicated device (like a router, especially for larger networks).

A simple Firewall (known as packet filter or screening router) has a set of rules, which are applied to each data packet according to its attached metadata, (where it is from, where it should be sent to, ...), to decide whether to allow a packet through. In the simplest case, a firewall only refuses data packets based on its port. It works at the network OSI layer. Simply put, the private network is the castle, the firewall the bulwark and (network) ports holes that have to be drilled into it for access to the public network.

An application-layer firewall not only looks at the metadata but also at the actual data (so-called deep packet inspection, DPI). For example, a user of a private network could install a backdoor trojan by surfing to a Website with malicious code or by opening an email attachment.

A firewall can run on the computer in the private network connected to the public network, for example, on a personal computer running Linux or Windows connected to the Internet. However, the larger the private network, the more points of failure (such as out-of-date or disabled firewalls), including other devices such as a printer or TV. Therefore, it is prudent in larger private networks (such as those in a company) that connect to a public network to have a dedicated firewall for the whole network.

Gateway. A **Gateway** is a device, often a dedicated computer, that works as a gateway from the computers of one to those of another network; that is, that joins two different networks, usually an internal (local-area) network to a wide-area network such as the Internet. For example, the connection of a modem to the Internet via an Internet Service Provider (ISP) is shared among the computers of a home or company network via a router or a firewall. For these computers, the router or firewall is the gateway. However, gateway is more general a term than router or firewall, because it takes care of all possible conversions between different network architectures, for example, from one protocol or character encoding to another: say from TCP-IP to a proprietary protocol used by a subnetwork. It works at level 4 and higher of the OSI reference-model.

NAT. Network Address Translation (**NAT**) translates public IP addresses into private ones in an Internet Protocol(IP) network by changing the the source or destination address in every packet header (and adjusting the checksums): It replaces the host's internal source address in the IP packet header by the NAT device's external IP address. Typically, NAT is implemented on gateways (such as router or firewalls).

> Network Address Translation (**NAT**): translates the public IP addresses of an IP network into private ones.

Port Address Translation (PAT) replaces the host's source port number in the TCP (or UDP) header by one from a pool of available ports. The NAT device stores an entry in a translation table that maps the host's internal IP address and source port to the source port it was replaced with. While the internal host knows the IP address and TCP (or UDP) port of the external host, the external host only knows the public IP address of the NAT device and the port used to communicate with the internal host. However, a host's internal applications that use multiple simultaneous connections (such as an HTTP request for a web page with many embedded objects) can deplete available ports. To avoid this, the NAT device tracks the destination IP address in addition to the internal port (thus sharing a single local port with many remote hosts).

For peer-to-peer applications such as VOIP and VPNs, external hosts must connect directly to a particular internal host. The internal addresses all map to the same publicly accessible address of the NAT device. This poses no problem

if the NAT is a *full cone* or has *Endpoint-Independent Mapping and Filtering* (as categorized in RFC 3489 or RFC 4787), that is, maps an internal IP address to a static TCP/UDP port. However, NAT traversal (NAT-T) is needed, for example, if

- the NAT is a (port) *restricted cone* or has *Address-Dependent (and Port) Filtering*, that is, only allows incoming traffic (on a specific port) only in reply to outgoing traffic,
- the NAT is *symmetric* (RFC 3489) or *Address-Dependent Mapping* (RFC 4787), that is, the source external socket (IP address and TCP/UDP port) depends not only on the source's internal socket, but only on the destination socket.

To this end IPsec-VPN encapsulates Encapsulating-Security-Payload (ESP) packets into UDP packets using port 4500. VOIP employs

-either the Session Traversal Utilities for NAT (STUN) protocol (as specified in RFC5389) that uses *hole punching*: For Alice to establish a connection to Bob, a public relay server first passes Alice's public socket (IP address and UDP port) to Bob. Bob then sends an initial (usually rejected) packet to this socket so that Bob's firewall allows a connection from it (as reply to this outgoing connection),
- or, if STUN fails, the Traversal Using Relay NAT (TURN) protocol (as specified in RFC 5766), in which the public relay server not only mediates the socket information but the entire traffic between Alice and Bob.

Masquerading. Most networks use NAT to enable multiple hosts on a private network with different private IP addresses to connect to the Internet using a single public IP address assigned through an Internet Service Provider (ISP). That is, NAT hides an entire (private network) address space behind a single IP address in the public domain address space.

NAT keeps track of the "state" of the network connections and uses *translation tables* whose entries can be filled by the network administrator (static NAT or port forwarding). This allows traffic originating in the public "external" network to reach selected hosts in the private "internal masqueraded" network. In particular, Port Address Translation (PAT) translates TCP or UDP connections made to a host and port on an outside network (Internet) to a host and port on an inside network (LAN) by mappings different internal IP addresses to

different outside ports. This way a single external IP address is used for many internal hosts; almost as many as there are ports: over 64000 internal hosts.

Host-to-Host NAT-Traversal (NAT-T). NAT changes the source or destination address in the packet header (and adjusts the checksums). In particular, PAT attaches to an IP packet a new IP address and source port. IPSec authenticates (and encrypts) the data packet, but the Network-Address Translation in-between breaks authenticity:

- The encapsulated address of the source computer (in the payload, the actual data) does NOT match the source address of the IKE packet because it is replaced by the address of the NAT device. Any change to the IP addresses (what NAT is all about) lets IKE discard the packet:
- In addition, the IP addresses and ports are encrypted in IPsec. Depending on the encryption level, the payload and in particular the headers are encrypted in IPSec ESP. NAT cannot access this encrypted information, therefore cannot exchange neither addresses nor ports.

Because ESP does not use ports that could be "translated", it fails on networks that use PAT (like common home routers) and only a single client in the local network can establish a VPN tunnel. Therefore connections between hosts in private TCP/IP networks, which use NAT devices; for example, of peer-to-peer and VoIP applications, cannot be established.

NAT Traversal (**NAT-T** or UDP encapsulation) solves this incompatibility between NAT and IPSec: After detecting one or more NAT devices, NAT-T adds a layer of User Datagram Protocol (UDP) encapsulation to IPsec packets, so they are not discarded after address translation. RFC3947 defines the negotiation during the IKE phase and RFC3948 the UDP encapsulation (both from 2005). (According to loc.cit.: "Because the protection of the outer IP addresses in IPsec AH is inherently incompatible with NAT, the IPsec AH was left out of the scope of this protocol specification.") For NAT-Traversal, three ports must be open on the NAT device:

- UDP port 4500 (used for NAT traversal),
- UDP port 500 (used for ISAKMP/IKE), and
- IP protocol 50 (ESP).

NAT-T encapsulates both IKE and ESP traffic within UDP(source and destination) port 4500:

1. If a NAT device has been determined to exist, NAT-T will point all ISAKMP/IKE packets change from UDP port 500 to UDP port 4500.
2. NAT-T encapsulates the Quick Mode (IPsec Phase 2 IKE) exchange inside UDP 4500 as well.
3. NAT-T adds a UDP header, which encapsulates the IPSec ESP header.

This encapsulating UDP packet is NOT encrypted; therefore the NAT device can change its addresses and process the message.

Endpoints. The VPN can connect:

- gateway-to-gateway; the simplest case, where both endpoints are directly accessible with hard-coded addresses and ports, for example, on the same LAN or both publicly accessible.
- host-to-gateway (Remote Access). This is the common case of one of the two parties being behind a gateway; for example, a notebook of a sales representative or a home office computer, behind a (public of private) router that connects to a public server (for example, the company's central server) via the Internet which can be reached under a fixed IP address or domain on the Internet.
- host-to-host (Peer-to-peer). This case is less common. Network Address Translation (NAT) allows an Internet Protocol(IP) network to translate public IP addresses into private ones (of hosts behind a gateway); often by mapping many private IP addresses to a single public one by assigning different ports to them (Port Address Translation, PAT). While sometimes possible, generally direct NAT-to-NAT connections are infeasible on modern networks because most NAT routers are strict about randomizing the port (that is, port address translation as needed for NAT), making it impossible to coordinate an open port for both sides ahead of time. In this case, both ends will usually forward through an intermediary bounce server. Instead, a signaling server (such as STUN) must be used that stands in the middle and communicates which random source ports are assigned to the other side. (Session Traversal Utilities for NAT, STUN, is a standardized protocol for such address discovery including NAT classification. STUN allows applications to discover the public IP address

and TCP/UDP port mappings that they can use to communicate with their peers.) Both clients make an initial connection to the public intermediate "signalling" server, then it records the random source ports and sends them back to the clients. (This is how WebRTC works in modern P2P web apps.) Even with a signalling server and known source ports for both ends, sometimes direct connections are impossible because the NAT routers are strict about only accepting traffic from the original destination address (the signalling server), and will require a new random source port to be opened to accept traffic from other IPs (for example, the other client attempting to use the originally communicated source port). This especially happens in cellular networks, and is where NAT-T (see below) comes in.

Underlying Transport Protocols. A VPN connection commonly uses one of the following two protocols underneath:

- either **TCP** (Transmission Control Protocol):
  - the more reliable, but slower option.
  - useful for obfuscating VPN traffic to look like regular HTTPS traffic which has less chance of being blocked.
- or **UDP** (User, or, jocular, Unreliable, Datagram Protocol): Packets are sent without any confirmation; that is, no guarantee that sent data arrived correctly. This duty is shifted to applications that use the protocol (for example, VOIP applications). The faster and preferable option for connecting a VPN, if the above two restrictions that TCP circumvents do not apply.

  UDP: Internet Protocol used for fast transport of data across a TCP-IP network due to the absence of reliability checking;

Ports. A **Port** is a software, rather than a hardware concept. It is a number (between `0` and `65535`) that stands for a data channel into and out of a computer in a network. The header of a packet of the Transmission Control Protocol (TCP) (or the User Datagram Protocol, UDP) contains a source and destination port number. This TCP (or UDP) packet is encapsulated in an Internet Protocol

(IP) packet, whose IP header contains a source and destination IP address. An **Internet Socket** can be defined as the pair of an IP Address and TCP Port.

Port numbers fall into three distinct ranges:

- the Well-Known Ports (`0` – `1023`);
- the Registered Ports (`1024` – `49151`), and
- the Dynamic (or Private) Ports (`49152` – `65535`).

Some *Well-Known* (or *Dedicated*) ports are dedicated to certain protocols, for example, port 80 usually to the HTTP protocol for retrieving web pages. Their numbers are assigned by the Internet Assigned Numbers Authority (IANA). They can be used only by system processes (or by privileged users' such as root).

Table 27: Well-Known Ports of well-known protocols

port	protocol
13/tcp	Daytime Protocol
17/tcp	Quote of the Day
21/tcp	FTP: The file transfer protocol - control
22/tcp	SSH: Secure logins, file transfers (scp, sftp) and port forwarding
23/tcp	Telnet, insecure text communications
25/tcp	SMTP: Simple Mail Transfer Protocol (E-mail)
53/tcp	DNS: Domain Name System
53/udp	DNS: Domain Name System
79/tcp	Finger
80/tcp	HTTP: HyperText Transfer Protocol (WWW)
88/tcp	Kerberos Authenticating agent
110/tcp	POP3: Post Office Protocol (E-mail)
119/tcp	NNTP: used for usenet newsgroups
139/tcp	NetBIOS
143/tcp	IMAP4: Internet Message Access Protocol (E-mail)
443/tcp	HTTPS: used for securely transferring web pages

Table 28: Well-Known Ports of well-known protocols

port	protocol
21/tcp	FTP: The file transfer protocol - control
22/tcp	SSH: Secure logins, file transfers (scp, sftp) and port forwarding
25/tcp	SMTP: Simple Mail Transfer Protocol (E-mail)
53/tcp	DNS: Domain Name System
53/udp	DNS: Domain Name System
80/tcp	HTTP: HyperText Transfer Protocol (WWW)
143/tcp	IMAP4: Internet Message Access Protocol (E-mail)
443/tcp	HTTPS: used for securely transferring web pages

IPsec. Internet Protocol Security (**IPsec**) is a stack of protocols that secures Internet Protocol (IP) communications by authenticating (and optionally encrypting) each packet (over public and insecure networks). It is mainly used for VPNs and is, at least in the business market, the most established protocol. IPsec was developed by the Internet Engineering Task Force (IETF) as an integral part of IP version 6. Recommend official sources to gain an overview of the entire IPsec protocol suite are the documentation roadmap RFC2411 and its security architecture RFC4301. (See also Friedl (2005) for an illustrated informal guide.)

IPsec as a VPN offers Interoperability with IP protocols: It operates at the Internet Layer of the Internet Protocol Suite (comparable to network layer in the OSI model). Other common Internet security protocols, such as Secure Sockets Layer (SSL), Transport Layer Security (TLS) and Secure Shell (SSH), operate on a higher application layer protocol. This makes IPsec more flexible, because applications can ignore IPsec in contrast to the other higher-layer protocols.

Tunnel Mode. **Tunnel mode** is usually used between gateways through the Internet and connects two networks between two gateways. The end devices themselves connected via the two networks do not have to support IPsec. The security of the connection is only provided on the partial route between the two gateways. Tunnel mode encrypts the whole IP packet. A new external IP header

is used. The IP addresses of the two communication end points are located in the inner protected IP header.

Tunnel mode: encrypts the whole IP packet.

Transport Mode. Transport mode is usually used when the final destination is not a gateway. It uses an additional IPsec header between the IP header and the transported data. It is less secure than tunnel mode as it only encrypts the data portion but leaves the original IP addresses as plaintext.

Tunnel mode: encrypts the data portion but leaves the original IP addresses as plaintext.

IPsec Protocols. IPSEC essentially consists of the Internet Key Exchange (IKE) and Encapsulated Security Payload (ESP) protocol. IKE is the technical implementation of the Internet Security Association and Key Management Protocol (ISAKMP) framework. IKE uses UDP at port 500 for the initial key exchange (IKE) and port 50 for the IPSEC encrypted data. ESP (for NAT traversal) uses UDP port 4500 and TCP port 10 000.

Internet Key Exchange (**IKE**):. Establishes a common secret key

- either manually through the exchange of public keys,
- or automatically by the certificates from a trusted certificate server.

The cryptographic Internet Security Association and Key Management Protocol (ISAKMP) defined by RFC 2408 describes the key exchange protocol, but does not specify the used cryptography. IKE implements the ISAKMP and establishes a mutual secret key by the Diffie-Hell. IKEv2 is a tunneling protocol that is standardized in RFC 7296 (a joint project between Cisco and Microsoft) and it stands for Internet Key Exchange version 2 (IKEv2). It extends IKE and simplifies configuration and connection establishment. The first version of IKE came out in 1998 and version 2 in December 2005.

To be used with VPNs for maximum security, IKEv2 is paired with IPSec. In comparison to other VPN protocols, the single most important benefit of IKEv2 is its ability to reconnect quickly after VPN connection loss. In particular useful on mobile devices, which usually support IKEv2 (natively).

1. IKE Phase 1 establishes a secure connection channel. This includes the authentication between the parties, either with certificates or keys shared before. This secure connection channel is then used to securely negotiate the parameters from Phase 2 between the VPN partners.
2. IKE Phase 2 negotiates the encryption and authenticity parameters with which the actual data is secured. After negotiating the Security Associations (SAs; a unidirectional secure flow of data between two gateways defined by a destination address, a Security Parameter Index (SPI) and a security protocol) in IKE phase 2, (usually) the ESP protocol is used to transport the encrypted data.

   Internet Key Exchange (**IKE**): uses UDP at port 500 for the initial key exchange, either manually or automatically by certificates

Authentication Header (**AH**):. Provides authentication and protection against replay attacks, but no confidentiality. That is, the user data is not encrypted and can therefore be read by anyone. AH protects the invariant parts of an IP datagram: IP header fields that can be changed by routers on their way through an IP network (for example, TTL) are not considered. If routers with activated Network Address Translation (NAT) are passed on the way through the network, then they change the actually invariant parts of an IP datagram, and authentication is therefore no longer possible. Thus NAT and AH are incompatible by design! Instead, ESP is possible, which essentially superseded AH.

> Authentication Header (**AH**): Provides authenticity, but no confidentiality. Essentially superseded by ESP.

Encapsulated Security Payload (**ESP**):. ESP (specified in RFC 3948) encrypts all critical information by encapsulating the entire inner TCP/UDP data packet into an ESP header. An IP protocol like TCP and UDP (OSI Network Layer 3), but without port information like TCP/UDP (OSI Transport Layer 4). Unlike Authentication Header (AH), ESP in transport mode does not provide authenticity for the entire IP packet. In Tunnel Mode, where the entire original IP packet is encapsulated and a new packet header added, ESP protects the whole inner IP packet (including the inner header) while the outer header remains unprotected. It provides:

- confidentiality (by encryption),
- authenticity (detection of tampering by hashing and every party is who it claims to be by certificates), and
- protection against replay attacks (by nonces).

  Encapsulated Security Payload (**ESP**): encrypts all critical information by encapsulating the entire inner TCP/UDP data packet. An IP protocol like TCP and UDP (OSI Network Layer 3), but without port information like TCP/UDP (OSI Transport Layer 4).

Other VPNs. IPsec, thanks to its long development history:

- is secure (was proved and improved time and again: (However, leaked documents by Edward Snowden, a former U.S. National Security Agency (NSA) employee suggest that the protocol has been deliberately weakened by the NSA. Therefore, commercial implementations as black boxes of IPsec are no longer considered trustworthy.)
- is standardized:
  - VPN gateways have special IPsec processors for high performance
  - IPsec is implemented in devices and operating systems of many manufacturers The IPSec protocol is as an extension to the IP stack implemented in the operating system (kernel); therefore, unlike OpenVPN, no additional software must be installed.

However:

- it is comparatively difficult to configure. Still, since the introduction of IKEv2, IPSec has caught up.
- its authentication protocol IKEv2 is easier to block than other protocols such as OpenVPN due to its reliance on fixed protocols and ports. Ikev2 protocol only works on standard ports, which are commonly blocked on many corporate, school, and public networks.

Other options are:

SSL-VPN. An alternative to IPsec is SSL-VPN that builds on SSL/TLS. Whereas IPSec provides network security as a whole, SSL VPN's only to certain applications and for remote client access.

- The most important advantage of SSL-VPN is a quasi client-less operation: IPSec software has to be set up on all client machines before being able to remotely connect, Whereas with SSL, the remote user only requires a web browser (and possibly installing a browser plug-in). SSL VPN is accessed via a web portal front end after a secure HTTPS connection has been established between the client and server. From here the user (usually an employee) can access the authorized (usually enterprise) applications. (However, if other applications are to use this connection, then a browser plug-in, such as Java or ActiveX, is needed, which could contain security holes.)
- The IPSec protocol is often blocked in public networks, where SSL is usually always open.

OpenVPN. **OpenVPN** is a popular unstandardized open-source VPN protocol over the UDP or TCP protocol that uses TLS for key exchange (and OpenSSL for encryption). It supports dynamically assigned IP addresses behind NAT gateways. Using TLS, it is incompatible with IPSec. It is implemented as software which is

- is stable and secure, thanks to the use of OpenSSL,
- runs on all common operating systems, like Windows, Linux, macOS, Solaris, OpenBSD and Android, and
- can scale up to thousands of clients.

In comparison to IPsec:

- OpenVPN software is implemented as software applications whereas IPsec in the operating system kernel and commonly on special hardware. Therefore, OpenVPN is more portable, but often needs additional software installation and is slower.
- as an open source solution the sofware VPN OpenVPN is a cost-effective alternative to commrcial IPsec implementations.
- OpenVPN needs one (UDP or TCP) port whereas IPsec up to four (50, 500 and 4500 for the IP protocols ISAKMP, ESP, AH, NAT_T). Firewalls let OpenVPN's (UDP or TCP) packets through, but often block those

of IPSec: Whereas OpenVPN traffic resembles that from an HTTPS connection, for IPSec the firewall either needs to be aware of (or ignore) packets of the IP protocols ISAKMP, ESP, AH, NAT-T).
- For key exchange, OpenVPN uses the universal TLS protocol whereas IPsec the custom IKE protocol.

WireGuard.

WireGuard: a minimalist and modern open-source VPN software built into Linux kernel 5.5 (and above).

Wireguard is a minimalist and modern open-source VPN protocol over the UDP protocol implemented in software. WireGuard

- is simple, user-friendly and easy to set up,
- is secure thanks to latest cryptographic algorithms and best practices; for example, the key exchange uses perfect forward secrecy (that is, every connection session uses a different key pair).
- has short source code (initially around 4000 lines in comparison to hundreds of thousands in, say, OpenVPN).
- is relatively young, and thus lacks all the years of security audits established VPNs such as IPsec and OpenVPN have gone through.
- only allows UDP on IPv4 or IPv6; TCP support is missing. In contrast, OpenVPN, also offers TCP and thus works in an environment where only TCP/80 and TCP/443 are open, such as public Wi-Fi networks. (Third party or anyway additional code is required to use TCP as the tunneling protocol); (DSVPN is a Dead Simple VPN in the minimalist spirit of WireGuard that was made to address this common use case of a client on an untrusted and restricted network connecting to a VPN server.)
- does not verify the identity of the server by certificates. For authentication, Wireguard uses a public/private key pair (whereas, for example, OpenVPN (by default) a username with password). For example, to generate the key-pair, the command:

```
wg genkey | tee privatekey | wg pubkey > publickey
```

creates the two files `publickey` and `privatekey` (which should be generated on the device that requires the private key and then the public key distributed, rather than the other way round.)

- cannot manage IP addresses dynamically; the client's addresses are permanently assigned and visible on the VPN server: The client needs to be permanently assigned an IP address that is uniquely linked to its key on each VPN server. A user's IP address could be found out by an attacker (say by WebRTC) and then matched with records from a VPN provider (obtained, say, by theft or legal enforcement). For this reason, many providers refrain from using WireGuard for fear of their customers' privacy (despite zero-log policies).

VPN Software. To securely use a shared VPN, one must trust its operator and users: though most users are well-behaved citizens, a single one possibly not, so that, say under law enforcement, all network traffic eventually could be scrutinized. To set up one's own VPN, there are several software options:

SoftEther. `SoftEther` ("Software Ethernet") VPN is free and open-source,

- runs on Windows, Linux, Mac, FreeBSD and Solaris, and
- Easy to establish both host-to-gateway and host-to-host VPN.
- can use many popular VPN protocols such as SSL-VPN (HTTPS), OpenVPN and IPsec, and
- supports NAT traversal via SSL-VPN Tunneling to run VPN servers behind firewalls by using HTTPS, so that even deep packet inspection (that looks at the metadata as well as data) is unable to detect SoftEther's VPN transport packets.

openvpn-install. `OpenVPN` is complex; see for example the overview Archwiki (2020). The shell script `openvpn-install` at Nyr (2019) simplifies the setup of a VPN server on a UNIX operating system for the inexperienced user. It starts from entering the following one-liner in her terminal:

```
wget https://git.io/vpn -O openvpn-install.sh && bash openvpn-install.sh
```

Algo. `Algo` is a set of Ansible scripts (a tool to automatize the set up of computers on a network) that simplifies the setup of an (IPSEC) VPN. It uses only the software necessary and the most secure protocols available, works with common hosting services, and does not require client software on most devices.

Self-Check Questions.

1. On which layer of the DoD model is the IPsec protocol? 1., *2.*, 3., or 4. layer.
2. Which transport protocol underlie the IPsec protocol? TCP, UDP, TLS, HTTPS
3. Which one of these two protocols the is faster, but less reliable, one? TCP or UDP?
4. How many ports uses IP sec to establish a secure connection? 1., 2., *3.*, or 4?

13.3 Transport Layer Security

Transport Layer Security (**TLS**) and its predecessor, Secure Sockets Layer (**SSL**), are cryptographic transport protocols that provide authentication, confidentiality and authenticity for data sent over a reliable transport protocol, typically TCP. It encrypts data in both directions and (almost always) guarantees the identity of the server and (optionally) the client. Originally developed by the Netscape Corporation, it is now supported by all the major browsers and the most common security protocol used on the World Wide Web.

> **TLS/SSL**: cryptographic transport protocols that provide authentication, confidentiality, and authenticity for data transmitted over a reliable transport, typically TCP

For an application programmer, TLS (and SSL) provide a protocol that can be accessed almost like plain TCP. For a user, they establish a safe channel over the Internet to allow the user's private information, such as credit card or banking account numbers, to be safely transmitted via certificates, public and symmetric keys (indicated by a small padlock on the web browser's address bar).

TLS sits on top of the transport layer (in the OSI reference-model, layer 4) as it requires reliable data transfer. Therefore, it sits at least at layer 4 (and thus, in the IP reference-model, at layer 4).

It deems sensible to situate it at:

- Layer 4, because TSL itself is a transport protocol that provides, in addition to reliability, also security, such as confidentiality and authenticity to prevent eavesdropping, tampering, and message forgery.
- Likewise, one layer above, at Layer 5, the session layer, because its principal duty is to establish a secure key exchange to encipher all ensuing communication by the so-called TSL handshake that establishes authenticity and session-management, such as start-up and tear-down.
- Layer 6, the presentation layer that converts data, such as by encoding, compression or encryption. (However, this conversion usually concerns application data, not transportation data.)
- Or on the top (application) layer of the Internet Protocol Suites OSI (and DoD).

X.509 Certificates. Authentication and encryption is established by **X.509 certificate**. Principally, a file that contains the name, address and public key of the web site and is signed by a certificate authority. These are organized hierarchically and pass trust from the upper to the lower level; those at the top, which are trusted unconditionally, are called root authorities. In practice, this unconditional trust is achieved by the deployment of their self-signed certificates, for example, as part of an Internet browser installation.

> **X.509 certificate**: a file, signed by a certificate authority, that contains the name, address and public key of the Web site.

The signature of a `X.509` certificate is the encryption by the private key of the hash of the concatenation

$$(V, SN, AI, CA, TA, A, KA)$$

where

- `V` = version `X.509`,
- `SN` = serial number of the certificate,
- `AI` = algorithm identifier number,

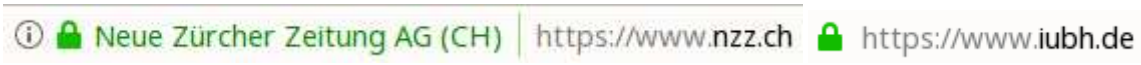
- `CA` = name of the certifying authority,
- `TA` = validity interval time of the certificate,
- `A` = name of subject, and
- `KA` = subject's public key.

The scheme of *hierarchical authorities* was created to establish trust through machines and has the comfort that the key exchange can be automated. However, trust is a human matter, and has as its Achilles' heel the (absolute) trust in (root) authority. The user must:

- trust that the key *public* belongs to the authority;
- trust that the *private* key to authority is not compromised;
- trust that authority does not *abuse* its power thus granted; for example, by charging high prices. To compare,
  - the recent (intermediate) authority `Let's Encrypt` provides free certificates and has a budget of 3 Million $,
  - whereas the company `GlobalSign` charges $224 a yer per certificate.
- trust that the authority will do its duty, for example, in the verification of the identity of the third party by the authority. To this end, each root certifier is subject to periodic audits (which leads us to ask whether the same goes for the auditors, the auditors' auditors, ... ?!).

The level of security is reflected by the shape of the padlock in the browser's address bar Whereas Let's Encrypt's certificates only verify via e-mail the ownership of the domain, companies offer (GlobalSign at $469.50) an Extended Validation (EV) certificate that verifies the identity of the owner:

- When issuing a `common certificate`, like a free one from `Let's Encrypt`, the verification is completely automated without any personal off-line verification. To obtain the certificate, access to the domain suffices. For example, this can be proved to the authority by uploading a file received by it.
- When issuing an `Extended Validation (EV) certificate` the verification of the site owner is done in person.



To prevent *DNS Hijacking*, for example, to make sure the site belongs to the intended owner (for example, to avoid confusion between `deutschebank.de` and `deutschbank.de`), it is important to verify in the address bar the padlock indicates an `extended certificate`. for example, the browsers `Firefox` and `Chrome` indicated it by the green color of the entity name before 2020, but more recent versions (> 70 respectively 77) abandoned it V. (2019) because it reportedly took up valuable screen estate, especially on mobile devices, and distracted the user.

By the common certificate, the user is only ensured to communicate with the owner of the domain, but not that it belongs to the company or organization that the site appears to represent. Thus, the common certificate,

- does prove that the owner of the private key is the owner of the server at this address; thus, it avoids, for example, a man-in-the-middle attack by `DNS Cache poisoning`, where the name address (for example, `deutschebank.de`) is resolved to the numeric `IP` address of another server.
- however, it does *not* prove that the server at this address belongs to the alleged (legal) person. thus, it allows a `MITM` attack by a user's confusion between the address and the (legal) person.

Handshake. The heart of the TSL/SSL protocol is the handshake that sets up the session encryption, whose steps are given. The definitive reference is RFC:5426; a splendid step-by-step illustration

- that traces every single exchanged byte is shown at Driscoll (2019).
- of the entire batch of protocols carried out on loading a web page is https://subtls.pages.dev/.

The following steps are the first steps between a client and the server, for example, an e-commerce site, to establish an encrypted connection (for example, to receive credit card data from the client).

