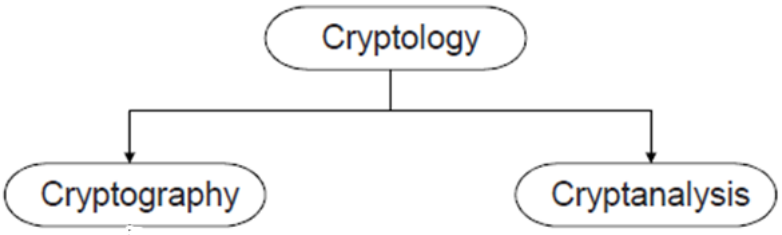
Introduction to Cryptology

What is cryptology?

The word "cryptology" comes from the Greek κρυπτός (kryptos), which means "hidden" or "secret," and -λογία (-logia), which means "study of." Together, the word "cryptology" means "the study of hidden or secret (communication)."

So, cryptology is the study of techniques for secure communication and storage of information. It encompasses both *cryptography*, which is the practice of creating and using codes to protect information, and *cryptanalysis*, which is the study of methods for breaking codes and reading encrypted messages.



Cryptography

The word "cryptography" comes from the same root as "cryptology." It is made up of κρυπτός (kryptos) and -γραφία (-graphia), which means "writing" or "drawing." Together, the word "cryptography" means "the practice of writing or drawing in secret code."

Cryptography is a practical application of the principles and techniques of cryptology. It involves designing and analyzing algorithms and protocols for secure communication. There are many situations where cryptography is used to secure communication or protect data, for example:

1. **Online shopping**: When you shop online and enter your credit card information, the website uses cryptography to secure the transmission of your personal and financial information.

2. **Email communication**: When you send an email, it is usually transmitted over the internet in an unencrypted form. This means that anyone who has access to the network could potentially intercept and read your email. To prevent this, many email providers use cryptography to secure the transmission of emails.

3. **Mobile phone communication**: When you make a phone call on your mobile phone, the call is usually transmitted over the air in an unencrypted form. This means that anyone with the right equipment could potentially listen in on your call. To prevent this, many mobile phones use cryptography to secure the transmission of calls.

4. **File storage**: When you store files on your computer or in the cloud, you may want to protect them from unauthorized access. Cryptography can be used to secure the files by encrypting them with a password or a key.

5. **Access to restricted areas/systems**: Cryptography is often used to control access to restricted areas or systems. For example, an employee might use a smart card with a cryptographic key to access a secure building or computer system.

Most important cryptographic goals

Cryptography has several goals, including confidentiality, integrity, authentication, and non-repudiation.

- Confidentiality means ensuring that information is only accessible to those who are authorized to see it. This can be achieved with encryption, which scrambles the information in such a way that it can only be read by someone who has the proper decryption key.
    - For example, when you send an email, it is usually transmitted over the internet in an unencrypted form. This means that anyone who has access to the network could potentially intercept and read your email. To prevent this, many email providers use cryptography to secure the transmission of emails.

- Integrity means ensuring that information has not been modified or tampered with during transmission or storage. This can be achieved by using hash functions, which produce a fixed-size output (called a "hash value") that is unique to the input. Any change to the input will result in a different hash value, making it easy to detect whether the information has been altered.
    - For example, when you download a file from the internet, the file may be digitally signed with a cryptographic hash to ensure that it has not been modified during transmission.

- Authentication means verifying the identity of a user or device. This can be achieved using passwords, biometric authentication, or digital certificates.
    - For example, when you log in to a website, your password is encrypted and sent to the server, which uses cryptography to verify your identity.

- Non-repudiation means ensuring that a sender cannot deny having sent a message. This can be achieved with digital signatures, which use encryption to bind a sender's identity to the message.
    - For example, when you send an email, it may be digitally signed with your private key to prove that you are the sender and cannot later deny having sent it.

Cryptanalysis

Cryptanalysis is concerned with finding weaknesses in cryptographic systems and algorithms, and with developing methods for exploiting those weaknesses to defeat the security of the system. It is an important field of study, as it helps to identify and fix vulnerabilities in cryptographic algorithms and protocols.

Some main types of cryptanalysis are:

- Ciphertext-only cryptanalysis involves trying to decrypt a message when the attacker has only the encrypted message (ciphertext) and no other information about the encryption process. Cryptanalysis may include trying different encryption keys or analyzing patterns in the ciphertext to try to find clues about the encryption method.

- Known-plaintext cryptanalysis involves trying to decrypt a message when the attacker has access to both the original message (called plaintext) and the encrypted version of the same. Cryptanalysis would include comparing the ciphertext and plaintext and looking for patterns or other clues about the encryption method.

Exercise 1

a) Which cryptographic goal is about ensuring that information is only accessible to authorized parties?
b) Which cryptographic goal is to ensure that a sender cannot deny having sent a message?
c) Which cryptographic goal involves verifying the identity of a user or device?
d) Which cryptographic goal involves ensuring that information has not been modified or tampered with during transmission or storage?

Exercise 2

a) What is the type of cryptanalysis that involves trying to decrypt a message when the attacker has only the encrypted message and no other information about the encryption process?
b) What is the type of cryptanalysis that involves trying to decrypt a message when the attacker has both the encrypted message and some unencrypted messages that were encrypted using the same encryption key?

Exercise 3

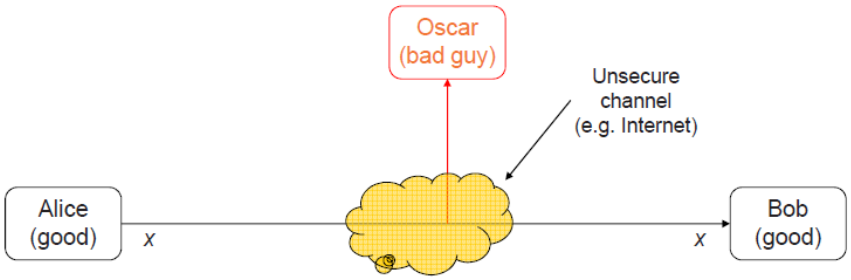
Discuss how cryptography may be applied in the following activities. Specify which cryptographic goals one may want to reach for in each case (and why).

a) Secure communication
b) E-commerce
c) Password protection
d) File storage
e) Network security

Symmetric Cryptography

Communication over an unsecure channel

The general setting where cryptography may be applied, involves communicating data over an unsecure channel. Traditionally in cryptography, the *parties* of the communication are called Alice (A) and Bob (B). Alice and Bob can represent users or devices. A possible *adversary* or attacker is here denoted by Oscar (traditionally "Eve"). There can be different scenarios as for what the adversary is capable of: hacking an internet router, eavesdropping radio signals, interfering with the data in the channel etc.



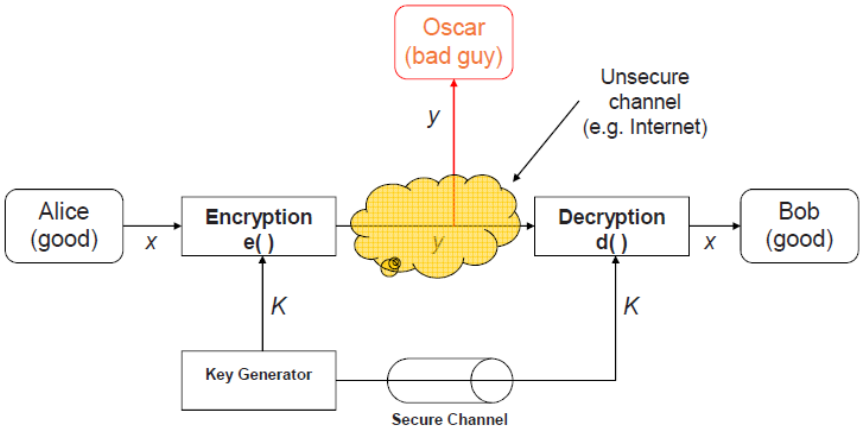
1 Understanding Cryptography by Christof Paar and Jan Pelzl

Some essential terminology and notations:

- *Plaintext* x is the original data
- *Ciphertext* y is the data in an unreadable form
- *Encryption* means transforming the plaintext into ciphertext
- *Decryption* means reversing the ciphertext back to the original plaintext
- *Key* K is an element of the *key space* $\{K_1,\ K_2,\ \ldots,\ K_n\}$, which is the set of all possible keys
- *Key length* $|K|$ is the length of an individual key in bits. For example, key length 40 yields key space of size 2^{40}.
- *Encryption algorithm* e, encryption operation $e_K(x) = y$
- *Decryption algorithm* d, decryption operation $d_K(y) = x$
- A specific algorithm for performing encryption or decryption is a *cipher*

Symmetric ciphers

Ciphers can be broadly classified into two categories: symmetric-key ciphers and asymmetric-key ciphers. Symmetric ciphers have been used for centuries to protect sensitive, and they continue to be an important part of modern cryptographic systems. The principle of symmetric ciphers is that the same secret key K is used both for encrypting the plaintext and decrypting the ciphertext.

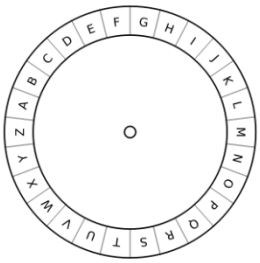


The security of symmetric ciphers is dependent on the secrecy of the key. There is a need for a secure channel for key agreement (or to use pre-stored keys), which is a potential disadvantage of symmetric ciphers. On the other hand, they tend to be fast and lightweight compared to other ciphers. Examples of popular symmetric ciphers are AES, DES, and Blowfish.

Some historical symmetric ciphers

Some of the oldest known symmetric ciphers are the simple substitution ciphers that date back to ancient civilizations. Some examples include:

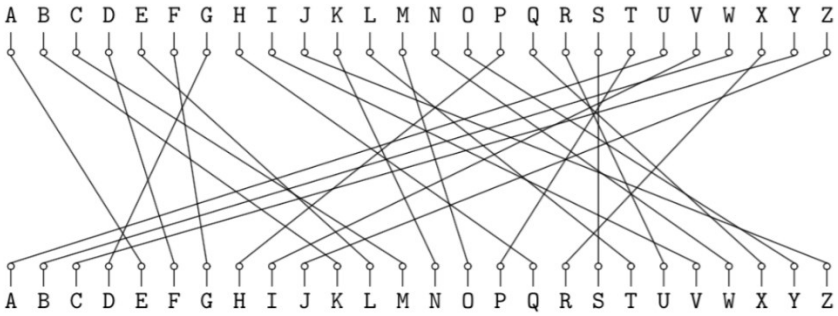
- Caesar cipher: a simple substitution cipher that replaces each letter in a message with a letter a fixed number of positions down the alphabet. The Caesar cipher is over 2000 years old.



- Affine Cipher: a type of monoalphabetic substitution cipher similar to the Caesar cipher. Instead of using a fixed shift, it uses a linear function to transform the plaintext letter into the corresponding ciphertext letter.

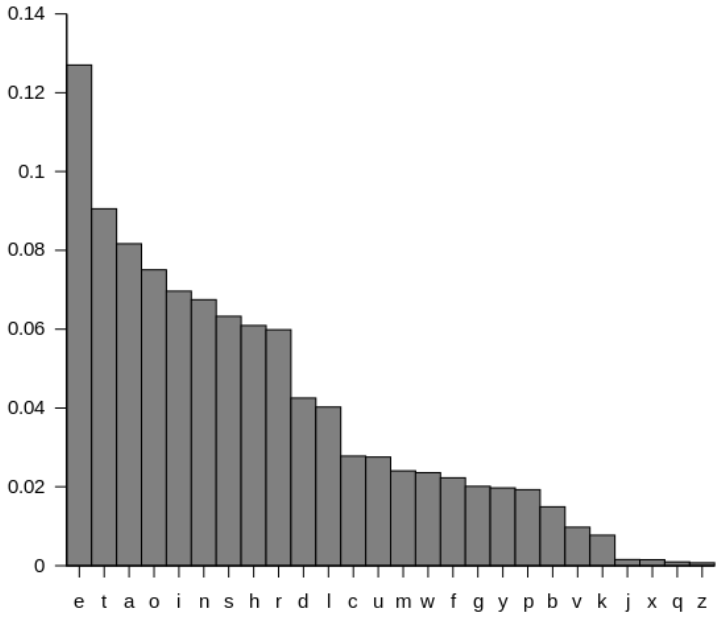
- Vigenère cipher: a polyalphabetic substitution cipher that uses a different Caesar cipher for each letter in the message, with the shift value determined by a repeating key. The Vigenère cipher is also an ancient symmetric cipher, dating back to the 16th century.

Historical symmetric ciphers provide an example of how elemental the key space size is for the security of any cipher. A small key space allows for an efficient brute force attack, a type of attack in which an attacker systematically exhausts every possible key in order to decrypt a message or crack a password.



Substitution cipher is an example of a historical symmetric cipher, which has an extensive key space and thereby is quite safe against brute force attack. However, it is vulnerable to another type of attack called frequency analysis. Since certain letters and groups of letters appear with different frequencies in each language, these frequencies can be used to identify the most likely mapping of ciphertext letters to plaintext letters.

To perform frequency analysis, we count the frequency of each letter or group of letters in the ciphertext and compare them to the known frequencies of letters or groups of letters in the plaintext language. The letters or groups of letters that have the highest frequency in the ciphertext are assumed to be the most common plaintext letters or groups of letters, such as the letter "e" or the group of letters "th" in the English language.



3Letter frequency in the English language

% calculated		% expected	
E	405×	13.98%	
T	296×	10.22%	
I	227×	7.84%	
A	220×	7.59%	
R	204×	7.04%	
S	180×	6.21%	
N	180×	6.21%	
O	165×	5.7%	
C	141×	4.87%	
H	134×	4.63%	
P	111×	3.83%	
L	105×	3.62%	
Y	74×	2.55%	
F	68×	2.35%	
M	64×	2.21%	
D	63×	2.17%	
U	56×	1.93%	
G	50×	1.73%	
B	38×	1.31%	
K	37×	1.28%	
X	27×	0.93%	
V	23×	0.79%	
W	18×	0.62%	
Q	8×	0.28%	
Z	3×	0.1%	

4*Letter frequency in this chapter*

Mathematical description of Caesar and Affine Ciphers

Let us formulate mathematically the encryption and decryption operations of Caesar and Affine ciphers. We assume to be working with the English alphabet, which has 26 letters. Each letter of the plaintext is encoded as a number: $a \to 0, b \to 1, \dots, z \to 25$, and then we operate on these numbers modulo 26 to ensure that we keep within the alphabet range. So, plaintext x and ciphertext y are both in Z_{26}.

A	B	C	D	E	F	G	H	I	J	K	L	M
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
13	14	15	16	17	18	19	20	21	22	23	24	25

The encryption and decryption operations of the Caesar cipher are defined as follows:

- encryption function $e_k(x) = x + k \ (mod\ 26)$
- decryption function $d_k(y) = y - k \ (mod\ 26)$

The encryption and decryption operations of the Affine cipher are defined as

- encryption: $e_k(x) = ax + b \ (mod\ 26)$
- decryption: $d_k(y) = a^{-1}(y - b) \ (mod\ 26)$

  with the key $k = (a, b) \in Z^2$, where $\gcd(a, 26) = 1$

Example. Let the key $k = (a, b) = (15,19)$. We note that $\gcd(15,26) = 1$.
Hence a has an inverse a^{-1}, namely $a^{-1} = 7$.
[To check this, we find that $15 \cdot 7 = 105 = 4 \cdot 26 + 1 \equiv 1 \ (mod\ 26)$.]

Let us encrypt message $ATTACK = x_1x_2 \dots x_6 = 0,19,19,0,2,10$ by computing

$y_1 = 15 \cdot 0 + 19 \ (mod\ 26) = 19,$
$y_2 = 15 \cdot 19 + 19 \ (mod\ 26) = 18$
$\vdots$
resulting in ciphertext $y = y_1y_2 \dots y_6 = 19,18,18,19,23,13 = TSSTXN$.

To decrypt the ciphertext, we compute

$x_1 = 7(19 - 19)(mod\ 26) = 0,$
$x_2 = 7(18 - 19)(mod\ 26) = 19$
$\vdots$

resulting back in $x_1x_2 \dots x_6 = 0,19,19,0,2,10 = ATTACK$

Exercise 4

a) Knowing that the ciphertext Rpalz huk khyaz was encrypted using Caesar cipher, find the plaintext and the key k that was used for encryption.
b) Use the Caesar cipher with key $k = 14$ to encrypt and the plaintext SECRET. Show the steps of computation.

Exercise 5

a) Which one of the following is a valid key for Affine cipher and why?
   a. $k = (8,17)$
   b. $k = (21,8)$
   c. $k = (13,14)$

b) Use the valid key from above to encrypt the plaintext PRIVATE. Show the steps of computation.

Exercise 6 Performing a brute force attack, on average we have to try 50 % of all keys before finding the correct one.

Suppose you are trying to break a system protected by a 5-digit PIN code. Each digit of the code is a number between 0 and 9. It takes just 2 seconds to generate and test one code, but after every three failed login attempts the system will be locked for 10 minutes. How long will it take on average to break the system by brute force?