1. The "Hello" between client and server, where the client proposes, and the server chooses, a *cryptographic package*; that is, the set of cryptographic algorithms,

- to authenticate himself:
  - a cryptographic checksum (`MD5`, `SHA`, ...), and
  - an asymmetric cryptographic algorithm (`RSA`, ...),
- to exchange a symmetric key: an asymmetric cryptographic algorithm (`RSA`, `ECC`, ...),
- to encrypt the communication: a symmetric algorithm (`AES`, `Camellia`, `RS4`, ...). Often, the server does not choose the most secure symmetric algorithm, but the most economical one.

For example, the cryptographic package `TLS_RSA_WITH_3DES_EDE_CBC_SHA` (identification code `0x00 0x0a`) uses

- `RSA` to authenticate and exchange the keys,
- `3DES` in `CBC` mode to encrypt the connection, and
- `SHA` as a cryptographic hash.

Besides this, both, the user and the server create a *nonce*, that is, a number used once, for a single use, which contains

- `4` bytes to set the time, and
- `20` random bytes,

to avoid a replay attack; that is, the re-use of authentications for other sessions.

2. The server identifies and authenticates itself by its *X.509* certificate, which contains (principally):

   - the server address and its public key (that uses the asymmetric algorithm initially agreed on to exchange the symmetric key),
   - to authenticate, the name of a certificate authority (such as a root certificate authority, for example, `VeriSign`), and its digital signature (that uses the cryptographic hash and asymmetric algorithm initially agreed on); that is, the encipherment, by the authority's private key, of the cryptographic hash
     - of the server address, and
     - of the server's public key.

   For example, in the picture, the server is `www.iubh.de`

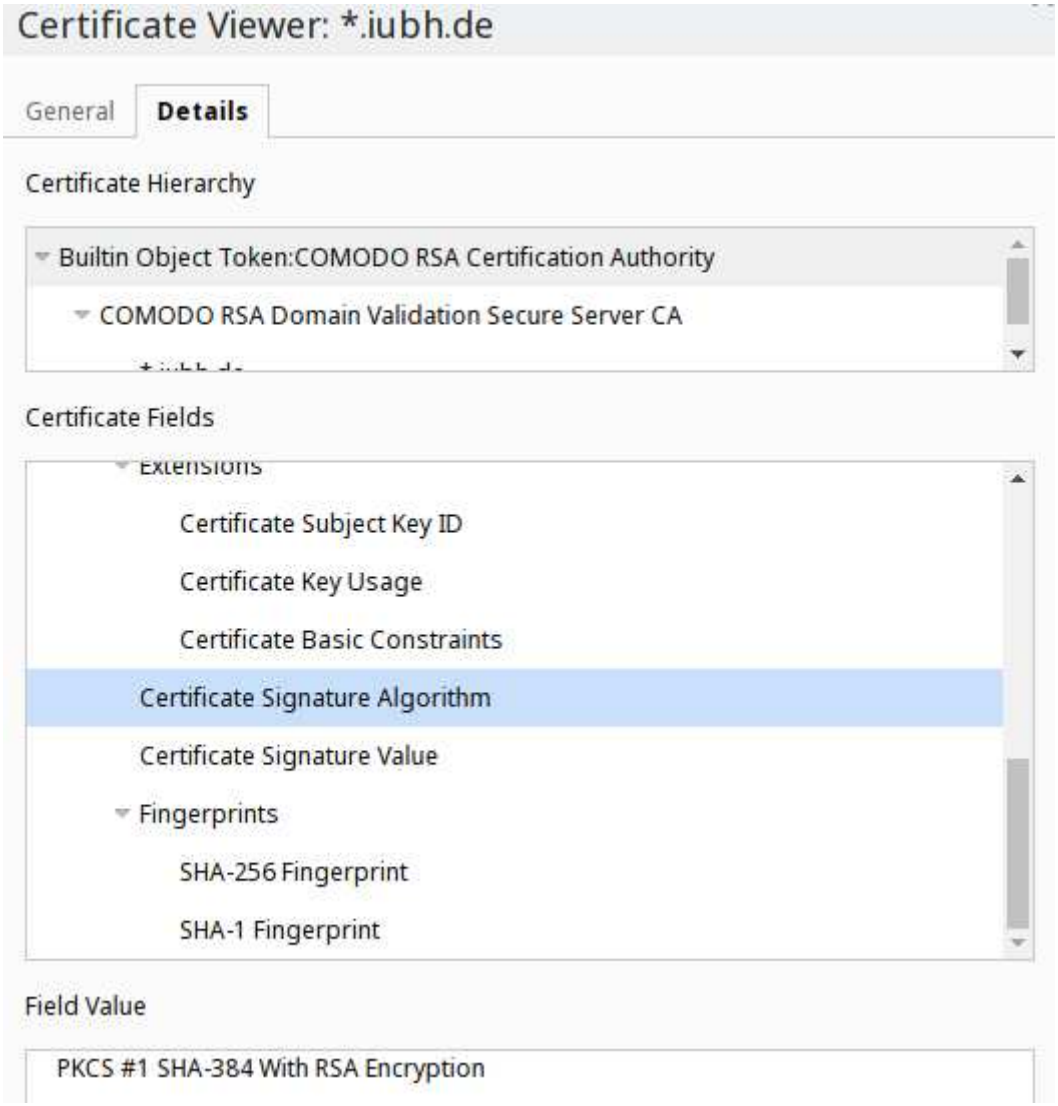


Figure 60: certificate `X.509`

- whose certificate is signed by the `Comodo RSA Domain Validation Secure Server CA` intermediate authority
- whose certificate is signed by the root authority `Comodo RSA Certification Authority`,
- whose certificate is self-signed (that is, signed by herself).

The client looks for the (root) certificate authority's public key indicated on the certificate (and which is usually included in the browser), and uses it to decipher this digital signature. If the result is the expected hash (that is, that of the server address and its public key), then

- the digital signature actually comes from the designated certificate authority, and
- the certificate authority trusts this server.

Since the client (or, more accurately, its browser) trusts the root authorities unconditionally, at this point it is certain the public key truly belongs to the target server. (Optionally, at this point also the client authenticates itself by a certificate).

3. The client

   - creates a *pre-secret*, a random (pseudo-)number of `48` bytes,
   - enciphers it using the public key (using the *asymmetric* algorithm initially agreed on), and
   - sends it to the server.

   The server

   - deciphers the *pre-secret* using its private key.

4. The client and server calculate the *secret* (`master secret`), a number of `48` bytes, by a function `PRF`,

   ```
   master_secret = PRF(pre_master_secret, ClientHello.random + ServerHello.random)
   ```

   which uses as input

   - the pre-secret, and
   - the "nonces", that were communicated during the "Hello",
     - from the client, and

– from the server.

The client and the server derive four symmetric keys (for the algorithm initially agreed on, for example, if it is `AES`, each one `16` bytes long) from the secret. (That each side has a different key is due to the best practice of using a different key for each different use.) Namely:

- `client_write_MAC_secret`,
- `server_write_MAC_secret`,
- `client_write_key`, and
- `server_write_key`.

Among these,

- the first two serve to check data authenticity, and
- the last two serve to encrypt the data.

Observation: Optionally,

- In addition to the mandatory authentication of the server (by a certificate), the client authenticates itself in the same way (by a certificate);
- the server and the client exchange an ephemeral *asymmetric* key to send the symmetric key:
  - while the permanent asymmetric server key almost always uses the `RSA` algorithm (and is used to sign the ephemeral encryption keys),
  - the ephemeral asymmetric key almost always uses an algorithm that is based on Diffie-Hellman key exchange (for example, `ECC` or `ElGamal`).

  This ensures that a compromised private key of the signing key stored on the server does not compromise the exchanged session data.

Self-Check Questions.

1. Above which layers of the OSI model does TSL sit? *1.*, *2.*, *3.*, and *4.* layer.
2. Above which layers of the IP (or DoD) model does TSL sit? *1.*, *2.*, *3.*, and *4.* layer.
3. Which cryptographic algorithms are agreed on during the TLS handshake and for which purposes?

- an asymmetric algorithm (such as RSA) to authenticate and exchange the keys,
- a symmetric algorithm such as AES to encrypt the connection, and
- a cryptographic hash algorithm such as SHA256.

13.4 Secure E-Mail

Most Internet protocols, among them the ones for e-mailing, such as POP3, IMAP and SMTP, initially ignored security concerns and exchanged all data in plain text. Since then, various approaches have surged to encrypt the data

- either only during *transport* (for example, TLS), more convenient, that is, easier to set up and use;
- from *end-to-end* (for example, S/MIME or OpenPGP), more secure: In end-to-end encryption, the data is encrypted and decrypted at the end points, the recipient's and sender's computers. Thus, an e-mail sent with end-to-end encryption is unreadable to the mail servers (hosted by, say, Hotmail or Gmail). Thus, for example, no third party (such as a sensitive organization that hosts its own e-mail server) can scan e-mail for malware; instead, it has to be done by the user('s computer) after decryption.

However, end-to-end protocols require additional effort and still only provide partial protection:

- require the user to set up pairs of public and private keys and publish the public keys,
- protect only the content of the e-mail, but no metadata, so that a third party can still observe who sent e-mail to whom, and

TLS. The most common e-mail protocol for encryption during transport is **STARTTLS**. It is a TLS (formerly SSL) layer over the plaintext protocol (such as IMAP$_4$ and POP$_3$ defined in RFC2595) that allows e-mail servers to encrypt all exchanged data between the servers as well as between servers and clients. However, certificate verification is optional, because a failure of verification is considered less harm than failure of e-mail delivery. That is, most e-mail is delivered over TLS provides only opportunistic encryption.

> **STARTTLS**: a TLS layer over the plaintext protocol that allows e-mail servers to encrypt all exchanged data between all relay servers.

Use of STARTTLS is independent of whether the e-mail's contents are encrypted or not. An eavesdropper cannot see the encrypted e-mail contents, but it is decrypted and thus visible at each intermediate e-mail relay. This is, the encryption takes place between the servers, but not between the sender and the recipient. This is convenient

- for the sender and recipient as encryption is automatic on sending e-mail.
- for the relays, as they can check the contents, for example, run virus scanners and filter spam, before delivering the e-mail to the recipient.

However, because every relay can easily read or modify the e-mail, this is also insecure. If the receiving organization is considered a threat, then end-to-end encryption is necessary.

Transport layer encryption using STARTTLS must be set up by the receiving organization. This is typically straightforward; a valid certificate must be obtained and STARTTLS must be enabled on the receiving organization's e-mail server. To prevent downgrade attacks organizations can send their domain to the 'STARTTLS Policy List'

S/MIME. The Secure Multipurpose Internet Mail Extension (**S/MIME**) is a protocol that standardizes public key encryption and signing of e-mail encapsulated in MIME using certificates emitted by an authority. S/MIME's IETF specification in Ramsdell (2009) enhances the Privacy Enhanced Mail (PEM) specifications of the 90s. Initially RSA public-key encryption was used, but since RFC:5753 (from 2010) ECC as well.

> **S/MIME**: An e-mail encryption protocol that uses certificate authorities to establish trust for key distribution.

Most e-mail clients, such as Microsoft Outlook or Mozilla Thunderbird, support S/MIME secure e-mail. Before use, one must install an individual key certificate, that is, before installing it:

1. create a (set of) key pair(s), and
2. send it to the certification authority (CA) (be it in-house or public) for them to sign it.

After successful verification, the certification authority creates a certificate of the key by signing it with its private signature key.

1. The certificate consists of the public key itself (including the algorithm used and other specifications), personal data (like the owner's e-mail address and name) and the signature (including the algorithm used and other specifications).
2. For the private signature key used for signing, there is a public verification key with which the signature can be verified.
3. There is also a certificate for this verification key of the certification authority, the CA certificate, which in turn is signed by a certification authority. In this way, a chain of CA certificates is created. The last link in this chain is called the root CA certificate. The root CA certificate was signed by the root CA itself and thus has to be unconditionally trusted.

Types of Certificate. Depending on the security class 1, 2 or 3, the certification authority checks more or less strictly whether the public key truly belongs to the applicant:

1. The certification authority (CA) authenticates the applicant's e-mail address. That is, the personal certificate verifies the owner's "identity" only insofar as it declares that the sender is the owner of the "From:" email address, in the sense that the sender can receive email sent to that address.
2. The CA verifies the personal data, such as the applicant's name, via copies of ID cards, extracts from the commercial register, ...
3. The applicant must identify himself personally at the CA.

Free Certificates. (Class 1) S/MIME certificates, which are mostly intended for private use, are available for free: For example, from `CAcert`, a non-commercial, community-operated CA. However, a common e-mail client or web browser does not recognize it as a trusted certificate authority. Thus, by default, bare of a manual installation of its certificate, a user who receives an e-mail with an `S/MIME` certificate signed by `CAcert` is warned that the origin of the certificate is unverified. Companies, which, in contrast to CAcert, are also recognized as trustworthy by common software:

- `GlobalSign` as a Free Trial PersonalSign 1 Certificate (valid for one month),

- `Secorio S/MIME` [3] (valid for one month; uses certificates from `Comodo`), or
- `WISeKey` as free secure email eID (valid for one year). (Attention: the "private-key" is generated on the server and is therefore known to the provider!)

Web of Trust. The *web of trust* is based on propagation of personal trust. It has the principal advantage that is a peer-to-peer system, that is, it is independent of any particular third party such as an authority. However, as its major inconvenience, it needs personal maintenance. It also

- reveals information such as
  - the social connections of each participant,
  - the time and place of interaction between the confidants.
- does not scale well; that is, the amount of storage for the web of trust of the whole world would be immense. For the level of authentication it provides, it makes no sense at these scales.
- The web is as safe as its least safe node; that is, a (easily) compromised node puts all its connecting nodes at risk.

  For example, so-called "key-sign parties" commonly gather strangers who sign their keys to each other. This is the opposite of what trust means: It should only be passed on when we know the owner of the key we sign!

  A compromised key can be used to sign any other key on the web of trust. For example, imagine that Alice wants to encrypt a message to Bob. Let Charles's key be compromised by a `man-in-the-middle`.

  - The `man-in-the-middle`
    1. creates a key in Bob's name;
    2. signs this key using Charles's key;
    3. send this key to Alice;
  - Alice, by the web of trust,
    1. if she trusts Charles,
    2. then,she trusts Bob (that is, the key of the man-in-the-middle!).

Even if this fraud is discovered by others (the idea of the web of trust is that it is impossible to fool everyone), some damage has already been done.

In practice, instead of the web of trust, it is more feasible to establish trust through another impersonal channel (by exchanging the fingerprints of public keys), for example, by post, by phone, by a messenger like WhatsApp, ...

OpenPGP.

OpenPGP: An e-mail encryption protocol that uses the web of trust for key distribution.

Since trust is a personal matter, the automatic unconditional trust of the user placed into root certificate authorities (principally companies) is unsatisfactory. The alternative **OpenPGP** protocol relies on the web of trust to communicate by e-mail. Still, few people use it: Most regard the (concrete personal) effort needed for maintaining the keys (which inherently requires the user's estimate of her trust in the keys) disproportionately large for the (abstract) benefit (of greater privacy and security) received. Unfortunately, conceivably because so few people use OpenPGP, the usability of dedicated programs has improved little in recent years:

- OpenPGP key management should be easier:
  - A public key directory server (for example, pgp.mit.edu) accepts any key without even confirming (by an activation e-mail) that the sender has access to the account of this e-mail address;
  - Several software solutions (presented below) have emerged that automate the key exchange in the OpenPGP protocol. Even if the user loses (in part) control over trust, gaining comfort helps to spread this protocol to laymen. The security they offer is Opportunistic Security as explained in Dukhovni (2014): as long as nobody's interested, it is safe; Otherwise, it is vulnerable to a man-in-the-middle attack.
- The OpenPGP protocol has its conceptual shortcomings:

– the lack of Perfect Forward Secrecy (PFS); even if the typical user wants to save the written and read emails and encipher them on her computer by one and the same key, it is safer to encrypt the e-mail for transport (through the sender and recipient server) by an ephemeral key.
– the vulnerability of the web of trust (as outlined above); Instead, it is nowadays possible to establish trust through other channels.
– a signature not only proves the origin of the message to its recipient, but also to third parties (who can use this proof against the sender);
– because the e-mail protocol by default does not use encryption, it is easily omitted by misuse of the e-mail client (as long as it does not exclusively send encrypted e-mails; in the distant future).

Off-the-Record. The **Off-the-Record** e-mail protocol specified in Borisov, Goldberg, and Brewer (2004) was designed to address the shortcomings of OpenPGP. It offers:

- Perfect Forward Secrecy: the exposure of private keys does not expose previously encrypted conversations (because an ephemeral key is created for the sending of each message and deleted upon receipt).
- Authenticity: guarantee of the identity of the correspondent (because every message is signed by the sender).
- Repudiability: inability to prove the origin of signatures after correspondence (by a so-called *group signature*, which shares the secret key for signing among all correspondents; due to this feature the need to use one key for each correspondent arises!)

Off-the-Record: e-mail protocol that provides Repudiability and Perfect Forward Secrecy.

The recent program opmsg at Krahmer (2019) (as an alternative to GPG presented below) implements this protocol (partially). It

- recommends (creating and) using one key (= persona in opmsg terminology) for each correspondent,
- insists on checking the public key fingerprint by another channel (and does not implement the web of trust),
- does *not* encrypt private keys with a password.

Examples of OpenPGP Programs. We present some programs that use the OpenPGP protocol, such as

- the command line program GPG to create keys and (de)encrypt and sign/authenticate for them,
- the extension Enigmail for the e-mail client Thunderbird, and
- the extension Mailvelope for the Internet browsers Firefox and Chrome to encrypt e-mails on web interfaces like gmail.com and Hotmail.com.

GnuPG. Gnu Privacy Guard, for short GnuPG or GPG, was written to offer *open and free* cryptographic methods to the public. It is a command-line program for

- encrypting and decrypting data (for example, e-mails), and
- creating and verifying digital signatures (to ensure authenticity of data).

It underlies the cryptographic functionality of many cryptographic applications with *a graphical user interface* (GUI applications). It is installed on most Linux distributions, and is under macOS and Microsoft Windows. The development of GPG by the German *Werner Koch* started in 1997 (and hasn't stopped to this day) to have a free alternative to the commercial e-mail encryption program Pretty Good Privacy (= PGP) by Phil Zimmermann.

- Version 1.0.0 was announced in 1999, in 2000 the German Federal Ministry of Economics and Technology sponsored a port to Microsoft Windows, and
- 2006 Version 2.0 was released, which brought significant changes in the architecture of the program.
- Up to this day, he is the leading developer and relied mainly on donations as its only source of income. For example, by early 2015, he was running out of resources and asked for financial help, which he received amply Angwin (2015).

Among the many functions, GPG creates a pair of keys for you, one public and the other private, where it lets you choose

- the algorithm (for example, RSA),
- the size (for example, 2048 bits),
- the validity (for example, one year),

```
→ ~ LC_ALL=en gpg --full-gen-key
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
@What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
@Key is valid for? (0) 1y
Key expires at Mon Jan 11 23:03:23 2021 CET
@Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

@Real name: Fulano
@Email address: fulano@bar.org
@Comment:
You selected this USER-ID:
    "Fulano <fulano@bar.org>"

@Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
```

Figure 61: Creation of a key pair in GPG on the command line; Koch et al. (2020)

- a password for the private key, and
- the identity: the name and e-mail address of the owner.

The *public* key is intended for disclosure, to encrypt and check signatures. The *private* key is stored and protected by a password, to decrypt and sign.

Enigmail. The Enigmail program is an extension to the graphic e-mail program Thunderbird that adds to it the functions to

- encrypt and decrypt,
- sign and check e-mail signatures.

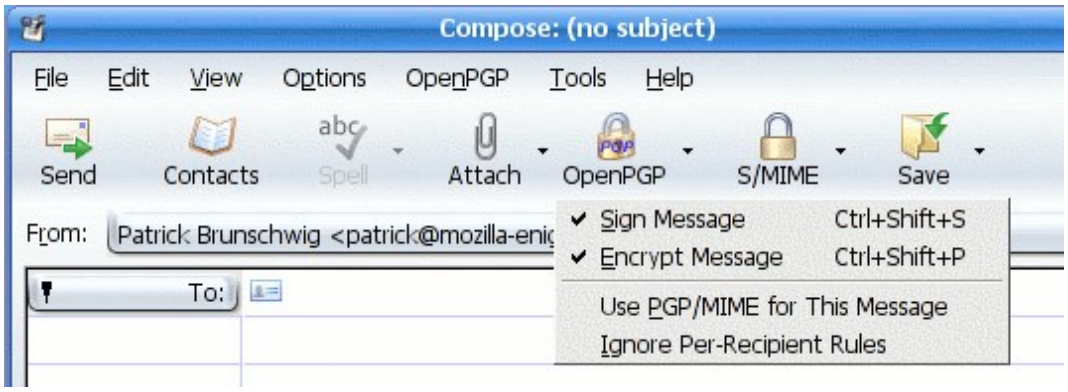


Figure 62: Enigmail; Brunschwig (2018)

The user can access these functions by buttons in Thunderbird itself; to implement these, it uses GnuPG underneath. According to Snipes (2019), the Thunderbird 78 release, planned for summer 2020, has the functionality for e-mail encryption and digital signatures using the OpenPGP standard built-in and replaces the Enigmail add-on, and therefore the dependency on GnuPG, whose installation was a hassle for beginners.

Mailvelope. Mailvelope is an extension for the browsers Firefox and Chrome, developed by the Mailvelope GmbH, which adds encryption and decryption functions to the web interface of common e-mail providers such as Gmail, Hotmail.com, and Yahoo!

For example, enciphering and deciphering messages (using the OpenPGP standard), files on your hard drive, and send encrypted e-mail attachments. Mailvelope is open source and based on OpenPGP.js, an OpenPGP library for JavaScript. It is comfortable, but comfort comes at the expense of security (and thus it is safer to use an e-mail client, such as Thunderbird); for example, it is potentially vulnerable to Cross-Site Scripting (XSS) attacks, where one site accesses local data stored for another

- with the user's consensus, the e-mail provider has access to the user's secret keys to synchronize them between devices (which is comfortable, but also risky).
- the Javascript language in which the extension is written is not the most appropriate for secure encryption; among others, is susceptible to the

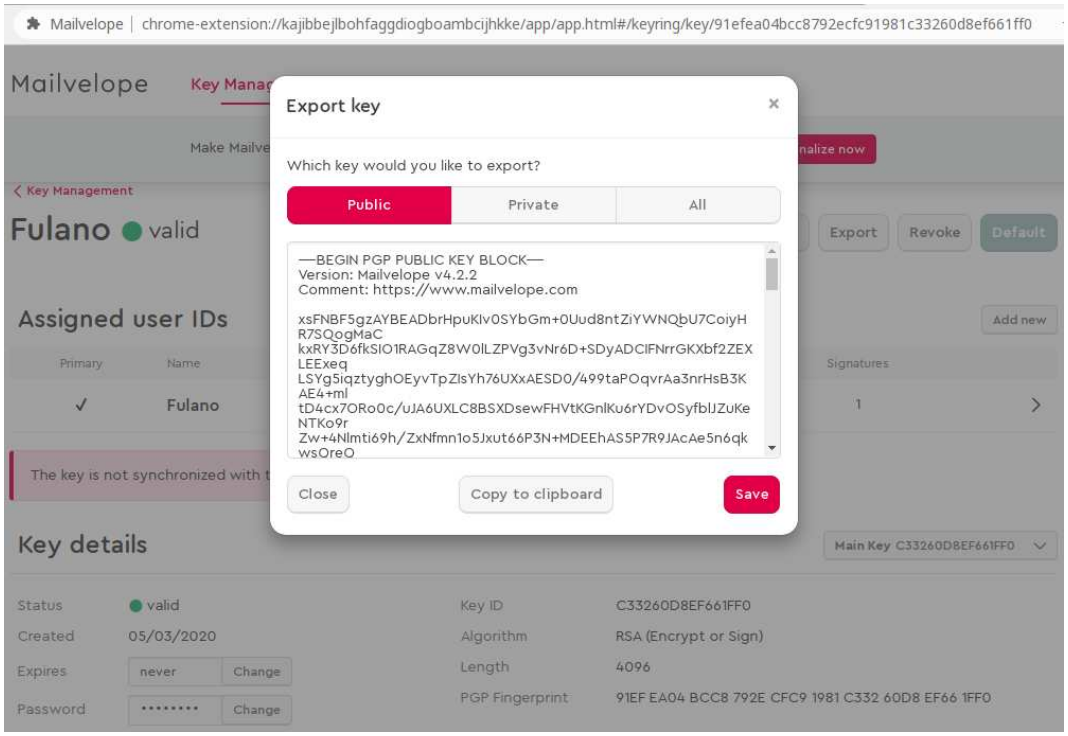


Figure 63: A public key created by `Mailvelope`; GmbH (2020)

following flaws:

- other scripts in the browser can read the secret key;
- it is impossible to delete the secret key from memory;
- in particular, private keys are saved in the browser storage, which is potentially vulnerable to Cross-Site Scripting (`XSS`) attacks, where one site accesses local data stored for another;

Automatic Key Exchange. Programs such as

- The extensions `AutoCrypt` for Thunderbird and `prettyeasyprivacy` for Outlook,
- the messenger `Delta-Chat` for `Android`,

offer as described in Dukhovni (2014) only *Opportunistic Security*: Protection against passive, but not active, eavesdroppers; that is, the encryption only protects the user as long as nobody is interested in her! Such a program,

precisely because of the lack of verification of the owner of the private key that corresponds to the public key, is vulnerable to the *man-in-the-middle* (MITM) attack in which the attacker interposes himself between the two communicating parties. For example, it is perfectly possible to use someone else's name on an e-mail or WhatsApp account. Therefore, encryption of the communication only prevents it from being read by a third party, but does not guarantee the other correspondent's identity. To avoid this attack, one must personally (or via another channel, for example, via telephone) check the fingerprint (a cryptographic check sum) of the other correspondent's public key. Tedious, but unavoidable.

Autocrypt. The program Autocrypt automates the exchange of public keys and is supported by many e-mail clients, such as the graphical e-mail client Thunderbird, the command-line client Mutt or K-9 Mail for Android. It was initiated by the European Union in response to the revelations by Edward Snowden (Krekel, McKelvey, and Lefherz (2018)).

Autocrypt automatically adds a line to the e-mail header (normally invisible to the user) that contains the sender's certificate (name, e-mail address and reference to her public key). This information is then automatically used by the recipient to encrypt her response. (Additionally, it is encrypted by her own key for secure local storage.) Therefore, all but the first e-mail exchanged between two Autocrypt users are encrypted.

Instead of the automatic usage of the Alice's public key by the recipient Bob, more secure would be if Bob's e-mail insisted on checking Alice's fingerprint through another channel (for example, by telephone).

prettyeasyprivacy. Another program to automatically send encrypted e-mails as Autocrypt that supports commercial programs such as the e-mail client Microsoft Outlook is prettyeasyprivacy. Founded by a private initiative, it is a graphical interface for GPG(4Win) that it installs and uses underneath for its cryptographic functions,. A convenient feature for us humans is to use so-called safe words instead of hexadecimal encoding to verify the fingerprint of a public key, that is,

- instead of transmitting 40 hex characters like 72F0 5CA5 0D2B BA4D 8F86 E14C 38AA E0EB,

Figure 64: Functionality of Autocrypt; Autocrypt Team (2019)

- correspondents can verify it by five words of their mother tongue, for example, by ocean contamination goose arenas survey.

Delta-Chat. The application Delta-Chat (available under Linux, Windows, macOS, Android and iOS) https://delta.chat/ uses the user's e-mail account to send automatically encrypted instant messages. It uses the same open protocol as e-mail (IMAP, which is old with many deficiencies, but is the established time-tested standard with a large ecosystem). Therefore:

- it does not depend on relaying all data through the vendor's servers (compared to many other applications that send automatically encrypted instant messages, such as WhatsApp),
- it is compatible with recipients who do not use Delta Chat as they still receive the messages sent in Delta-Chat by e-mail.

This interoperability, for example, that a hotmail.com user can communicate with another gmail.com user, is called *federation*. The dependency on a single company brings the following problems:

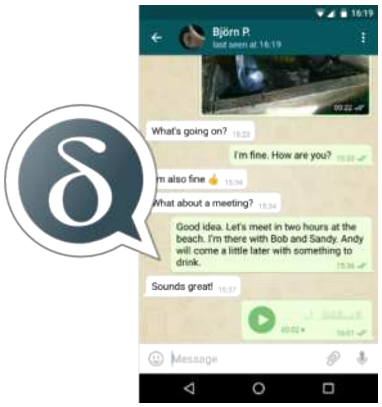


Figure 65: Delta-Chat; Merlinux GmbH (2018)

- the user has to place all her trust in this company, about which she usually knows little on a personal level, where trust is established, for example:
  - the company could exploit the compiled users' (meta-)data stored on their servers for its own profit;
  - even if not, then it could change its mind about its business model.
- the company is as a single point of failure, for example:
  - the company's servers could be compromised,
  - the government could suspend or block these servers (which for WhatsApp has occasionally happened, for example, in Brazil http://www.dw.com/pt-br/whatsapp-volta-a-ser-suspenso-no-brasil/a-19413134 and continues to happen in China).

That being said, the meta-data between the e-mail servers is still unencrypted. However, the user can consciously choose a mail server which does not exploit the users' meta-data; for example, she sets up her own server or pays a monthly fee to a trusted provider.

Conversations. The messenger Conversations proposes a modern XMPP protocol that uses less meta-data than the IMAP e-mail protocol. (Apparently the messenger Signal uses less meta-data than Conversations, but almost all servers are maintained by the vendor Open Whisper Systems itself, in contrast to Conversations. After all, the only secure solution is to run one's own server!).

It is a secure messenger, but unfortunately little established; for example, there are many IMAP mail servers, but few use XMPP.

Self-Check Questions.

1. How does secure e-mail via TLS compare to S/MIME and OpenPGP? TLS encrypts only during transport (but on no relay) while S/MIME and OpenPGP from end to end.
2. How does authentication to establish trust compare between S/MIME and OpenPGP? S/MIME trusts in certificate authorities while OpenPGP only in personally authorized keys.
3. What kind of attack is automatic key exchange via the OpenPGP protocol susceptible to? A man-in-the-middle attack.
4. What kind of security does automatic key exchange via the OpenPGP protocol offer? Opportunistic Security, Some Protection Most of the Time, as defined in Dukhovni (2014).

13.5 Secure DNS

The Domain Name System (**DNS**) is a distributed database analogous to a phone book of the Internet.

> **DNS**: database of all Internet domain names distributed on hierarchically organized servers over the Internet.

DNS. DNS translates human-friendly alphabetic Internet domain address, such as https://www.ongel.de, to a computer-friendly numeric IP address, such as `194.6.193.105` (as can be found out by the Unix command-line program `nslookup`). This Internet-domain address takes the form of

- the domain name of a machine,
- followed by a top-level domain (TLD),
- separated by dots (periods).

For example, `ongel.de` has the domain name `ongel` and the TLD `de`. Domain and TLD names were initially registered and governed by Inter Network Information Center (InterNIC), a cooperation between the US government

and the company Network Solutions Inc, which managed the registration and maintenance of .com, .net, and .org top-level domain names. In 1998 the US government liberalized the process of registration and set up the ICANN that administers Registrar companies for registering domains; for example, the company VeriSign registers domains ending in the TLDs .com and .net. Generic Top-Level-Domains (gTLD) such as .com are more strictly controlled by the ICANN than country-code Top-Level-Domains (ccTLD) such as de as a concession to state sovereignty. (An alternative Network Information Center is OpenNIC, which adds its own top-level domains such as .pirate, .geek or .libre to those from ICANN and operates free DNS servers.)

Fully Qualified Domain Name. A Fully Qualified Domain Name (**FQDN**) (such as www.ongel.de) is a unique (worldwide) name (to address an IP address) on the Internet and which can be freely chosen under the rules determined by the Internet Corporation for Assigned Names and Numbers (ICANN):

- Every FQDN ends in a *top-level domain* (such .de),
- a subdomain (such as www) can be added to a FQDN (such as ongel.de) by prepending it with a dot (to result in, say, www.ongel.de). Common subdomain labels are, for example, www. for Web servers and mail., smtp., pop3. and imap. for (outgoing and incoming) mail servers.
- Each domain (or *label*, such as ongel) may contain at most 63 characters and the entire FQDN (such as www.ongel.de) at most 255 characters.

The FQDNs are put in correspondence with the IP address by the entries of name servers.

> Fully Qualified Domain Name (**FQDN**): a unique (worldwide) name (to address an IP address) on the Internet

The labels of a FQDNs are represented as nodes of a tree. A FQDN is then a path of the tree:

1. The highest node is the null or root label that represents an empty name.
2. Below the highest node are those that represent a top-level domain (such as de).
3. Below the nodes on the first level are those that represent a domain (such as ongel)

4. Below the nodes on the second level are those that represent a subdomain of the domain (such as `www`) ...

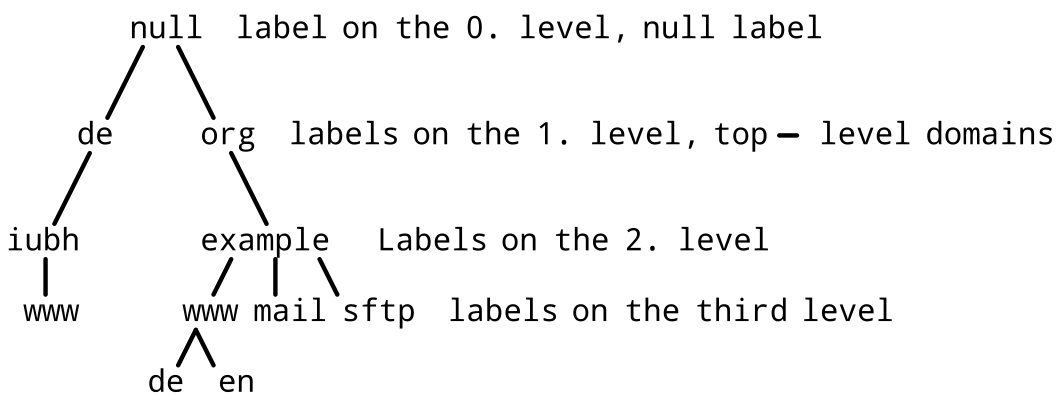


Figure 66: FQDNs represented as paths of a tree

DNS Hierarchy. DNS is defined in Mockapetris (1987) and uses the UDP or TCP protocol on Port 53. DNS servers use a set of databases distributed on servers over the Internet that are organized hierarchically.

> DNS **zone**: the subset of the DNS hierarchy which is described by a **zone file**, a list of entries that map a FQDN name to its IP address.

A DNS **zone** is the subset, often a single domain (say `ongel.de`), of the DNS hierarchy which is described by a **zone** (text) **file**; a list of entries called resource records (**RR**s) that map a FQDN name to its IP address. The zone file format, as originally specified for the Berkeley Internet Name Domain (BIND) software, is used by most DNS server software. It contains

- usually the `ORIGIN` keyword, that specifies the starting point for the zone in the DNS hierarchy; (If omitted, then the starting point is inferred by the server software from the reference to the zone file in its server configuration.)
- exactly one Start-of-Authority RR (SOA-RR), usually at the beginning of the file, that contains
  - the e-mail address of an administrator of the zone file (say `admin@ns.ongel.de`),

  - the primary authoritative name server for the zone (say `ns.ongel.de`), and
  - the frequency with which the RR are updated from the master name server (say once a day);

- at least one name-server RR, usually following the SOA-RR, for the authoritative name servers for this zone (among which figures at least the primary authoritative name server);
- possibly one or more NS-RR that delegates the DNS resolution of a subdomain to another name server (say resolution of all subdomains below `www.ongel.de` is delegated to `ns.www.ongel.de`).

root server: is the name server that resolves all FQDNs of a top-level domain (TLD)

A root name server, or **root server** for short, is the name server that resolves all FQDNs of a top-level domain (TLD) such as `.com`.

History. The concept of a domain name server came around in the 80s: As the size of computer networks grew, it became increasingly difficult for humans to keep track of which machine corresponded to which number. Before DNS, names were resolved into IP addresses by a list (such as `/etc/hosts` on Unix operating systems) that had to be available on every computer on the Internet. Changes were first made manually on a master server and then downloaded by the clients. As the number of IP subscribers increased, this procedure became increasingly unwieldy. In 1983 Paul Mockapetris specified the Domain Name System (DNS), the first DNS software JEEVES was developed and the first three DNS root servers went into operation.

In the early 80s, the DNS software `BIND` for UNIX (Berkeley Internet Name Domain) was developed at the University of Berkeley, whose version 4 became the worldwide standard. Further development of the software was taken over, for a short time, by the company DEC and then by Vixie Enterprises led by Paul Vixie. Starting with version 4.9.3, `BIND` became the responsibility of the non-profit organization `ISC` (Internet Systems/Software Consortium). Version 8 was completed in 1997. In 1999 `ISC` commissioned Nominum Inc. to develop version 9, which has been the standard since 2007 and forms the backbone of the worldwide Domain Name System.

DNSsec.

DNSsec: DNSsec protocol to authenticate the resolutions of a domain name to an IP address.

The DNSsec protocol allows the client to authenticate its requested resolution of a domain name to an IP address. For this, the resolution is signed on registration by the DNS server responsible for the zone file. This provides authenticity of the resolution, but neither confidentiality (that is, the request and its resolution are unencrypted) nor authentication of the DNS server. To this end, further secure DNS protocols, such as DNScrypt, DNS-over-TLS and DNS-over-HTTPS have been devised and will be discussed below.

The DNSSEC protocol as extension to the DNS protocol was standardized in RFC 25352 in March 1999. However, this version proved in practice unsuitable due to elaborate key management. The roll-out of DNSSEC was delayed till the completely rewritten version RFC 40331, RFC 40343, and RFC 40354 was published in 2005, which obsoleted RFC 25352. In May 2010 DNSSEC was introduced on all 13 root servers; in July the root zone key was published. In the meantime, 90% of the top-level domains are signed with DNSSEC and marked as signed in the root zone. A few are still testing DNSSEC without an entry in the root zone. The distribution of DNSSEC at domain level for some TLDs is now 50% or more. On average, about 10 % of domains validate.

Chain of Trust. The *chain of trust* along the hierarchy of the DNS architecture ensures that the public key in the resolver's DNSKEY record in the zone file is correct by automatic successive signature verification up to the trust anchor. The **trust anchor** is the first key in the chain of trust, the key upon which the chain resides, the public key associated with the root name server (which is entered manually). To establish trust along the chain, every server below must know this key. See Sandia Corporation (2014) for a tool that visually analyses the DNSSEC authentication chain for a domain name and its resolution path in the DNS namespace.

Currently, the private DNSSEC key for the root zone is managed at two US locations. Critics accuse ICANN, after it had chosen the American company Verisign as its exclusive signing partner, of putting the independence of the Internet at risk through exclusive DNSSEC key management in the USA.

trust anchor the key upon which a trust chain resides.

DNSSEC defines four levels of trust in a record:

- Secure: A full chain of trust, all signatures up to a trust anchor can be verified.
- Insecure: A valid partial chain of trust, that is, one that terminates before a trust anchor by the lack of a signed key.
- Bogus: An invalid chain of trust, for example, by signatures using unsupported algorithms, by missing data or attempted tampering.
- Indeterminate: The chain of trust has no trust anchor; the default operation mode.

Protocol. DNSSEC is a protocol that extends DNS. It therefore includes all entries (Resource Records; RRs) of a DNSSEC packet. RFC 4034 specifies the addition of the four RRs: DNSKEY , RRSIG , NSEC and DS.

- DNSKEY The DNSKEY record is passes a public key between the resolver and the name server. This public key is the one associated with the private key with which the authority server will sign hashes of RRSET records. The resolver will use the public key in the DNSKEY record to authenticate the message of the authority server by verifying its signature.
- RRSIG record contains the signer's name and the signature of the record sent by the authority server; the signature that the resolver will later verify. There is one RRSIG record for each zone record in the signed zone file.
- The NSEC record is used for proof of non-existence. It contains the name of the next authority domain or point of delegation for the request and the records that exist for that name.
- Next Domain Name field: Contains the name of the next authority domain for the request according to the RRsets order relationship. For the name following the last RRset, the name of the parent field is returned.

Signatures. DNSSEC authenticates each resource record (RR) by a digital signature, as follows: Owner of a RR is the primary authoritative name server as defined in the Start-of-Authority RR (SOA-RR) entry of the zone file.

1. For each zone a ZSK (a pair of a public and private key) is generated:
2. The public key is added to the zone file in the DNSKEY-RR.

3. Each RR of the zone file is then (hashed and) signed by the private zone signing key.
4. The resolver obtains the authoritative name server's public key by the DNSKEY record.
5. The resolver can then verify authenticity of the passed RR by verifying whether the decrypted (hash of the) RR using the public key (corresponding to the primary name server's private key) matches (the hash of) the RR.

If no server matches the request, then DNSSEC proves that no such RR exists uses a new type of record that sends the name of the first (in alphabetic order) existing domain.

Key Management.

Zone Signing Key (**ZSK**) respectively Key Signing Key (**KSK**): the key to sign the (hashed) RRs respectively to sign the ZSK.

The more encrypted data, the more information to infer the keys used to encrypt the data: Because of the large amount of data encrypted by the key to sign the (hashed) RRs (the so-called Zone Signing Key; **ZSK**), this "working" key must be renewed regularly. To avoid network administrators having to renew the ZSKs too often, the Key Signing Key (**KSK**) has been introduced, which is longer and has to be renewed less regularly. While the public keys are stored in DNSKEY-RRs, the KSK and ZSK private keys are stored offline. In more detail:

- The Zone Signing Key (ZSK) is the "working" key, the (private) key to sign a zone, that is, all the RRs requested by other servers which know the corresponding public key to authenticate the replies. The ZSK is usually valid for one month.
- The Key Signing Key is (the private) key to sign a zone key (ZSK), that is, to sign a DNSKEY record. The KSK is part of the chain of trust. It is longer than the ZSK and usually valid for 13 months.

Virtual Hosting. Virtual hosting makes it possible to host multiple DNS names on a single (web) server, in particular, on the same IP address. This reduces server maintenance and the number of IP addresses (which are scarce

in IPv4; every IP address assignments must be justified to the regional Internet registry).

> **Virtual hosting**: allows multiple DNS names to be hosted on a single server (usually a web server) on the same IP address.

For example, a server receives requests for the domains, www.ongel.de, www.ongel.org and www.ongel.net, which resolve all to the same IP address. But for www.ongel.de, the server sends the html file /var/www/user/de/site/index.html, while he responds accordingly for the top level domains .org and .net. Likewise, two subdomains of the same domain can be hosted together: for example, mail.ongel.de and ftp.ongel.de.

The distinction which domain on the server was requested is made at the application level: For example, the requested domain is sent, always unencrypted,

- in the SMTP protocol, during the SMTP handshake.
- in the HTTP protocol, by the HTTP header field Host sent by the client as is obligatory since HTTP/1.1 (which is commonplace today, but whose inclusion cannot be enforced by the server).
- in the HTTPS protocol, for a suitable assignment of certificates to domains, both, client and server, must support Server Name Indication (SNI):

Server Name Indication (SNI). The biggest gripe with name-based virtual hosting is that of multiple secure websites running TLS/SSL. In HTTPS, the TLS handshake happens before the server has received the HTTP headers, and it can therefore not send the certificate for the requested domain name. Therefore, the HTTPS server can only serve one domain on a given IP address.

The TLS protocol extension Server Name Indication (**SNI**) defined in RFC 6066, addresses this problem by sending the domain name during the TLS handshake. This allows the server to choose the virtual domain earlier and thus send the certificate corresponding to the DNS name asked for. Therefore, with SNI-aware clients, a single IP address can be used to serve a group of domains without a common certificate.

> **SNI**: extension of TLS that sends the domain name during the TLS handshake.

By 2013, most browsers and TLS libraries implemented SNI, but about 20% of users still had software that was incompatible with SNI 5. However, by 2019, this number had fallen below 3%.

Observation. The host name is not sent encrypted in SNI. (ESNI, Encrypted Server Name Indication, as drafted in Rescorla et al. (2018), is supposed to solve this security hole.) SSL/TLS with `SNI` reveals more information than SSL/TLS without SNI, since the server certificate then transmitted also contains the domain(s) for which it was issued in plain text. If instead the certificate were valid for multiple domains, then the full requested host name would not be transmitted.

Reverse DNS. A reverse DNS lookup (or resolution), rDNS, queries the Domain Name System (DNS) to determine the domain name associated with an IP address. That is, it is the reverse of the usual DNS lookup of an IP address from a domain name. It uses (so-called PTR, pointer) records through the reverse DNS database of the Internet rooted in the .arpa top-level domain. Although the informational RFC 1912 (Section 2.1) recommends that "for every IP address, there should be a matching (PTR) record", not all IP addresses have a reverse entry, for example, when a web server hosts many virtual domains.

Secure DNS. In DNSsec authenticates DNS replies by a signature from the authoritative DNS server on which the domain name was registered; not necessarily the DNS server that answered the DNS request. DNSsec thus offers authenticity of the DNS entries but:

- neither confidentiality as all data exchanged is unencrypted, (However, usually the DNS server of Internet provider is used for DNS resolution, who can see which IP addresses the user visits to anyway.)
- nor authentication as the correct DNS server does not need to authenticate. (However, this problem is solved by TLS/SSL certificates; in particular, over HTTPS.)

Secure DNS protocols

- to prevent the former, encrypt all exchanged data, and
- to prevent the latter, authenticate the DNS server to the DNS client.

The three principal contenders for encrypted and authenticated DNS queries and replies are:

- DNSCrypt protocol (supported by Cisco OpenDNS, among others),
- DNS-over-TLS (or DoT), DNS resolution over TLS (supported by Cloudflare, Google, and OpenDNS), and
- DNS-over-HTTPS (or DoH), DNS resolution over HTTPS (supported by Cloudflare and Google).

The DNS server SecureDNS.eu, operated by Dutchman Rick Lahaye, supports all three of these: DNSCrypt, DNS-over-TLS and DNS-over-HTTPS.

Neither Windows, macOS nor Linux support encrypted DNS queries by default at the time of writing. However, Android 9, supports DoT. Firefox supports encrypted DNS queries by DoH and Google, at the time of writing, is testing DoH over Chrome. DNSCrypt has less backing from the big companies.

Privacy Consideration. Because all the other sent metadata in particular usually include the requested DNS, encrypting DNS queries mainly shifts it from one party (the DNS server without encryption) to another (the DNS server with encryption). Since most requests on the Internet leak the domain name (for example, reverse DNS lookup or by protocol headers, see below), the merit of secure DNS queries lies less in the protection against eavesdropping, but more in the authentication of the DNS server. However, authentication (of the requested domain) is usually already provided by TLS. On the downside, centralization of all DNS queries (away from the ISPs) to the DNS provider permits their bundled processing: While the data between the client and the server is encrypted during transport, it is decrypted at each end.

Reverse DNS Lookup. The domain name can be inferred from the IP address by reverse DNS lookup: The Internet service provider (ISP), who connects the client to her destination by its IP address, can still (to a good measure) figure out the client's DNS destination by Reverse DNS lookups: (Though this is not completely reliable, for example, the web hosting service can lodge many domains at the same IP address by virtual hosts or move a domain from one IP address to another.) Thus, not only the DNS server can track the visited websites, but the ISP as well. Therefore, for privacy it is preferable to use the

ISP's DNS server, because the ISP already implicitly has the information passed to the DNS server.

Headers. The domain name is sent as plaintext:

- in the HTTP protocol, by the HTTP header field Host sent by the client as is obligatory since HTTP/1.1 (which is commonplace today, but whose inclusion cannot be enforced by the server).
- in the HTTPS protocol, for a suitable assignment of certificates to domains, both, client and server, must support Server Name Indication (SNI) that sends the DNS name of the domain as part of the TLS handshake (so that the server can send the certificate corresponding to the requested DNS name on an IP with various domains.)
- Likewise for e-mail servers over STARTTLS.

DNSCrypt. In DNSCrypt the client, instead of relying on X.509 certificates emitted by trusted certificate authorities as found in web browsers, has to explicitly trust a public signing key used to verify a set of certificates (retrieved using conventional DNS queries). These certificates contain short-term public keys used for key exchange, as well as an identifier of the used cipher suite. Clients should generate a new key for every query; servers should rotate short-term key pairs every 24 hours.

> **DNSCrypt**: unstandardized secure DNS protocol without trusted certificate authorities.

According to DNSCrypt Team (2019a), the **DNSCrypt** protocol sits over the TCP (mandatory) and UDP (optionally) transport protocols. The protocol has been around since 2013, but is not standardized (say, in an RFC).

Protocol Steps.

1. The client sends a (non-authenticated) DNS query to a DNSCrypt resolver, which encodes the certificate versions supported by the client, as well as a public identifier of the provider requested by the client.

2. The resolver responds with a set of signed certificates, that must be verified by the client using a previously distributed public key, the *provider public key*. Each certificate includes a validity period, a serial number, a version that defines a key exchange mechanism, an authenticated encryption algorithm and its parameters, as well as a short-term public key, known as the *resolver* public key. (A resolver can offer multiple algorithms and resolver public keys.)
3. The client picks a certificate (that with the highest serial number among the valid ones that match a supported protocol version) and encrypts by the resolver public key to send an encrypted query, which includes a (magic) number (to identify the chosen certificate) and the client's public key.
4. The resolver
    1. decrypts the query using the private key that corresponds to the resolver public key of the certificate,
    2. verifies the query, using the client public key, and
    3. encrypts the response using the client public key.

Features. Advantages:

- DNSCrypt's is among all secure DNS protocol the one closest to normal DNS.
- By using the UDP port 443, address are resolved relatively fast and little likely to be blocked by a firewall.

Disadvantages: - DNSCrypt does not rely on trusted certificate authorities, but the client has to trust a chosen public signing key. That signing key is used to verify certificates that are retrieved via conventional (unencrypted) DNS requests and used for key exchange - While many DNS services use DNSCrypt (such as CleanBrowsing, which blocks adult content domains, and Cisco OpenDNS, which blocks malicious domains), recent DNS services (including Google, Cloudflare, and Quad9) opted instead for DNS over TLS and DNS over HTTPS.

DNS-over-HTTPS (DoH). **DNS-over-HTTPS** was specified in RFC8484 and uses HTTPS to be indistinguishable from any other HTTPS traffic and is thus practically never blocked. Firefox has added support for DoH through the DNS servers of Cloudflare. (Google plans on testing DOH with Chrome.)

DNS-over-HTTPS: DNS-lookup protocol that uses HTTPS.

DNS-over-TLS (DoT). DNS-over-TLS, thanks to the IETF standardization in RFC7858, is the most widely supported in software; For example, Android 9 supports DoT. DoT clients authenticate the service they connect to using Simple Public Key Infrastructure (SPKI) which is a joint effort via the IETF to simplify traditional X.509 PKI and a supported standard of establishing trust.

DNS-over-TLS: DNS-lookup protocol that uses TLS.

DoT is plain DNS traffic within a TLS connection using a dedicated port 853 (and occasionally on port 443). That is, up to the encryption by TLS, it is the same as DNS over TCP/IP instead of UDP. Since TLS is the encryption protocol used to secure almost all other Internet services, the technology is well understood and constantly improved.

Comparison. See DNSCrypt Team (2019b) for a comparison (to DNSCrypt).

DNSCrypt versus DNS over HTTPS:.

- Advantages:
  - DNScrypt has been well-tested, and many servers support the protocol.
  - DNSCrypt is faster (as it runs over UDP)
- Disadvantages:
  - Has a complete specification since 2013, but the specification hasn't been submitted to the IETF yet.

DNS over TLS versus DNS over HTTPS:.

- Advantages:
  - DNS over TLS is considerably faster.
- Disadvantages:
  - With DNS over HTTPS an eavesdropper will not be able to tell whether DNS queries are being made or web content is being retrieved, unlike with DNS over TLS.

DNS over TLS versus DNSCrypt:.

- Advantages:
  - DoT is a proposed IETF standard.
  - Straightforward: encrypts standard DNS requests over TCP by the established TLS protocol.
- Disadvantages:
  - DoT uses the dedicated port 853, which is usually closed by firewalls and can thus be blocked easily
  - TLS encryption slows the DNS query down (by a factor two or three).
  - If a provider retires a certificate and starts using a new one, there is no clean way to update the SPKI data on clients other than cutting and pasting it into the configuration file.

Self-Check Questions.

1. Which port uses the DNS protocol? *DNS is defined in Mockapetris (1987) and uses the UDP or TCP protocol on Port.*
2. Which data reveals the server visited by the user other than her DNS request?
   - *IP address*,
   - *in the* HTTPS *protocol, the Server Name Indication (SNI)*, and
   - *in the* HTTP *protocol, the header field* Host.

3. List three protocols to securely look up domain names:

    - *DNSCrypt protocol* ,
    - *DNS-over-TLS (or DoT), DNS resolution over TLS*, and
    - *DNS-over-HTTPS (or DoH), DNS resolution over HTTPS*

Summary

Internet Protocol Suite. The protocols that standardize communication on the Internet can be stacked in to layers: the **Open Systems Interconnection (OSI) reference-model** with *seven* layers (more of a theoretical abstraction), and the **Internet Protocol Suite** with *four* layers (the practical standard). The layers are ordered according to how much structure the processed data has, the higher the layer, the closer to the user's applications. The two most important Internet protocols (and those that were defined first) are the Transmission Control Protocol (TCP), and the Internet Protocol (IP), that specify how data should be formatted, addressed, transmitted, routed and received at the destination.

IPsec. A virtual private network (VPN) is a private network made up of two or more closed (spatially separate) networks connected via an open network (such as the Internet). The IPsec VPN uses one of two Modes: The transport mode establishes point-to-point communication between two end points, while the tunnel mode connects two networks via two gateways. In IPSEC tunnel mode, IP packets are encapsulated (tunneled) in other IP packets.

Transport Layer Security (TLS). Transport Layer Security (TLS) Protocol and its predecessor, Secure Sockets Layer (SSL), are cryptographic transport protocols that provide authentication, confidentiality, and authenticity for data transmitted over a reliable transport, typically TCP. Authentication and encryption is established by `X.509` certificate. Principally, a file that contains the name, address and public key of the web site and is signed by a certificate authority. These are organized hierarchically and pass trust from the upper to the lower level; those at the top, which are trusted unconditionally, are called root authorities.

Secure E-Mail. E-mail is encrypted: either only during *transport* (for example, TLS), more convenient, that is, easier to set up and use; from *end-to-end* (for example, S/MIME or OpenPGP), more secure: In end-to-end encryption, the data is encrypted and decrypted at the end points, the recipient's and sender's computers. Thus, e-mail sent with end-to-end encryption is unreadable to the mail servers.

SecureDNS. The Domain Name System (DNS) is a distributed database on the Internet that resolves human-readable domain names into machine-readable IP addresses. DNS offers neither privacy as all data exchanged is unencrypted, nor trust, as the DNS server does not need to authenticate. DNSsec offers authenticity by signing all DNS records, whereas more recent secure DNS protocols encrypt all exchanged data and authenticate the DNS server to a client.

Questions

1. How many layers does the OSI model have?

   - ☐ 4
   - ☐ 5
   - ☐ *7*
   - ☐ 10

2. How many layers does the TCP/IP reference model have?

   - ☐ 3
   - ☐ *4*
   - ☐ 5
   - ☐ 7

3. Which protocol is *not* part of the IPsec protocol family?

   - ☐ Internet Key Exchange (IKE),
   - ☐ Authentication Header (AH),
   - ☐ Encapsulated Security Payload (ESP), and
   - ☐ *Session Traversal Utilities for NAT (STUN)*

4. Which kind of (cryptographic) algorithm is *not* agreed on during the TLS handshake?

   - ☐ an asymmetric algorithm such as RSA,
   - ☐ a symmetric algorithm such as AES,
   - ☐ a cryptographic hash algorithm such as SHA256,
   - ☐ *an error-correction check sum such as* CBC.

5. Which security feature is neither part of the S/MIME nor OpenPGP protocol?

   - ☐ asymmetric encryption and decryption
   - ☐ digital signature
   - ☒ perfect forward secrecy
   - ☐ integrity verification

Required Reading

Read the carefully crafted illustrated guides on IPsec Friedl (2005) respectively TLS Driscoll (2019).

Further Reading

Read Dukhovni (2014) on opportunistic security.

14 Practical aspects of cryptology

Study Goals

On completion of this chapter, you will have learned ...

- where, why and how random numbers in cryptography are generated; in particular, the distinction between physical and pseudo-random number generation.
- how to ensure long-term security (in 10 to 20 years of time, for example, of health data) by
  - knowing the future strength of key lengths,
  - using means such as perfect forward security to ensure that exchanged data cannot be decrypted afterwards,
  - knowing about the possible future threats by quantum computing and how to counter them.
- best practices for using cryptography in application development, by
  - risk analysis, and
  - using proven common libraries rather than home-brewed solutions.
- to conform to legal regulatory requirements by using encryption for data protection, of utmost importance, for example, in health care,
- to be aware of government trap doors and security holes potentially kept secret by intelligence agencies

Even though the presented algorithms are secure in *theory*, in practice a lot may go wrong when implementing cryptography; thus, as a software developer, caution must be taken and best practices adopted:

Most importantly, use what is vetted by the test of time; for example, as a software developer, to use (open-source) software libraries that implement cryptographic functions such as encryption, decryption, signing and verification. Besides the cryptographic algorithms, critical is the implementation of the random number generator, which is notorious for exploits: As we saw, in ECC, elliptic curve cryptography, if the same ephemeral key , usually randomly generated, for signing is used twice to sign different documents by the same private signature key, then the ephemeral becomes known and reveals the secret signing key.

Even when well-known time-proven cryptographic open-source libraries, for example, OpenSSL can have security holes, such as the Heartbleed bug contributed by a PhD-student at the Fachhochschule Münster, that made common web servers reveal on request currently processed secret data, such as passwords or server keys. Even though quickly fixed, the question remains if in the meanwhile intelligence agencies exploit these, on top of back doors built into cryptographic software (usually on the behest of government agencies).