Exercise 7 Oscar is trying to break ciphertext y on brute force. It takes him 1 second to generate a key k' and check if $d_{k'}(y) = x$. How long will it take him (on average) to decipher y, when the size of key space K is

a. 2^{10}
b. 2^{30}

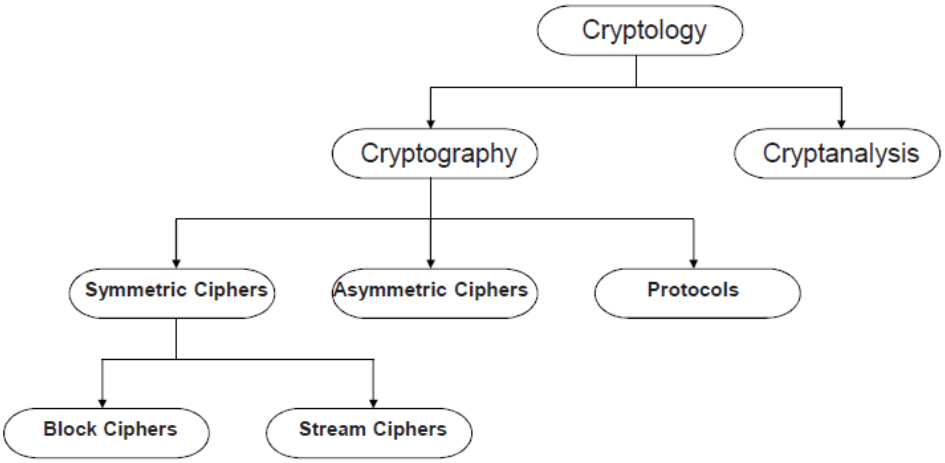
Exercise 8 Run a frequency analysis on the text of this Wikipedia article on cryptography: https://en.wikipedia.org/wiki/Cryptography .

Analyze the frequencies of letters A-Z for single characters, bigrams and trigrams.

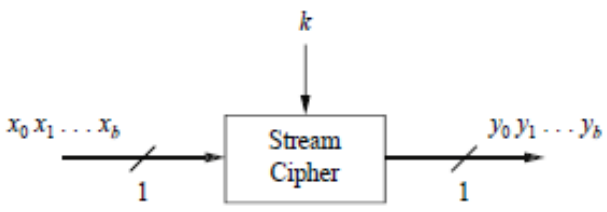
Show your results. What can you say about the results compared to the respective general frequencies in English?

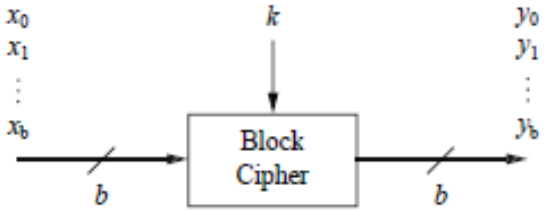
Stream Ciphers

Stream ciphers and block ciphers



Symmetric ciphers can be divided in two categories: stream ciphers and block ciphers. A stream cipher encrypts plaintext one bit at a time, while a block cipher encrypts a fixed number of bits (called a block size) at a time. This means that stream ciphers are more efficient for encrypting data of an unknown or variable length, while block ciphers are better suited for fixed-length messages or data. Stream ciphers typically use a simpler encryption algorithm than block ciphers, as they only need to encrypt small amounts of data at a time.





Stream ciphers are widely used in cryptography due to their simplicity and efficiency in encrypting data streams. Primarily they are used for real-time, low-latency encryption of data that is being transmitted over a communication channel. They are often used in wireless and mobile communications, such as cellular networks, satellite communications, and wireless LANs, as well as in various industrial control systems, SCADA systems and IoT devices.

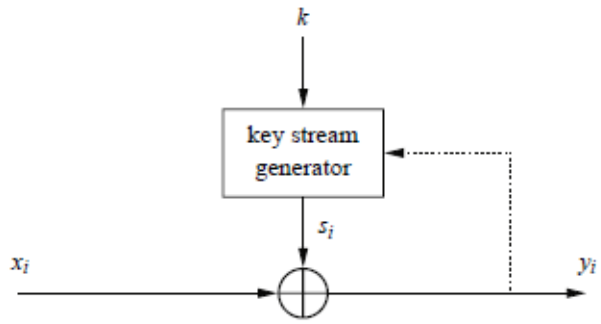
Stream ciphers can also be used for encrypting files and other forms of data, but this is less common due to the lack of random access, which makes it difficult to update or modify the encrypted data without decrypting and re-encrypting the entire file.

Some specific examples of stream cipher use:

- RC4 is widely used as the default cipher for WPA (Wi-Fi Protected Access) wireless networks.
- Salsa20 is a family of stream ciphers, which is used in a number of applications, including the Tor anonymity network and the Linux kernel's random number generator.
- A5/1 and A5/3 are stream ciphers used in the GSM mobile communication standard.

Synchronous and asynchronous stream ciphers

There are two types of stream ciphers that differ in how they generate the keystream used for encryption: synchronous and asynchronous stream ciphers. The main difference is that the former generates keystream *independently* of the plaintext and ciphertext, while the latter generates keystream *based on* the plaintext and ciphertext.

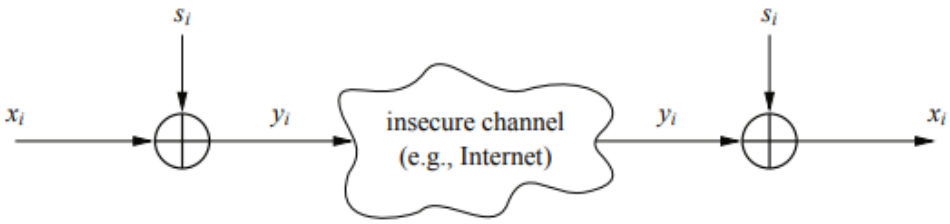


A synchronous stream cipher generates the keystream s based on a fixed key k and an internal state. The keystream is generated by repeatedly applying a pseudorandom function to the internal state, independently of the plaintext and ciphertext. The plaintext is then encrypted by XORing it with the keystream. The encryption process is deterministic, meaning that the same plaintext will always be encrypted to the same ciphertext when the same key is used.

An asynchronous stream cipher on the other hand, generates the keystream based on the plaintext and ciphertext. The key is used to initialize the cipher, and the keystream is then generated by repeatedly applying a pseudorandom function to the plaintext and ciphertext. The keystream is again used to encrypt the plaintext by XORing it with the keystream. The encryption process here is non-deterministic, as the same plaintext will not necessarily be encrypted to the same ciphertext when the same key is used.

Synchronous stream ciphers are generally considered to be more secure than asynchronous stream ciphers because the keystream is not dependent on the plaintext and ciphertext. However, asynchronous stream ciphers have the advantage of being more efficient, as they do not require the storage of large keystream buffers.

Stream cipher encryption / decryption operation



In a stream cipher, encryption and decryption operations are both based on the XOR operation. In encryption, the plaintext is XORed with a keystream generated by the stream cipher. In decryption, the process is reversed: the ciphertext is XORed with the keystream to obtain the original plaintext. To express the operations in terms of modular arithmetic, we use modulus 2:

Encryption: $y_i = e_{s_i}(x_i) = x_i + s_i \ (mod\ 2)$

Decryption: $x_i = d_{s_i}(y_i) = y_i + s_i \ (mod\ 2)$

Example. Use key stream $s_0 s_1 \ldots = 0101101 \ldots$ to encrypt letter $A = 65_{10} = 1000001_2$ in ASCII code.

The fact that encryption and decryption are the same is due to a property of XOR operation called *invertibility* or *reversibility*. Another important property of XOR is that it is *balanced*: for any input bit, the probability for the output being 1 is equal to the output being 0. These two properties make XOR a great choice for encryption in stream ciphers.

x_i	s_i	$y_i \equiv x_i + s_i \bmod 2$
0	0	0
0	1	1
1	0	1
1	1	0

It is one of the main benefits of stream ciphers that the encryption and decryption operations can be performed very quickly and efficiently since they only involve simple bitwise operations. Additionally, stream ciphers can encrypt data in real-time, as the keystream can be generated and used to encrypt the data as it is being transmitted or stored.

Exercise 1.

a) What is the main difference between a stream cipher and a block cipher?

b) What are some common uses of stream ciphers?

c) How does a stream cipher generate the keystream used for encryption?

d) What is the main benefit of using a stream cipher for encryption?

e) How are the encryption and decryption operations performed in a stream cipher?

Random Number Generators

The key in a stream cipher is used to generate or initialize the generation of the key stream. An attacker who knows the key or can predict the keystream can easily decrypt the ciphertext. Therefore, the security of a stream cipher relies on the secrecy of the key and the unpredictability of the keystream.

A random number generator (RNG) is a mathematical function that generates a sequence of random numbers. Random numbers are often used in computer programs, such as games, simulations, and cryptography. There are several different types of RNGs, including true random number generators and pseudorandom number generators.

- True random number generators (TRNGs) use physical processes, such as radioactive decay or thermal noise, to generate truly random numbers. TRNGs are nondeterministic, and relatively slow and expensive to use.

- Pseudorandom number generators, (PRNGs) use mathematical algorithms to generate numbers that appear statistically random but are not truly random. Pseudorandom number generators are deterministic, and typically faster and more efficient than TRNGs. Pseudorandom number generators can be further categorized to *ordinary* PRNGs and *cryptographically secure* PRNGs (CSPRNGs).
    - An ordinary PRNG is a simple algorithm that generates a sequence of numbers that appears random but is not necessarily difficult to predict. Ordinary PRNGs are typically used in non-security related applications, such as generating random numbers for simulations or games.
      Some examples of commonly used ordinary PRNGs:
        - Linear Congruential Generator (LCG) is a simple PRNG that uses a linear congruence equation to generate a sequence of numbers. It is one of the oldest and most widely used PRNGs.
        - Middle-Square Method generates a sequence of numbers by repeatedly squaring an initial value and taking the middle digits of the resulting number.

    - A cryptographically secure PRNG is a type of PRNG that is specifically designed to be used in security-related applications, such as generating random numbers for encryption keys. CSPRNGs are designed to be difficult to predict, even if an attacker knows the algorithm and the seed. They are also designed to be resistant to attacks that attempt to predict the next number in the sequence.
      Some examples of cryptographically secure PRNGs are:
        - The Advanced Encryption Standard (AES) in counter mode
        - The Blum Blum Shub algorithm
        - The Yarrow algorithm

The next-bit test

The next-bit test is a statistical test that is used to evaluate the quality of a pseudorandom number generator. It tests how well a PRNG imitates the unpredictability of a TRNG in that each bit in the generated sequence of numbers appears to be independent of the previous bits.

The PRNG will pass the next-bit test, if given some n bits of key stream, it is computationally infeasible to predict the next (or previous) bit with better than 50 % chance of success.

$$?, s_i, s_{i+1}, s_{i+2}, \dots, s_{i+n-1}, ?$$

The next-bit test is a simple, efficient, and widely used test. It is not comprehensive and it is recommended to use a combination of statistical tests to evaluate the quality of a PRNG.

The one-time pad

The one-time pad is a method of encryption that is considered to be completely unbreakable, as long as it is used correctly. It is essentially a stream cipher that uses a truly random key that is the same length as the plaintext message. The key is used only once to encrypt a single message.

OTP is not practical to use in most situations, due to the key distribution and management challenges. However, theoretically it is "the perfect design". It is *unconditionally secure* in that if the key is kept secret, it cannot be mathematically broken even with unlimited resources.

Exercise 2 True or false?

a) A truly random number is generated by a physical process, such as radioactive decay.

b) Pseudorandom number generators are typically slower and less efficient than true random number generators.

c) Cryptographically secure pseudorandom number generators are designed to be resistant to attacks that attempt to predict the next number in the sequence.

d) Ordinary pseudorandom number generators can be used in security-related applications.

e) The one-time pad encryption method is considered unbreakable as long as the key is truly random and kept secret.

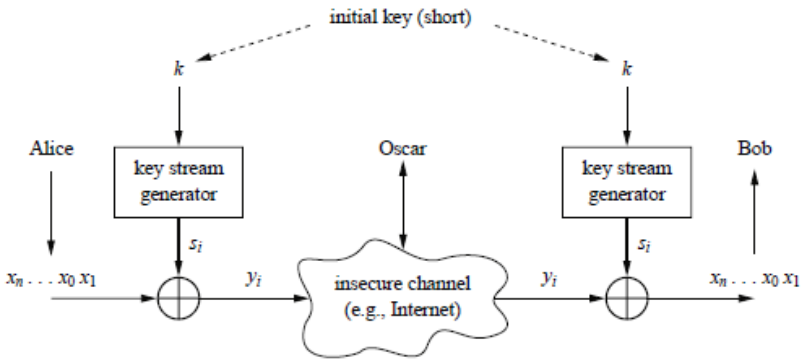
Exercise 3 An adversary has access to a plaintext-ciphertext pair (a known-plaintext attack). They know that the method of encryption is stream cipher. The texts are as follows:

Plaintext: 0100 1000 0100 0101 0100 1100 0101 0000

Ciphertext: 0001 0011 1111 1000 1011 1011 1011 1111

Show how to extract the key stream used for encryption of the plaintext. Why it is vital that the key stream appears random and does not give away any previous or subsequent key stream bits?

Principles of practical stream ciphers



Building a stream cipher for practical use, we would want to emulate the one-time pad but replace TRNG by CSPRNG and use a key k as a seed to initialize the key stream. A stream cipher can be optimized for software or hardware implementation. It is possible to use a block cipher as a stream cipher by using a mode of operation that adapts the block cipher to work on a stream of data.

One such mode of operation is called *counter mode* (CTR). In CTR mode, a counter is used as the input to the block cipher and the output of the block cipher is XORed with the plaintext to produce the ciphertext. This way, a unique keystream is generated for each plaintext block, allowing the block cipher to encrypt the plaintext one block at a time.

Another mode of operation that can be used to turn a block cipher into a stream cipher is called *Output Feedback Mode* (OFB) which encrypts the plaintext one block at a time by using the previous ciphertext block as an input to the block cipher.

These modes of operation use the same key for all plaintext blocks, making it less secure than using a different key for each plaintext block. Also, if the same key is used multiple times with the same counter or feedback value, the output will be the same, revealing the plaintext to an attacker.

Linear Shift Feedback Registers

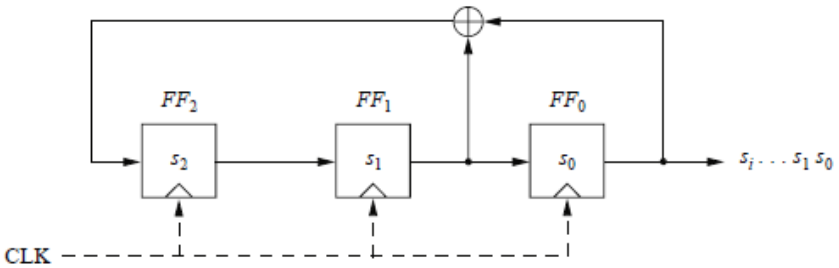
Many stream ciphers use linear feedback shift registers (LFSRs) to generate pseudorandom sequences. A LFSR is a type of finite state machine that uses shift and feedback operations (typically XOR operation) to generate a sequence of bits.

LSFRs are widely used in stream ciphers and cryptographic systems due to their high speed, low area complexity and high-level of parallelism. They are also widely used in hardware and software implementations of PRNGs. However, single LFSRs are vulnerable to certain types of attacks. Therefore, many stream ciphers use *combinations* of several LFSRs to build stronger cryptosystems.

A LFSR consist of flip-flops and storage elements. The shift operation moves the bits in the register, discarding the bits that fall out of the register and introducing a new bit at the other end. The discarded bits form a pseudorandom number sequence. The feedback operation uses some of the bits in the register to generate a new bit, which is then introduced into the register.

- The number of flip-flops is the *degree* of the LFSR.
- The *period* of the sequence is determined by the number of states of the register.
- The maximum period (sequence length) generated by an LFSR of degree m is $2^m - 1$.

Example. A LFSR of degree 3 with $s_{i+3} = s_{i+1} + s_i$ and period 7

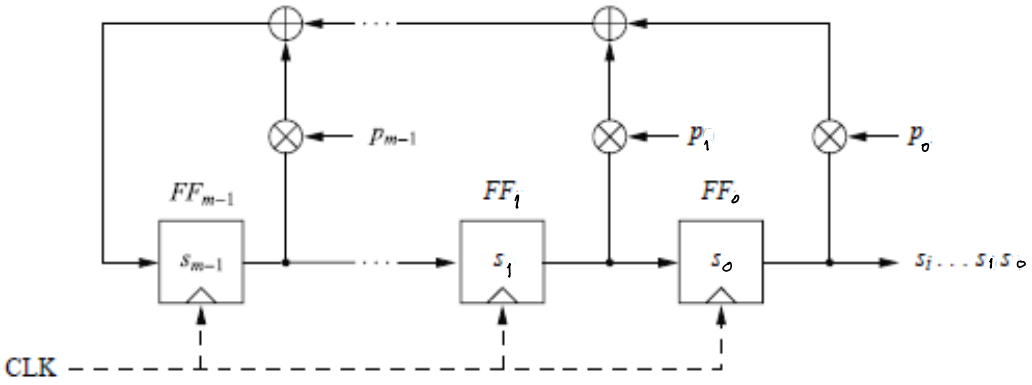


Example. We find the sequence produced by the LFSR of the previous example, given initial values $s_0 = 0,\ s_1 = 1,\ s_2 = 0$.

Clock cycle i	FF_2	FF_1	$FF_0 = s_i$

Mathematical description of LFSRs

Below is a general LFSR of degree m with initial values $s_0, s_1, \dots, s_{m-1}$. The feedback coefficients $p_0, p_1, \dots, p_{m-1}$, either have value $p_i = 0$ (switch open) or $p_i = 1$ (switch closed).