Even best cryptographic practice does not stop Moore's law that predicts a doubling of the computing power every 18 months that progressively weakens keys. Besides continual progress, there may be technological leaps, such as the quantum computer, which would break many common asymmetric ciphers such as Diffie-Hellman, RSA and Elliptic Curve Cryptography and more involved alternatives have been found.

14.1 Random number generation

Cryptography requires random numbers to generate

- Keys for (symmetric and asymmetric algorithms), and
- Nonces (Numbers only used ONCE), Salts and IVs (Initialization Vectors), numbers that are (usually) randomly generated, disclosed, used once in a cryptographic process to improve its security by making it unique. For

example, a session between a client and a server or a hash function used to store a password.

True randomness is critical for the generation of keys but less so and can be dispensable nonces (where sometimes a key is used as a nonce, see below), for which uniqueness is can be sufficient. For example,

- The GnuPG strong random number generator (for cryptographic keys) builds a pool of 600 entropy (physical disorder) bytes and hashes these by SHA-1.
- The GnuPG nonce generator adds 20 bytes containing the process ID number (PID) and the time in seconds and 8 bytes taken at random from a strong random number generator, and hashes these with SHA-1.

Secrets. A secret (that is, a secret sequence of bits, or, equivalently, a secret number) often must be generated by the computer; for example,

- if the secret needs to be generated automatically: for example, to extend the Diffie-Hellman key exchange to a public-key algorithm, the El Gamal algorithm generates for every plaintext to be encrypted or signed an ephemeral secret key. The popular signature algorithms DSA (Digital Signature Algorithm) and its analogue over elliptic curves, ECDSA, which the El Gamal signature algorithm underlies, generate an ephemeral key pair for every plaintext to be signed.
- if the secret needs to satisfy a specific format: for example, in RSA with modulus $N = pq$ for prime numbers p and q, one key E is a number without any (prime) factor in common with $\phi(N) = (p-1)(q-1)$.
- if the secret needs to be guaranteed to be random: To ease memorization, humans choose passwords with patterns.
- if the (bit) length of the secret needs to be significantly larger than what a retainable password can provide. For example, for decryption of a ciphertext encrypted using RSA to take as long as an exhaustive search of a 112 bit long secret key, the secret RSA key must be 2048 bits long, around 340 ASCII (Armor) letters.

For secrecy, it is necessary that the output of the generation is unpredictable; since the generation algorithm is usually known, its input must be unpredictable. This excludes, for example, using as input computer times or a number of a known sequence.

Uniqueness. For example, if the secret ephemeral key used by the signature algorithms DSA (Digital Signature Algorithm) and its analogue over elliptic curves, ECDSA is known, then the signee's permanent secret key can be inferred; that is, an attacker can forge the signee's signatures!

In particular, if two of the same signee's ephemeral public keys used for different documents coincide, then the secret ephemeral key used by either of these signature algorithms can be inferred. (For example, the standard (Java) library for the generation of a random number on Android generated repetitive random numbers, thus allowing to forge signatures of users of an Android Bitcoin app, see Ducklin (2013)) Therefore, in this case, the generated number must be unique.

Random Number Generators (RNGs). While true randomness of numbers are critical for security-sensitive applications and can be generated using hardware random number generators, pseudorandom numbers are often sufficient for less confidential ones, for example, for (probabilistic) experimental simulations such as the Monte Carlo method.

Pseudorandom Number-Generator (PRNG). A Pseudorandom Number-Generator produces sequences of numbers which appear independent of each other, that is, which satisfy statistical tests for randomness (which require careful mathematical analysis such as the BigCrush Test Suite L'Ecuyer and Simard (2007)), but are produced by a definite mathematical procedure, However, this apparent randomness is sufficient for most purposes The random number generators provided by most software libraries are pseudorandom. See "List of Random Number Generators — Wikipedia, the Free Encyclopedia" (2020) for a list of such pseudo-generators. An ancient influential and simple such is the Linear congruential generator, which for a multiplier a, offset c and modulus m produces the sequence $X_0, X_1, \ldots$ inductively given by

$$X_{n+1} \equiv aX_n + c \bmod m$$

Its successor that use linear feedback shift registers replace arithmetic in $\mathbb{Z}/m\mathbb{Z}$ by that in the binary polynomial ring $\mathbb{F}_2[X]$. An efficient one that passes the BigCrush Test Suite is, among others, Xorshift+ 128 , an adaption (to pass the test suite) of Xorshift that iteratively multiplies a nonzero initial n -bit string by an invertible matrix of order $2^n - 1$. Other generator that pass the test

suite are the hash function SHA-1 and the symmetric algorithm AES (with initial values).

Hardware Random-Number Generator (HRNG). A hardware random number generator is a computer device that generates random numbers from a physical process which is theoretically completely unpredictable, for example, thermal noise, voltage fluctuations in a diode circuit or, quantum optics (which can provide instant randomness). In Unix operating system, randomness is gathered from the devices /dev/random and /dev/urandom. GnuPG, in absence of these, uses process statistics, but also supports the hardware RNGs inside the Padlock engine of VIA (Centaur) CPUs and x86 CPUs with the RDRAND instruction.

Self-Check Questions.

1. List at least three uses of random numbers to generate keys:

    1. *purely random key*
    2. *long key*
    3. *specially formatted key*
    4. *automatically generated key*

14.2 Long-term security

How to ensure that data encrypted today will still be secure in the decades to come? This is in particular relevant for certain long-lived applications, for example, health data. In practice, the security of a cipher, and thus the recommended key sizes, relies foremost

- on its resistance to the most efficient (known!) methods of cryptanalysis (using a back door), and
- the computational effort needed to check all keys (taking the front door) by checking the decrypted output for probable patterns of a plaintext.

One has to take into account

- increasing computing power to apply the strongest known cryptanalysis algorithms, and

- new algorithms (less predictable) and devices (more predictable) on the horizon; in particular quantum computing.

Moreover, preventive measures can be taken by diversifying the keys used for encryption so that a single compromised key compromises as little ciphertext as possible: *Perfect Forward Security* generates a new key (pair) for every new session (that is, exchange of ciphertexts).

Key Lengths. World's fastest supercomputer, IBM's Summit (taking up 520 square meters in the Oak Ridge National Laboratory, Tennessee, USA) has around 150 petaflops, that is, $1.5 \cdot 10^{17}$ floating point operations per second. The number of flops needed to check a key depends for example, on whether the plaintext is known or not, but can be very optimistically assumed to be 1000. Therefore, Summit can check approximately $1.5 \cdot 10^{13}$ keys per second; thus, a year having $365 \cdot 24 \cdot 60 \cdot 60 = 31536000 \approx 3 \cdot 10^8$ seconds, approximately $4.5 \cdot 10^{21}$ keys a year.

To counter the increasing computing power, one prudently applies Moore's Law that stipulates that computing power doubles every other year. Therefore, every twenty years computing power increases by a factor $2^{10} = 1024 \approx 10^3$. Therefore, to ensure that in, say, sixty years, a key not surely be found during a yearlong search by world's fastest supercomputer at least $4.5 \cdot 10^{30}$ key combinations have to be used.

For a key of bit length n, the number of all possible keys is 2^n. If $n = 80$, then there are are $2^{80} \approx 1.2 \cdot 10^{24}$ possible key combinations. While this number is sufficient for now, the probability for the key to be found during a yearlong search by world's fastest supercomputer being around $1/250$, the projected fastest super computer in twenty years will likely find it in half a year. Instead, to be safe against worlds fastest yearlong supercomputing efforts in 40 years, a minimal key length of 112 is recommended.

AES. For the symmetric algorithm AES , the fastest known algorithm currently is exhaustive key-search, to try out all possible keys, whose complexity (= the number of operations) is 2^n . The minimal AES key length is 128 bits; that is, there are are $2^8 \approx 3.4 \cdot 10^8$ possible key combinations. Therefore the chance that world's fastest supercomputer in say, sixty years, finds the secret key is around

10^{-8} , a millionth percent. We conclude that the minimal AES key length is safe against brute-force attacks for the years to come.

Asymmetric Algorithms. In contrast to single-key cryptography whose cryptanalysis exploits statistical patterns, by its reliance on computationally difficult mathematical problems (that is, whose runtime grows exponentially in the bit-length of the input), the cryptanalysis of two-key cryptography is that of computational mathematics: to find an algorithm that quickly computes the solutions of the difficult mathematical problem. According to Arjen K. Lenstra (2006), for the classic asymmetric algorithms,

- the prime factor decomposition used in RSA, or
- the discrete logarithm in the Diffie-Hellman key exchange,

the fastest algorithm is the *General Number Field Sieve* whose number of operations, roughly, for a large number of input bits M, putting $N = \log(2)M$, is $L(N, 1/3, (64/9)^{1/3})$ where

$$L(N; r, c) := \exp(cN^r(\log N)^{1-r}).$$

To compare this to the number of operations 2^m required for exhaustive key-search of a key of bit length m, we put $n = \log(2)m$, equate both numbers of operations, and obtain

$$N\log(N)^2 \approx (9n/64) \approx n^3/7$$

which must be solved numerically for N; for example, for $m = 80$, that is, $n \approx 55$ we find $M = 1024$, that is, $N \approx 709$, to satisfy $\log(N)^2 = 43$; therefore $N\log(N)^2 \approx 30500$ and $55^3/7 \approx 24000$. That is, at least as much computational effort is needed for finding a private 1024 bit long key for RSA and Diffie-Hellman by the General Number Field Sieve as for finding a secret 80 bit long key (say, for AES) by exhaustive key-search.

For the logarithm over a finite elliptic curve, the fastest algorithm today is the generic baby step, giant step algorithm (or, slightly faster, Pollard's ρ -algorithm) that complexity is roughly $2^{\sqrt{n}}$. Therefore, for the computational effort for finding a private key of bit length N for Elliptic Curve Cryptography to be comparable to that for finding a secret key of bit length n (say, for AES) by exhaustive key-search, the key length must double, that is $N \approx 2n$.

By these formulas, we can calculate that

- the key length of a (private) RSA or Diffie-Hellman key must be at least 3072 bits to be as secure as that of an AES key of 128 bits, and
- the key length of an ECC key must be at least 256 bits to be as secure as that of an AES key of 128 bits.

These key sizes are therefore sufficient for the decades to come, assuming that no faster algorithm than those known is discovered to solve the underlying mathematical problem.

Quantum Computing. Quantum researchers hope to build computers that harnessing a phenomenon known as superposition where a quantum system is in many possible states before a measurement "collapses" the system into a single state:

Quantum Computer. A classical computer stores information in bits; each bit is either on or off. A quantum computer uses qubits, which can be "entangled", in-between on and off so that it can carry out multiple calculations at the same time and whose final output depends on the interferences generated by them. However, actually building a useful quantum computer has proved difficult:

- a quantum computer must keep its qubits *entangled* long enough to complete a computation;
- errors caused by inevitable interactions with the environment need to be filtered out and corrected; and
- since measuring a quantum system disturbs its state, reliable methods of extracting information must be developed.

Quantum States. The properties measured on one of one particle depend on the operations carried out on all the others: If a particle can take two states (denoted 0 and 1), a system of two particles can take for example states 00 or 11, or even a superposition of these states: For example, each of the two particles, considered in isolation, is measured randomly as a 0 or a 1, but the particles are "twins", that is the measurement of a state of one of the two particles forces the other particle one into the same state. The violation of a statistical inequality (predicted by John Bell in 1964 and verified experimentally, among others, by

Alain Aspect in 1982) proves that this is a characteristic of the particles and not due to a (hidden) link.

A physical quantity in quantum mechanics is called *observable* and is usually in a superimposed state; only in special (so-called *eigen*)states it has a uniquely determined (so-called *eigen*)*value*. In general, the eigenstate is the result of applying an observable to a (superimposed) state and choosing an eigenvalue.

Formally, a (superimposed) *state* is a complex unit vector, that is, a vector of length one with complex entries and an *observable* A is a (self-adjoint) operator on the vector space of all states. Because A is self-adjoint, there is a basis b_i such that $Ab_i = a_i b_i$ for real a_i; each b_i is an eigenstate and a_i its eigenvalue.

An eigenstate b_i is an eigenvector of A and interpreted as the possible outcome of a measurement for a given observable; A general state $|\psi\rangle = \sum c_i b_i$ is a linear combination of eigenstates. The complex coefficient c_i of b_i is called the *probability amplitude* of b_i and its absolute value $|c_i|$ in

$$0,1$$

is the probability of the measurement of b_i.

Quantum Bits. Whereas a digital computer processes bits, that can be in one of two states, 0 or 1, a quantum computer processes quantum bits (qubits) whose states superpose:

- whereas the state space, the set of all possible states, of a computer with n bits is given by the set of all strings

  $$Q_n := \{0,1\}^n = \{(0,\ldots,0),\ldots,(1,\ldots,1)\}$$

  of n bits, of which there are 2^n;
- the state space of of a quantum computer with n qubits is given by all **probability amplitudes** (or Schrödinger wave functions) on Q_n , that is, all strings of 2^n entries of complex numbers of unit length: a probability amplitude is a vector of unit length with complex entries indexed by Q_n , that is, a string $\lambda\colon Q_n \to \mathbb{C}$ of 2^n complex numbers

  $$\lambda = (\lambda_{(0,\ldots,0)},\ldots,\lambda_{(1,\ldots,1)})$$

  such that $\sum_{q\in Q_n} |\lambda(q)|^2 = 1$.

The basis states are those probability amplitudes that have a single nonzero entry of value 1, of which there are 2^n. The state whose entry of value 1 is at σ in Q_n is denoted by $|\sigma\rangle$.

Example. A single qubit superimposed between 0 and 1 is denoted by $a \cdot |0\rangle + b \cdot |1\rangle$ such that $a^2 + b^2 = 1$, the probability amplitude of two qubits together superimposed between 00, 01, 10 and 11 is denoted by

$$\alpha \cdot |00\rangle + \beta \cdot |01\rangle + \gamma \cdot |10\rangle + \delta \cdot |11\rangle ,$$

with $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$.

Each elementary operation on the state space is then described by a (unitary) matrix of 2^n columns (orthogonal to each other): each column is the probability amplitude obtained by applying the operation to the corresponding basis state.

Post-Quantum Cryptography. A quantum computer can solve many classical problems faster than a classical computer. For example,

- to find an item among an unordered list of N items on a classical computer, on average N/2 operations are needed, and one cannot do better than that. However, a quantum algorithm exists that achieves this in only about $\sqrt{N}$ operations!
- Simon's problem: given a function f that transforms n -bit strings into n -bit strings, find the nonzero bit-string s such that $f(x \oplus s) = f(x)$ for all n -bit strings x. To solve Simon's problem on a classical computer, one searches for a collision that needs around $\sqrt{2^n}$ evaluations of f. However, a quantum algorithm exists that achieves this in only about n operations!
- Shor's Algorithm Peter Shor of AT&T in 1994 in his article "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer" gave a quantum algorithm that factors a number N (for $n = \log N$) in $O(n^3)$ operations and $O(n)$ memory units. Instead, on a classical computer, solving the factoring, discrete logarithm (DLP), and elliptic curve discrete logarithm (ECDLP) problems takes subexponential time, around $O(N^{1/3})$ operations. Therefore, given a sufficiently large quantum computer, current public key encryption schemes are easily broken.

Shor's algorithm requires a quantum computer of around 8000 entangled qubits to factorize a 4096-bit RSA key, and is therefore not imminent.

Therefore, given a sufficiently large quantum computer, current public key encryption schemes are easily broken. (In compensation, quantum mechanical effects offer a new method of secure communication known as quantum encryption.) RSA, for example, uses a public key N which is the product of two large prime numbers. One way to crack RSA encryption is by factoring N, but with classical algorithms, factoring becomes increasingly time-consuming as N grows large; no classical algorithm that factors in $O((n)^k)$ operations for some k is known.

However, while a quantum computer can solve many classical problems faster than a classical computer, not all of them; the problems solved are all *subexponential*, that is, exponential in a root of the bit length n of the input (for example, in the cube root $n^{1/3}$ for prime factorization and discrete logarithm), whereas genuinely exponential problems, in particular problems that are NP-hard (such as finding the closest vector to a lattice) are conjectured to remain hard even on a quantum computer.

Lattice Ciphers: the Goldreich–Goldwasser–Halevi (GGH) Cipher. This cipher was published in 1997 by Oded Goldreich, Shafi Goldwasser, and Shai Halevi, and uses a trapdoor one-way function that relies on the closest vector problem which is NP-hard. Though this specific algorithm was later cryptanalyzed by Phong, it illustrates the principles on which modern withstanding lattice based ciphers stand, such as NTRU: Given any basis of a lattice, it is easy to generate a vector close to a lattice point by adding a small error vector to the latter. However, to return from this shifted vector to the original lattice point, a particular type of basis is needed.

The private key consists of

- a lattice basis b, that is, an invertible matrix with integer entries which consists of nearly orthogonal vectors; for example, a diagonal matrix that scales every basis vector by an integer multiple;
- a *unimodular matrix* u, that is, an invertible matrix with integer entries whose inverse has integer entries; that is, it transforms vectors with integer entries into vectors with integer entries. (Equivalently, an integer matrix whose determinant is ±1; in particular, it preserves volumes.)

The public key is the basis $\mathrm{B} = ub$ of the lattice L.

If a message m (a vector $(m_1, \ldots, m_n)$ of integers) and a public key B are given, then to encrypt the plaintext m :

1. compute the lattice point vector $v = m \cdot \mathrm{B}$,
2. choose a small error vector e, for example, one with entries in $0, \pm 1$, and
3. compute the ciphertext $c = v + e (= m \cdot \mathrm{B} + e)$.

If a ciphertext c and the private key consisting of a lattice basis b and unimodular matrix u are given, then to decrypt the ciphertext c :

1. Compute

$$c \cdot b^{-1} = (m \cdot \mathrm{B} + e) b^{-1} = m \cdot u \cdot b \cdot b^{-1} + e \cdot \mathrm{B}^{-1} = m \cdot u + e \cdot \mathrm{B}^{-1};$$

   While $m \cdot u$ is a vector of integers, the error term $e \cdot \mathrm{B}^{-1}$ is fractional.
2. Therefore the integral summand $m \cdot u$ can be distinguished from the fractional summand $e \cdot \mathrm{B}^{-1}$; remove the fractional error term $e \cdot \mathrm{B}^{-1}$ to obtain the integer vector $m \cdot u$.
3. Compute the plaintext $m = (m \cdot u) \cdot u^{-1}$

Code-Based Cryptography: McEliece's Cipher. Code-based post-quantum ciphers are asymmetric ciphers that are based on error correcting codes to transmit bits over a noisy channel: To avoid that Alice sends, say 01, but Bob receives 11, a simple solution would be that Alice repeats each bit thrice, 000111, and Bobs takes for each group of three the bit that appears most often: For example, 100110 would be decoded to 01. However, this encoding scheme is limited to one erroneous bit in each group of three bits. Instead, a linear code multiplies the bit vector v with a matrix M, that is, computes $w = v\mathrm{M}$. For any number of erroneous bits n, the matrix M can be chosen such that at most n erroneous bits in w are correctable (to v).

The first encryption scheme based on linear error-correcting codes was developed by McEliece in 1978 and is still unbroken; due to the (secure) public key size of around 500 kilobytes, it was shrugged off before the advent of quantum computing. More exactly, the algorithm uses the error-correcting code Goppa codes. Goppa codes are easy to decode, but distinguishing them from a general linear code is known to be NP-hard. The Post Quantum Cryptography Study

Group installed by the European Commission has recommended this type of cryptography against quantum computers.

The private key are three matrices, an error correction matrix c, and invertible matrices S and P; and the public key is $C = SCP$ that can correct n error bits.

To encrypt: compute $w = vC + e$ for an error term e containing n errors.

To decrypt:

1. knowing P^{-1} and c allows one to remove Se;
2. then apply S^{-1}.

Hash-Based Cryptography. Hash-based cryptography is based on the security of cryptographic hash functions rather than on the hardness of mathematical problems. Lamport showed how to derive a one-time signature scheme from any one-way function, such as a cryptographic hash function, and Merkle improved on it by the Winternitz one-time signature (WOTS) in Merkle (1990); "one-time", because a private key can only be used securely once:

Let k be the private key; fix an integer L. The public key is $K = \mathrm{Hash}^{L}(k)$, the L -fold nested application of the hash function to k.

$$\mathrm{Hash}^{L}(k) = \mathrm{Hash}(\mathrm{Hash}(\ldots(\mathrm{Hash}(k)))).$$

The signature S of a message, given by an integer $M < L$, with the private key k is the M -fold nested application of the hash function to k, that is, $S = \mathrm{Hash}^{M}(k)$. The signature S is checked by the equality

$$\mathrm{Hash}^{L-M}(S) = \mathrm{Hash}^{L}(k) = K.$$

This simple scheme is insecure. For example, signatures can be forged: From the signature S of M, one derives the signature of $M + 1$ by

$$\mathrm{Hash}(\mathrm{Hash}(K))^{M+1} = \mathrm{Hash}(S).$$

Therefore, one must sign not only M by k, but also $L - M$ by a different key. Still, this scheme

- is computationally too expensive: For example, for a message of 100 bits, the computation of the public key needs $2^0 - 1$ computations. Therefore, messages need to be split into shorter ones whose encryption is computationally feasible.
- can only be used to securely sign one message per key. If a private key signs more than one message, then a signature can be forged: If messages m and M are signed with key k' and L – m and L – M with key k'', then further iterated application of H to the signature s' or S′ corresponding to the smaller value between m and M for k' (and likewise for k'') forge signatures for messages in-between m and M.

Modern hash-based signatures are more sophisticated than WOTS; however, they can still either only securely sign one message for each key, or, such as SPHINCS, sign a limited number of messages but produce large signatures.

Perfect Forward Security. Perfect Forward Secrecy means that after the correspondents exchanged their (permanent) public keys and established mutual trust,

1. *before* correspondence the correspondents *create an (ephemeral) session key* and sign it with their (permanent) private keys (to avoid a man-in-the-middle attack),
2. *after* correspondence each correspondent *deletes the (ephemeral) private key.*

This way, even if the correspondence was eavesdropped and recorded, it cannot be deciphered later on; in particular, it cannot be deciphered by obtaining a correspondent's private key. For example, the TLS protocol, which encrypts much communication over the Internet, supports since version 1.2 Perfect Forward Secrecy: More specifically, in the handshake between client and server,

1. after the client has received (and trusted) the server certificate,
2. the server and the client exchange an *ephemeral* public key which serves to encrypt the communication of this correspondence. This ephemeral key is signed by the (permanent) public key of the server. (The creation of this asymmetric key in Perfect Forward Secrecy makes the creation of a symmetric *preliminary* key by the client in the penultimate step in the handshake in the TLS protocol superfluous.)

The mutual secret key can be established, for example, by the DHE (Diffie-Hellman Ephemeral) or ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) protocol This key is then used as input to generate a secret symmetric key, for example, for the AES encryption algorithm.

Self-Check Questions.

1. List three types of algorithms considered secure in the advent of quantum computers:

    1. *symmetric cryptographic algorithm*
    2. *lattice-based cryptographic algorithm*
    3. *cryptographic algorithm based on error correcting codes*
    4. *cryptographic algorithm based on hash functions*

14.3 Incorporating cryptography into Application Development

Risk Analysis. Risk analysis measures the likelihood that an organization's security will be breached (for example, an intruder exploiting network vulnerabilities) and calculates the inflicted damages from such a breach, be they material (such as data privacy violation) or immaterial (such as a reputation loss). The risk is often measured as a financial risk and countered similar to insuring against threats such as theft. Risk analysis

- assesses the threats (by an intruder or company's insider) to the company's computer network. According to the CSI/FBI Survey, the biggest ones are, in this order

    – viruses,
    – abuses of the network by employees,
    – Denial of Service (DoS), and
    – stolen intellectual property.

- assesses the values of the company's assets, consisting of:

    – material values, such as the company's computer network, and
    – immaterial values, such as the company's reputation.

- the company's vulnerability, that is, where, how and how a security breach could happen.
- prescribes preventive measures, for example,
  - random generation and regular revision of passwords,
  - anti-virus software, and
  - intrusion detection systems, such as firewalls.
- builds plans to quickly recover from an exploit.

Don't roll your own crypto!. The First Rule of Cryptography: Don't implement cryptography yourself in production code. Instead, leave it to the experts and use a proven library that withstood the test of time under the scrutiny of cryptanalysts rather than a home-made one:

1. In practice, a cryptographic algorithm is secure if it has proved resilience by defying all attacks. Therefore, an established cryptographic algorithm should be used and proven ciphers such as `AES` and `RSA` are recommended. If instead a new one is used, no safety can be assured. In particular, it is a bad idea to design a proper new cryptographic algorithm.
2. Similarly, a software (library) is secure if it has proved resilience by defying all attacks. Therefore, an established software (library) should be used and common software libraries such as `Libsodium` (or the widely-used `OpenSSL`), are recommended.

Authenticated Encryption. Programmers often confuse the encryption and authentication. While encryption ensures confidentiality, authentication ensures integrity. When they have to be done separately: Encrypt *then* authenticate (that is, compute a MAC, Message Authentication Code). Verify the MAC *before* decryption. A priori, there are three options for authenticated encryption:

- Authenticate and Encrypt Together: The sender computes a MAC of the plaintext, encrypts the plaintext, and then appends the MAC to the ciphertext. This is what SSH does.

  In this order, the MAC (over plaintext) leaks information about the plaintext; for example, whether two messages have the same plaintext (by their identical MACs).

- First Authenticate then Encrypt: The sender computes a MAC of the plaintext, then encrypts both the plaintext and the MAC. This is what SSL does.
- First Encrypt then Authenticate: The sender encrypts the plaintext, then appends a MAC of the ciphertext. This is what IPsec does.

  This way, one can verify the MAC and discard texts without decryption; thus

  - diminishing denial of service attacks by discarding forged packets faster;
  - second, it reduces your "attack surface". One of the most important rules of computer security is that every line of code is a potential security flaw; if you can make sure that an attacker who doesn't have access to your MAC key can't ever feed evil input to a block of code, however, you dramatically reduce the chance that he will be able to exploit any bugs.

Comparison. Only Encrypt-then-Authenticate is provably secure in theory, that is, secure against IND-CCA (indistinguishability of ciphertext for chosen *ciphertexts*), that is, an attacker can ask for any pair of ciphertexts to be deciphered (excluding the ciphertext in question), and still cannot distinguish which one among two plaintexts corresponds to the ciphertext:

1. The oracle creates a secret key.
2. The attacker asks for the decryption of any ciphertext (except the one in question), and creates two plaintexts M_0 and M_1 of equal size.
3. The oracle
   - randomly picks a bit b in $0,1$
   - encrypts M_b, and
   - passes the ciphertext to the attacker.
4. The attacker asks for the decryption of any ciphertext (except the one in question), and
5. chooses a bit b' in $\{0,1\}$.

Example. Bleichenbacher's attack on PKCS#1 from 1998 is not secure against IND-CCA.

A cipher is secure against IND-CPA (*indistinguishability of the ciphertext for chosen plaintexts*), if no attacker can distinguish which one of two plaintexts, that he selected before, corresponds to the ciphertext that he receives afterwards. Bellare and Namprempre showed in 2000 for a symmetric cipher that if

- the cipher resists against IND-CPA, and
- the unidirectional function resists (is not *forgeable*) against an attack of chosen messages,

then the cipher with "Encrypt-then-MAC" resists an "IND-CCA" attack.

That neither Encrypt-and-Authenticate nor Autenticate-then-Encrypt aren't secure in theory, but Encrypt-then-Authenticate is, neither means in practice that the former two are insecure, nor that the latter is secure. However, there have been a number of vulnerabilities for the former two, while none for the latter: For example,

- Vaudenay's Attack, and
- Bleichenbacher's Attack

Recommendation. Therefore, it is best to use a library that takes care of authenticated encryption as a whole, instead of only offering these functions separately, where one has to be composed correctly oneself; sticking to the golden rule that as much as possible is reused instead of reimplemented:

The AEAD modes (Authenticated Encryption with Associated Data, where Associated Data is whatever must be authenticated but not encrypted) is a modern mode to encrypt and authenticate a message in the same operation, Reliable implementations of AEAD are, for example,

- AES-GCM, the Advanced Encryption Standard (a.k.a. Rijndael cipher) in Galois/Counter Mode, available OpenSSL, and
- ChaCha20-Poly1305 that combines the ChaCha20 stream cipher with the Poly1305 Message Authentication Code, available in Libsodium.

Password Storage. To store a password, as soon as it is received,

1. hash it using a key derivation hash function such as `scrypt` (that uses, besides the password, a nonce and iteration count to counter rainbow attacks) or `PBKDF2` (but NOT using a fast hash function, such as `MD5`, more vulnerable to rainbow table attacks), and
2. erase the plaintext password from memory.

Utmost care must be taken even for less sensitive applications because some users might reuse these passwords for more sensitive ones.

Finally, we recall that Encoding, for example `base64` encoding, and Compression, such as Zip compression, aren't encryption, as they hardly obfuscate information: Encoding and compression algorithms are both reversible, keyless transformations of data:

- Encodings transform data for better processing, while
- Compression reduces data as much as possible.

Self-Check Questions.

1. To authenticate and encrypt a message, one should:

    - Authenticate and Encrypt Together,
    - First Authenticate then Encrypt, or
    - *First Encrypt then Authenticate*?

2. Cryptographic functions should be

    - reimplemented, or
    - *reused*j ?

3. Which cryptographic hash function should *not* be used for storing passwords:

    - `bcrypt`,
    - `scrypt`,
    - `PBKDF2`, or
    - *MD5*?

14.4 Legal and Regulatory Aspects

The General Data Protection Regulation (EU) 2016/679 (GDPR) governs the exposure of personal data, processed and stored by hand or by computers, and applies since 25 May 2018 to all companies in the European Union. Besides the GDPR, other national data protection law may apply, for example, in Germany,

- the federal data protection act Bundesdatenschutzgesetz (BDSG), and
- the data protection acts of the German federal states.

GDPR wages the interests of companies and consumers in the digital age and protects every citizen's fundamental right of informational autonomy by granting the concerned citizen transparency and ultimate authority in the processing of her own personal data; that is,

- the processing of personal data must always have a clear purpose that has been expressly confirmed by the concerned person, and
- the data must be protected and deleted as soon as no longer needed for the bespoken purpose or when it expires (for example, credit bureau information, for example, by the SCHUFA in Germany).

Personal Data Protection. Every staff member who processes personal data must be instructed on data secrecy. In general, forwarding personal data to third parties is inadmissible without consent of the concerned person. If exceptionally admission is granted, then data must be encrypted and sent separately for each purpose, so that third parties neither an eavesdrop nor collect data.

What is personal data according to law, for example, the GDPR? General data:

- general personal data (such as name, date and age of birth, place of birth, address, e-mail address, telephone number, etc.)
- identification numbers (such as social security number, tax identification number, health insurance number, identity card number, matriculation number ...)
- physical characteristics (such as sex, skin, hair and eye colour, stature, dress size ...)

- ownership characteristics (such as vehicle and real estate ownership, land registry entries, license plates, registration data ...)
- value judgements (such as school and work certificates ...)
- bank data (such as account numbers, credit information, account balances ...)
- customer data (such as orders, address data, account data ...)
- online data (such as IP address, location data ...)

Special data that needs special protection is in general (but also listed similarly, for example, in Paragraph 4, 9 of the GDPR):

- ethnic origin, political opinions, religious beliefs, sexual orientation, or syndicate membership
- biometric data, such as
  - gene data,
  - health records: for example,
    * examination of the body or genetic data relating to diseases (current, previous or in risk), disabilities.
    * predispositions of the data subject due to family history, the measurement of health data in fitness studios and data collected by fitness and health apps and Smart Watches.

Thus, for example, in health care all patient information is strictly confidential by law. For example, in Germany, documents can must be transmitted either encrypted or by fax. Only data necessary for treatment can be collected. The patient data must be confidentially stored and kept confidential by the staff. For example, in Germany the unauthorized disclosure of patient data subject to professional secrecy can be punished, by Section 203 of the Criminal Code (StGB), with a monetary fee or up to one year of prison.

Government Trap Doors. Leaked secret documents show that the American National Security Agency (NSA)

- searches for vulnerabilities, such as the `Heartbleed` bug; see the section below.

- has sabotaged international encryption standards, that is, the algorithms were purposefully weakened for later decryption:
  - in the specification of the first U.S. Data Encryption Standard (DES) in the 1970s, the NSA was suspected of weakening DES by shortening the key length.
  - Juniper VPN Backdoor: use of NSA mandated an insecure random number generator that allowed decryption of eavesdropped VPN traffic.
  - Backdoor in Lotus Notes used by several European governments: though the version with stronger cryptography was acquired, NSA knew part of a key so that they could still decrypt the files.
  - the insertion of a backdoor into the cryptographic machines from the Swiss company Crypto AG; see the section below.
- U.S. export regulations lead to the `EXPORT` ciphers in `SSL` used outside the USA, because the stronger ciphers could not be exported, leading to the so-called `FREAK` attack.
- GCHQ mandated weak cryptography into the GSM standard to decrypt mobile communication.
- In 2004, Greece government officially mandated an interface for wiretapping into telephone talks; though disabled, it was still in the firmware. This was used to eavesdrop on government members.

`EES`. The Escrowed Encryption Standard (EES) is a chip-based symmetric encryption system developed in the USA in April 1993. The developer of the algorithm is the secret service NSA. It was developed as part of a U.S. government project to provide electronic devices sold to the general public with a security chip. The encryption key was to be provided to the government, which would then be able to eavesdrop on communications if necessary.

The main difference to other encryption methods is that, if necessary, US authorities can get access to the keys used by two users to exchange data. The procedure is specified in such a way that two keys are required for eavesdropping, which are deposited with different authorities and which should only be released at the same time by court order. This official access possibility is not achieved by a built-in back door in the technical sense, but by depositing two partial

keys. If the legal conditions are met, the two parts of the key are issued and joined together.

Skipjack was the algorithm used for encryption within the Clipper chip. The chip was designed to resist external modification and allowed the government to access the data in plaintext through a mechanism called Law Enforcement Access Field (LEAF). After the appearance of an attack in 1994 the project was abandoned in 1996. The appearance of software such as PGP, which was not under government control, made the Clipper chip obsolete and the Skipjack algorithms was made public in 1998.

Bullrun Program. `Bullrun` (and its British equivalent called `Edgehill`) is a secret American program by the NSA (respectively by the GCHQ) to break the encryption systems used in the most widespread protocols on the Internet, such as Secure Sockets Layer (SSL), Virtual Private Networks (VPN) or Voice over IP (VoIP). The existence of the program was revealed in September 2013 by Edward Snowden, showing that the agencies have been working on the main protocols or technologies used in the Internet (HTTPS/SSL, VPN) or 4G for mobile telephony to intercept and decipher in real time large Internet data volumes; for example, those circulating on Hotmail, Yahoo, Facebook and especially Google.

`RSA BSAFE` is cryptography library in `C` and `Java` by `RSA Security`; it used to be common before the RSA patent expired in September 2000. From 2004 to 2013 (till revealed by Snowden) its (supposedly cryptographically secure) default pseudorandom number generator `Dual_EC_DRBG` contained an alleged kleptographic (allowing for stealing information securely and subliminally) backdoor from the NSA who held the private key to it, as part of its secret Bullrun program:

Cryptographers had been aware that `Dual_EC_DRBG` was a very poor Pseudo Random Number Generator (PRNG) since shortly after the specification was posted in 2005, and by 2007, it seemed to be designed to contain a hidden backdoor usable only by NSA via a secret key. NSA can potentially have weakened data protection worldwide, in case NSA's secret key to the backdoor is stolen. RSA Security did not explain their choice to continue using `Dual_EC_DRBG` even after the defects and potential backdoor were discovered in 2006 and 2007, and has denied knowingly inserting the backdoor.

Operation Rubikon. The Swiss Crypto AG was an internationally active company in the field of information security. Between 1960 and 1990, at the height of the Cold War, Crypto AG was a leading company for encryption devices and produced for countries for more than 130 countries. The CIA was concerned about being unable to decipher foreign messages and approached European manufacturers, including Crypto AG. The (West) German foreign intelligence service BND and the US intelligence service CIA secretly bought the company in 1970. They arranged for many states to be supplied with machines with weaker encryption that could be decrypted by the BND and CIA (Operation Rubikon).

The company enabled these two services to decipher encrypted messages between the 1960s and 2010, While the suspicious Soviet Union and China were never among Crypto's clients,the CIA could however learn about some of their exchanges thanks to third countries equipped with tampered devices. For example, the CIA estimates that it could

- read around 85% of the Iranian coded messages sent in the late 1980s,
- spy on Egyptian communications during the Camp David negotiations in 1978,
- on Argentinian messages during the Falklands War in 1982, and
- gather decisive information during the invasion of Panama in 1989.

In February 2020, after evaluating a 280-page dossier, Swiss Radio and Television, ZDF and The Washington Post published a joint investigation which proved that the German BND and the US CIA were the owners of Crypto AG and delivered manipulated ciphering devices to some 130 states as part of Operation Rubikon to eavesdrop on communication.

A Security Hole supposedly kept secret by Intelligence Agencies. *Heartbleed* is a vulnerability in the SSL implementation OpenSSL, which is used in the popular web servers `Apache` and `nginx`, which are running two-thirds of the web pages at the time of the bug The *Heartbeat* allows server and client to keep a TLS connection alive by sending a message of any content (payload) from one end to the other, which is then sent back exactly the same way; to show that the connection is alive:

The RFC 6520 `Heartbeat` Extension tests TLS/DTLS secure communication links by allowing a computer at one end of a connection to send a "Heartbeat

Request" message, which consists of a payload, typically a string, along with the length of that payload as a 16-bit integer. The receiver must then send the exact same payload back to the sender.

Versions of OpenSSL subjecet to the `Heartbleed` bug allocate a memory buffer for the message to be returned based on the length field in the request message, regardless of the actual payload size of that message. Because of this failure to check the appropriate limits, the returned message consists of the payload, possibly followed by whatever else is allocated in the memory buffer.

The problem with implementing the TLS heartbeat feature in OpenSSL was that the program does not check how long the received payload is. The attacker can write arbitrary values in the payload_length field provided for this purpose in the header of the payload packet and thus read the memory of the remote peer.

This `Hearbleed` attack works in both directions; let us assume that a client is attacking a server:

1. The attacker sends the server a heartbeat payload that has 1 byte, but claims that it has, for example, 16 kilobytes.
2. The server writes the attacker's byte into its memory in a buffer called `pl`. Since the actual size of the payload is not compared, the server assumes the size specified by the attacker (payload) when the payload is returned.
3. The server therefore reserves 16 Kilobytes of memory (and a little more for administrative information):

   ```
   buffer = OPENSSL_malloc(1 + 2 + payload + padding);
   bp = buffer;
   ```

4. The server then copies the payload size given by the attacker (about 16 Kilobytes) at this place for the response:

   ```
   memcpy(bp, pl, payload);
   ```

But the source `pl` to be copied only has a single byte from the incoming heartbeat! The following bytes consist of any other data that the server is currently processing; such as passwords, data of another user that was just decrypted, or secret keys of the server. The server then sends the data packet `bp` to the client, which can repeat this attack at will.

The tapping leaves hardly any traces on an attacked computer. It is therefore not certain to what extent the error has been exploited. However, the news agency Bloomberg reported, citing "two informed persons", that the Hearbleed bug was used by the NSA from the beginning; this was immediately denied by the NSA director: "This government takes seriously its responsibility to maintain an open, secure and trustworthy Internet. There is a national interest in disclosing any such vulnerability as soon as it is discovered."

Self-Check Questions.

1. When can personal data be collected ?

   With a clear purpose and express consent of the person.

2. Name at least three examples of specially protected personal data:

   - *ethnic origin, political opinions, religious beliefs*
   - *biometric data, such as gene data and health records*

Summary

Many things can go wrong when implementing cryptography; thus, as a software developer, caution must be taken:

1. First, it is recommended to reuse what is already available and vetted by the test of time; for example, as a software developer, to use (open-source) software libraries that implement cryptographic functions such as encryption, decryption, signing and verification. In particular, it is out of question to roll develop one's proper cryptographic algorithm instead of basing one's cryptography on the established solutions such as AES for symmetric and RSA for asymmetric cryptography. For example, the cryptocurrency IOTA implemented its proper ternary cryptography and hash function, which was quickly unravelled by the Digital Currency Initiative at the MIT.
2. Thus, even though most was already implemented by cryptography professionals, still great care has to be taken in choosing the implementation and in implementing what is left: For example, the implementation of random number generators is notorious for exploits and the first function to audit on a system. as we saw, in ECC, elliptic curve cryptography, if the

same ephemeral key , usually randomly generated, for signing is used twice to sign different documents by the same private signature key, then the ephemeral becomes known and reveals the secret signing key. For example, the `SecureRandom` class of the Java Android Crypto library that implements cryptographic functions on Android did not properly initialize the underlying Pseudo Random Number Generator, leading to the same ephemeral key being used more than once.

3. Even when reusing well-known time-proven cryptographic open-source libraries, for example, OpenSSL, bugs might creep in, such as the Heartbleed bug contributed by a PhD-student at the Fachhochschule Münster, that made common web servers reveal on request currently processed secret data, such as passwords or server keys. These bugs, once known, are quickly fixed; the question remains if in the meanwhile intelligence agencies exploit these.
4. Finally, governments make laws that oblige the use of encryption, but government agencies might also work against it, for example, by demanding back doors to be built into cryptographic software or devices.

Even when best cryptographic practices are applied, Moore's law predicts that continuous technological progress doubles the computing power every 18 months and thus weakens the future security of keys. Therefore, a security margin for long-term security, usually for around twenty years, has to be added. Finally, there may be technological leaps, such as the looming construction of a quantum computer, which would break many common asymmetric ciphers such as Diffie-Hellman, RSA and Elliptic Curve Cryptography and more involved alternatives have been presented.

Questions

1. Which part of a cipher is in practice most susceptible to be exploited?

   ☒ the pseudo-random number generator,
   ☐ the decryption algorithm,
   ☐ the encryption algorithm, or
   ☐ the key generation algorithm.

2. Which algorithm is secure against a quantum computer?

   ☐ RSA

- ☒ AES
- ☐ ECC
- ☐ Diffie-Hellman

3. For at most how many years will projectively an AES 128 -bit key stay secure?

   - ☐ 10
   - ☒ 20
   - ☐ 40
   - ☐ 60

4. To which AES key size does an RSA key of 2048 bits compare?

   - ☐ 80
   - ☐ 112
   - ☐ 128
   - ☒ 192

5. To which AES key size does an ECC key of 256 bits compare?

   - ☐ 80
   - ☐ 112
   - ☒ 128
   - ☐ 192

15 Applications

Study Goals

On completion of this chapter, you will have learned ...

- the European standard protocol for home banking, FinTS, in particular, its cryptographic operations;
- how a modern voting schemes such as Scantegrity II lets the voter verify the correct tallying of her ballot while maximally guarding its secrecy;
- how steganography embeds a secret message into a public message and could be statistically detected;
- how the Tor project guarantees anonymity on the internet by onion routing, where all exchanged data is encrypted so that each node of a network only knows how to pass it to the closest neighbors.

Introduction

Cryptography secures data exchange on the Internet; for example, it protects the content of a financial transaction, be it between a customer and her bank or users of a cryptocurrency, from eavesdropping (confidentiality) and tampering (authenticity). Because correspondents rarely met in person before, asymmetric cryptography (such as RSA) is the cornerstone of secure data exchange on the Internet that makes possible

- the initial exchange of a secret (usually symmetric, say, AES) key to encrypt all further communication, and
- authenticate the messages by digital signatures (that is, its encryption, usually of a cryptographic hash, with the private asymmetric key). Indispensable especially for financial transactions.

However, while cryptography hides the content of the messages, their metadata is not, for example:

- that the data is encrypted, possibly pitting cryptanalysts against it: Steganography is the art of imperceptibly embedding sensitive content into public content; for example, a key to decrypt a file into an image file.
- who exchanged data with whom; that is, the identity is not hidden. The Tor project achieves anonymity by transmitting the data over a network of nodes where each one of can only transmit it to its closest neighbors.

In voting cryptography allows the voter to check whether her vote has been correctly tallied. However, to avoid coercion, secrecy of the ballot, that is, anonymity of the vote, must be maximally preserved while the tallying process must stay comprehensible for the average voter. The Scantegrity II voting scheme achieves this by pen and paper probabilistic audits.

15.1 Online banking

The Home Banking Computer Interface is an open protocol specification originally conceived by the two German decentralized saving banks Sparkasse and Volksbanken (and German higher-level associations such as the Bundesverband Deutscher Banken) to unify the online access of the client to her bank by standardizing the homemade software clients and servers. HBCI is the counterpart

for the European market of the IFX (Interactive Financial Exchange), OFX (Open Financial Exchange) and SET for the north-american market. HBCI

- is independent of the operating system.
- supports accounts on multiple banking companies.
- uses DES and RSA-encryption and signatures.
- stores the key on a chip card.

The Financial Transaction Services (FinTS) specification succeeds HBCI 3.0. It is publicly available on the website of the ZKA (Zentraler Kreditausschuss) and supported by Sparkasse, Volksbanken und Raiffeisenbanken, Commerzbank, Deutsche Bank and more than 2000 other financial institutions. First published as version 3.0 in 2002.

FinTS supports

- online banking with SWIFT (Society for Worldwide Interbank Financial Transactions, started in 1977 and is owned by the member banks), protocols and standards for international payment systems between thousands of financial member institutions in 194 countries, among them the central banks of most countries.
- issuing legally binding digital signatures of financial transactions.

Starting from version 4.0 from 2004

- all data is encoded into the universal XML (Extensible Markup Language) format and
- data exchanged by the HTTP, HTTPS (the synchronous case of a permanent connection between client and bank) and SMTP (the asynchronous case of an unstable connection between client and bank) protocols to facilitate integration with other payment systems (and ensure communication with clients behind a firewall). The underlying transport protocols is chosen according to the application protocol: While unencrypted protocols, such as the PIN/TAN method, must use an encrypted HTTPS protocol, encrypted protocols (such as RAH-7 or RAH-9) may use the unencrypted HTTP protocol.

In 2014 version 4.1 was published that contains improvements gained by years of practical experiences and to adopt:

- SEPA (Single Euro Payments Area), a self-regulatory initiative by the European banking sector represented in the European Payments Council to set (technical) standards that uniformize the payment transactions in the European Union (BIC, IBAN, ...).
- PSD2: The European Union passed in 2015 the revised legal framework Payment Services Directive (PSD2) that all payment service providers of the member states had to respect till 2018,
  - to make online payments more secure, and
  - in particular to protect the customers' rights.

  PSD2 was supplemented in 2017 by technical regulatory standards (2018/389) for (by two-factor) client authentication for secure online payments that had to be adopted till September 2019.

Business Transactions. FinTS predefines business transactions codes, but also allows each bank to define their proper codes.

Code	Name
DKPAE	Change PIN online
HIISA	Transmission of a public key
HKISA	Public key request
HKCCS	SEPA (Single European Payments Area) payment
HKAUB	foreign bank transfer
HKEND	End of dialogue

FinTS specifies bank parameter data (BPD) and user parameter data (UPD). If the user is represented by an intermediary, then the IPD are the largely identical counterpart (for the intermediary) to the UPD.

- BPDs specify all business transactions of the bank and are used to validate the data sent to it; they contain, for example, the supported security procedures, compression procedures and business transactions. Customer software often comes with bank parameter data from common banks.

- UPD are sent by the bank and define the user's access rights for certain accounts and business transactions, and The customer software uses the UPD to check whether the user is authorized to execute one of the business transactions specified in the BPD. For example, the UPD contain the authorizations of the business transactions for each one of the user's account (for example, to enter payment orders or as a signatory) and other information, such as its currency, a limit ...
- UPD and IPD are each a subset of the BPD, that is, they can only restrict and detail the BPD, but not extend it.

Message Confirmation Codes. Client and bank communicate via confirmation codes that are classified by the first digit of the code:

- Success (class 0) confirms that all commands have been completely processed.
- Note (Class 1) confirms that an intermediate command has been completely processed.
- Warning (Class 3) no command has been rejected, but warnings exist.
- Error (class 9) the order part or message is partially incorrect and individual orders may have been rejected

A message is syntactically valid if it obeys the underlying XML schema; if not, they are answered with the confirmation code 9110 for "Unknown structure".

Code	Text
0020	Information received without errors
1040	BPD no longer current. Current version will follow
1050	UPD no longer current. Current version will follow
3330	Keys are already available
9010	Order rejected
9210	Language is not supported

The confirmation codes enable customer software to react automatically to messages from the credit institution; for example, if the response is "wrong institution", then the software can automatically request the correction of the institution (as part of the IBAN). While the "confirmation text" gives the user plain text information, the confirmation code facilitates customer queries.

Roles. FinTS distinguished between the following roles:

- the *Messenger* signs and transmits the message and does not need to know about its (possibly encrypted) contents.
- the *Issuer* signs the order part of a user message and possibly encrypts (and can be the messenger as well).
- the *Witness* signs the order part of a user message in addition to the issuer if the issuer's signature alone is insufficiently authorative.
- the *Intermediary* mediates between the customer and bank as a technical interface endowed with varying authorizations towards the bank (for example, those of the issuer or of the messenger). Communication between the intermediary and the credit institution is always encrypted.

Encryption and Signing. Transactions are encrypted

- either on the FinTS protocol layer using a key stored
  - either on a chip card, more exactly a banking signature cards using the SECCOS operating system of the Deutsche Kreditwirtschaft,
  - or in a file secured by a password
- or on the TSL protocol (underlying the HTTPS-Protocol) using PIN/TAN with indexed (iTAN) and mobile (mTAN) transaction numbers, that is, one-time passwords. (iTAN was abandoned in 2019 by the EU payment regulations directive PSD2.)

Using a chip card is the most secure because

- all cryptographic operations are achieved without the secret key ever leaving the chip card securely stored on it,
- and the PIN is entered by the card reader keyboard.

If a file is used, then the key must be encrypted by a password chosen by the user and only accessible after manually having entered it.

The PIN/TAN method is more convenient because it does not require a card reader (for example, while travelling).

All asymmetric cryptography uses exclusively the RSA algorithm: The key pairs of the user are to be generated by the customer (from the chip card), and

those of the bank are to be generated by the bank; according to the following procedure:

- A constant public exponent $e = 2^{16} + 1$ (= 4th Fermat prime number), and a modulus n individual for each user is used for each RSA key system used.
- The modulus n of each RSA key system has a length of $N \leq 2047$ bit and no leading 0 bit (so that $2^{N-1} \leq n < 2^N$).

Encryption between client and server is hybrid and uses

1. RSA for the initial key exchange.
2. A randomly generated message 32 byte key for the symmetric encryption algorithm:
    - 3DES up to version 4.0, the *RDH* method (RSA-DES-Hybrid);
    - AES since version 4.1, the RAH method (RSA-AES Hybrid).

the current one-time key is encrypted with the public key of the recipient. The length of the one-time key of 256 bits is extended to the modulus length of the public encryption key (2048 bits) by the ZKA padding specified in the crypto-catalogue of the German Banking Industry.

FinTS uses RSA signatures to authenticate transactions (for example, in RAH-9 and RAH-10) and, in RAH-7, to sign transactions by a certificate. The signing key pairs is

- either authenticated by the bank via an initial "Ini-letter" (for example, RDH-1, RDH-5, RDH-8 and RDH-9),
- or already authenticated by the bank in the client's chip card (for example, RDH-3, RDH-6 and RDH-7).

An encrypted message is authenticated by signing its plaintext. All messages must be encrypted, with the notable exceptions of those that

- are already encrypted by the transport protocol, for example, HTTPS,
- initially ask for keys or block keys,
- were received anonymously, for example, to be informed about business transactions or submit unauthenticated orders, and
- keep the connection alive (heartbeat message).

Self-Check Questions.

1. List the roles that a customer can assume in FinTS business transactions: *issuer, messenger, witness or intermediary*
2. Which encryption algorithms uses the FinTS RAH protocol? *RSA-2048 and AES-256*

15.2 Voting

For a vote, like every sensitive transaction, for example, a financial transaction,

- the voter must be authenticated, and
- the vote must be
  - authentic (that is, unaltered between emission and reception) and
  - confidential (that is, only the sender and receiver know it).

However, the transaction, the ballot, must be *anonymous* as well; that is, the receiver cannot know the sender: there is no link to between the voter and her ballot (in particular, that there is no receipt that reveals the voter's choice). *Secrecy of the Ballot* guarantees that only the voter herself, but no one else, knows of her choice (with the notable exception of a vote by a physically handicapped person instructing her assistant to give her vote and possibly, though forbidden by law, postal vote).

Anonymity versus Integrity. Integrity is about:

1. whether a vote is **cast as intended** (which is obvious in a paper ballot),
2. whether a vote is **registered as cast** (which, for paper ballots, relies on trusting the election officials)
3. whether all the votes are **tallied as registered** (which, for paper ballots, relies on blind trust into the election officials).

Integrity can only be achieved on the expense of Anonymity, and precautions must be taken to preserve it as much as possible. In an *end-to-end auditable (or voter verifiable)* voting system, each voter receives an encrypted ID, which she can use to check on a public list whether her choice was likely cast, registered

and tallied as intended. The more voters check and errors are committed, the more probable that one of them is detected.

While secrecy can be strengthened by shuffling the assignment of bubbles to candidates on each ballot, the integrity check nevertheless demands a database that assigns each bubble of each ballot to a candidate, and which is owned by the election authorities at some point. Therefore, while integrity can likely be achieved, to break the secrecy of the ballot, it suffices:

- to identify the ballot by its ID and the filled bubbles,
- to identify the voter of a ballot, for example, her fingerprint on the ballot and access to a database of fingerprints, and
- to identify the candidate that corresponds to a bubble by access to the assignment database.

To ensure that this assignment between the bubbles of the ballots and candidates is left unaltered during the casting of the ballots, the election authorities commit to it before the election by

1. cryptographically hashing
    - the information of each ballot (such as its ID and the bubble numbers), and
    - the assignment database.
2. publicizing the hashes.

Integrity versus Traceability. In many countries, the voting process that underlies a democracy should be understandable by everybody (instead of having to trust the computer), thus ruling out the use of cryptography. For example,

- In its decision of 3 March 2009, the Federal Constitutional Court declared the use of the voting computers in the election to the 16th German Bundestag (and European Parlament) unconstitutional due to insufficient public traceability: The system used for this was closed source, an audit of the integrity of the source code was not allowed to the interested public. However, "the essential steps of the voting process and the determination of results must be verifiable by the citizen reliably and without special expertise" by Article 38 with Article 20 (1) and (2) of the Grundgesetz

(Basic Law) which require "that all essential steps of the election are subject to public scrutiny".
- In 2014 Namibia was the first country in Africa to use voting machines for national elections to the National Assembly and Presidency. The use of the machines without a verification printout was declared unconstitutional by Namibia's Supreme Court in February 2020.

Instead, most countries opt for paper ballots that are hand counted. Therefore, instead of using voting machines to cast, register and tally votes, voting procedures such as Scantegrity (see below) are explored that use pen and paper to cast, register and tally votes, but cryptographically identify the ballot with the voter's choice to ensure authenticity while preserving anonymity.

History of Electronic Voting Machines.

1. The first generation of electronic voting machines, the DRE (Direct Recording Electronic voting machine), is characterized by direct electronic recording without printing of the vote. In Brazil, such a microcomputer is used for the collection and tallying of votes. In the USA, this type of voting machine is forbidden because it does not meet the "Principle of Software Independence in Election Systems" that the election result can be audited independently of the machine (as given in the "Voluntary Voting System Guidelines" by the U.S. federal agencies Election Assistance Commission (EAC) and National Institute of Standards and Technology (NIST)).
2. The second generation accomplishes the "Software Independence Principle in Election Systems" which registers the vote digitally and on paper. In Brazil such a device was proposed in an electoral reform in 2015, however rejected by the STF (Federal Supreme Court) in 2018 because human intervention would increase the chances for fraud.
3. The third generation of voting machines is characterized by the scanning of the ballot and an encrypted registration that allow the voter to check herself (without any technical equipment) the correct cast, registration and counting of her vote, but is incapable to prove her vote to others. A prominent example is the Scantegrity voting scheme, which we will present below:

Scantegrity. The Scantegrity voting scheme is an end-to-end voter verifiable voting scheme, that is, each voter receives an ID which she can use to check

on a public list whether her choice was likely cast, registered and tallied as intended (and otherwise file a dispute). It was devised by David Chaum et al. in 2008 and supersedes the Punchscan scheme by David Chaum and since then continually evolved into versions I, II to III after trial elections. The paper ballot contains:

- a list of candidates which have, right next to each one of them, a bubble that is identified by a letter code, and
- a detachable human-readable ballot ID and an undetachable barcode of it (to prevent a human to easily identify the cast ballot).

The voter can use her ballot to

- vote by marking an admissible number of bubbles,
- spoil her vote, for example, by marking no bubble or more bubbles than votes, or
- audit a ballot by asking for two ballots among which she selects one to audit (and the other to vote).

1. To vote, the voter fills a bubble on the ballot,
2. to register, an optical scanner recognizes the bubble position and ballot ID, and
3. to verify her vote, the voter can keep the code of her chosen bubble and the ballot ID by detaching it from the ballot.

A database assigns to each bubble of each ballot ID the candidate shown on the ballot. While this assignment could be achieved in a single table P

- whose rows correspond to the ballots,
- whose columns to the candidates, and
- whose cells are the bubble codes of the ballot and bubble of the corresponding row and column,

for public auditing of the registration and tally of votes, this assignment is split into two steps and the table P kept secret and expanded into three tables:

1. a table Q in which every row corresponds to a ballot (ID) and the cells of each row contain the bubble codes (in no particular order so as not to reveal that on the paper ballot),

2. a table S with a column for each candidate and as many rows as ballots and in which each cell is flagged if and only if the corresponding bubble on a bullet was filled, and
3. a table R that contains two columns in which each row assigns to every cell in Q, that is, bubble of each ballot, a column (and row) in S, that is, a candidate (plus a column to flag the bubbles that were voted).