The output sequence of the LFSR has formula

$$s_{i+m} = \sum_{j=0}^{m-1} p_j \cdot s_{i+j} \pmod 2$$

where $s_i, p_j \in \{0,1\}; i = 0,1,2, \dots$

Example. Let us find the sequence produced by the LFSR with $m = 4$ and feedback coefficients $p_0 = p_1 = p_2 = p_3 = 1$, given initial values

a) $s_0 = s_1 = s_2 = s_3 = 1$
b) $s_0 = s_1 = 1,\ s_2 = s_3 = 0$

Clock cycle	FF_3	FF_2	FF_1	FF_0

Clock cycle	FF_3	FF_2	FF_1	FF_0

Exercise 4

The stream cipher can be generalized to work in alphabets other than the binary. Consider a scheme which operates on the English alphabet represented by the numbers 0,1,…,25. Note that in this case, the encryption and decryption functions are not identical.

a) What are the encryption and decryption functions here?
b) Decrypt ciphertext `rviodthvays` which was encrypted using the key `rsidpydkawo`.

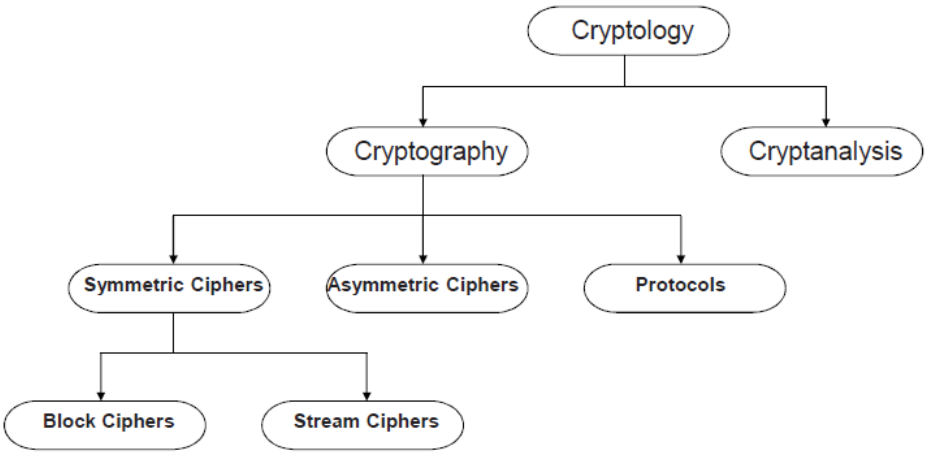
Exercise 5

Consider the LFSR with degree $m = 3$ and feedback coefficients $p_0 = 1, p_1 = 1$ and $p_2 = 0$. Find the sequence generated from the initial values $s_0 = 0, s_1 = 1, s_2 = 1$.

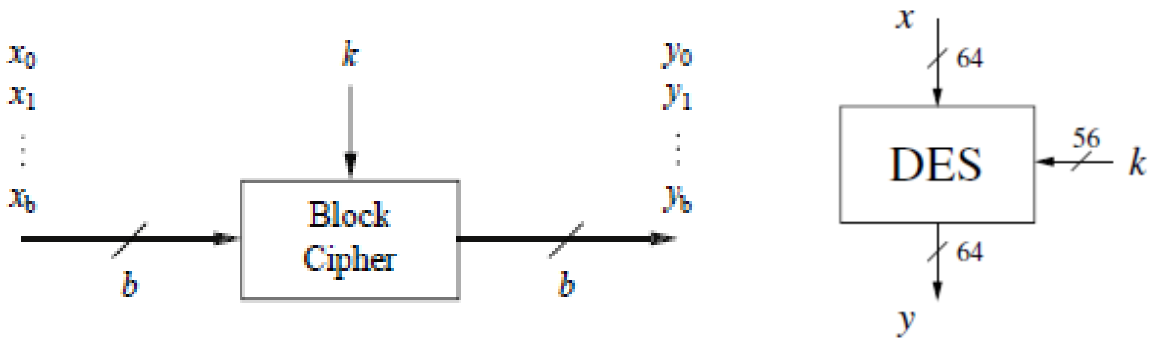
Exercise 6

Consider the LFRS with degree $m = 4$, feedback coefficients $p_0 = 0, p_1 = 1,\ p_2 = 0, p_3 = 1$ and initial values $s_0 = 0, s_1 = 1, s_2 = 1, s_3 = 1$. Find the period of this LFRS. Is it a maximum-length LFSR?

Block Ciphers



A block cipher is a symmetric-key cipher that encrypts a fixed-size block of plaintext at a time. We will study more closely two common block ciphers, namely DES (Data Encryption Standard) and AES (Advanced Encryption Standard).



DES is a block cipher that was developed in the 1970s by IBM and was later adopted as a standard by the National Institute of Standards and Technology (NIST) of USA. DES uses a 56-bit key to encrypt 64-bit blocks of data. DES has been widely used in the past, but it is weak by today's standards due to the small key size which makes it vulnerable to brute-force attacks. Despite intensive research, no efficient analytical attacks were found during the lifetime of DES.

Although DES is not recommended for use in new systems, it is still widely used in legacy systems. As DES has been the foundation of many other encryption algorithms developed since then, learning about it provides a good base for understanding the principles and designs of block ciphers in general. Understanding DES can help one appreciate the improvements in encryption technology that have been made since its development and understand the importance of regularly reviewing and updating encryption standards.

Encryption process of DES

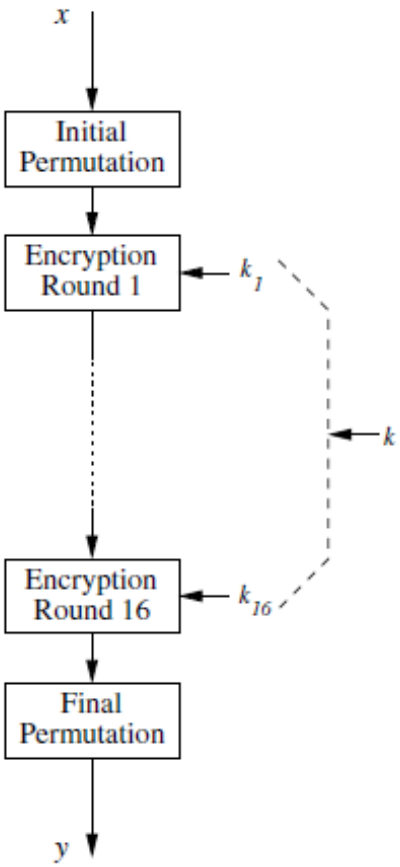
The block size of DES is 64 bits. The DES algorithm has an iterative structure. The encryption process involves a series of operations including substitution, permutation, and key mixing, which are applied to the plaintext block in multiple rounds. The number of rounds is 16, and for each round a subkey is derived from the 56-bit main key k.

The encryption process of DES can be broken down into the following steps:

1. **Initial Permutation** (IP): The plaintext block is subjected to an initial permutation to shuffle the bits in the block.
2. **Round Function**: The encryption process is divided into 16 rounds, with each round consisting of the following operations:

   - *Expansion permutation* (E/P): The 32-bit right-half of the plaintext block is expanded to 48 bits by a permutation.
   - *Key Mixing*: The 48-bit expanded right-half is mixed with the round key by applying a bitwise XOR operation.
   - *Substitution Box* (S-box): The mixed block is then divided into 8 groups of 6 bits, and each group is replaced by a 4-bit value obtained from a substitution box.
   - *Permutation* (P-box): The 32-bit output of the S-boxes is then permuted by another permutation called P-box.

   The final output of the round is XORed with the left-half of the plaintext block.
3. **Final Permutation** (FP): The final output of the 16th round is subjected to a final permutation to shuffle the bits in the block again. The result is the ciphertext.



Decryption process of DES

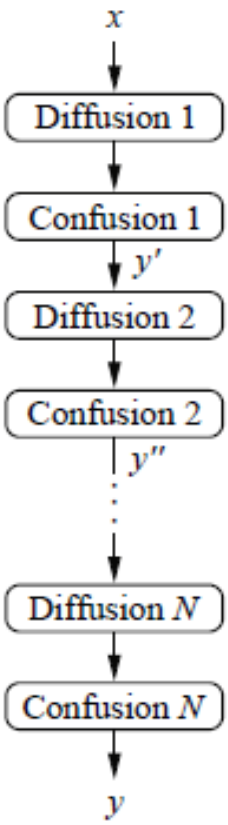
In the decryption process, the same key and algorithm are used, but the process is applied in reverse order. The ciphertext is permuted and passed through 16 rounds of the same substitution and permutation operations in reverse, to finally get the original plaintext.

It's worth noting that the encryption and decryption process in DES is complex and computationally intensive, which makes it less efficient than other modern encryption algorithms such as AES.

Confusion and diffusion

All modern block ciphers including DES are product ciphers, which are designed to achieve two fundamental properties in symmetric-key cryptography, namely confusion and diffusion. Product ciphers combine two or more simple cryptographic operations to create more secure and robust encryption systems. The main idea is to use the strengths of different encryption algorithms to compensate for each other's weaknesses.

Confusion is the process of making the relationship between the encryption key and the ciphertext as complex and non-linear as possible. This will make it difficult for an attacker to determine the encryption key by analyzing the ciphertext. In DES, confusion is achieved by several operations of the round function, such as expansion permutation, key mixing, and using substitution boxes. S-boxes are *non-linear*, meaning that the output of the S-box is not a simple function of the input. This non-linearity makes it difficult for an attacker to predict the output of the S-box based on the input.



Diffusion is the process of spreading the plaintext information uniformly over the entire ciphertext, so that a change in one plaintext bit is likely to affect many ciphertext bits. The goal of diffusion is to make it difficult for an attacker to determine the plaintext by analyzing a small portion of the ciphertext. In the case of DES, the initial permutation and final permutation operations contribute to diffusion, as well as the repeated application of the substitution and permutation operations in each round.

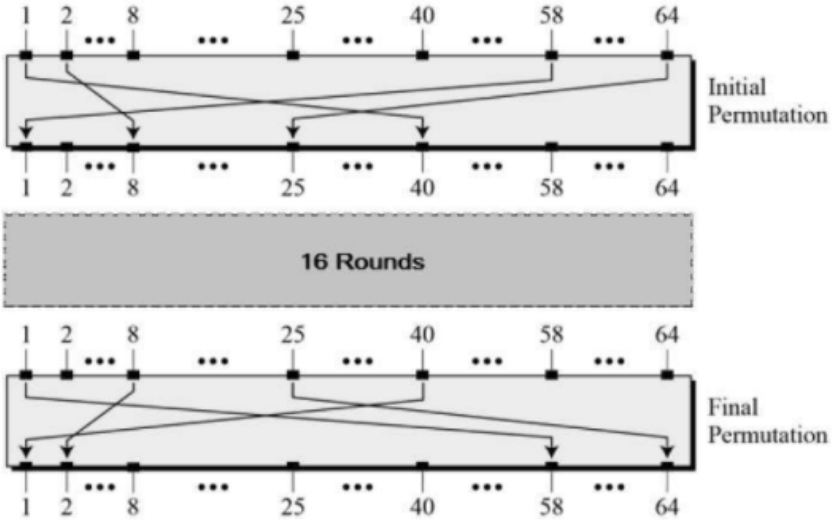
The avalanche effect refers to the property of cryptographic systems that states that small changes in the plaintext or key should result in large, unpredictable changes in the ciphertext. It is considered a desirable property in cryptographic systems, as it means that even small changes to the plaintext or key will result in completely different ciphertext, making it difficult for an attacker to determine the plaintext or key by analyzing the ciphertext. In DES, the round function is designed to achieve the avalanche effect through a combination of operations, including substitution boxes and permutation boxes.

Exercise 1.

a) What is the main weakness of DES and why it is no longer considered a secure encryption algorithm?
b) What is the goal of confusion in symmetric-key cryptography and how does the round function of DES achieve it?
c) How do the initial permutation and final permutation of DES help to achieve diffusion?
d) How does diffusion differ from the avalanche effect?

Internal structure of DES: permutations

The initial permutation operation is used to shuffle the bits of the plaintext before encryption and the final permutation operation is used to shuffle the bits of the ciphertext after encryption.



The final permutation of DES is the inverse of the initial permutation.

- Initial permutation IP

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

- Final permutation IP^{-1}

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Additional two permutations occur within each round of the round function of DES, namely the expansion permutation (E/P) which expands 32 bits (half a block) to 48 blocks, and the permutation P to finish of each round.

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

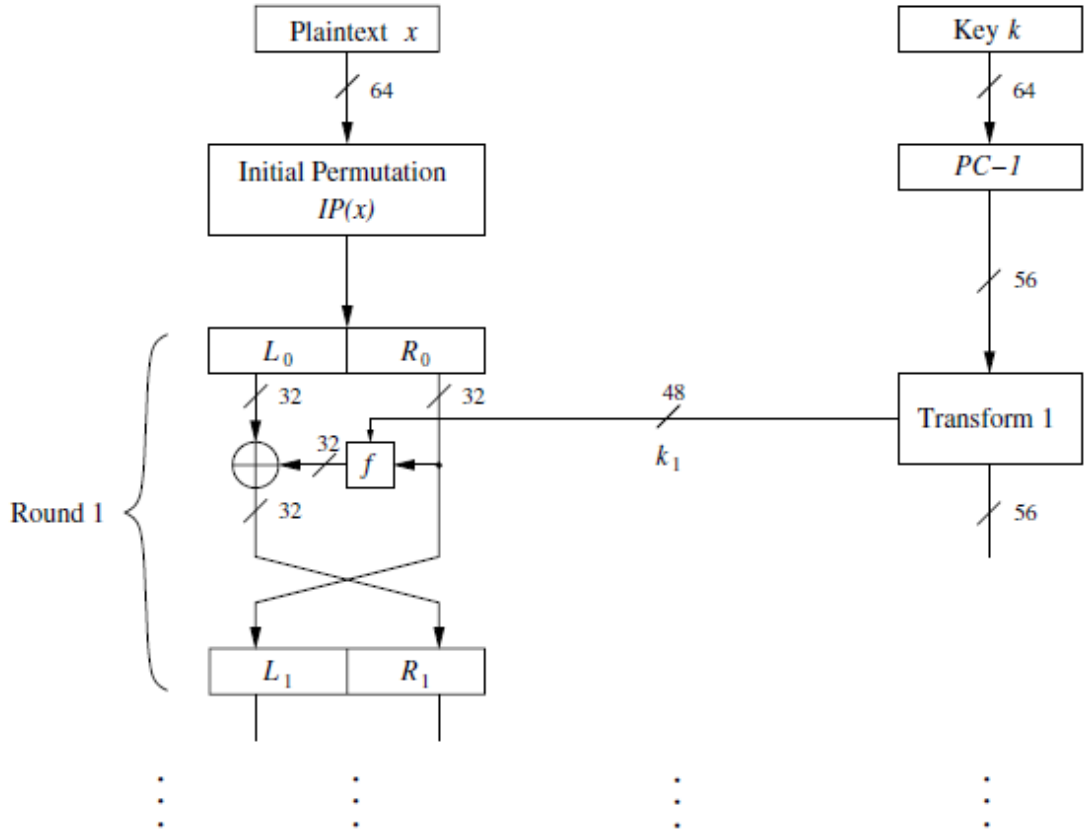
P							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

The round structure of DES: Feistel Network

A Feistel network is a type of symmetric-key cryptographic structure that was first used in DES. It has since been used in several other symmetric block ciphers such as Blowfish.

The Feistel network is designed to achieve diffusion. The basic structure consists of a series of rounds. On each round, the plaintext is divided into two halves, and one half is used as input to the substitution operation, while the other half is used as input to the permutation operation. The output of the substitution operation is then combined with the output of the permutation operation, typically by performing a bitwise XOR operation, to produce the output.

In DES, the Feistel Network is implemented as shown below.



For each round, the left and right sides of the block are given by:

$$L_i = R_{i-1}$$

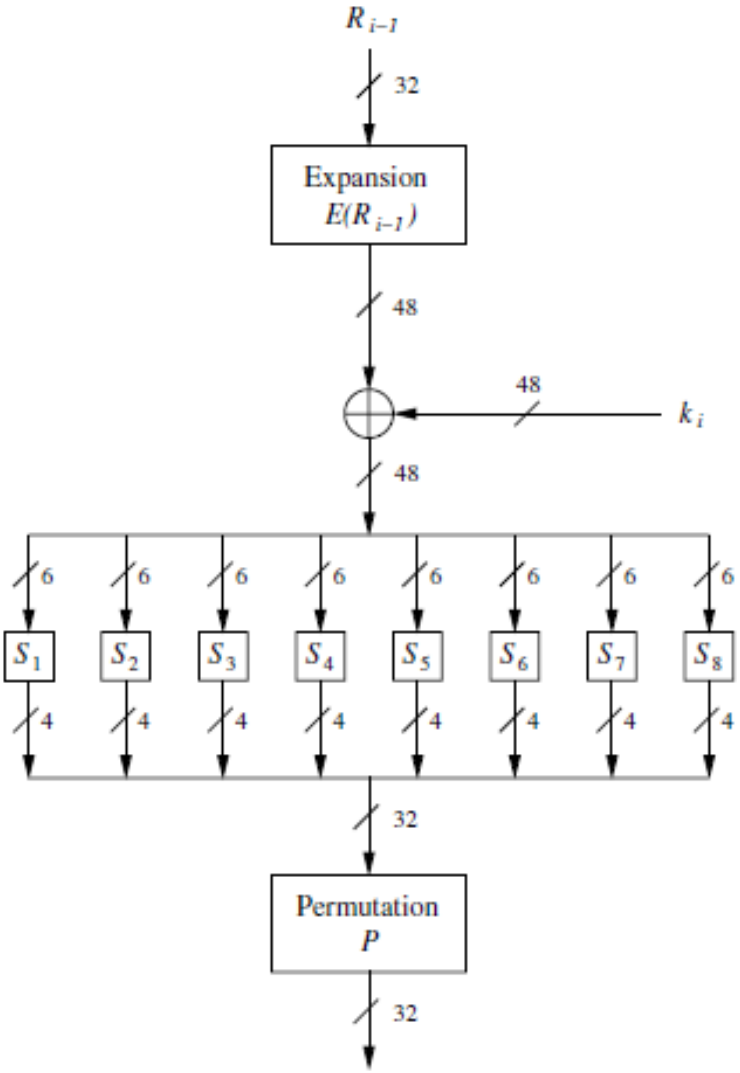
$$R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$$

where $i = 1, \ldots 15$ and f is the DES function.

The DES function (f)

The DES function f contains the following phases:

- Expansion permutation E expands the right side of the block from 32 to 48 bits.
- Key mixing: $E(R_{i-1}) \oplus k_i$, where k_i is the unique subkey for round i.
- 48 bits are divided into eight 6-bit blocks, which become inputs for eight substitution boxes $S_1, S_2, \ldots, S_8$ called S-boxes.
- Each S-box gives a 4-bit output, which in turn are concatenated back into a 32-bit block.
- Finally, the 32-bit block is permuted using permutation P.



Internal structure of DES: S-boxes

Substitution boxes form the core of the cryptographic strength of DES. Each S-box is a fixed, non-linear function that maps an input of 6 bits to an output of 4 bits. The S-boxes have a strong avalanche effect: small changes in the input to the S-box are likely to result in large, unpredictable changes in the output of the S-box. This makes it difficult for an attacker to predict the output of the S-box based on the input, which contributes to confusion.

S-box S_1

S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Example. Find the output of S-box 1 given input sequence a) 100001 b) 110110

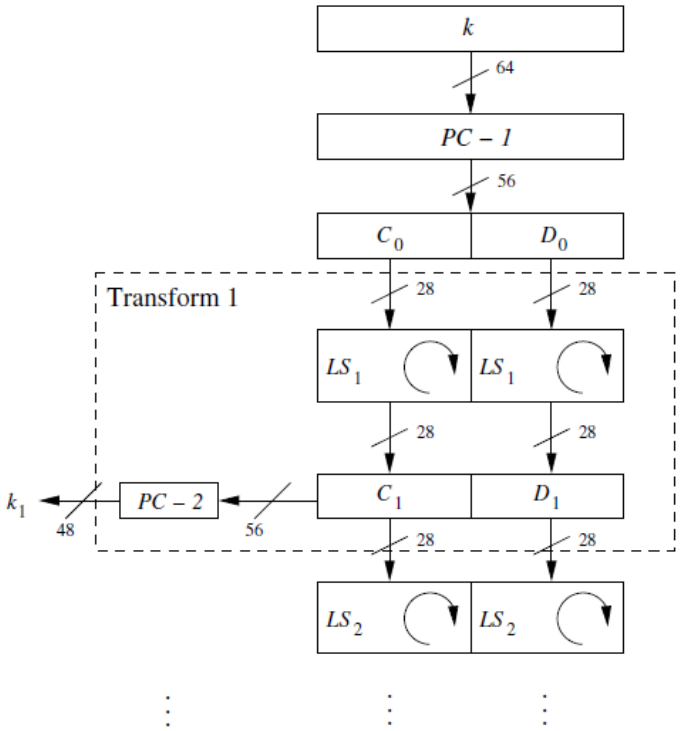
DES Key schedule

The original key k has 64 bits, of which 8 are parity bits. As parity bits don't contribute to key space, DES is effectively a 56-bit cipher. The initial key permutation PC-1 strips off the parity bits, after which the key is split into two 28-bit halves. To derive the subkeys for each round, each half is first left-shifted by 1 or 2 bits and then permuted again using permutation PC-2.

The key schedule is very easy to implement and to reverse for decryption.

$PC-1$							
57	49	41	33	25	17	9	1
58	50	42	34	26	18	10	2
59	51	43	35	27	19	11	3
60	52	44	36	63	55	47	39
31	23	15	7	62	54	46	38
30	22	14	6	61	53	45	37
29	21	13	5	28	20	12	4

$PC-2$							
14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32



Exercise 2.

Find the values $S_1(x_1)$ and $S_1(x_2)$ where S_1 is the first S-box of DES and inputs are as follows:

a) $x_1 = 000000, x_2 = 000001$
b) $x_1 = 111111, x_2 = 100000$
c) $x_1 = 101010, x_2 = 010101$

Present the values $S_1(x_1)$ and $S_1(x_2)$ in binary.

Exercise 3.

DES has the following *weak keys* and *semi-weak key pairs*:

weak key (hexadecimal)	C_0	D_0
0101 0101 0101 0101	$\{0\}^{28}$	$\{0\}^{28}$
FEFE FEFE FEFE FEFE	$\{1\}^{28}$	$\{1\}^{28}$
1F1F 1F1F 0E0E 0E0E	$\{0\}^{28}$	$\{1\}^{28}$
E0E0 E0E0 F1F1 F1F1	$\{1\}^{28}$	$\{0\}^{28}$

Table 7.5: *Four DES weak keys.*

C_0	D_0	semi-weak key pair (hexadecimal)	C_0	D_0
$\{01\}^{14}$	$\{01\}^{14}$	01FE 01FE 01FE 01FE, FE01 FE01 FE01 FE01	$\{10\}^{14}$	$\{10\}^{14}$
$\{01\}^{14}$	$\{10\}^{14}$	1FE0 1FE0 0EF1 0EF1, E01F E01F F10E F10E	$\{10\}^{14}$	$\{01\}^{14}$
$\{01\}^{14}$	$\{0\}^{28}$	01E0 01E0 01F1 01F1, E001 E001 F101 F101	$\{10\}^{14}$	$\{0\}^{28}$
$\{01\}^{14}$	$\{1\}^{28}$	1FFE 1FFE 0EFE 0EFE, FE1F FE1F FE0E FE0E	$\{10\}^{14}$	$\{1\}^{28}$
$\{0\}^{28}$	$\{01\}^{14}$	011F 011F 010E 010E, 1F01 1F01 0E01 0E01	$\{0\}^{28}$	$\{10\}^{14}$
$\{1\}^{28}$	$\{01\}^{14}$	E0FE E0FE F1FE F1FE, FEE0 FEE0 FEF1 FEF1	$\{1\}^{28}$	$\{10\}^{14}$

Table 7.6: *Six pairs of DES semi-weak keys (one pair per line).*

What is the meaning of these terms, and why should these keys not be used?

Exercise 4. There are two variants of DES that were designed to enhance the security: DES-X and Triple DES. What are the basic mechanisms/ideas for each of them to achieve the goal of elevated security compared to original DES?

Exercise 5. Find all equivalence classes for modulus $m = 5$, show at least six elements for each of them. Are the following congruences correct? Why, or why not?

a) $4 \equiv -24\ (mod\ 5)$ b) $-2 \equiv 38\ (mod\ 5)$

Exercise 6. Consider integer ring Z_{29}. Name the following:

Neutral element with respect to addition
Neutral element with respect to multiplication
Additive inverse of element $\overline{5}$
Additive inverse of 235

Exercise 7. Is element 5 invertible in Z_{11}, Z_{12} or Z_{13}? Why? If yes, then find the multiplicative inverses of 5 in each integer ring. You can do a trial-and-error search using a calculator or computer.

Block Ciphers: AES

Advanced Encryption Standard (AES)

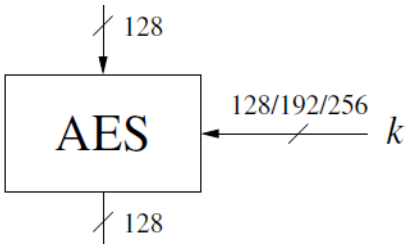
As the limitations of DES and triple-DES with respect to security and efficiency had become obvious, NIST (the US National Institute of Standards and Technology) called for proposals for a new Advanced Encryption Standard in 1997. The requirements for the candidates included:

- block cipher with 128 bit block size
- supports three key lengths: 128, 192 and 256 bits
- efficient implementation for both software and hardware

After multiple evaluation rounds by NIST and the international scientific community, a block cipher called *Rijndael* was chosen amongst the candidates as the new AES in 2000, and formally approved as US federal standard in 2001. AES soon became and continues to be the dominant symmetric-key algorithm for many industry standards and commercial systems.

Overview of AES

AES has block size 128 and options for key sizes 128, 192 and 256. It has an iterative structure with 10, 12 or 14 encryption rounds. The number of internal rounds depends on chosen key length.



key lengths	# rounds = n_r
128 bit	10
192 bit	12
256 bit	14

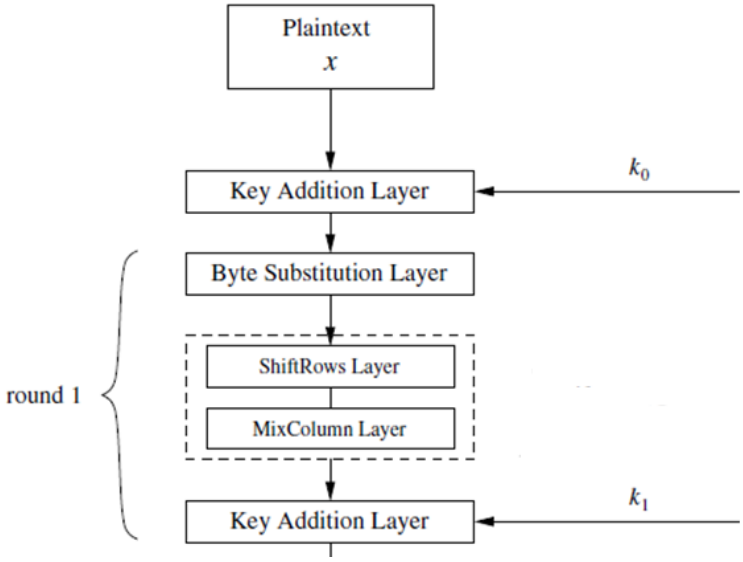
AES internal structure: bytes

AES is byte-oriented, meaning that operations are performed on byte-level. Each stage of the 128-bit data sequence is called state of the algorithm. State A is arranged in a 4x4 byte matrix of 16 bytes $A_0, A_1, \ldots, A_{15}$. AES then operates on elements, columns, or rows of the current state matrix.

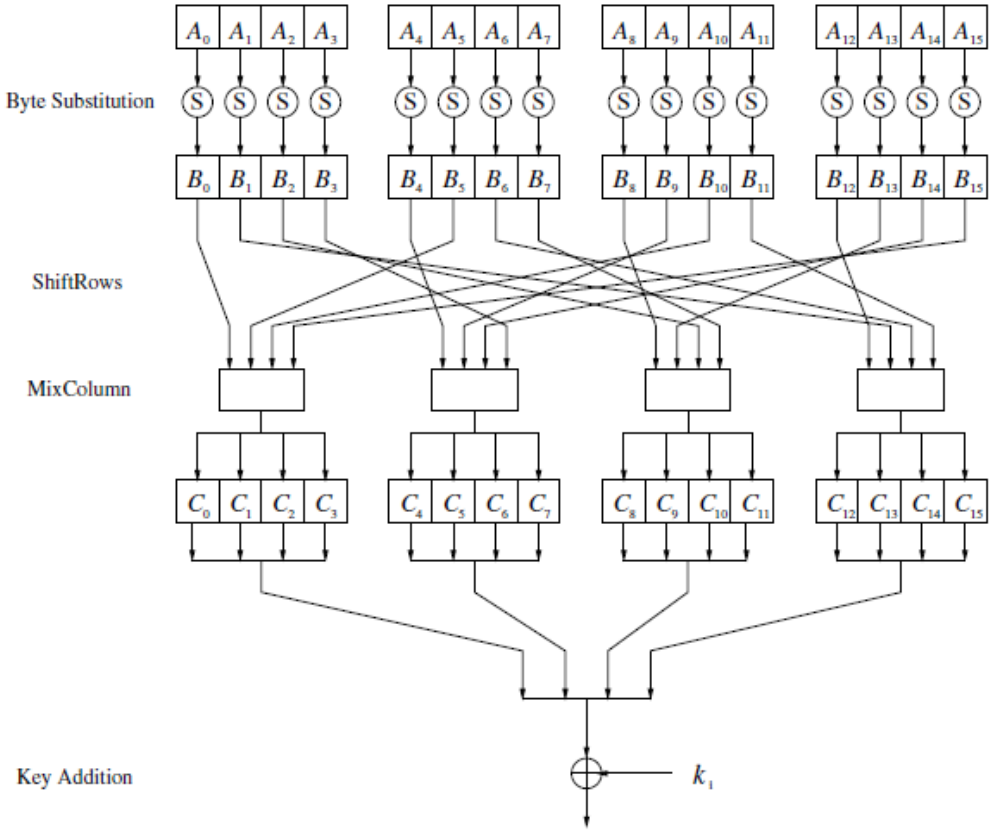
A_0	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

AES internal structure: layers

AES has no Feistel structure. Each round consists of multiple **layers**. All rounds are identical except for the last one. For each round, the key schedule computes subkey k_i from the key k.



Each layer has a unique function. **Byte Substitution Layer** introduces confusion by substituting each byte by another one using an S-box**. ShiftRows Layer** is a byte permutation, and **MixColumn Layer** is a matrix operation performing an affine mapping. Together these two form the *Diffusion Layer* to provide diffusion. Within **Key Addition Layer** the round key k_i is XORed to current state.

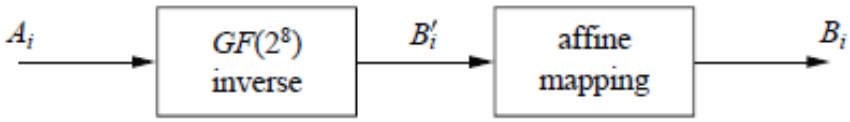


Byte Substitution layer

In Byte Substitution layer, each element is substituted by another one using a S-box. The S-box is a bijective function, meaning that there is a one-to-one mapping between input and output bytes. It follows that the S-box is invertible. It is also the only nonlinear element of AES, i.e.

$$ByteSub(A_i) + ByteSub(A_j) \neq ByteSub(A_i + A_j)$$

The S-box takes state A_i as input, interprets it as an element of the Galois extension field $GF(2^8)$, and finds its multiplicative inverse (B_i'). It then performs an affine mapping modulo 2 on B_i', and the result is the output B_i.



$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} \equiv \begin{pmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{pmatrix} \begin{pmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \text{ mod } 2.$$

In software implementations, the AES S-box is usually implemented as a lookup table. Below is a lookup table with substitution values in hexadecimal notation for input byte (xy).

x \ y	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

Example. Find the S-box value $S(A)$ for byte $A = 10100111$.

ShiftRows Layer and MixColumn layer

On ShiftRows Layer, the input data is permuted cyclically on a byte level:

Input matrix

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

Output matrix

B_0	B_4	B_8	B_{12}	no shift
B_5	B_9	B_{13}	B_1	← one position left shift
B_{10}	B_{14}	B_2	B_6	← two positions left shift
B_{15}	B_3	B_7	B_{11}	← three positions left shift

MixColumn step is a linear transformation which mixes each column of the state matrix. Each 4-byte column is a vector and multiplied by a fixed 4x4 matrix, e.g. the first output column is given by

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

where 01, 02 and 03 are given in hexadecimal notation. All arithmetic is done in the Galois extension field $GF(2^8)$.

KeyAddition layer

KeyAddition step performs a bitwise XOR operation between its two inputs, namely the 16-byte state matrix *C* and a 16-byte subkey k_i*:*

$$C \oplus k_i$$

We note that XOR operation is equivalent to addition in the Galois Field $GF(2)$. The subkeys are generated in the key schedule.