α	κ_0	κ_1	κ_2	κ_3	β	γ
01			PL		8860	3478
02				SA	3813	2448
03	AY	EY	SI	XW		3492
04		ZK			1337	4592
05			SH		3492	3472

(a) Table **Q**

	Adams	Burr	Jefferson	Pinckney
0	A		✓	
1	✓		✓	A
2		A		
3				
4	✓		A	

(b) Table **S**

Q-Pointer	Marks	**S**-Pointer
$(03, \kappa_3)$	A	(4, Jefferson)
	✓	
$(03, \kappa_0)$	A	(0, Adams)
	✓	
$(03, \kappa_1)$	A	(2, Burr)
	✓	
$(03, \kappa_2)$	A	(1, Pinckney)
	✓	

(c) Table **R**

Figure 67: The tables Q, S and R that assign a candidate to every bubble code; from Clark (2011)

In table Q, in each row, the cells whose bubble codes have been revealed on the ballot are revealed. In table S, the cells that corresponds to the bubble of a ballot that has been revealed is marked.

The public audit reveals a random half of the assignment between each bubble on a ballot and candidate in table R, that is, either the cell in table Q or the column (and row) in table S. While the chance to detect a given manipulated bubble is 50%, the odds to overlook a manipulated bubble among, say, ten manipulated bubbles is below 1 to 1000, that is, the probability is $< (1 - 0,5)^{10} \approx 0,1\%$.

If the voter chooses to audit a ballot, then this ballot is

1. excluded from the tally by the poll worker,
2. the full assignment between every bubble and candidate of that ballot are revealed on tables Q, S and R, and
3. all the bubble codes are revealed, and their assignments are checked.

Again, while the chance to detect a given manipulated ballot is 50%, the odds to overlook a manipulated ballot among, say, ten manipulated ballot are below 1 to 1000, that is, the probability is $< (1 - 0,5)^{10} \approx 0,1\%$.

Scantegrity II. As countermeasures to perceived attacks during trial elections with Scantegrity I, while in Scantegrity I the codes of the bubbles are randomly permuted among a set of letters that identify the bubbles, in Scantegrity II (or the basic system): The voter

1. fills out a bubble next to her chosen candidate with a special pen that reveals a unique code of multiple letters written in invisible ink that encrypts the cast vote, and
2. receives for later verification the ballot number and the revealed code to be exposed online.

This way,

- the voter cannot ensure to choose a certain bubble code because every ballot contains different bubble codes; therefore she cannot be forced to randomize her vote by choosing a fixed letter.
- a voter can prove online and anonymously, without a receipt, that she chose a certain bubble, because:
  - all but the marked code are hidden, and
  - each code is sufficiently unique to be improbable to be guessed.

  This avoids

  - being pressurized to omit filing an error by retribution, and
  - taking away from the voter the possibility to file an error by taking away her receipt.

Scantegrity III. In Scantegrity III, to prevent a human from reading off the information contained in a cast ballot (where all codes given to the voter were detached):

- The ballot ID is encoded, say as a bar code, and
- the bubble codes turn dark after a couple of minutes by having been printed in a slow-reacting invisible ink.

To prevent a voted ballot being turned into an audited or a spoiled ballot, two "authenticated status" codes are added to the ballot and committed to before the election. Each code is printed on the ballot in slow-reacting invisible ink, and individually detachable (for example, using a perforation).

If the voter:

- votes, then **both** status codes are given to her.
- chooses to audit, then **one** of the status codes (chosen by her) is revealed and given to her. The other one is either destroyed or kept as a record by the election authority.
- spoils the ballot, then **no** code is given to her and both status codes are destroyed.

If a voter later on the website,

- has voted and finds that
  - an incorrect confirmation code appears online, then knowledge of **one** (correct) **bubble** code suffices (instead of the physical receipt, stamped or otherwise) to prove her diverging choice,
  - her cast ballot has been filed as an audited or spoiled ballot, then knowledge of **both status** codes suffices to prove that the ballot was voted.
- has audited her ballot and finds that
  - her audited ballot has been filed as a spoiled ballot, then knowledge of **one status** code suffices to prove otherwise,

  - her audited ballot has been filed as a voted ballot, then knowledge of **all bubble** codes suffices to prove that the ballot was not voted on.

- has spoiled her ballot and finds that her ballot is misrepresented (as voted or audited), then, in theory, two additional status codes could be added to detect the misrepresentation; however, in practice it is sufficiently effective to prevent it at the voting booth by destroying all ballot IDs.

Self-Check Questions.

1. List four criteria that an end-to-end voter verifiable voting system should adhere to:

    1. *confidentiality*
    2. *authenticity*
    3. *anonymity*
    4. *traceability*

2. Which election steps in an end-to-end voter verifiable voting system should be verifiable by the voter:

    Whether the vote is:

    1. *cast as intended*,
    2. *registered as cast*, and
    3. *tallied as registered*.

15.3 Steganography

Steganography from Greek *steganos*, covered, is the art of concealing a message by embedding it within another message so that nobody but the intended recipient knows of the existence of the message. One example is given in The Histories of Herodotus where Histaeus shaved the head of his most trusted slave and punctured a message on his head which became hidden as soon as his hair had grown back.

In contrast, in cryptography the existence of the hidden message itself is not hidden, but only its content (for example, when sending an encrypted e-mail). Thus it makes public, presumably important, information is deliberately being

hidden, drawing attention to the fact and possibly pitting interested cryptanalysts against it.

> **Steganography**: the art of concealing a message by embedding it within another message so that nobody but the intended recipient knows of the existence of the message.

Implementation. In computing, the message is typically hidden in a picture, for example, a compressed JPEG, but also an audio file, say, in the MP3 format, is conceivable. This can be used, for example, to hide, in the picture itself, a copyright notice of the owner of a picture (Coded Anti-Piracy). Because a typical picture file has hundreds of kilobytes, a few bytes can be changed to convey a secret message without noticeable change. The secret information is usually stored in the marginal parts of the image: The simplest picture format is a bitmap, that is, a picture

- is a matrix of pixels (of dimensions, for example, 1024×768),
- each pixel has a color, and
- each color is defined by the intensities of the three primary colors Red, Green and Blue.

Because man cannot distinguish more than $256 = 2^8$ degrees of each primary color, it suffices to represent each pixel by three bytes.

Least-Significant Bit Substitution: The hidden message, as a stream of bits, can then be stored, for example, in the least significant bit of the eight for each primary color of each pixel: For each byte, as a number b between 0 and 255, the hidden bit is 1 if and only if b is odd.

Steganalysis. To read the hidden content in a medium, it must be

1. detected, and
2. extracted.

Steganalysis is the art of detecting and extracting the hidden content in a medium without knowledge

- where the hidden content is, and
- which algorithm was used.

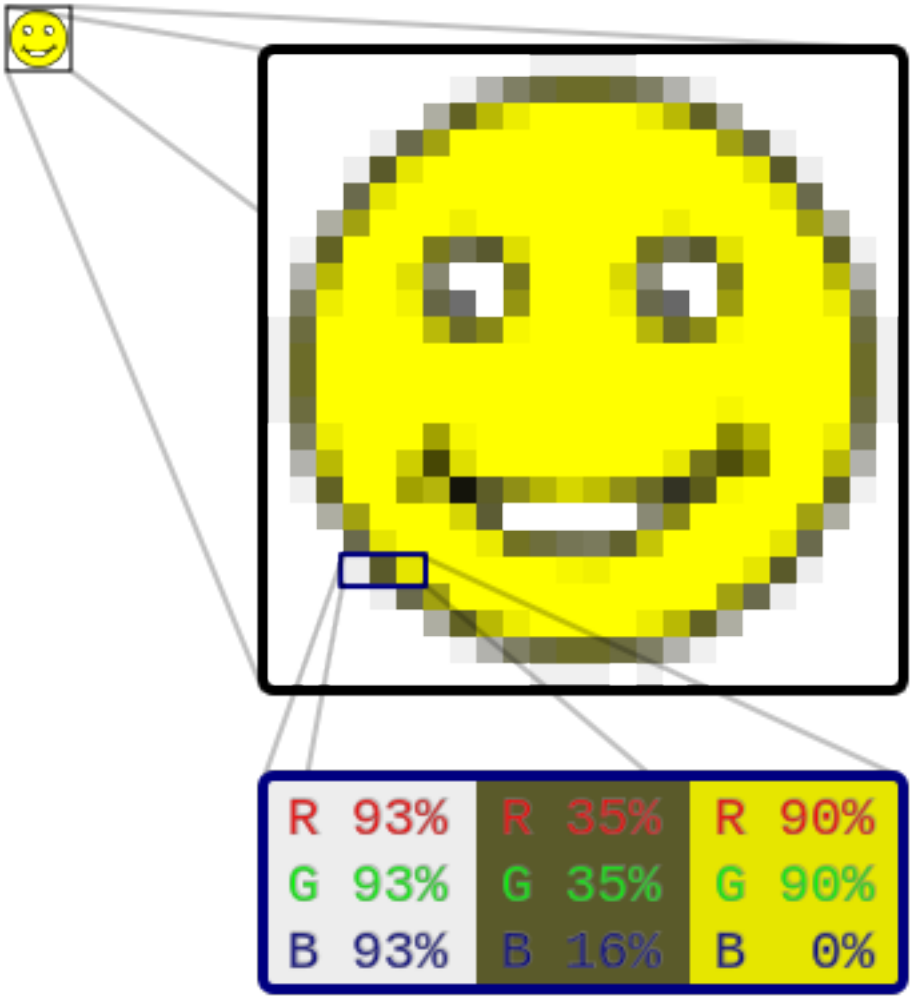


Figure 68: Primary color degree map; Commons (2020)

Hidden content can be detected statistically by looking for significant deviations to similar reference data without hidden content. For these deviations to be statistically less significant, the hidden content itself should be statistically random, pattern free; this is for example achieved by encrypting it.

For example, *Least-Significant Bit Substitution* can be detected by a *histogram*, a diagram of bars, one for each color in the picture whose height is proportional to the number of pixels of that color. Because bytes that were originally

- odd can only be decremented by 1 and
- even can only be increased by 1,

the heights of each pair of neighboring bars, that is, the frequency of pixels of neighboring colors, will be significantly closer to their mean than in the original picture. Therefore *Least-Significant Bit Substitution* can be reliably detected when the hidden information makes up less than 1% of the whole picture.

To counter this detection method, Least-Significant Bit Matching, instead of Least-Significant Bit *Substitution*, adds or subtracts 1 to the byte randomly, instead of exclusively adding to even and subtracting from odd bytes. That is, if the bit of the hidden message is 1, then, a coin flip decides whether the byte is incremented or decremented (instead of decrementing the byte if it is odd and decrementing it if it is even). Then, like in Least-Significant Bit *Substitution*, for each byte, as a number b between 0 and 255, the hidden bit is 1 if and only if b is odd. However, the heights of each pair of neighboring bars in the histogram are no longer significantly closer to their mean than statistically expected and the problem of detecting .

Kerckhoff's principle, that an attacker who knows of the steganographic algorithm used to store the hidden information cannot detect it, is however more difficult to achieve in steganography than in cryptography.

Application. The command-line tool Tomb by Denis Roio (aka Jaromil), is a Linux shell script that encrypt folders by `dm-crypt`, which is part of the kernel; the encrypted folders are consequently called Tombs (whereas `dm-crypt` calls them containers). These can be created and integrated into the running system with a few commands on the command line (which, however, partially need administrative rights). For highest security, and tomb and its key should not be stored on the same device. For example, if the tomb is on a PC or notebook, the tomb could be stored on a USB stick. If however, the key must be stored on the same device as the tomb, then Tomb offers to hide the key in a JPEG picture using steganography. This hides the key from unauthorized eyes and helps remember the key's location.

Tomb can mount folders that other applications need at runtime, for example, the mailbox of an e-mail client. To this end, it needs the package `steghide`. The key can be hidden in a small JPEG picture by `tomb bury` and extracted by `tomb exhume`:

- to hide a key in a picture: `tomb bury secret.tomb -k picture.jpg`
- to retrieve a key from a picture: `tomb exhume picture.jpg -k secret.key`
- to open a tomb with a steganography key: `tomb open secret.tomb -k picture.jpg`

Self-Check Questions.

1. How does steganography differ from cryptography?

   Steganos means cover while *crypto* means *hide* in Greek. Steganography covers up a secret message by a plain message while cryptography reversibly scrambles its content.*

15.4 Mix-Nets

Chaum introduced in 1981 Mix-Nets to communicate anonymously on the network by passing the data through relays that can only forward to their nearest neighbors.

Principles.

- Message Pooling: To always ensure many potential endpoints, messages are pooled and redistributed among the endpoints, either a single message randomly or a batch of messages at once.
- Rearrangement of Messages: The order of the outgoing messages should not depend on that of the incoming messages.
- Deletion of Duplicates: Duplicate incoming messages become duplicate outgoing messages and as such indicate their source; thus they better be deleted.

Sending and Receiving Scheme. Each relay has a public key pair with public keys $p_1, \ldots, p_n$ numbered in the order in which the message travels through them, 1 for the relay after the sender, ..., n for the relay before the receiver.

Sender. To send a message m anonymously and confidentially, the receiver sends:

- the message m (encrypted so that only the receiver knows its plaintext), and
- a list of addresses (so that each relay only knows the address of its successor).

1. The sender selects the (IP) addresses a_r of the receiver and of each relay $a_1, \ldots, a_n$.
2. The sender chooses a random number for each relay (to ensure uniqueness of each message).
3. The sender
    1. encrypts the message m using the public key of the receiver if available.
    2. encrypts, using the public key p_n, the IP address a_r, the (possibly encrypted) message and a random number.
    3. ...
    4. encrypts, using the public key p_1, the IP address a_2, a random number and all the data encrypted in the last step.

    and sends this encrypted data to the first relay.

Receiver. To receive data anonymously and confidentially, the receiver sends to the sender s the following data (which in turn is successively sent to each relay):

- a list of addresses (encrypted so that each relay only knows the address of its successor), and
- a list of symmetric keys (encrypted so that each relay only knows its own key) to encrypt the messages, one for the sender and one for each relay, numbered in the order in which the message travels through them, 1 for the relay after the sender, ..., n for the relay before the receiver. (Because the symmetric key serves as a random number for the uniqueness of each message, a random number is no longer needed.)

The message m sent anonymously and confidentially as described above could, for example, be this (encrypted) list of addresses and symmetric keys to receive data anonymously and confidentially.

1. The receiver selects the (IP) addresses a_s of the sender and of each relay $a_1, \ldots, a_n$.
2. The receiver creates symmetric keys for the sender k_s and each relay $k_1, \ldots, k_n$.
3. The receiver

    1. encrypts, using the public key p_n, the symmetric key k_n and the receiver's IP address a_r,
    2. ...
    3. encrypts, using the public key p_1, the symmetric key k_1, the IP address a_2 and all the data encrypted in the last step,
    4. attaches the symmetric key k_s of the sender and the IP address a_1 to all the data encrypted in the last step.

    and sends this data to the sender.
4. The sender

    1. decrypts the packet using its private key to obtain the symmetric key k_s and the address a_1 of the first relay (after the sender), and
    2. encrypts the packet using k_s, and
    3. sends it to the first relay.
5. The first relay (after the sender)

    1. decrypts the packet using its private key to obtain the symmetric key k_1 and the address a_2 of the following relay, and
    2. encrypts the packet using k_1, and
    3. sends it to the following relay.
6. ...
7. The last relay (before the receiver)

    1. decrypts the packet using its private key to obtain the symmetric key k_n and the address a_r of the receiver, and
    2. encrypts the packet using k_n, and

3. sends it to the receiver.

Tor Project. Tor is an implementation of second-generation **onion routing** to guarantee anonymity on the Internet. It was originally sponsored by the US Naval Research Laboratory, then the Electronic Frontier Foundation (EFF) between late 2004 and 2005.

> **Onion Routing**: A chain in which every node only knows its immediate predecessor and successor, and in which all traffic between both endpoints is indecipherable to every node but the endpoints.

Connection to Network. To connect to the network, each client:

1. Fetches a list of Tor nodes from a server.
2. Automatically chooses a random path (that may change after a while).
3. Builds a circuit in which each node knows only its predecessor and successor.

The first node in the circuit knows the requested IP address. But from the second node on, the negotiation is done through the already built partial circuit, so that the second node, for example, will only know the IP address of the first node (and eventually of the third node). The packets to be routed are identified by a code (chosen at the time the circuit is built) of the owner of the circuit (the person who built it). Each node of the circuit receives its own private (asymmetric) key encrypted by the public (asymmetric) key dedicated to that node.

Packet Exchange between Client and Server. Before dispatching a TCP packet to the server, the client encrypts it as many times as there are nodes:

1. with the public key corresponding to the last node, numbered n ;
2. with the public key of the penultimate node, numbered $n-1$;
3. with the key of $n-2$;
4. with that of $n-3\ldots$;

...

n. the last time, with that of the first node.

At this point, all layers of the onion enclose the TCP packet. The onion is peeled when the client dispatches it to the circuit she has built :

1. the first server in the circuit decrypts the packet with key number `1` and sends it to the second server;
2. the second server decrypts this packet with key number `2` and sends it to the third server;

...

n. the last server decrypts this packet with its own private key number n and receives the original plaintext packet.

Proxy. A user on the Tor network can set up her web browser to use a personal proxy server to access Tor (such as `Privoxy`); for example, to connect to `ongel.de`:

1. Her web browser sends the HTTP request to `Privoxy`;
2. `Privoxy` removes the non-anonymous information and passes the information via SOCKS to the Tor client.
3. The Tor client
    1. builds a circuit (if it hasn't already done so),
    2. encrypts the data to be sent, and
    3. passes it to the first node;
4. The first node decrypts part of the envelope and forwards the data to the exit node;
5. This exit node sends the request to `ongel.de`.

For the `ongel.de` website to connect to the user, the steps are carried out in inverse order.

Implementations. The most accessible use of the Tor network without advanced computer skills is the Tor Browser:

- The Tor Browser is a web browser, available for Linux, Microsoft Windows and Mac that tunes Mozilla Firefox for leave as few traces as possible on the network and the computer:
  - Browser traffic is by default redirected through the Tor instance started at initialization,
  - lack of browsing history,
  - `duckduckgo.com` as the default search engine, and
  - the `NoScript` and `HTTPS-Everywhere` extensions enabled by default.
- Tails (The Amnesic Incognito Live System) is an operating system that uses the Tor network by default and designed to leave no trace on the computer being used. It is built to run on removable media (such as USB drives) and is based on the Linux distribution Debian.

Self-Check Questions.

1. How many times does a client encrypt a packet before sending it to the server through ten nodes in a Tor network?

   the client encrypts it as many times as there are nodes, that is, ten times

Summary

The online banking protocol FinTS standardizes data exchange between the customer and her bank by the XML (Extensible Markup Language) format, and ensures best cryptographic practices such as

- storage of the secret keys on a chip card that never leaves it, but only is used to respond to challenges based on knowledge of the key,
- permanent encryption and authentication (by signing) of all sensitive data,
- using proven encryption algorithms.

The blockchain replaces a third trusted party, the bank, that mediates financial transactions, by a database of entries that successively point to each other. This pointer is a hash of the whole other entry, ensuring the database's integrity: thus, a change of the entry entails a change of the hash, thus invalidates the pointer, thus the whole chain. Because the hashes must have many leading zeros, finding valid entries, aka mining, demands much computational power and practically turns impossible.

Verifiability of the tallying in an election by cryptography puts additional requirements on its implementation because

- the casting of votes must be anonymous, and
- the tallying must be comprehensible to the average voter.

While some anonymity has to be put at risk to database breaches by colluding election officials, the Scantegrity II pen-and-paper voting-scheme lets the voter check the tallying of her vote probabilistically by an intermediate private database that connects each ballot to the candidate choices.

While cryptography hides a message by reversibly scrambling it, steganography hides it by embedding in a plaintext message, say, by embedding a secret key inside an image. The most common approach is slightly altering each pixel color according to a set rule; for example, to set the least significant bit (out of eight) to that of the message. If done too naively, for example, as described, than statistical analysis gives away the existence of concealed information; however, if done with care, then it is hardly discovered.

The Tor network achieves anonymous data transfer on the Internet by so-called onion routing, where traffic passes through a chain of nodes in which

- every node only knows its immediate predecessor and successor, and
- all traffic between both endpoints is indecipherable to every node but the endpoints (by every node iteratively encrypting the content with its own key).

Questions

1. Which home banking protocol standard precedes FinTS?

   ☒ HBCI,
   ☐ IFX,
   ☐ OFX, or
   ☐ SET

2. How high is, roundabout in percents, the chance that ten manipulated ballots go undiscovered in the Scantegrity II voting scheme?

   ☐ 10
   ☒ 1
   ☐ 0,1
   ☐ 0,001

3. What does the Greek word Steganos translate to?

   ☐ hide
   ☒ cover
   ☐ obfuscate
   ☐ embed

4. What routing technique does the anonymity of data transfer in the Tor network rest on?

   ☒ onion routing
   ☐ starshape routing
   ☐ cross routing
   ☐ local routing

16 Blockchain

Study Goals

On completion of this chapter, you will have learned the anatomy of the blockchain that stores and secures the transactions of cryptocurrencies such as Bitcoin.

When using a common currency traders trust a third party, the bank, that keeps a ledger of all transactions. When using a *cryptocurrency*, they put their trust instead into a *blockchain*: a public ledger of all transactions, split into blocks, packets of transactions, each containing around 2000 of them, immutably bound together. It is maintained (and replicated) by a network of thousands of computers to make it practically unforgeable: The whole network can read all the blocks and it will only accept the addition of valid blocks.

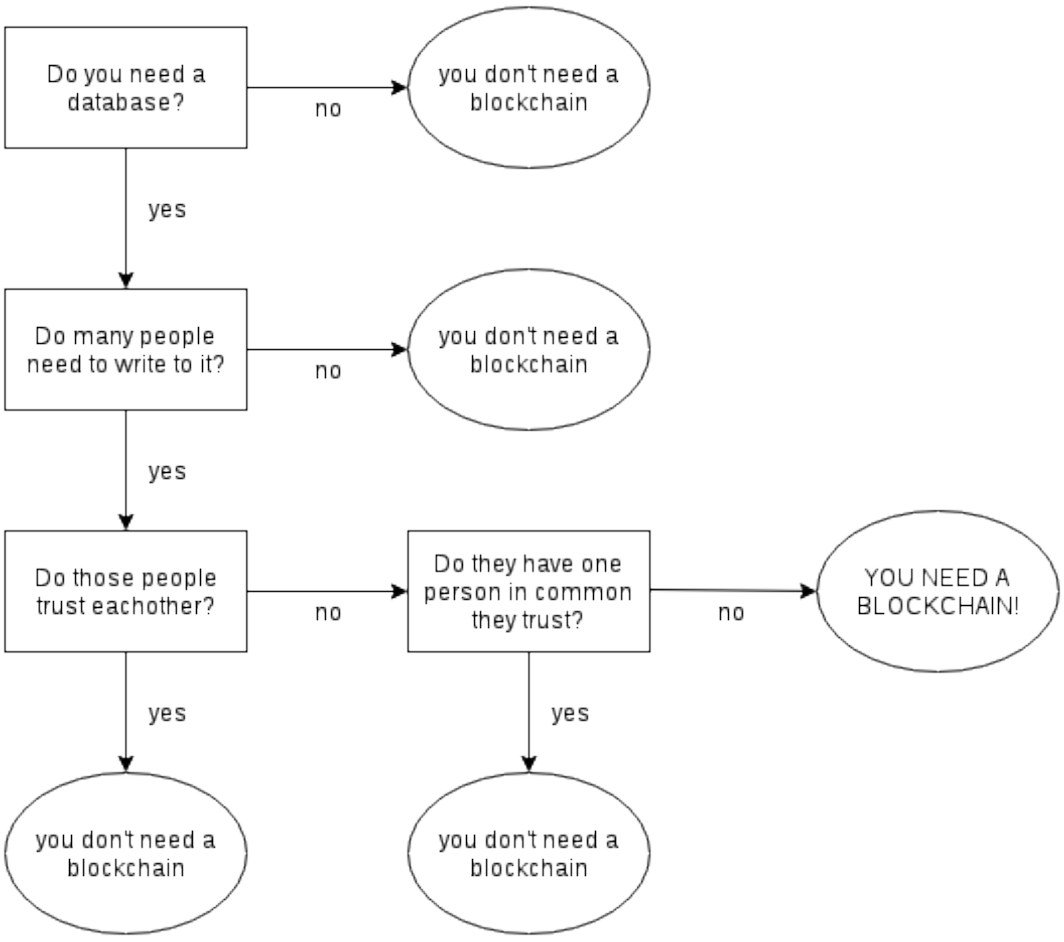


Figure 69: A blockchain is useful as a database for multiple users who trust neither each other nor anyone else.

The Bitcoin network prevents the (malevolent) alteration of the (existing) blockchain by requiring a *proof of work* for appending a block to it: the compu-

tation of an input to a cryptographic hash function that yields an output with many leading zeros.

There are other approaches to make cryptocurrencies costly, for example:

- the `BurstCoin` uses a "proof of space" instead of a proof of work, that is, the reservation of a hard disk space, to earn coins. Like the `Bitcoin`, it also has the drawback that all transactions are public (with the knowledge of the person behind the public key [the `Bitcoin` address] as the only cover-up).
- For increased privacy, the cryptocurrency
  - `Monero` uses *group signatures* (see Section 4.3) to confirm transactions between traders, thus obscuring the cryptocash flow, and
  - `Zcash` which uses `zk-SNARKs` instead of group signatures, a *non-interactive zero knowledge proof*.

We will see

1. how the chain of blocks works is built,
2. what a block (of transactions) consists of,
3. how bitcoins are generated, so-called *mining*,
4. why this hard work, *proof of work*, is indispensable to ensure the integrity of the chain, and
5. how bitcoins are transferred.

16.1 Overflight

The blockchain, literally, is a chain of blocks. There is an initial block, the Genesis block and blocks that point to their predecessors. This arrow pointing to the predecessor is a hash, an identification of the entire contents of the previous block; it is an address that appears in the block header. The trunk of the block consists of an average of 2000 transactions between Bitcoin users.

Hashes. Since the hash of the block depends on the entire contents of the block, the change of a single bit changes its hash, that is, invalidates the arrow pointing to its successor! Thus, for the blocks to continue to form a chain, this arrow must be changed. Thus, the hash of the successor block changes! Thus, for blocks to continue to form a chain, it is necessary to change the arrow of the successor of the successor, and so on; a chain reaction! That is, if we change one detail, for example, a transaction in the first block, all the arrows (= hashes) that follow must be recalculated!

Mining. Enters so-called "mining", which makes this change very difficult, because only blocks whose hashes are small, that is, start with many zeros, are accepted by the network (that is, by its verifier nodes). Thus, it is not enough to change a transaction and calculate the new hashes of all successor blocks. This change has to be such that all blocks are accepted, that is, their hashes are small! It is highly difficult to find such a change: Currently, searching such a block takes a billion years on a regular computer; but only ten minutes on the mining network. While the bad guy started searching for admissible blocks, the network has already created several others, thereby invalidating his work!

Transactions. All existing bitcoins were generated by mining, that is, they were given (by the *coinbase transaction*) as a reward to the one who created a block (with a small hash). All other transactions have an input and output: As input a sufficient amount is gathered together to pay what will be spent. Better almost the full amount be spent, because everything what is not spent will go to the miner who will eventually make this transaction happen (by including it in the block he mined). Therefore usually a transaction includes the sender as a recipient as well to receive the unspent money back; the *change transaction* (whereas usually only about $0,001$ bitcoin is paid to the miner as a transaction fee). The recipient is designated by his public key, and only at the time he will spend the received coins he will need to prove that he owns the corresponding private key, by signing the transaction.

Elliptic Curves. Bitcoin uses encryption by finite elliptic curves to sign transactions: Diffie-Hellman uses finite rings of number such as $\{0,1,...,m\}$ (for example, $m = 12$ for the clock, and a prime number such as $p = 2^{255} - 19$ with 100 digits in Bitcoin). The concept used by Bitcoin resembles that of

Diffie-Hellman, only instead of numbers 0, 1, 2, ... it uses pairs of numbers (x, y) that designate Cartesian coordinates of points on a (so-called elliptical) curve. The beauty of this curve is that one can add points on them: $p + q + r = 0$ if these three points p, r and q lie on the same line. Instead of multiplying the same number several times as in Diffie-Hellman, we add the same point several times. It is easy to add points, but it is very difficult to know how many times a point was added to itself to obtain the resulting point. Encryption corresponds to iterated addition, decryption to the knowledge of how many iterations.

Finally, the signature scheme is a variation of ElGamal's signature: the signature shows that the owner of the private key was able to solve a difficult equation; so difficult that it is practically impossible to solve it without this private key that provides a shortcut.

Self-Check Questions.

1. How many transactions does a block of the Bitcoin blockchain contain on average?

    1. *1000*
    2. *2000*
    3. *100 000*
    4. *1 000 000*

2. How long does the addition of a block to the Bitcoin blockchain currently take on average on a regular computer?

    1. *1 000 hours*
    2. *1 000 days*
    3. *1 000 000 years*
    4. *1 000 000 000 years*

16.2 Chain

The *blockchain* is a chain of blocks that are linked, that is, each block contains a hash of another block, which we think of as a pointer (or address) to the *previous* block.