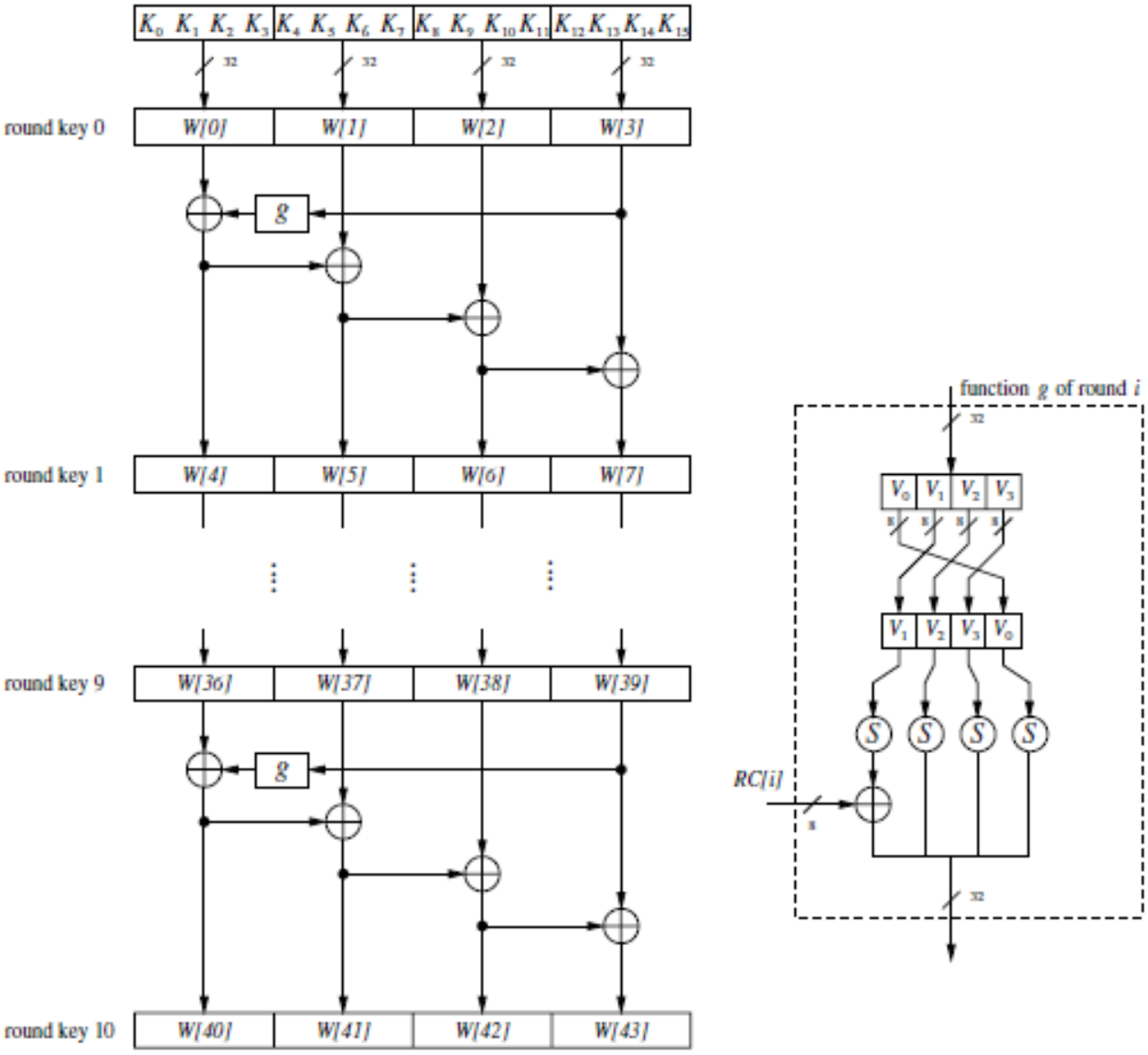
KeyAddition is the final step of each encryption round. It is also performed as the very first step of AES encryption, thereby creating an inherent *key whitening* step to AES.

Key Schedule

The key schedule derives a number of subkeys k_i from the original key k. The number of subkeys depends on the key size as follows:

Key length (bits)	Number of subkeys
128	11
192	13
256	15

The AES key schedule is different but similar for the three different key lengths. In each case it is *word-oriented*, meaning that the data is handled as words of 32 bits each. Below is the chart of key schedule for 128-bit AES.



Decryption

To decrypt AES, all layers must be inverted. We have the inverses:

- MixColumn layer → Inv MixColumn layer
- ShiftRows layer→ Inv ShiftRows layer
- Byte Substitution layer → Inv Byte Substitution layer
- Key Addition layer is its own inverse

The inverse operations are fairly similar to the layer operations of the encryption. The subkeys are needed in reversed order, so there is need for a reversed key schedule: This is mainly achieved simply by precomputing and storing all subkeys first.

Exercise 1. Find some industrial standards or commercial applications in which AES is used.

Exercise 2. True or false?

a) Byte Substitution Layer provides confusion to AES.
b) Key length affects the block size of AES.
c) AES has an inherent key whitening step.
d) The irreducible polynomial for multiplications in $GF(2^8)$ is determined in the AES standard.

Exercise 3.

a) What is AES T-box (or T-table)?
b) What can be said about AES T-box / S-box lookup table implementations and side-channel attacks? [Well, a lot could be said. I'm expecting a short(ish) answer here.]

Exercise 4.

Find A^{-1} for AES byte $A = 11001001$ from the AES multiplicative inverse lookup table. Verify by performing the multiplication $A \cdot A^{-1}$ $(mod\ x^8 + x^4 + x^3 + x + 1)$.

Exercise 5.

Using the AES multiplicative inverse table, find the inverses for bytes 29, F3 and 01, where each byte is given in hexadecimal notation.

Exercise 6.

Using the AES S-box lookup table, perform the Byte substitution operation for bytes 29, F3 and 01.

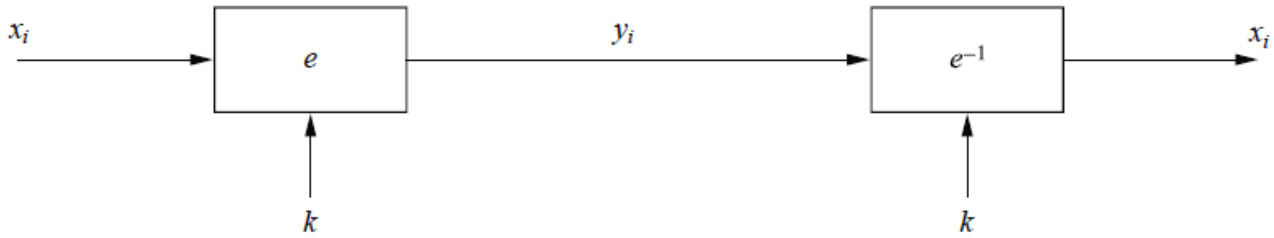
Block Ciphers: modes of operation

There are several ways of encrypting plaintexts with a block cipher. The following are the most used modes of operation for block ciphers:

1. Electronic Codebook (ECB) mode
2. Cipher Block Chaining (CBC) mode
3. Counter (CTR) mode
4. Output Feedback (OFB) mode
5. Cipher Feedback (CFB) mode
6. Galois Counter (GCM) mode

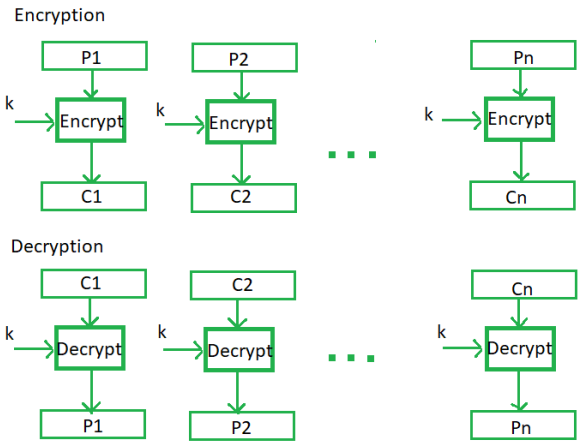
Each mode has its own advantages and disadvantages in terms of security, implementation complexity and efficiency. It is important to choose the right mode of operation depending on the specific requirements and the security constraints of a system.

Electronic Codebook (ECB) mode



ECB mode is the simplest mode of operation, where each block of plaintext is encrypted independently using the same key. The advances of ECB include:

- there is no need for block synchronization between sender and receiver
- possible bit errors in transmission only affect a single block
- both encryption and decryption can be parallelized



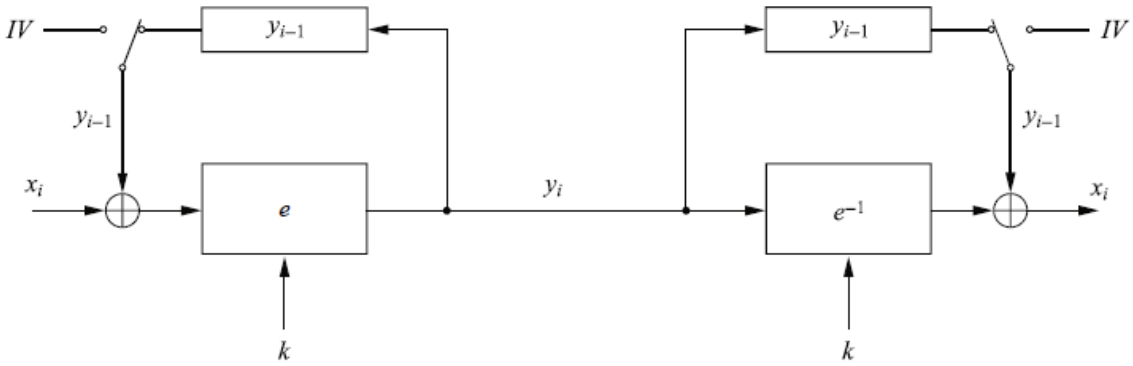
The properties of ECB allow for high-speed implementations and working in noisy channels. However, ECB is highly deterministic, and identical plaintext blocks encrypt into identical ciphertext blocks (aka the codebook effect).

Therefore, ECB is not recommended for most applications due to its lack of diffusion. ECB may still be appropriate for small-scale systems or applications with low security requirements, where the amount of data being encrypted is not very large.

Original image, RGB values split into many **b**-bit blocks

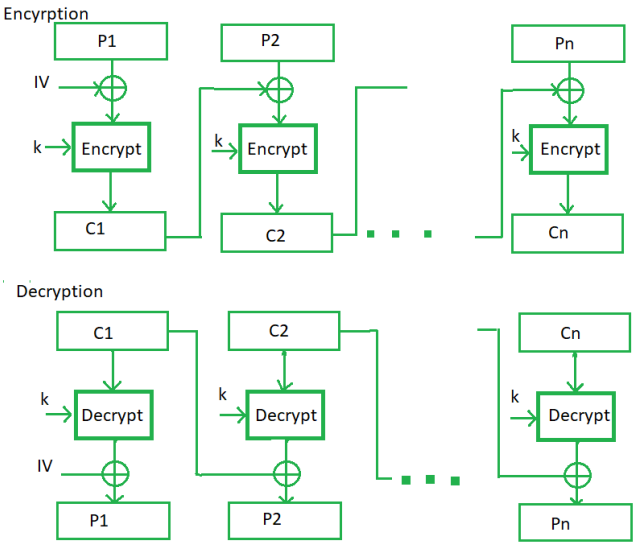
Encrypted with ECB and interpreting ciphertext directly as RGB

- **Cipher Block Chaining (CBC) mode**



In CBC mode, encryption of all blocks is *chained*: each block of plaintext is XORed with the previous ciphertext block before being encrypted with the key. This provides diffusion and helps to prevent patterns in the plaintext from appearing in the ciphertext. Encryption is randomized by using an *initial value* or *initialization vector* (IV). IV does not have to be a secret, but usually it is a *nonce*. This helps to obtain nondeterminism, thus avoiding the codebook effect of ECB mode.

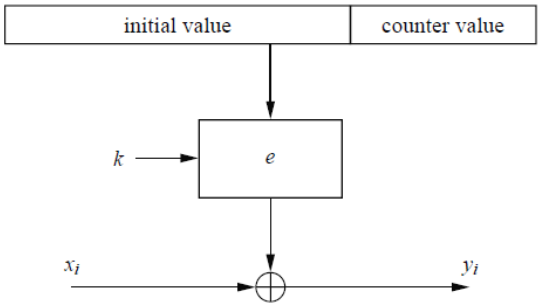
CBC is commonly used in secure socket layer (SSL) and transport layer security (TLS) protocols for Internet security.



Original image, RGB values split into many **b**-bit blocks

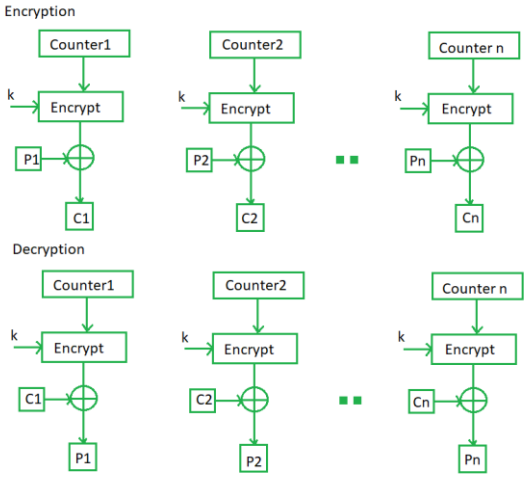
Encrypted with CBC and interpreting ciphertext directly as RGB

Counter (CTR) mode

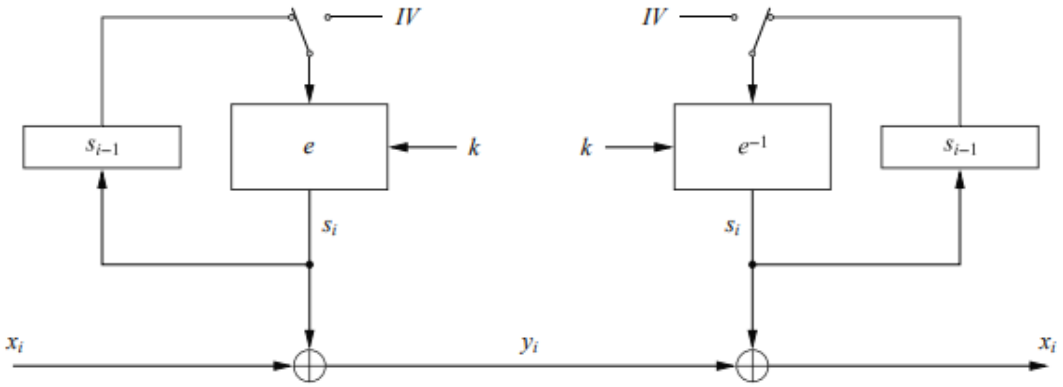


CTR mode is essentially used to build a stream cipher from a block cipher. It operates by encrypting a sequence of counter values using the key, and XORing the resulting ciphertext with the plaintext to generate the final ciphertext. A nonce is used as an initial value and the counter value is incremented for each block, which helps to avoid the codebook effect.

CTR mode provides *parallelism* and possible bit errors only affect a single block. This makes CTR mode suitable for applications with high throughput demands, such as in disk encryption and network communication.

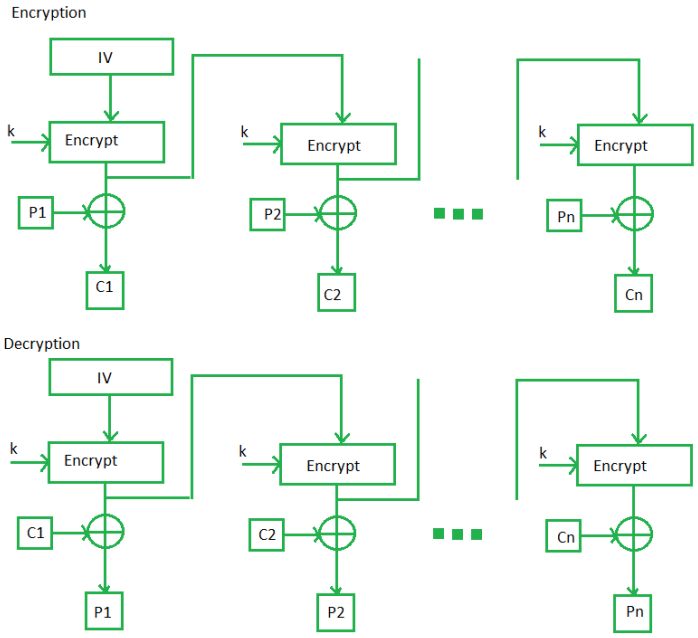


Output Feedback (OFB) mode

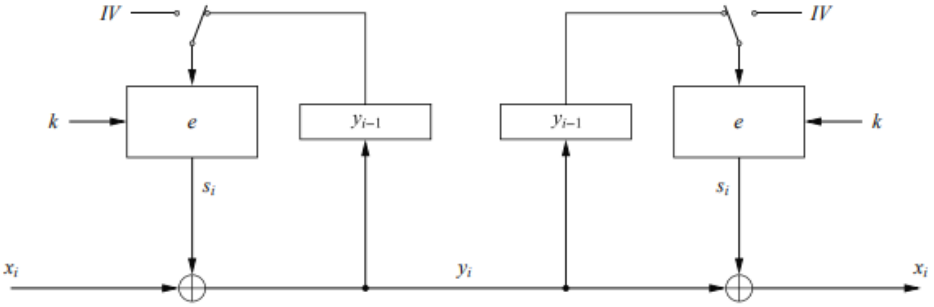


OFB mode is used to build a synchronous stream cipher, where the key stream is generated blockwise. An IV is used to initialize generating the first key stream bits, which are then both fed back to the encryption process to generate more key stream, and XORed with the plaintext to generate the ciphertext. Encryption and decryption are the same operation in OFB mode.

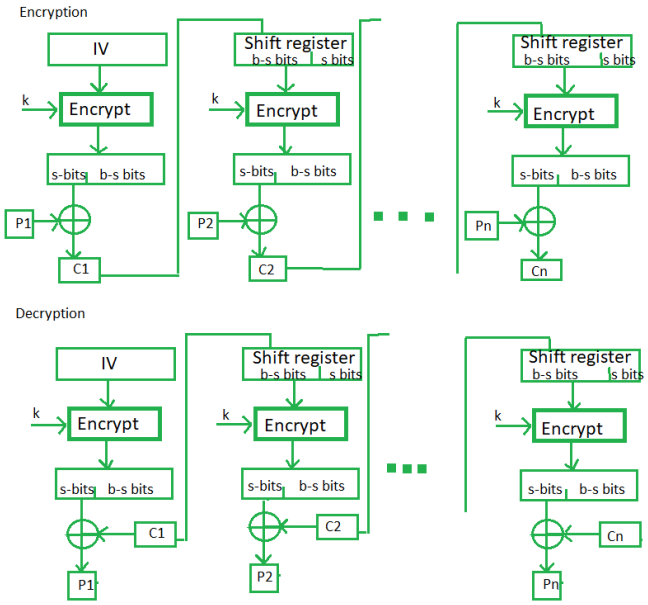
As here the key stream computations are independent of the plaintext, the key stream material can be precomputed. A nonce should be used as an initial value to provide nondeterminism.


Encryption
IV
k
Encrypt
P1
C1
k
Encrypt
P2
C2
k
Encrypt
Pn
Cn
Decryption
IV
k
Encrypt
C1
P1
k
Encrypt
C2
P2
k
Encrypt
Cn
Pn

Cipher Feedback (CFB) mode



CFB mode is similar to OFB mode, but instead of previous key stream we feed back the previous ciphertext block. This is again encrypted using the key k, and the resulting key stream is XORed with the plaintext to generate next ciphertext block. Again, an IV is used to initialize encryption and to provide nondeterminism. In CBC, encryption cannot be parallelized, but decryption can.

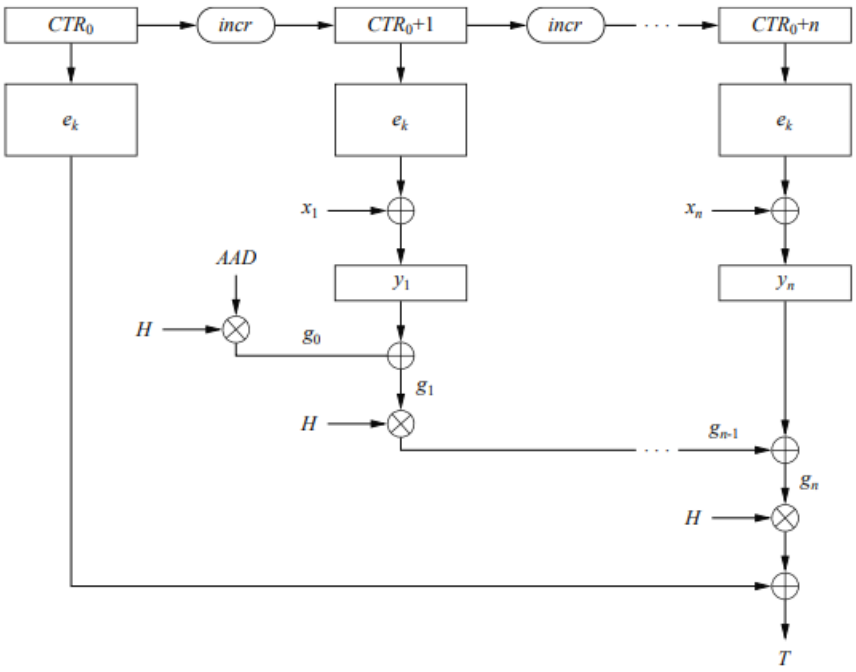


Galois Counter Mode (GCM)

GCM mode is a mode of operation that provides confidentiality, authenticity and integrity of the data being encrypted. In GCM mode, a block cipher (typically AES) is used to encrypt a counter value, which is then XORed with the plaintext to generate the ciphertext. The counter value is encrypted using an IV. The GCM mode also calculates a message authentication code (MAC), which is included with the ciphertext to provide data authenticity and protect against tampering.

A key component that is used to calculate MAC is the Galois field multiplier. It is used to perform the multiplication operation in the calculation of the MAC. The multiplications are performed in $GF(2^{128})$ with the irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$.

The Galois field multiplier is an important part of GCM because it provides the mathematical foundation for the calculation of the MAC, ensuring the authenticity of the encrypted data. The use of the Galois field in GCM helps to provide a high level of security, as it is difficult to tamper with the data without being detected.



CRT_i counter value i

AAD additional authenticated data

H authentication subkey

$\otimes$ Galois Field multiplication

T final authentication tag

- Receiver gets the packet $[(y_1, \ldots, y_n), T, AAD]$
- To check authenticity, they compute tag T' and see if $T' = T$.

One of the advantages of GCM over other modes of operation is its high speed. Encryption and decryption can be performed in parallel, and authentication process is separate from them. Additionally, GCM provides protection against replay attacks, since the IV must be unique for each encryption operation. GCM is widely used in secure communication protocols, such as SSL/TLS and IPsec. It is also used in disk encryption and wireless communication, among other applications.

Example: AES in PyCryptodome

PyCryptodome is a Python package with some cryptographic primitives:

https://pycryptodome.readthedocs.io/en/latest/src/introduction.html

AES in PyCryptodome:

https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html

Procedure images for ECB, CBC, CFB, OFB, CRT

https://www.geeksforgeeks.org/block-cipher-modes-of-operation/

Exercise 1. True or false?

a) ECB mode provides confidentiality and authenticity for the data being encrypted.

b) In Cipher Block Chaining mode, each block of plaintext is encrypted using a different key.

c) Counter mode is not suitable for parallel encryption and decryption.

d) CFB mode is not recommended for most applications due to its lack of diffusion.

e) The choice of mode of operation in symmetric ciphers depends solely on the amount of data being encrypted.

Exercise 2. True or false?

a) Galois Counter Mode (GCM) is a mode of operation for asymmetric-key cryptographic block ciphers.

b) In GCM mode, the message authentication code (MAC) is calculated using a symmetric key.

c) The Galois field multiplier is used to calculate the encryption key in GCM mode.

d) GCM is commonly used in disk encryption and wireless communication.

Exercise 3. What is the role of an IV and why should we use a nonce and not a fixed sequence?

Exercise 4. For which modes of operation are the following statements true?

a) It is used to build a stream cipher

b) It is deterministic (i.e. encrypts the same PT into the same CT every time)

c) In the encryption process, some data is fed back to the system.

d) A change in one plaintext block affects the encryption of every following block.

e) Encryption process can be parallelized.

f) Decryption process can be parallelized.

	ECB	CBC	CTR	OFB	CFB
a					
b					
c					
d					
e					
f					

Exercise 5. Suppose there is an error in the plaintext block x_i on Alice's side. Which plaintext blocks are affected on Bob's side

a) when using the ECB mode?

b) when using the CBC mode?

Exercise 6. Suppose a bit error occurs in one ciphertext block y_i during the transmission. Which plaintext blocks are affected in decryption

a) when using the ECB mode?

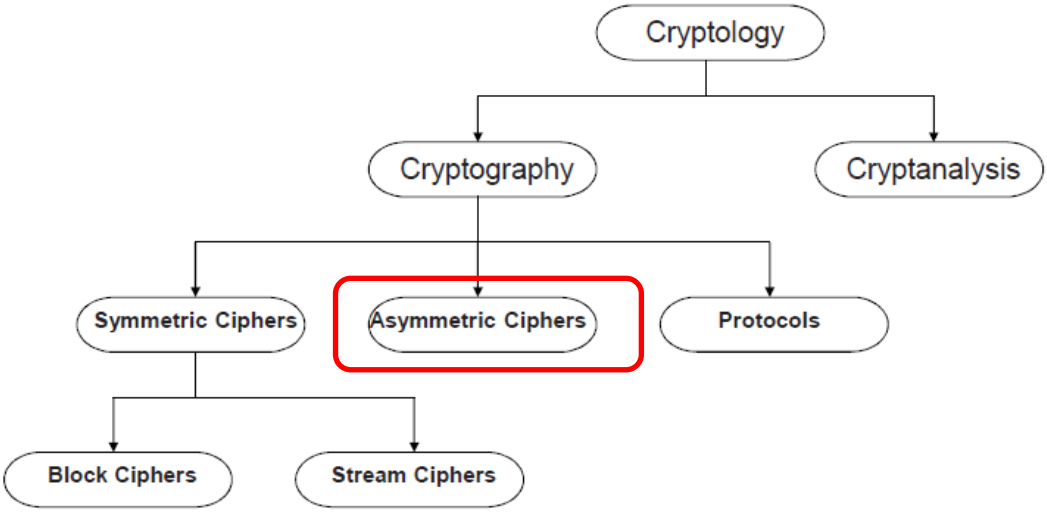
b) when using the CBC mode?

Exercise 7.

a) What are the benefits of using GCM mode over other modes of operation, such as CBC or ECB?

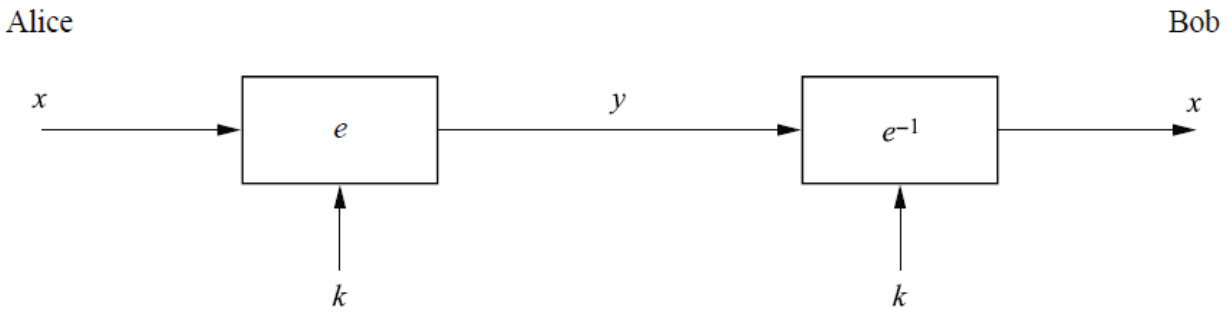
b) What are some potential security concerns when using GCM mode, and how can these be addressed?

Introduction to Public-Key Cryptography



Recap: Principle of Symmetric Cryptography

In symmetric cryptography, we use the same secret key k for encryption and decryption. Depending on the cipher, encryption and decryption functions are the same or similar. A good analogy for a symmetric cipher is a safe box with one lock and two identical keys.

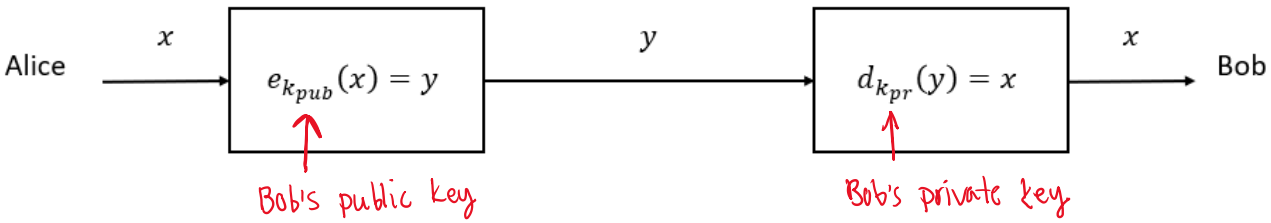


Symmetric ciphers are fast and generally light to implement, but they have some shortcomings. First, distributing and managing the keys can be tedious. Alice and Bob must find a way to establish or share the key in a secure manner. If the system has multiple users, then the number of keys grows fast. Each pair of users need a separate key pair, thus for n users there is a need for $\frac{n(n-1)}{2}$ key pairs. Each user will have to store and manage $n-1$ secret keys. Secondly, as both parties of any communication have the same information, there is no mechanism for *non-repudiation* to protect them from each other.

Principle of Asymmetric Cryptography

An asymmetric or public-key cipher has separate functions and keys for encryption and decryption. Each user has their own public and private key pair and there is no need for pre-shared secret. We usually denote:

- encryption function e, decryption function d
- k_{pub} is Bob's public key; used by Alice to encrypt message x
- $k_{pr}$is Bob's private key; used by Bob to decrypt message y



Asymmetric ciphers play an important role in securing digital communications over public networks. They are used for a variety of cryptographic tasks, including:

- **Key exchange**. Asymmetric ciphers can be used to securely exchange a shared secret key between two parties, enabling them to communicate using a symmetric cipher.

- **Authentication**: Asymmetric ciphers can be used for authentication purposes, such as allowing a user to prove their identity to a remote server.

- **Digital signatures**. A digital signature is created by taking a message (or a hash) and encrypting it with the private key of the sender. The recipient can then use the sender's public key to verify the signature. Digital signature allow for verifiable authentication of the origin of a message, thereby providing integrity, authentication and non-repudiation.

- **Encryption.** Asymmetric ciphers can also be used to encrypt messages so that only the recipient with the matching private key can decrypt and read the message.

Some practical aspects of public-key cryptography

In terms of key managements, public-key ciphers are more convenient than symmetric ciphers for distributing keys. The public key can be made publicly available without compromising security, whereas symmetric keys must always be securely exchanged between parties before communication can occur. As every user has their own private key, public-key systems enable building protocols that provide non-repudiation.

Symmetric ciphers are generally faster than public-key ciphers and can operate on larger blocks of data at once. Public-key ciphers are slower due to the mathematical complexity of the algorithms and longer keys.

Public-key ciphers are typically used for key exchange, digital signatures, and secure communication, whereas symmetric ciphers are used for bulk encryption of data. The two types of ciphers are often used together in a complementary manner to provide both security and efficiency. Often a public-key cipher is used to securely exchange a symmetric key, which is then used for the bulk encryption of data. Some examples of such hybrid protocols include:

- SSL/TLS: a protocol for secure communication on the Internet.
- IPsec: a protocol for secure communication at the Internet Protocol level.
- PGP: an email encryption software ciphers to provide end-to-end encryption for email messages.
- SSH: a protocol for secure communication over an insecure network, such as the Internet.

Authentication of Public Keys

Public-key schemes do not require a secure channel for key exchange, but they do require authenticated channel for the distribution of public keys. We need to be able to trust that a given public key belongs to a certain user.

Public keys are authenticated with digital certificates, which are issued by trusted third-party organizations known as certificate authorities (CA). The CA acts as a trusted intermediary that issues a certificate binding the public key to an entity's identity. Public-key authentication is only secure if the client can trust the CA that issued the certificate. If the CA is compromised, an attacker could issue a fraudulent certificate binding a different public key to the entity's identity. To mitigate this risk, many organizations use multiple CAs and require clients to have a list of trusted CAs built into their systems.

Rank	Issuer	Usage	Market Share
1	IdenTrust	43.4%	48.9%
2	DigiCert	16.6%	18.7%
3	Sectigo (Comodo Cybersecurity)	13.8%	15.5%
4	Let's Encrypt	7.2%	8.2%
5	GoDaddy	5.4%	6.1%
6	GlobalSign	2.4%	2.7%

The six largest CA's (Wikipedia, July 2022)

Public-Key Algorithms

Definition 6.1.1 One-way function
A function $f()$ is a one-way function if:

1. $y = f(x)$ *is computationally easy, and*
2. $x = f^{-1}(y)$ *is computationally infeasible.*

All public-key schemes are based on the idea of one-way functions. A one-way function is a mathematical function that is easy to compute in one direction, but extremely difficult to invert.

One-way functions are based on mathematically hard problems. The problems for which public-key cryptosystems are based on are:

- **Integer factorization**: given a composite integer n, find its prime factors. Given a prime factorization, it is easy to determine the original number by multiplying the factors, but given a large number, it is computationally infeasible to determine its prime factorization.

- **The Discrete logarithm problem**: given integers a, y and m, find x such that
$$a^x = y \ (mod\ m)$$
Here, finding x is very hard. On the other hand, finding y given a, y and m is very easy.

Additionally, elliptic curve schemes are based on so called generalized discrete logarithm problem.

Note that there is no proof of existence for one-way functions, or that these problems are hard.

Algorithm Family	Cryptosystems	Security Level (bit)			
		80	128	192	256
Integer factorization	RSA	1024 bit	3072 bit	7680 bit	15360 bit
Discrete logarithm	DH, DSA, Elgamal	1024 bit	3072 bit	7680 bit	15360 bit
Elliptic curves	ECDH, ECDSA	160 bit	256 bit	384 bit	512 bit
Symmetric-key	AES, 3DES	80 bit	128 bit	192 bit	256 bit

Comparison of key lengths for different families of encryption schemes

Exercise 1. Discuss the following:

a) What are the advantages of using symmetric ciphers over asymmetric ciphers for encryption and decryption?

b) What are the advantages of using asymmetric ciphers over symmetric ciphers for encryption and decryption?

Exercise 2. Considering one-way functions, answer the following questions:

a) What is a trapdoor?

b) Give some examples of the uses of one-way functions in cryptography.

Exercise 3. Assume a company has 150 employees. The new company policy requires encrypted message exchange with a symmetric cipher. How many keys are required if encryption is needed for communication between any two employees?

Exercise 4. Find $\gcd(7035,864)$

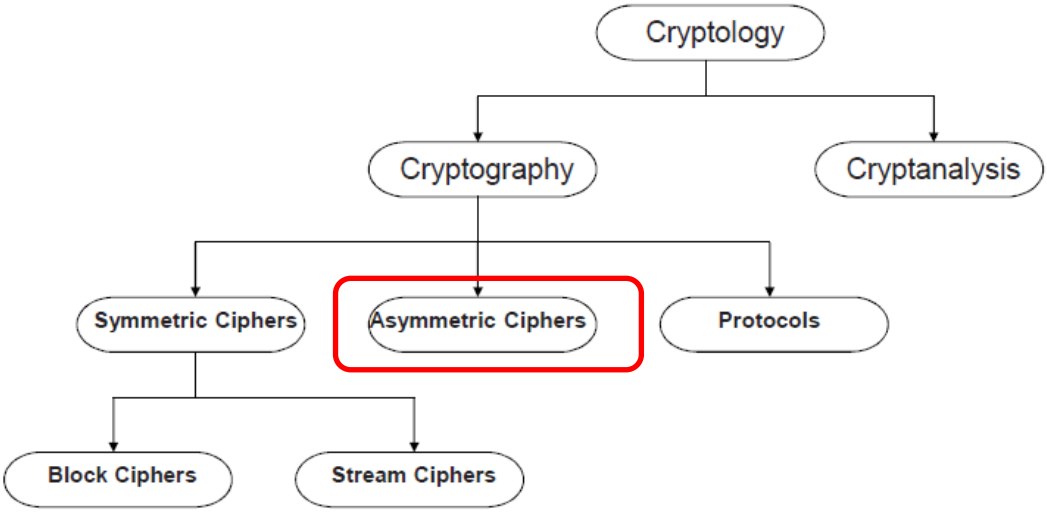
a) based on their respective prime power factorizations

b) using the Euclidian algorithm.