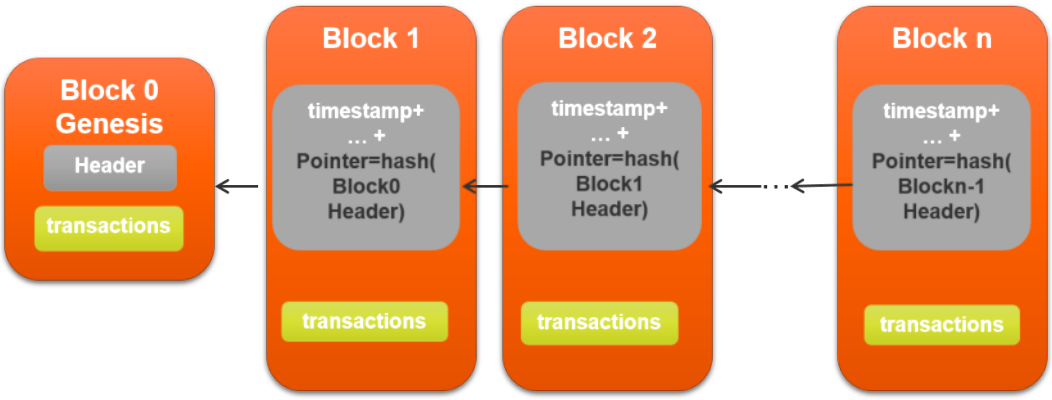


Figure 70: Blockchain

Branches, *orphaned* blocks, occur; however, only the longest chain (more exactly, whose construction was computationally heaviest; see proof of work in Section 16.4) is considered valid. As it is very difficult to extend the chain by another block, branches rarely have more than one block.

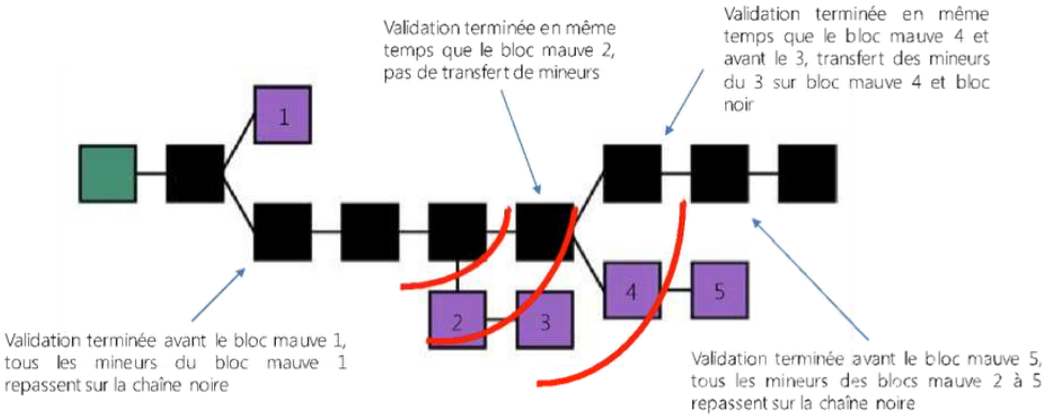


Figure 71: Bifurcation Management

The chain was lanced on January 3, 2009 at 18:15 UTC, probably by Satoshi Nakamoto, with the first block, the *genesis block*.

Each block consists

- of its *head*, the meta-data, which contains
  - the pointer to the previous block, and

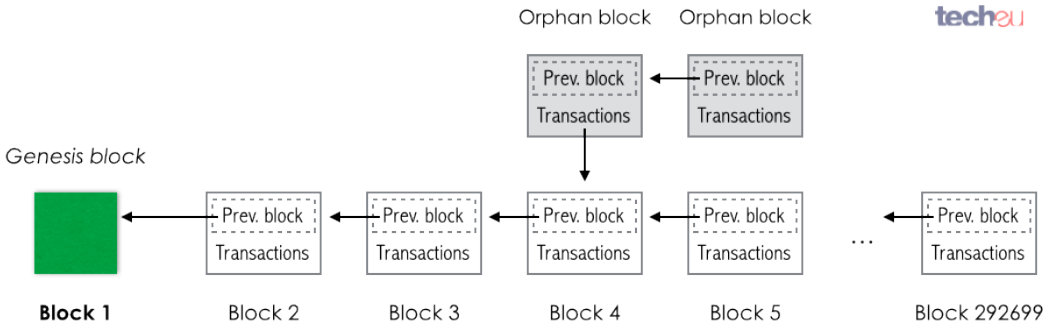


Figure 72: Genesis block

  - information about his creation;

- of its *content*, the data, many (about one megabyte of) *transactions* among Bitcoin users (grouped in a *Merkle tree* for fast retrieval; see Section 3.8).

The hash of the block depends on all of its content. That is, (practically) any change causes its hash to change; for example:

- the alteration of one of its transactions,
- changing the pointer to the previous block.

For example:

1. If one of its transactions changes, then its hash changes.
2. Thus the pointer to it in the next block changes, thus the hash of the next block changes.
3. Same for the hash of the block following the next block, and
4. ...so on.

In other words, a chain reaction occurs: The change of a single transaction in one block invalidates all the hashes of its later blocks.

Therefore, for the blocks to continue to form a chain, the pointers (that is, the hashes of the later blocks) of all the blocks that follow the changed block must be recomputed: Normally, these new blocks invalidate the blockchain because these new hashes no longer conform to the required pattern (= a sufficient number of leading 0s) to be accepted in the blockchain.

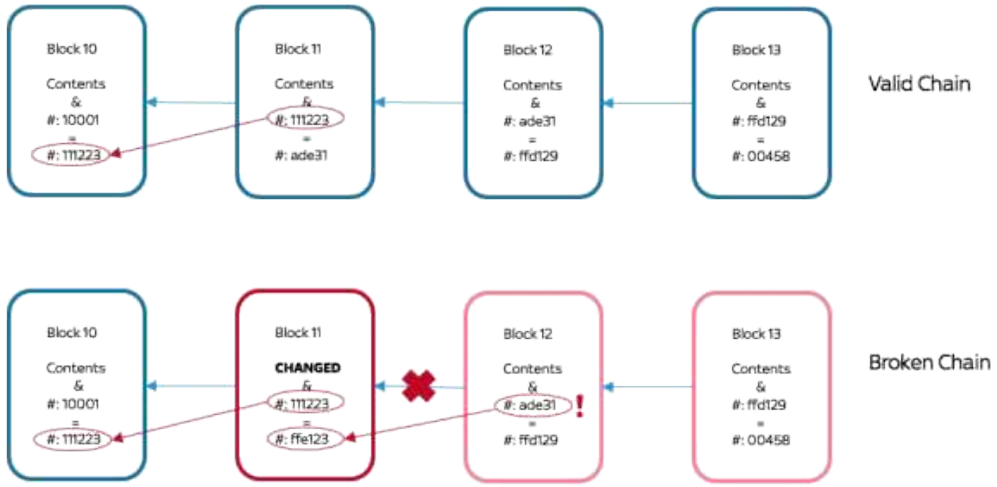


Figure 73: Invalid block

16.3 Block

Each *block* groups *transactions* between users in a *Merkle* tree.

Structure.

Field	Description	Size
Magic	= 0xD9B4BEF9	4 bytes
Header	Contains 6 items	80 bytes
Block	Size	4 bytes
Number of transactions		1 – 9 bytes
Transactions		

Each block groups transactions (which have, on average, 5000 bytes). First its size (up to 1 Megabyte) and number of transactions (on average 20000) are indicated.

The first transaction, the `coinbase` transaction, the reward for the work done for its creation, is written by the creator of the block and, therefore, commonly she and her collaborators are put as recipients.

The other transactions are transmitted to the network by the senders, that is, payers, and all block creators, the *miners*, decide which transactions they include in the block they are creating. To encourage the inclusion of a transaction, the sender may pay a fee to the creator of the block; hence often the miner includes as many transactions as possible, about 1 Megabyte.

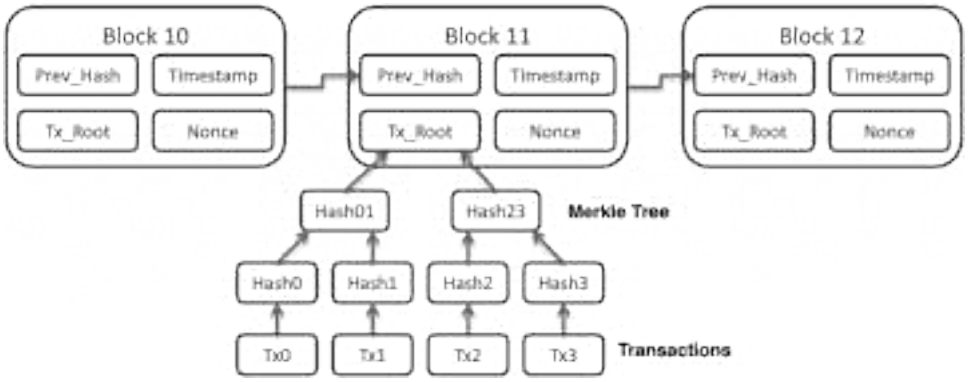


Figure 74: To quickly check transactions, they are grouped in a Merkle tree, a (binary) tree, whose vertices are hashes and whose leaves are transactions.

Header. The header of each block contains:

1. the version of the software used,
2. the hash of the previous block (for the blocks to form a chain),
3. the hash of the root of the Merkle tree of transactions,
4. the timestamp, the creation date (in seconds counted from 1970-01-01 at 00:00 UTC), and
5. the difficulty, roughly the number of zeros with which the block hash needs to start in order for it to admissibly extend the chain, and
6. a nonce, a field without (semantic) content, which serves to change the hash of the block without changing its (semantic) content.

The cryptographic hash function used by `Bitcoin` is `SHA-256`. In more detail:

Field	Updated when...	Size
Version	the software is updated	4 bytes
Hash of the previous block	a new block is created	32 bytes

Field	Updated when…	Size
Hash of the tree root	a transaction was accepted	32 bytes
Date of creation	a few seconds passed	4 bytes
Difficulty	every 2016-th block	4 bytes
Nonce	another hash is proven	4 bytes

The "body" of the block, its content (in contrast to the header metadata), is formed by the transactions. These are grouped in a "Merkle" tree, a (usually binary) tree of hashes where the hash of a node is calculated by those of its successors; the data, here the transactions, constitute its leaves.

The `nonce` is used to search for a block with a sufficiently small hash without changing its (semantic) content. For quite some time the 4 bytes of the `nonce` are insufficient to find a sufficiently small hash. That is, after 2^{32} increments, no found hash is small enough. In this case, `ExtraNonce`, the `coinbase` transaction message (which has 100 bytes) is iterated. However, then the hash of the Merkle tree root needs to be recomputed.

Coinbase. The first transaction, the `coinbase` transaction, the reward for the work done for its creation, is created by the creator of the block and, therefore, commonly he and his collaborators are the recipients. Each transaction has

- an "input", and
- an "output".

In all (except the initial transaction) the input collects outputs from transactions (by referring to their hashes) whose sum is greater than (or equal to) the sum of the outputs. In the initial transaction of the block, the input is arbitrary, and the amount is the reward given to the miner, initially 50 bitcoins, 6.25 in 2021.

The first block of the chain, mined by Satoshi Nakamoto, the anonymous creator of Bitcoin, contains the title of page one of the Financial Times:

```
The Times 03/Jan/2009
Chancellor on brink of second bailout for banks
```

However, as mentioned above, the content nowadays is often not human readable; instead, it is a value for technical purposes, for example, to alter the block such that its hash is small enough.

16.4 Chain Extension

To append a block to the blockchain, a *proof of work* must be given, the calculation of (a *head* of) a block such that its hash is small, that is, that its binary expansion starts with a large number of zero digits (in January 2021, with 20 zeros in the hexadecimal expansion or 80 zeros in the binary expansion). Perhaps the most used software for this purpose is currently CGMiner; there are versions for all operating systems, some adapted for graphics processors (GPUs) and others for processors specifically programmed for mining (ASICs).

The hash of a mined block on January 13, 2021:

```
00000000000000000010f32aa4a0a862d4761ae7a997fbf8590ce2191dfc064
```

Irreversibility. While the high computational cost is useless to build a blockchain, it is essential for its integrity because it makes it practically impossible to (malevolently) alter previous blocks. That is, it ensures the *irreversibility* of the blockchain: Once a transaction is in a block which has been extended by, say, at least five other blocks, it is practically impossible to replace the blockchain blocks because every such effort is outpaced: while these blocks are hard searched for, the blockchain has already been extended by other blocks.

Since each block contains in particular as an entry the hash of the previous block, and the hash of the entire block depends in particular on this entry, changing a block requires changing every subsequent block; since each block needs a hash with many initial zeros to be accepted into the blockchain, it is hard to find these subsequent blocks. A lot of work against time:

To change a block in the chain, the single miner needs to

- recalculate all subsequent blocks (such that its hashes begin with a sufficient number of zeros),
- while all other miners append other blocks to the blockchain!

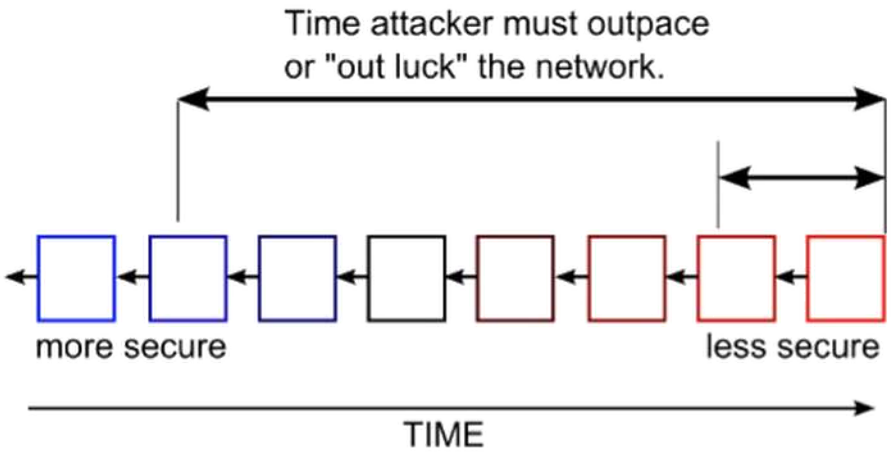


Figure 75: Outpacing: While one is hard pressed finding admissible alternative blocks, others have already been added.

This irreversibility lets one dispense with a third party for the traders, that is, a fiducial authority. Instead, it is the nodes of the network agree on the validity of the transaction.

However, if a miner has more computing power than all other miners, then he can apply the 51 % attack:

1. Send a transaction,
2. await its confirmation, that is, its inclusion in a blockchain block,
3. at the time it is included in a block, branch off the blockchain by extending the previous block by another block that does not include this transaction,
4. continue to extend this branch by creating other blocks,
5. when the transaction is confirmed, that is, 6 blocks have been appended, send to the network the blocks in the meanwhile created. By his higher computational power, his branch is longer than the blockchain; so the verifier nodes accept it as new blockchain!

Proof of Work. A *proof of work* shows information that is computationally costly to obtain. Often, the work consists of repeating an operation till an unlikely event occurs, that is, brute force.

`Bitcoin` uses the proof of work introduced by `Hashcash` to prevent spam by a

proof of work required for the sending of each email to each recipient; so that the proof of work makes mass sending costly.

In Bitcoin, the proof of work is required to extend the chain by a new block.

Hash. The work consists in searching for a block whose hash with 32 bytes by the SHA-256 algorithm is less than a certain target number:

- initially, in 2009 a number that starts with 4 bytes that are 0,
- currently in 2021, a number that starts with 10 bytes that are 0.
- The number of zeros is continuously (every 2016-th block) adjusted such that the (worldwide!) search takes on average 10 minutes.

Other common Hash algorithms (instead of SHA-256) for proof work are, for example,

- Scrypt, and
- SHA-3.

Example. We iteratively append to the string "Hello, world!" a nonce, a *n*umber used *once*, such that (the hexadecimal expansion [with the 16 digits 0 – 9 and A – F]) of its SHA-256 hash starts with 0000. There are $16^4 = 4096$ combinations of four hexadecimal digits; so if the values of the hash function are uniformly distributed, then we expect about 4096 attempts to find it. In fact, after 4251 attempts, which take a millisecond on a modern computer, we obtain:

```
"Hello, world!0" => 1312AF178C253F84028D480A6ADC1E25...
"Hello, world!1" => E9AFC424B79E4F6AB42D99C81156D3A1...
"Hello, world!2" => AE37343A357A8297591625E7134CBEA2...
...
"Hello, world!4248" => 6E110D98B388E77E9C6F042AC6B49...
"Hello, world!4249" => C004190B822F1669CAC8DC37E761C...
"Hello, world!4250" => 0000C3AF42FC31103F1FDC0151FA7...
```

However, in Bitcoin, the hashed object, the block header, is more complex, in particular because it contains the root of the (Merkle) tree of transactions.

Difficulty. The 'difficulty' field in the block header compares

- the computational effort that *is* currently on average required to find a new block attachable to the chain

with

- the computational effort that *was* on average required to find the first ("Genesis") block of the chain:

Initially, for the "Genesis" block to be accepted in the chain, the binary expansion of the hash of its header needed to start with 32 zeros (= the size, in bits, of the nonce in the block header); that is, it took $2^{32} \approx 4$billion) computations of hashes to find a block with such a hash (taking a couple of minutes on a notebook).

Since then, the number of zeros has been readjusted after every 2016-th block to ensure that the computation of a new attachable block over the network takes, on average, 10 minutes. (That is, computing 2016 new attachable blocks takes on average 2 weeks.)

Each readjustment (after 2016 blocks) calculates the new difficulty as a ratio between

- the target time $A = 2016 \cdot 10$ minutes (about two weeks), and
- the actual time U in minutes that the creation of the last 2016 blocks took;

that is,

$$\text{new difficulty} = A/U \cdot \text{previous difficulty}$$

To avoid too steep a jump, the new difficulty is ≤ 4. That is, even if $A/U > 4$ (that is, the network took less than three and a half days to append 2016 blocks), then the new difficulty is 4.

Mining. Computing a block header whose hash is smaller than the current target, that is, whose binary expansion begins with enough zeros, is called *mining*.

The difficulty is at each moment (with a delay of at most two weeks) proportional to the joined computational force of the miners: The more miners, the more difficult, the fewer miners, the easier. Currently, in 2022, the binary expansion of the hash has to start with around ≥ 80 zeros to be accepted by the network.

Computing Time. Since the hash used in bitcoin is cryptographic (currently SHA-256), that is, its output doesn't allow deducing the input, the only practical way to find a header such that its hash is small enough is by *brute force*, that is, trying out all possible combinations one after another. Since a hash function is almost uniformly random, that is, the probabilities of all outputs are almost equal, it takes on average about 2^n attempts until a header whose hash has a binary expansion starting with n zeros is found.

1. At the beginning of the blockchain, $n = 32$ and it took 2^{32} (about 4 billion) computations to find a block with such a hash (taking a couple of minutes on a notebook).
2. Currently, $n \approx 80$ and it takes on average around $2^{80} \approx 1.2 \cdot 10^{24}$ (= one trillion times one trillion) tries of different hashes. A fast processor of a microcomputer, for example the `Intel Core i7 2600`, computes around $2.4 \cdot 10^7 = 24$ million hashes per second. That is, it takes on average around $5 \cdot 10^{16}$ seconds ($> 10^9$ years, that is, a billion years).

This gives an idea of the current combined computational power of the miners, since they calculate this hash on average in ten minutes. Since the hash function used by `Bitcoin` is SHA-256, as discussed in Section 3.6, it is quickly computed by a CPU, a microprocessor for general use on a personal computer, and in particular,

- for the average consumer, a GPU, by a graphics processor; about 100 times faster than a CPU, and
- for someone with more resources, by an ASIC, a microprocessor suitable for a specific application such as the computation of SHA-256; it is about 100,000 times faster than a CPU.

In fact, the energy expenditure for mining is equivalent to that of the whole of Austria at any given time. For this reason, alternative concepts to the proof of work have emerged, for example, the *proof of stake* where the owner of the next block is determined by how much she owns instead of how much she can compute. However, so far these alternative concepts have not worked so well in practice.

Once such a block and its hash are found, given both, the verification that the hash is small enough is quickly done: simply compute the hash of the block and compare it to the given hash.

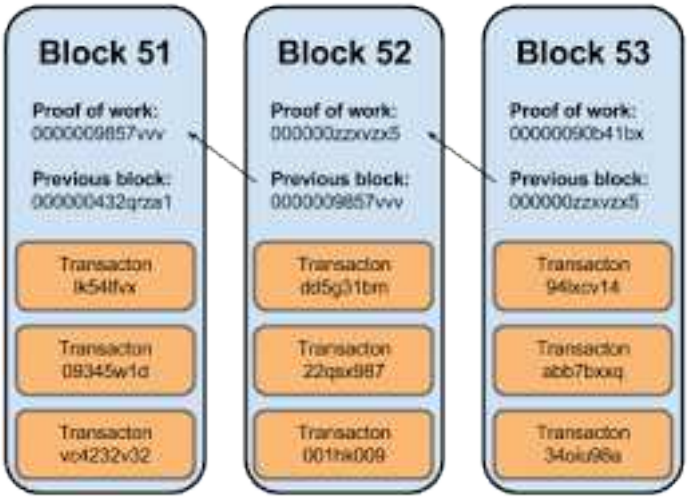


Figure 76: Blockchain

Time Available. The interval to create a new block is, on average, 10 minutes. While it will not always take almost exactly ten minutes or less, as a *Poisson process* the probability of a block been found in this 10 minute range is about 63% (more exactly, the probability is $1 - 1/e$).

- That is, almost two thirds of the blocks will be found in at most 10 minutes.
- In 30 minutes, the probability increases to 95, and in one hour to 99.7.

Change the Header. To change the block header, let us recall that the changeable header data consists of (cf. Section 16.3)

- the hash of the Merkle tree root of the transactions, and
- an nonce, a field without (semantic) content that serves to change the hash of the block without changing its (semantic) content.

In particular, the main content of the block, the transactions, enters into its hash only indirectly through the hash of the root of the Merkle tree.

Currently, since the number of initial zeros in the binary expansion of an admissible hash $n \geq 80$ and the field nonce only has 4 bytes (= 32 bits), rarely (that is, with a probability of only about 2^{32-80}) there is a value of the nonce such that the block hash is small enough. Therefore the miners change, in addition to the nonce,

- the `timestamp`, the creation date in seconds counted from `1970-01-01` at `00:00 UTC`,
- the transactions, for example, their order, and
- the `coinbase`, the message (having 100 bytes) that accompanies the first transaction of the block and transfers a reward to the miner for having found the block with a hash accepted by the blockchain network. While `coinbase` provides ample space to change the hash, it is a more expensive option than `timestamp` or `hash` because, being part of the transactions, its change implies the recomputation of the hashes of the Merkle tree that stores the transactions. (Recall though that it is only the root hash of the tree that enters the block header.)

Reward. When a new block is discovered, the miner transmits it to the network, and the nodes check, among others:

- that its header hash actually starts with the number of 0s required by the current difficulty, and
- that all transactions are valid.

After the blockchain has been extended by at least 100 other blocks following this block (so that the network ensured that this blocks will persist in the blockchain, the longest chain), its finder:

- earns a certain amount of bitcoins as a reward for the work of discovery: initially, in 2009, the reward was 50, currently, in 2021, it is 6.25; the value is split in half after 210000 blocks (as a block is mined every 10 minutes, this takes about 4 years).
- earns all transaction fees included in the block; paid as an incentive to include the more transactions with the most generous fees first in the block.

As the reward gradually fades out (until it runs out after $21000000 = (50 + 25 + 12.5 + \cdots) \cdot 210000$ blocks), these transaction fees will play a more and more important role in encouraging the miners to hold on mining.

To distribute the unpredictable reward from mining more evenly, many miners join their computational forces in groups, `clusters`, to share the rewarded bitcoins.

16.5 Transaction

All bitcoins are generated by mining: The miner, the creator of the block, can freely choose the recipients of the first transaction of the created block, the `coinbase` transaction.

Once generated, a `bitcoin` changes place by a chain of digital signatures: A `bitcoin` is transferred by its owner

- signing (with his private key) of (hashes of) previous transfers in which he received this bitcoin, and
- indicating the amount and (a hash of) the public key of the recipient.

The sum shown in the `Bitcoin Core` graphical user interface (initially developed by Satoshi Nakamoto under the name `Bitcoin-Qt` and used by 90 % of the traders) is the sum of all transactions that the owner received: For Alice to pay a certain amount, for example, 4 bitcoins, to Bob, the program

1. joins transactions whose sum is ≥ 4, for example,
   - one of 2, and
   - another one of 3 bitcoins,
2. transfers
   - the sum of 4 bitcoins to Bob, and
   - the change of 1 `bitcoin` to Alice.

That is, Alice received another transaction (although of a smaller value than the two initials).

Bitcoin address. The identity of each trader corresponds to his asymmetric key (`ECDSA`), a pair of

- a *public* key with 32 bytes, and
- a *private* key.

To get his *address*, several hash functions are applied to the public key. Then the final sequence of 25 letters is coded by the `Base58` whose 58 numbers are

- all the numbers,
- all lowercase, and

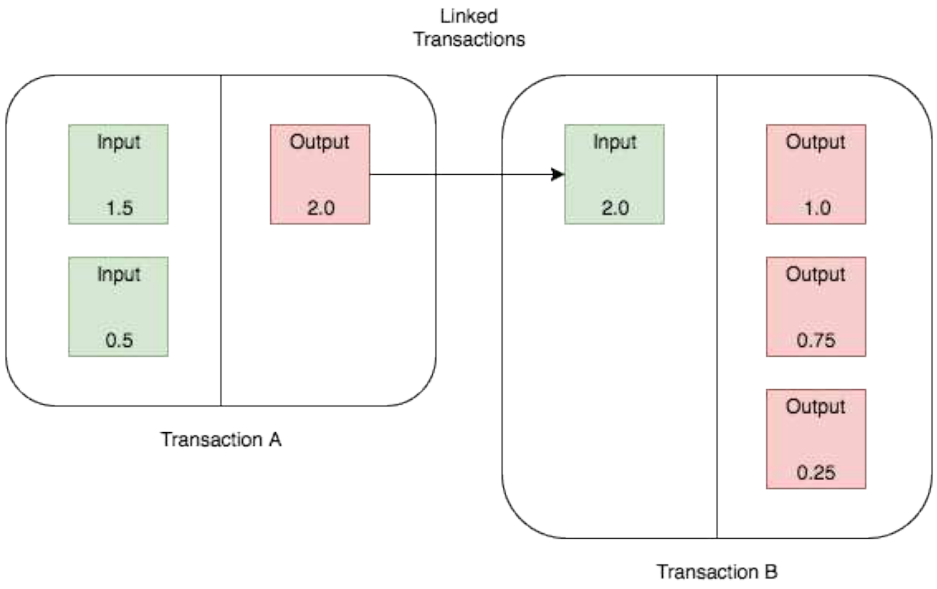


Figure 77: Exhaustion of the entries of a transaction

- all uppercase letters,
- except the number and the letter `0O` and the letters `Il` for the risk of mixing the letters of each pair up.

To calculate the hash of the public key:

1. Apply SHA-256 to the public key `ECDSA`,
2. apply `RIPEMD-160` (to the last result), and
3. prepend an instruction code (`0x0000` for P2PKH, `0x0005` for P2SH).

To add a checksum:

4. Apply once `SHA-256`,
5. apply again `SHA-256`,
6. use the first four bytes as a checksum, and
7. suffix the check sum to the result by 3.

To abbreviate the result (in a human-readable way):

8. Encodes it in Base58.

To see it in in action, visit https://gobittest.appspot.com/Address.

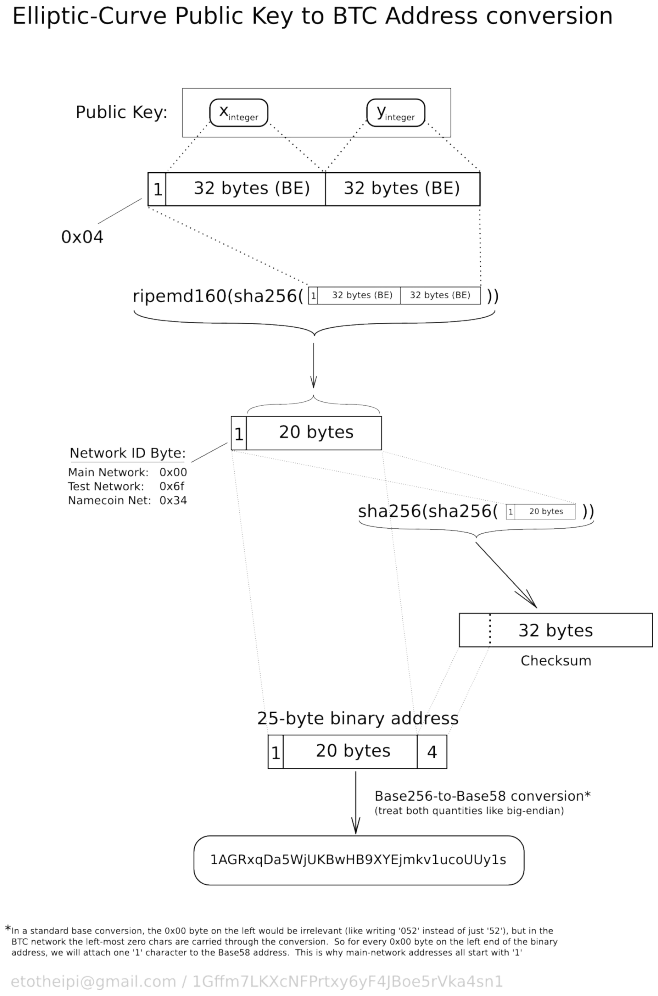


Figure 78: Public Key Conversion

To get the scriptPubkey format used in transactions:

1. Decode from Base58 to hexadecimal base,
2. remove the checksum, and
3. remove the prefix,

and finally add a certain instruction code.

The use of a hash instead of the public key as the address, besides shortening it,

- makes the transition to another asymmetric encryption algorithm more flexible in case the one currently employed (`ECDSA`) will be compromised one day, and
- hides the public key until the amount received is spent (for which the signature indicating the public key is required).

Types of Transactions. A *transaction* is a bitcoin transfer, first transmitted on the network, then collected in a block by a miner and published. All transactions are public, while (the hashes of) the public keys are anonymous, that is, they do not reveal a priori the owners' names.

There are two types of transactions:

- the *generation* of bitcoins when creating a new block, and
- the payment *between* two (groups of) users, the sender and the recipient.

In more detail:

- a *generation transaction* is determined by the miner:
  - the only information in the input is a 100 bytes `coinbase` field (instead of `scriptSig`) whose content is arbitrary. (It is often (ab)used as `ExtraNonce`, that is, additional `Nonce` for mining, since the size of `Nonce`, 4 bytes, is currently insufficient to find a block whose hash is small enough. However, unlike the `Nonce` field, `ExtraNonce` only indirectly enters the transaction through the hash of the Merkle tree root. Therefore, modifying `ExtraNonce` requires recomputing the hashes of the coinbase branch into the Merkle tree of transactions).
  - the `output` distributes the reward (of 6.25 bitcoins, plus the earned transaction fees, in 2021) to the recipients favored by the miner.
- a *payment transaction* has
  - an 'input', that lists transactions in which the sender received bitcoins, and

– an ‘output’ that transfers to the recipients an amount **equal** to the sum of the sender’s received bitcoins.

  Often, the transaction includes an **exchange**, that is, the sender appears as one of the recipients: If the sum is smaller, the difference is paid to the miner of the block that included the transaction as an incentive to swiftly consider it.

Once a block is included in the chain, and this block is extended by a sufficient number (≥ 6) of other blocks, the transaction can be considered irreversible; it is *confirmed*.

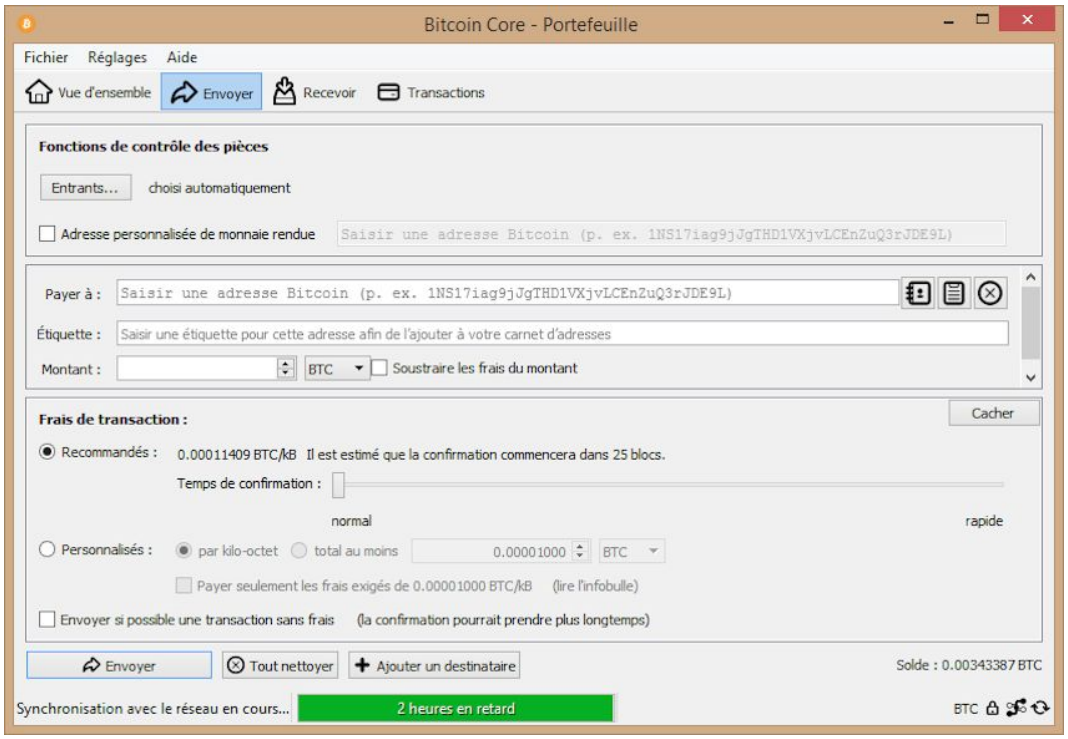


Figure 79: Transaction in `Bitcoin Core` application. Paying a transaction fee helps decrease the confirmation time.

Processing. In the bitcoin network, there are

- the *full nodes*, which *transmit and validate* the transactions, and
- the *miners*, who create the blocks that extend the chain.

A miner is incentivized

- to send the block mined by him to the nodes as soon as possible — otherwise the blockchain will be extended by someone else's block,
- to receive blocks mined by others to continue mining from this block as the end of the chain, and
- validate the transactions because every block with an invalid transaction is rejected by the nodes.

However, the miner has no incentive to forward a block mined by someone else to the node network. Instead, he rather hides its existence until he has mined an own block that extends it. This instantaneous transmission is up to the nodes.

A user is incentivized to maintain a node to ensure the integrity of the blockchain at all times. However, maintaining an entire node is above all an altruistic need that is essential to the functioning of blockchain: The chain is only safe using a partial node (which only knows part of the chain) while there are sufficiently many full nodes in the network that guarantee the validity of the blockchain.

Summary. We summarize how a transaction is processed, from its emission to its accomplishment:

1. Send a transaction through your portfolio application.
2. The transaction is diffused by the nodes and becomes part of the pool of unconfirmed transactions.
3. Miners
    1. choose transactions from this pool (preferably those that reward them the highest transaction fee, the difference between the joined amounts of all incoming respectively outgoing transactions),
    2. validate them (for example, whether the payer's balance is sufficient), and
    3. add them to the block they are trying to generate; commonly, until the maximum block size (of 1 Megabyte) is exhausted.

4. Each miner tries to modify the block so that its hash becomes small enough (currently starting with 10 null bytes). The probability of finding such a block, that is, the difficulty of the problem, is the same for all blocks and miners. This difficulty is adjusted every 2016-th block (that is, after about two weeks) so that the miners' total computational power to find such a block takes on average ten minutes.

   When a new block is appended to the chain, all the miners have to start again with another block since

   - the indicator (to the hash of the preceding block) has changed, and
   - furhter transactions were made.

5. Every miner who found a valid block transmits it to the network nodes.
6. The node validates the block, that is, checks whether

   - all its transactions are valid, and
   - its hash is small enough.

7. If the node confirms the validity of the block, then it appends it to the chain. A *confirmation* of a transaction is each extension of the chain by a block after the block containing the transaction. The extension of the chain by the block containing the transaction counts as the first confirmation, the block after this one as the second confirmation, and so on. The probability that the last six confirmations (after an average of one hour) are undone is practically zero. Therefore, a transaction is usually seen as accomplished after its sixth confirmation.

Validation. For the node to quickly check the validity of a transaction entry, that is, if the cashed in transactions have not yet been spent, he steadily updates the *Unspent Transaction Ouput* (UTXO) database of all transactions that have not yet been spent. Currently, the UTXO has about 4 GB and is stored in memory to ensure fast queries.

Let us recall that each transaction is unique, and can be spent only once and entirely. When a transaction is transmitted,

- the previous transactions used as input are removed from UTXO, and
- the new outgoing transactions are added to UTXO.

To prevent double spending, the node checks whether an outgoing transaction is in the UTXO: if so, it allows the transaction, otherwise it prevents it.

Confirmation.

Let us recall that a *confirmation* of a transaction is each extension of the chain by a block after the block containing the transaction. The extension of the chain by the block containing the transaction counts as the first confirmation, the block after this one as the second confirmation, and so on. Because it is practically impossible that the last six confirmations (after an average of one hour) are undone, a transaction is usually seen as accomplished after its sixth confirmation.

For example, the default graphical interface for bitcoin, `Bitcoin Core`, shows a transaction as confirmed when the confirmation count reached 6. However, this number of confirmations 6 is arbitrary.

In contrast, `bitcoins` *mined* can only be spent after the generated block has achieved a confirmation count of 100.

Summary

The blockchain replaces a third trusted party, the bank, that mediates financial transactions, by a database of entries that successively point to each other. This pointer is a hash of the whole previous entry, ensuring the database's integrity: thus, a change of the previous entry entails a change of the hash, thus invalidates the pointer, thus the whole chain. Because the hashes must have many leading zeros, finding valid entries, *mining*, demands much computational power and practically makes malevolent alterations impossible.

Questions

1. How long does the addition of a new block to the Bitcoin blockchain take on average?

   - ☐ one second
   - ☐ one minute
   - ☒ ten minutes
   - ☐ one hour

> **Availability of URLs**: Even after a certain URL `URL` has expired its content will still be available at the `archive.org` under the URL `https://web.archive.org/web/URL`. For example, the finite field function plotter http://graui.de/code/ffplot/ is available under https://web.archive.org/web/http://graui.de/code/ffplot/. In this case, the page was archived on May 5, 2018; its status on this date is available at https://web.archive.org/web/20180505102200/http://graui.de/code/ffplot/.

Abramson, Jay. 2019. "Graphs of Polynomial Functions." https://math.libretexts.org/Courses/Mission_College/Math_1_College_Algebra_(Carr)/04%3A_Polynomial_and_Rational_Functions/4.04%3A_Graphs_of_Polynomial_Functions.

Acdx. 2019. "File:digital Signature Diagram.svg — Wikimedia Commons, the Free Media Repository." https://commons.wikimedia.org/w/index.php?title=File:Digital_Signature_diagram.svg&oldid=362770490.

Ajtai, Miklós, and Cynthia Dwork. 1999. "A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence." In *STOC '97 (El Paso, TX)*, 284–93. ACM, New York.

Angwin, Julia. 2015. "The World's Email Encryption Software Relies on One Guy, Who Is Going Broke." https://www.propublica.org/article/the-worlds-email-encryption-software-relies-on-one-guy-who-is-going-broke.

Archwiki. 2020. "OpenVPN." https://wiki.archlinux.org/index.php/OpenVPN.

Aumasson, Jean-Philippe. 2017. *Serious Cryptography: A Practical Introduction to Modern Encryption*. No Starch Press.

Autocrypt Team. 2019. "Example Data Flows and State Transitions." https://autocrypt.org/examples.html.

Bach, Eric. 1984. *Discrete Logarithms and Factoring*. Computer Science Division, University of California Berkeley.

Barker, Elaine. 2016. "NIST Special Publication 800-57 Part 1, Revision 4."

BeEsCommonsWiki. 2015. "File:Skytale3d de.png — Wikimedia Commons, the Free Media Repository." https://commons.wikimedia.org/w/index.php?title=File:Skytale3d_de.png&oldid=154927861.

Bentz, Romain. 2019. "Kerberos En Active Directory." https://beta.hackndo.com/kerberos/.

Blum, Manuel, Paul Feldman, and Silvio Micali. 2019. "Non-Interactive Zero-

Knowledge and Its Applications." In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 329–49.

Boneh, Dan et al. 1999. "Twenty Years of Attacks on the RSA Cryptosystem." *Notices of the AMS* 46 (2): 203–13.

Borisov, Nikita, Ian Goldberg, and Eric Brewer. 2004. "Off-the-Record Communication, or, Why Not to Use PGP." In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, 77–84. https://otr.cypherpunks.ca.

Braden, R. 1989. "Requirements for Internet Hosts — Communication Layers." Internet Engineering Task Force. http://tools.ietf.org/html/rfc1122.

Brumley, David, and Dan Boneh. 2005. "Remote Timing Attacks Are Practical." *Computer Networks* 48 (5): 701–16.

Brunschwig, Patrick. 2018. "Screenshot of Enigmail." https://enigmail.net/index.php/en/.

Buonafalce, Augusto. 2014. "File:alberti Cipher Disk.jpg — Wikimedia Commons, the Free Media Repository." https://commons.wikimedia.org/w/index.php?title=File:Alberti_cipher_disk.jpg&oldid=121453667.

Carleton, Jodie. 2011. "Craft Clock." https://www.flickr.com/photos/huzzah16/3288666366.

Cesar, Jovansonlee. 2020. "Svgbob ASCII to SVG Converter." https://ivanceras.github.io/svgbob-editor/.

Chouhartem. 2016. "File:merkle-Damgard-Padding-Fr.svg — Wikimedia Commons, the Free Media Repository." https://commons.wikimedia.org/w/index.php?title=File:Merkle-Damgard-padding-fr.svg&oldid=220081315.

Clark, Jeremy. 2011. "Democracy Enhancing Technologies: Toward Deployable and Incoercible E2E Elections." PhD thesis, University of Waterloo. https://uwspace.uwaterloo.ca/handle/10012/5992.

Commons, Wikimedia. 2020. "File:rgb-Raster-Image.svg — Wikimedia Commons, the Free Media Repository." https://commons.wikimedia.org/w/index.php?title=File:Rgb-raster-image.svg&oldid=407479763.

Corbellini, Andrea. 2015a. "Elliptic Curve Cryptography: A Gentle Introduction." https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/.

———. 2015b. "Implementation of ECDHE in 'Python'." https://github.com/andreacorbellini/ecc/blob/master/scripts/ecdhe.py.

Cort, Bas de. 2018. "Protecting Your Users with Certificate Pinning." https://basdecort.wordpress.com/2018/07/18/protecting-your-users-with-certificate-pinning/.

CourtlyHades296. 2017. "File:enigma Rotor Wiring.png — Wikimedia Com-

mons, the Free Media Repository." https://commons.wikimedia.org/w/index.php?title=File:Enigma_rotor_wiring.png&oldid=260095687.

Daemen, Joan, and Vincent Rijmen. 1999. "AES Proposal: Rijndael." http://www.cs.miami.edu/home/burt/learning/Csc688.012/rijndael/rijndael_doc_V2.pdf.

———. 2002. *The Design of Rijndael*. Information Security and Cryptography. Springer-Verlag, Berlin. https://doi.org/10.1007/978-3-662-04722-4.

Diffie, Whitfield, and Martin Hellman. 1976. "New Directions in Cryptography." *IEEE Transactions on Information Theory* 22 (6): 644–54.

DNSCrypt Team. 2019a. "DNSCrypt Version 2 Protocol Specification." https://dnscrypt.info/protocol/.

———. 2019b. "Frequently Asked Questions." https://dnscrypt.info/faq/.

Driscoll, Michael. 2019. "Illustrated TLS Connection 1.3 with Each Download Explained." https://tls13.ulfheim.net/.

Ducklin, Paul. 2013. "Android Random Number Flaw Implicated in Bitcoin Thefts." Sophos Ltd. https://nakedsecurity.sophos.com/2013/08/12/android-random-number-flaw-implicated-in-bitcoin-thefts/.

Dukhovni, V. 2014. "Opportunistic Security: Some Protection Most of the Time." Internet Engineering Task Force. http://tools.ietf.org/html/rfc7435.

ElGamal, Taher. 1985. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." In *Advances in Cryptology (Santa Barbara, Calif., 1984)*, 196:10–18. Lecture Notes in Comput. Sci. Springer, Berlin. https://doi.org/10.1007/3-540-39568-7_2.

Esslinger, Bernhard et al. 2008. "CrypTool 1." https://www.cryptool.org/en/cryptool1.

——— et al. 2012. "CrypTool 2." https://www.cryptool.org/en/cryptool2.

——— et al. 2018a. "Screenshot of Diffie-Hellman Protocol in CrypTool 1 Taken by Author." https://cryptool.org/.

——— et al. 2018b. "Screenshot of RSA Encryption in CrypTool 1 Taken by Author." https://cryptool.org/.

Feige, Uriel, Amos Fiat, and Adi Shamir. 1988. "Zero-Knowledge Proofs of Identity." *J. Cryptology* 1 (2): 77–94. https://doi.org/10.1007/BF02351717.

Fiat, Amos, and Adi Shamir. 1987. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems." In *Advances in Cryptology—CRYPTO '86 (Santa Barbara, Calif., 1986)*, 263:186–94. Lecture Notes in Comput. Sci. Springer, Berlin. https://doi.org/10.1007/3-540-47721-7_12.

Frederick) Friedman, William F. (William. 1976. *The Classic Elements of Cryptanalysis: With New Added Problems for the Solver*. Vol. 3. Aegean Park Press.

https://www.marshallfoundation.org/library/wp-content/uploads/sites/16/2014/09/WFFvol08watermark.pdf.

Friedl, Stephen J. 2005. "An Illustrated Guide to IPsec." http://www.unixwiz.net/techtips/iguide-ipsec.html.

GmbH, Mailvelope. 2020. "Screenshot of Public Key Generated by Mailvelope." https://www.mailvelope.com/.

Goldwasser, Shafi, and Silvio Micali. 1984. "Probabilistic Encryption." *J. Comput. System Sci.* 28 (2): 270–99. https://doi.org/10.1016/0022-0000(84)90070-9.

Goldwasser, Shafi, Silvio Micali, and Charles Rackoff. 1989. "The Knowledge Complexity of Interactive Proof Systems." *SIAM J. Comput.* 18 (1): 186–208. https://doi.org/10.1137/0218012.

Gordon, Daniel M. 1993. "Discrete Logarithms in GF(p) Using the Number Field Sieve." *SIAM J. Discrete Math.* 6 (1): 124–38. https://doi.org/10.1137/0406010.

Grau, Sascha. 2018a. "Elliptic Curve Plotter." http://www.graui.de/code/elliptic2/.

———. 2018e. "Modular Function Plotter." http://graui.de/code/ffplot/.

———. 2018c. "Modular Function Plotter." http://graui.de/code/ffplot/.

———. 2018b. "Modular Function Plotter." http://graui.de/code/ffplot/.

———. 2018d. "Modular Function Plotter." http://graui.de/code/ffplot/.

Hellisp. 2014. "File:DES-f-Function.png — Wikimedia Commons, the Free Media Repository." https://commons.wikimedia.org/w/index.php?title=File:DES-f-function.png&oldid=141155336.

Heys, Howard M. 2002. "A Tutorial on Linear and Differential Cryptanalysis." *Cryptologia* 26 (3): 189–221.

Kahn, David. 1996. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet.* Scribner. https://books.google.de/books?id=SEH/_rHkgaogC.

Kaminsky, Alan. 2004. "Cryptographic One-Way Hash Functions." https://www.cs.rit.edu/~ark/lectures/onewayhash/onewayhash.shtml.

Kku. 2019. "File:web of Trust-En.svg — Wikimedia Commons, the Free Media Repository." https://commons.wikimedia.org/w/index.php?title=File:Web_of_Trust-en.svg&oldid=375289053.

Koch, Werner et al. 2020. "Screenshot of Key Generation by 'GnuPG 2.2.5' on the Command Line in Linux Taken by Author." https://gnupg.org/.

Kocher, Paul C. 1996. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems." In *Annual International Cryptology Conference*, 104–13. Springer.

Krahmer, Sebastian. 2019. “Opmsg.” https://www.cryptool.org/en/cryptool1.

Krawczyk, Hugo, Mihir Bellare, and Ran Canetti. 1997. “RFC2104: HMAC: Keyed-Hashing for Message Authentication.” Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc2104.

Krekel, Holger, Karissa McKelvey, and Emil Lefherz. 2018. “How to Fix Email: Making Communication Encrypted and Decentralized with Autocrypt.” *XRDS: Crossroads, The ACM Magazine for Students* 24 (4): 37–39.

L’Ecuyer, Pierre, and Richard Simard. 2007. “TestU01: A c Library for Empirical Testing of Random Number Generators.” *ACM Trans. Math. Softw.* 33 (4). https://doi.org/10.1145/1268776.1268777.

Lenstra, A. K., H. W. Lenstra Jr., M. S. Manasse, and J. M. Pollard. 1993. “The Number Field Sieve.” In *The Development of the Number Field Sieve*, 1554:11–42. Lecture Notes in Math. Springer, Berlin. https://doi.org/10.1007/BFb0091537.

Lenstra, Arjen K. 2006. “Key Lengths.” In *Handbook of Information Security, Volume 1: Key Concepts, Infrastructure, Standards and Protocols*. Wiley.

Lenstra, A., and E. Verheul. 2001. “Selecting Cryptographic Key Sizes.” *Journal of Cryptology* 14: 255–93.

“List of Random Number Generators — Wikipedia, the Free Encyclopedia.” 2020. https://en.wikipedia.org/w/index.php?title=List_of_random_number_generators&oldid=942835169. https://en.wikipedia.org/wiki/List_of_random_number_generators.

Lynn-Miller, Beau Danger. 2007. “How Hash Algorithms Work.” http://www.metamorphosite.com/one-way-hash-encryption-sha1-data-software.

Melnikov, A. 2011. “RFC6331: Moving DIGEST-MD5 to Historic.” Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc6331.

Menezes, Alfred J., Paul C. van Oorschot, and Scott A. Vanstone. 1997. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and Its Applications. CRC Press, Boca Raton, FL. http://www.cacr.math.uwaterloo.ca/hac/.

Menon-Sen, Abhijit, N Williams, A Melnikov, and C Newman. 2010. “Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms.” Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc5802.

Merkle, Ralph C. 1990. “A Certified Digital Signature.” In *Advances in Cryptology—CRYPTO ’89 (Santa Barbara, CA, 1989)*, 435:218–38. Lecture Notes in Comput. Sci. Springer, New York. https://doi.org/10.1007/0-387-34805-0_21.

Merlinux GmbH. 2018. “Screenshot of DeltaChat.” https://delta.chat/.

Mockapetris, P. 1987. “Domain Names — Implementation and Specification.” Internet Engineering Task Force. http://tools.ietf.org/html/rfc1035.

Moser, Jeff. 2009. “A Stick Figure Guide to the Advanced Encryption Standard (AES).” http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html.

National Institute for Standards and Technology. 2000. “AES Competition.” https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development.

Nyr. 2019. “Openvpn-Install.” https://github.com/Nyr/openvpn-install.

O’Connor, Jack. 2022. “The SHA-256 Project Developed for NYU Tandon’s Applied Cryptography Course.” https://github.com/oconnor663/sha256_project.

Omerta-ve. 2012. “File:kerberos-Funcion.svg — Wikimedia Commons, the Free Media Repository.” https://commons.wikimedia.org/w/index.php?title=File:Kerberos-funcion.svg&oldid=77112632.

Petticrew, Ian. 2018. “File:a Turing Bombe, Bletchley Park - Geograph.org.uk - 1590996.jpg — Wikimedia Commons, the Free Media Repository.” https://commons.wikimedia.org/w/index.php?title=File:A_Turing_Bombe,_Bletchley_Park_-_geograph.org.uk_-_1590996.jpg&oldid=289828731.

Quisquater, Jean-Jacques, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, and Soazig Guillou. 1989. “How to Explain Zero-Knowledge Protocols to Your Children.” In *Conference on the Theory and Application of Cryptology*, 628–31.

Ramsdell, B. 2009. “Secure/Multipurpose Internet Mail Extensions (s/MIME) Version 3.1.” Internet Engineering Task Force. http://tools.ietf.org/html/rfc3851.

Rescorla, Eric, Kazuho Oku, Nick Sullivan, and Christopher A Wood. 2018. “Encrypted Server Name Indication for TLS 1.3.” https://tools.ietf.org/html/draft-ietf-tls-esni-02.

Rivest, Ronald L, Adi Shamir, and Leonard Adleman. 1978. “A Method for Obtaining Digital Signatures and Public-Key Ciphers.” *Communications of the ACM* 21 (2): 120–26.

Sandia Corporation. 2014. “DNSViz, a Tool That Visually Analyses the DNSSEC Authentication Chain for a Domain Name and Its Resolution Path in the DNS Namespace.” https://dnsviz.net/.

Sasaki, M., M. Fujiwara, H. Ishizuka, W. Klaus, K. Wakui, M. Takeoka, S. Miki, et al. 2011. “Field Test of Quantum Key Distribution in the Tokyo QKD

Network." *Opt. Express* 19 (11): 10387–409. https://doi.org/10.1364/OE.19.010387.

Schneier, Bruce. 2007. *Schneier's Cryptography Classics Library: Applied Cryptography, Secrets and Lies, and Practical Cryptography*. Wiley Publishing.

Schnorr, C.-P. 1991. "Factoring Integers and Computing Discrete Logarithms via Diophantine Approximation." In *Advances in Cryptology—EUROCRYPT '91 (Brighton, 1991)*, 547:281–93. Lecture Notes in Comput. Sci. Springer, Berlin. https://doi.org/10.1007/3-540-46416-6_24.

Schoof, René. 1995. "Counting Points on Elliptic Curves over Finite Fields." In *J. Théor. Nombres Bordeaux*, 7:219–54. 1. http://jtnb.cedram.org/item?id=JTNB_1995__7_1_219_0.

Shoup, Victor. 1997. "Lower Bounds for Discrete Logarithms and Related Problems." In *Advances in Cryptology—EUROCRYPT '97 (Konstanz)*, 1233:256–66. Lecture Notes in Comput. Sci. Springer, Berlin. https://doi.org/10.1007/3-540-69053-0_18.

Simmons, Gustavus J et al. 2016. "Cryptology." In *Encyclopædia Britannica*. Chicago: University of Chicago. https://www.britannica.com/topic/cryptology.

SimplyScience.ch, Rédaction. 2014. "Le Décalage Revient à Tourner l'alphabet, Ici de 3 Positions." https://www.simplyscience.ch/archives-enfants/articles/les-alphabets-codes-pour-preserver-tes-secrets.html.

Singh, S. 2000. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Anchor Books. https://books.google.de/books?id=skt7TrLK5uYC.

Smith, Jan. 2009. "Obtuse Clock." https://www.flickr.com/photos/ric-rac/5977088713/in/set-72157632352987928.

Smith, Rick. 2008. "Stream Cipher Reuse: A Graphic Example." https://cryptosmith.com/2008/05/31/stream-reuse/.

Snipes, Ryan. 2019. "Thunderbird, Enigmail and OpenPGP." https://blog.thunderbird.net/2019/10/thunderbird-enigmail-and-openpgp/.

Sweigart, Albert. 2013a. *Hacking Secret Ciphers with Python*. Gratis.

———. 2013b. "Hacking Secret Ciphers with Python." https://inventwithpython.com/hacking/chapter24.html.

Thuresson. 2013. "File:marian Rejewski.jpg — Wikimedia Commons, the Free Media Repository." https://commons.wikimedia.org/w/index.php?title=File:Marian_Rejewski.jpg&oldid=107782161.

V., LeaderTelecom B. 2019. "Google and Mozilla Will Stop Displaying Company Name in the Address Bar of the Browser." LeaderTelecom B.V. https:

//www.leaderssl.com/news/492-google-and-mozilla-will-stop-displaying-company-name-in-the-address-bar-of-the-browser.

Walmart. 2019. “2 Round 7 Day Pill Box Medicine Organizer.” https://www.walmart.com/ip/2-Round-7-Day-Pill-Box-Medicine-Organizer-Daily-Weekly-Medication-Holder-Travel/199344006.

Wassermann, A. 2020b. “Simple Function Plotter.” http://jsxgraph.uni-bayreuth.de/wiki/index.php/Simple_function_plotter.

———. 2020a. “Simple Function Plotter.” http://jsxgraph.uni-bayreuth.de/wiki/index.php/Simple_function_plotter.

Webster, Noah. 1913. *Webster’s Revised Unabridged Dictionary of the English Language*. G. & C. Merriam Company. http://www.mso.anu.edu.au/~ralph/OPTED/index.html.

Zabala, Enrique. 2019a. “Rijndael Animation v.4.” https://formaestudio.com/portfolio/aes-animation/.

———. 2019b. “Rijndael Inspector v.1.1.” https://formaestudio.com/portfolio/aes-animation/.

Zeilenga, Kurt D. 2008. “CRAM-MD5 to Historic.” Internet Engineering Task Force (IETF). https://tools.ietf.org/html/draft-ietf-sasl-crammd5-to-historic-00.

Ziegler, Joachim. 2009. “Floating Point Numbers of Arbitrary Size and Precision (Class Bigfloat).” https://www.leda-tutorial.org/en/unofficial/ch06s03.html.