Exercise 5. Using Extended Euclidean Algorithm, compute the greatest common divisor of 198 and 243 and solve the Diophantine equation $198s + 243t = \gcd(198,243)$.

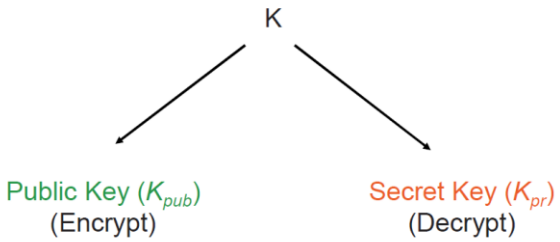
Exercise 6. Use Extended Euclidean Algorithm to find the inverse of 321 in Z_{2707}.
Note. If you get a negative answer from this one, you have to add the modulus to it once, to bring your answer to range $\{0,1,\dots,2706\}$.

The RSA Cryptosystem



Principle of Asymmetric Cryptography

The main principle of asymmetric cryptography was introduced by Diffie, Hellman & Merkle in 1976 in their paper "New Directions in Cryptography". They showed that it is possible to build a system with different keys for encryption and decryption, and that the encryption key can be public if there is no way to derive the decryption key from it.



In any asymmetric system, each user has a key pair (k_{pub}, k_{pr}) where encryption key k_{pub} is public and decryption key k_{pr} is private. There is no need for a pre-shared secret, and public keys can be publicly distributed to other users without compromising the system security. Therefore, there is no need for a secure communication channel.

Consider a simple encryption scenario. If (k_{pub}, k_{pr}) is Bob's key pair, then Alice will use k_{pub} to encrypt message x, and Bob will use k_{pr} to decrypt message y.

Alice		**Bob**
	$\xleftarrow{k_{pub}}$	$(k_{pub}, k_{pr}) = k$
$y = e_{k_{pub}}(x)$		
	$\xrightarrow{y}$	
		$x = d_{k_{pr}}(y)$

The RSA Cryptosystem

The RSA cryptosystem was suggested by Ron Rivest, Adi Shamir, and Leonard Adleman in 1978. The security of RSA is computational security and relies on the difficulty of integer factorization. Since its publication, RSA has been subject to research and attack attempts, but no general attacks have been found. Brute force and factoring attacks can be prevented by using long key lengths, and any side-channel attacks can be tackled by following certain best practices in implementation.

Newer cryptosystems such as elliptic curve cryptography (ECC) have been developed as alternatives to RSA. However, RSA is still widely used for various purposes, such as:

- Secure communication: secure email and instant messaging applications, protocols for secure web browsing (VPNs and SSL/TLS).
- Digital signatures
- Key exchange, for example in the SSH protocol used for remote login to servers.

RSA key generation

RSA is based on the assumption that integer factorization is a *one-way function*. We assume that it is computationally infeasible to find the totient $\varphi(n)$ for a given large integer n, and therefore it is also computationally infeasible to find the inverse e^{-1} of an integer e $(mod\ \varphi(n))$ without extra information. However, if one does know the factorization of n, then one can find e^{-1} $(mod\ \varphi(n))$ using the Extended Euclidean Algorithm. This is the *trapdoor* for RSA.

The key generation protocol of RSA is as follows:

1. Choose two large distinct prime numbers, p and q.
2. Calculate their product $n = pq$, which is the modulus of the RSA system.
3. Calculate the totient of n, $\varphi(n) = (p-1)(q-1)$.
4. Choose the public exponent, an integer $e \in \{2, \dots, \varphi(n)-1\}$ that is relatively prime to $\varphi(n)$, i.e., $gcd(e, \varphi(n)) = 1$.
5. Calculate the private exponent d, such that $de \equiv 1\ (mod\ \varphi(n))$, i.e., d is the multiplicative inverse of e modulo $\varphi(n)$.

Now, the public key is (n, e), and the private key is (n, d).

Example. Given parameters $p = 41,\ q = 17$, generate a pair of RSA encryption and decryption keys.

RSA encryption and decryption

RSA public key is the pair $k_{pub} = (n, e)$, where n is the modulus and e is the public exponent. RSA encryption operation is the modular exponentiation:

$$y = e_{k_{pub}}(x) = x^e \ (mod\ n)$$

RSA private key is pair $k_{pr} = (n, d)$, where private exponent d is the inverse of e in $Z_{\varphi(n)}$, i.e.,

$$ed = 1 \ (\mathrm{mod}\ \varphi(n))$$

RSA decryption operation is the modular exponentiation:

$$x = d_{k_{pr}}(y) = y^d \ (mod\ n)$$

The correctness of the RSA operations is based on Euler's theorem: when $\gcd(x, n) = 1$, then

$$x^{\varphi(n)} \equiv 1 \ (\mathrm{mod}\ n)$$

Hence the decryption process becomes:

$$\begin{aligned} y^d &= x^{ed} \\ &= x^{k\varphi(n)+1} \\ &= \left(x^{\varphi(n)}\right)^k \cdot x \\ &\equiv 1^k \cdot x \\ &= x \ \ (\mathrm{mod}\ n) \end{aligned}$$

Example. Encrypt and decrypt message $x = 100$ using keys $(n, e) = (253,3)$ and $(n, d) = (253,169)$.

Key lengths of RSA

The recommended key lengths for RSA in practice depend on the level of security required and the current state of technology. Generally, longer key lengths provide greater security but also require more computational resources and increase the size of the encrypted data. In practice, the key lengths used for RSA vary depending on the specific application and security requirements.

As of 2021, the NIST recommends a minimum key length of 2048 bits for RSA . This key length is considered secure against current and future attacks.

Fast exponentiation: Square-and-multiply algorithm

Modular exponentiation is a key operation in RSA. Without making use of modular arithmetic, it is computationally expensive to perform, especially with the long RSA parameters. To find x^e, the straightforward exponentiation

$$x \overset{SQ}{\to} x^2 \overset{MUL}{\longrightarrow} x^3 \overset{MUL}{\longrightarrow} x^4 \overset{MUL}{\longrightarrow} \dots$$

requires about $2^{|e|}$ multiplications. For $|e| = 1024$, this already becomes $2^{1024} \approx 10^{300}$ which is clearly infeasible. Thus, the naïve method is impossible to use, and a faster method is needed.

Luckily, there are several faster algorithms for modular exponentiation, each with different trade-offs between speed and memory usage. One of the most common algorithms for modular exponentiation is the square-and-multiply algorithm which works as follows:

Input:	Base element x Exponent $e = \sum_{i=0}^{t} e_i 2^i$ with $e_i \in \{0,1\}$ and $e_t = 1$
Output:	$x^e \ (mod\ n)$
Initialize:	$r = x$

1. For each bit in the binary representation of e, starting with the most significant bit:

   1.1 $r = r^2 \ (mod\ n)$
   1.2. If the current bit is 1, $r = r \cdot x \ (mod\ n)$

2. Return r.

To verbally describe the algorithm: Scan the exponent bits of e one by one from left to right

a) On each round, square

b) If the current bit is 1, multiply

Example. Perform a step-by-step computation of x^{26} using

a) the naïve algorithm
b) square-and-multiplication algorithm.

Example. Compute $x^e \ (mod\ n)$ for $x = 3,\ e = 197,\ n = 101$.

STEP		(mod 101)	power of x in binary
0 (initial step)	3^1	3	3^{1_2}
1A	3^2	9	3^{10_2}
1B	$9 \cdot 3$	27	3^{11_2}
2A	27^2	22	3^{110_2}
3A	22^2	80	3^{1100_2}
4A	80^2	37	3^{11000_2}
5A	37^2	56	3^{110000_2}
5B	$56 \cdot 3$	67	3^{110001_2}
6A	67^2	45	$3^{1100010_2}$
7A	45^2	5	$3^{11000100_2}$
7B	$5 \cdot 3$	15	$3^{11000101_2}$

Complexity of the Square-and-multiply algorithm

Consider exponent e with bit length $t + 1$, i.e.

$$\lfloor \log_2 e \rfloor = t + 1$$

On average, half of the bits are 1's. Thus, the number of squarings $\#SQ = t$ and the number of multiplications $\#MUL \approx 0.5t$, hence a total of $1.5t$ operations must be performed. We deduce that the square-and-multiply algorithm has a logarithmic complexity, whereas straightforward exponentiation has a linear complexity. Thereby, sq-and-mul algorithm greatly reduces the number of modular multiplications needed to compute the result. Additionally, the algorithm can be optimized by precomputing the squares of the base number, which can be reused in subsequent iterations of the algorithm.

Some speed-up techniques for RSA

In addition to fast exponentiation, there are other techniques to speed up the performance of RSA operations.

- Choosing short public exponents for fast encryption and signature verification. The public exponent can be chosen to be short, and also to have a low *Hamming weight* (= #1's in the binary presentation of the4 number). This means less operations in the process of encryption and correctly implemented will not compromise the security of RSA.

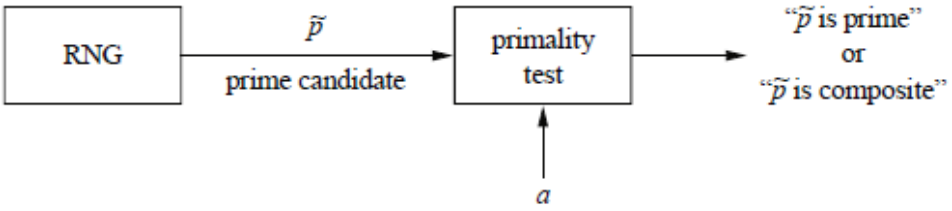
Public key e	e as binary string	#MUL + #SQ
3	11_2	3
17	10001_2	5
$2^{16}+1$	$1\,0000\,0000\,0000\,0001_2$	17

- Accelerating decryption based on the *Chinese Remainder Theorem* (CRT). For security reasons, RSA private exponents must always be long. The CRT is a fundamental result in number theory, which provides a method that allows to find $x^d \ (mod\ n)$ by computing two exponents of length approximately half the length of d. As the complexity of multiplication decreases quadratically with the bit length, the total speed-up obtained using this method is a factor of 4.

On finding large primes

Large primes p and q are needed for RSA setup. The principal approach to find large primes is the following:

1. Generate random number $\tilde{p}$
2. Test $\tilde{p}$ for primality



Primality test algorithms are *probabilistic* in a way that a negative result ("$\tilde{p}$ is a composite") is always correct and a positive result ("$\tilde{p}$ is a prime") is *probably* correct. Thus, there is a chance for false positive results.

Primality test algorithms are also *randomized*, meaning that there is a random parameter a involved in the algorithm. Test can be repeated for different values of a until the probability for false positive result becomes negligible.

Some examples of common primality test algorithms are the Fermat Primality Test and the Miller-Rabin Primality Test. (See for example https://brilliant.org/wiki/prime-testing/ .)

The probability for a random odd number to be prime is relative to its length:

$$P(\tilde{p}\ is\ prime) \approx \frac{2}{\ln(\tilde{p})}$$

Example. The probability for a random number of length 512 bits to prime is

$$P \sim \frac{2}{\ln(2^{512})} = \frac{2}{512 \cdot \ln(2)} \sim \frac{1}{177} = 0{,}00564\ldots$$

This means that one must test approximately 177 randomly picked odd numbers before hitting a prime.

Exercise 1. Find $\varphi(n)$ for numbers $n = 30, 301, \text{and } 3001$.

Exercise 2. Compute the inverse $a^{-1}\ mod\ n$ with Fermat's Little Theorem (if applicable) or Euler's Theorem:

a) $a = 4, n = 7$

b) $a = 5, n = 12$

c) $a = 6, n = 13$

Exercise 3. Let primes $p = 59$ and $q = 47$ be given as set-up parameters for RSA. Which of the following are valid parameter choices for RSA public exponent *e,* and why / why not?

a) $e = 17$

b) $e = 23$

c) $e = 31$

Exercise 4. A message was encrypted using the RSA public key pair $(n, e) = (86609,17)$. Given the set-up parameters $p = 257, q = 337$, decrypt the ciphertext $y = 12345$.

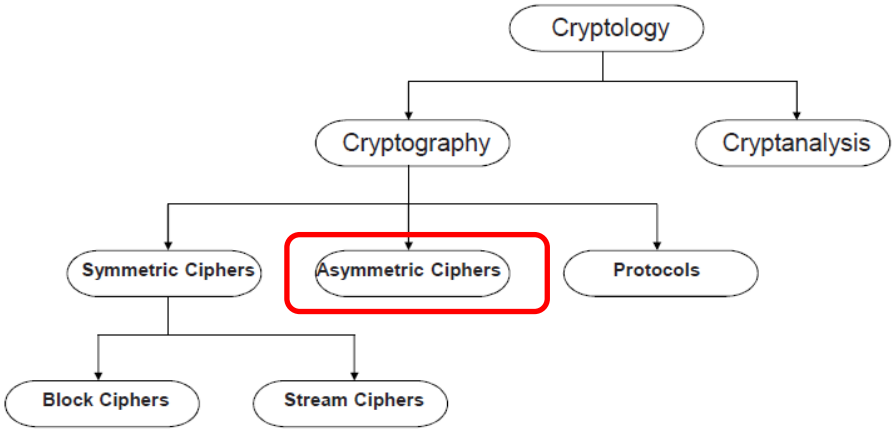
Exercise 5. Apply the square-and-multiply algorithm to compute $x^e\ (mod\ m)$, where $x = 2, e = 79, m = 101$. Show the exponent in binary notation after every iteration step.

Exercise 6. How many random odd integers do we have to test on average, until we expect to find a prime factor p for RSA modulus n of size 2048?

Exercise 7. Shortly explain the following terms:

a) Hamming weight
b) RSA trapdoor
c) false positive (in primality testing)

The Diffie-Hellman Key Exchange



The Diffie-Hellman Key Exchange (DHKE) was proposed by W. Diffie and M. Hellman in 1976. It was the first published asymmetric crypto scheme. The DHKE provides a method for two parties to establish a shared secret key over an insecure channel. The resulting key can be used for example as a session key for a symmetric algorithm like AES or 3DES. The DHKE is still widely used in various security protocols, such as IPsec, SSH, and TLS.

The security of the Diffie-Hellman key exchange relies on the difficulty of the *discrete logarithm problem*, which is considered to be a one-way function. In other words, it is easy to compute powers of a generator over a finite field, but it is very difficult to compute the discrete logarithm of a given number with respect to that generator.

The DHKE System setup & protocol

1) Choose modulus $p \in P$.
2) Choose a generator (primitive element) $\alpha \in Z_p^*$
3) Publish p and α.

After Alice and Bob have agreed on the *domain parameters* (or *system parameters*) p and α, the key exchange protocol is as follows:

Alice		**Bob**
Choose private key $a \in \{2, \dots, p-2\}$		Choose private key $b \in \{2, \dots, p-2\}$
Compute public key $A = \alpha^a \ (mod\ p)$		Compute public key $B = \alpha^b \ (mod\ p)$
	$\xrightarrow{A}$ $\xleftarrow{B}$	
Compute shared key $k_{AB} = B^a \ (mod\ p)$		Compute shared key $k_{AB} = A^b \ (mod\ p)$

It is easy to show that both computations yield the same key:

- Alice computes: $B^a = (\alpha^b)^a = \alpha^{ba} = \alpha^{ab} \ (mod\ p) = k_{AB}$
- Bob computes: $A^b = (\alpha^a)^b = \alpha^{ab} \ (mod\ p) = k_{AB}$

Example. Let Alice and Bob generate a common key using the DHKE protocol with parameters

a) $p = 29,\ \alpha = 2$
b) $p = 59,\ \alpha = 6$

Some computational aspects of the DHKE

In many computational aspects, the DHKE is similar to RSA. There is a need for a large prime parameter p, and the integer α must have some special properties (be primitive in Z_p^*).

From 2021, the NIST recommends the following parameter sizes for DHKE:

- 2048-bit modulus for key exchange
- 3072-bit modulus for long-term keys

These parameter sizes are considered sufficient to provide strong security against attacks by modern computing resources. Some applications may require even higher security levels.

To choose a and b, it is ideal to use a true random number generator. To compute the modular exponentiations to find A and B, one can use the SQ-and-MUL algorithm.

On finding primitive elements

To establish a DHKE setup, we must find a suitable generator α. The naïve method would be to pick any element $\alpha \in Z_p^*$ and try $\alpha, \alpha^2, \alpha^3, \ldots (mod\ p)$ to see if α is primitive. As p is very large, this is obviously not a computationally feasible approach.

Luckily, we can make use of the mathematical theorem which states that if an element $\alpha \in Z_p^*$ is *not* primitive, then its order is a factor of $p-1$. In a more mathematical notation:

$$\text{if } ord(\alpha) = k < p-1, \text{ then } k \mid p-1$$

This means that to find if α is primitive, we do not have to check all consecutive powers $2,3,4, \ldots$ of α modulo p, but rather it suffices to check if

$$\alpha^{k_i} \equiv 1\ (mod\ p)$$

for any proper divisor k_i of $p-1$.

This can be generalized to find primitive elements in any Z_n^*: If $\alpha \in Z_n^*$ is not primitive, i.e.

$$\text{if } ord(\alpha) = k < \varphi(n), \text{ then } k \mid \varphi(n)$$

Therefore, it suffices to check if

$$\alpha^{k_i} \equiv 1\ (mod\ n)$$

for any proper divisor k_i of $\varphi(n)$.

Example. a) Is 2 a primitive element of Z_{23}^*?

b) Is 5 a primitive element of Z_{233}^*?

Exercise 1. Find the members of the multiplicative group Z_7^*. Find the order of each element. Are there any primitive elements in Z_7^*? Is Z_7^* cyclic?

Exercise 2. Compute public keys A, B and the shared session key K_{AB} for the DHKE scheme with parameters $p = 467$ and $\alpha = 2$, and

a) $a = 3, b = 5$

b) $a = 228, b = 57$

Exercise 3. Show that $\alpha = 4$ is not a primitive element in the field Z_{467}.

Exercise 4. Compute the shared session key k_{AB} for DHKE parameters $p = 467$ and $\alpha = 4$ for

a) $a = 400, b = 134$

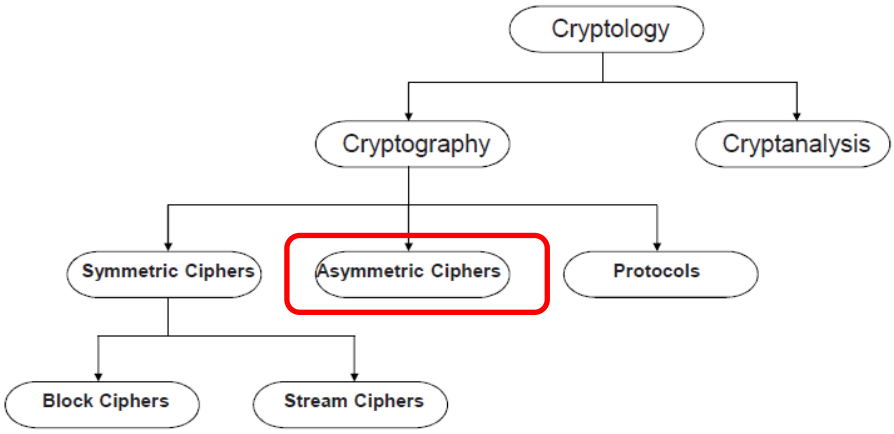
b) $a = 167, b = 134$

You will find that both cases result in the same shared key. How is this possible?

Exercise 5. In the DHKE protocol, the private keys are chosen from the set $\{2, \dots, p-2\}$. Discuss why the values *1* and $p - 1$ are excluded.

Exercise 6. Public key algorithms without public key authentication are vulnerable to the so-called *man-in-the-middle* attack. Describe the attack on the basic DHKE protocol.

Elliptic Curve Cryptosystems



Elliptic curve cryptosystems (ECC) are a class of public-key cryptosystems that use elliptic curves over finite fields as the underlying mathematical structure. They were independently invented and proposed by Victor Miller in 1986 and by Neal Koblitz in 1987.

ECCs are based on the generalized discrete logarithm problem, and they provide the same level of security as RSA or discrete logarithm systems, but with much shorter operands. Therefore, they require less computational resources compared to other public-key cryptosystems.

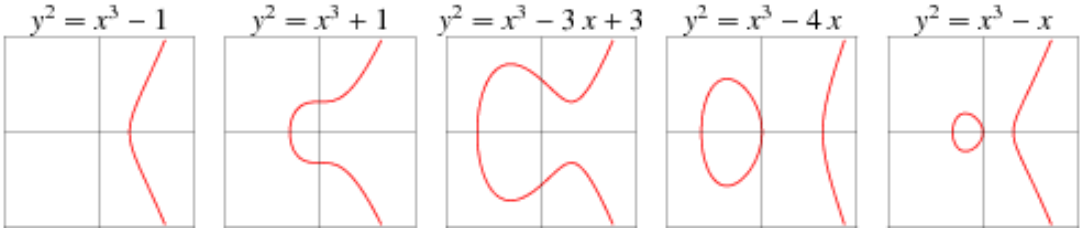
Elliptic curve cryptosystems have found widespread use in many applications including secure communication protocols, digital signatures, and key agreement protocols. They are used in various cryptographic standards such as TLS, SSH, and PGP.

Elliptic curve cryptosystems are known for their high level of security, but some attacks have been discovered against specific elliptic curves. Hence, it is important to carefully select the elliptic curve parameters to ensure sufficient security. In practice, the parameters are typically chosen by following well-established standards that specify criteria for selecting secure elliptic curves.

Elliptic curves

An elliptic curve E is a set of points defined by a polynomial equation called the *Weierstrass* equation:

$$E: \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

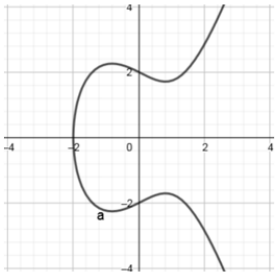


For cryptographic purposes, elliptic curves are defined over prime fields $GF(p)$ or Galois extension fields $GF(2^m)$. The equation for elliptic curve E can be written in simplified form:

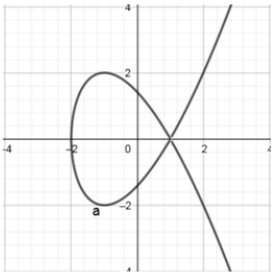
$$E\colon\ y^2 = x^3 + ax + b$$

Example. (Elliptic curves plotted over R)

$y^2 = x^3 - 2x + 4$



$y^2 = x^3 - 3x + 2$



For ECC, we only use curves that are non-singular, meaning that they don't have any self-intersections or cusps. Here, the curve on the right would not be fit for ECC because of the self-intersection at $(1,0)$.

Group operations on Elliptic Curves

We can define a mathematical group on an elliptic curve. The points of a specified curve E form the set of elements, and the group operation is defined as point addition $(+)$. To have all group properties, we must add a point at infinity O to the set of points. Its purpose is to serve as the identity element of the group. The group properties are the following:

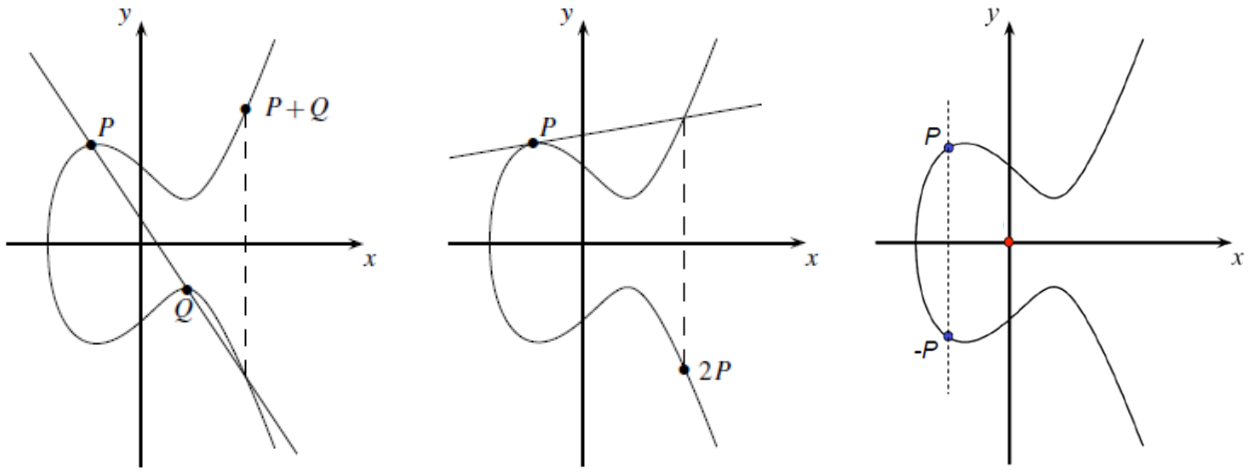
- The group is closed under operation
- Operation $+$ is associative: $(P + Q) + R = P + (Q + R)$
- There is an identity element O such that $P + O = P$ for all elements P.
- Every element P has an inverse $-P$ such that $P + (-P) = O$

Point operations on a curve

Adding two points $P + Q$, where $P \neq Q$, is called point addition

Adding $P + P$ is called point doubling $(2P)$.

Below you can see the graphic interpretation of the point operations addition, doubling and inversion on an elliptic curve:



Elliptic Curves over prime fields $GF(p)$

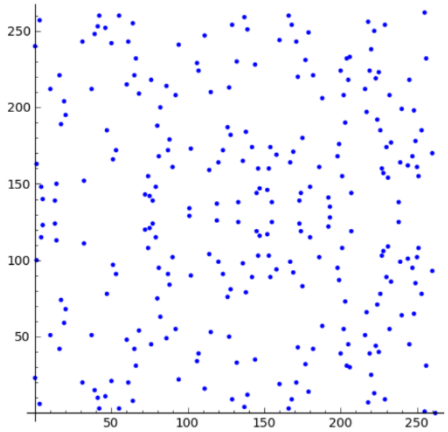
Definition. The elliptic curve over Z_p, $p > 3$, is the set of all pairs $(x, y) \in Z_p^2$ which fulfill

$$y^2 \equiv x^3 + ax + b \ (mod\ p)$$

where $a, b \in Z_p$ and $4a^3 + 27b^2 \neq 0 \ (mod\ p)$, together with an imaginary point of infinity O.

We note that condition $4a^3 + 27b^2 \neq 0$ is to ensure non-singularity of the curve.

Example. Points of $y^2 = x^3 + 2x + 3$ over $GF(263)$:



EC point addition and doubling over $GF(p)$

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points on an elliptic curve. Then the coordinates of the sum $R = P + Q = (x_3, y_3)$ are given by

$$\begin{cases} x_3 = s^2 - x_1 - x_2 \ (mod\ p) \\ y_3 = s(x_1 - x_3) - y_1 \ (mod\ p) \end{cases}$$

where the slope s is given by

$$s = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \ (mod\ p) \text{ for } P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1} \ (mod\ p) \text{ for } P = Q \end{cases}$$

Example. Consider the curve $E: y^2 \equiv x^3 + 2x + 2$ over $GF(17)$.

a) (Point addition) Find the sum $P + Q$ of points $P = (3,1)$ and $Q = (7,6)$

b) (Point doubling) Double the point $R = (5,1)$.

Point inversion and the point at infinity

We have defined the abstract point at infinity O as the **identity element** such that $P + O = P$ for all points on the elliptic curve, and the inverse $-P$ of a point P to be such that $P + (-P) = O$.

In $GF(p)$, the coordinates of the inverse for $P = (x_p, y_p)$ are simply given by $-P = (x_p, p - y_p)$.

Example. (Point inversion)
Find the inverses of points $P = (3,1)$ and $Q = (7,6)$ in $E: y^2 \equiv x^3 + 2x + 2$ over $GF(17)$

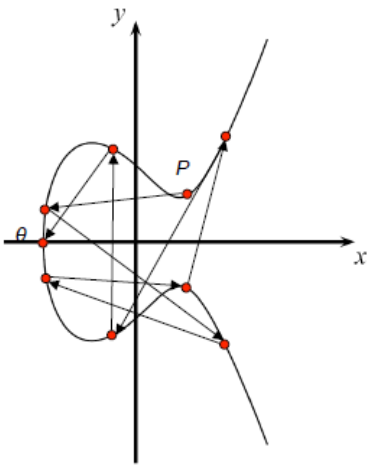
Cyclic groups on elliptic curve groups

We know that under certain conditions, all points on an elliptic curve form a cyclic group. Further, if the order $|G|$ of a group G is a prime, then *all* elements except for the identity element are primitive.

Example. (EC cyclic group)

Let us find the multiples of point $P = (5,1)$ on curve $E: y^2 = x^3 + 2x + 2$ over Z_p:

$$
\begin{aligned}
2P &= (5,1) + (5,1) = (6,3) \\
3P &= 2P + P = (10,6) \\
4P &= (3,1) \\
5P &= (9,16) \\
6P &= (16,13) \\
7P &= (0,6) \\
8P &= (13,7) \\
9P &= (7,6) \\
10P &= (7,11)
\end{aligned}
\qquad
\begin{aligned}
11P &= (13,10) \\
12P &= (0,11) \\
13P &= (16,4) \\
14P &= (9,1) \\
15P &= (3,16) \\
16P &= (10,11) \\
17P &= (6,14) \\
18P &= (5,16) \\
19P &= \mathcal{O}
\end{aligned}
$$



$P = (5,1) = (5,17 - 16)$ is the inverse of $18P$. Hence $18P + P = O$ by definition.

Elliptic Curves in implementations

Elliptic curve cryptosystems are based on the generalized discrete logarithm problem. To achieve cryptographic strength, the choice of curve E is crucial. The cyclic group formed by the curve points must be large and have a prime order, in order to guarantee the difficulty of the Discrete Logarithm problem. Popular elliptic curves in cryptology are those over prime fields $GF(p)$ and binary Galois extension fields $GF(2^m)$. Since checking curves for some wanted properties and unwanted weaknesses is nontrivial and computationally demanding, standardized curves and parameters (ANSI, NIST, NSA etc.) are used to ensure cryptographically strong properties.

Elliptic Curve Discrete Logarithm Problem

Definition. Given an elliptic curve E, a primitive element P and another element T, the ECDLP is the problem of finding $d \in \{1, \dots, |E|\}$ such that $dP = T$.

The ECDLP has very good one-way function characteristics and all DLP based protocols can be realized with elliptic curves. The only known attacks against cryptographically strong elliptic curves are generic algorithms to solve discrete logarithm, such as Pollard's Rho method and Shanks' baby-step giant-step method

Note. In EC cryptosystems, integer d is the private key and point T is the public key.

Note. Point multiplication dP is a shorthand notation for applying point addition operation d times: $dP = P + P + \cdots + P$. Point multiplication is analogical to exponentiation, and we can adapt the SQ-and-MUL algorithm into Double-and-Add algorithm to speed up the computations.

Example. Consider in group $E: y^2 = x^3 + x + 6$ over $GF(11)$. For $B = (5,9)$, find $6B$ using the Double-and -Add algorithm.

Elliptic Curve Diffie-Hellman Key Exchange: The ECDH System setup & protocol

1) Choose a prime p and the elliptic curve $y^2 \equiv x^3 + ax + b \ (mod\ p)$
2) Choose a primitive element $P = (x_p, y_p)$
3) Publish the domain parameters p, a, b, P.

After Alice and Bob have agreed on the domain parameters, they can perform the key exchange protocol as follows:

Alice		**Bob**
Choose private key $a \in \{2, \dots, \lvert E\rvert - 1\}$		Choose private key $b \in \{2, \dots, \lvert E\rvert - 1\}$
Compute public key $A = aP = (x_A, y_A)$		Compute public key $B = bP = (x_B, y_B)$
	$\overset{A}{\rightarrow}$ $\quad$ $\overset{B}{\leftarrow}$	
Compute shared key $aB = T_{AB} = (x_{AB}, y_{AB})$		Compute shared key $bA = T_{AB} = (x_{AB}, y_{AB})$

It is easy to show that both computations yield the same key:

- Alice computes: $aB = a(bP) = abP$
- Bob computes: $bA = b(aP) = abP$

Example. Let Alice and Bob generate a common session key T_{AB} using the ECDH protocol in

$$E: y^2 \equiv x^3 + x + 6 \text{ over } GF(11) \text{ with } P = (2,4)$$

Exercise 1. Solve for x (if possible) the Discrete logarithm problem $\alpha^x = \beta \pmod{p}$ when

a) $\alpha = 5, \beta = 2, p = 7$
b) $\alpha = 2, \beta = 5, p = 7$
c) $\alpha = 2, \beta = 4, p = 7$

Exercise 2. Learn about computational Diffie-Hellman assumption (CDH) and about decisional Diffie-Hellman assumption (DDH), and state them in a simple manner. What is the relationship between CDH and DDH?

Exercise 3. Given an elliptic curve $E: y^2 = x^3 + x + 7$ over Z_{17} and points $P = (1,3)$ and $Q = (2,0)$, find $P + Q$.

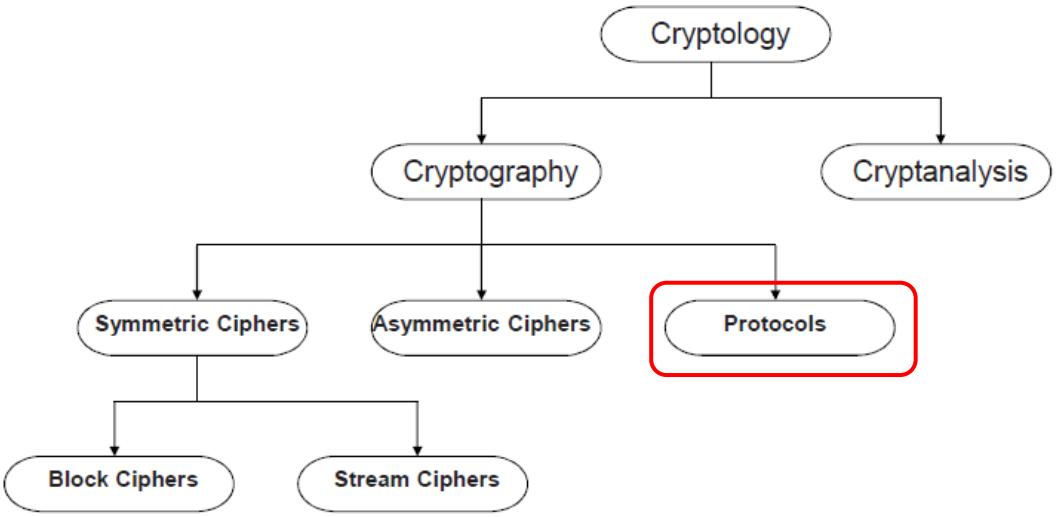
Exercise 4. Consider the elliptic curve $E: y^2 = x^3 + 4x + 20$ over $GF(29)$. Find

a) the inverse of $P = (8,10)$
b) $2P$

Exercise 5. Consider the elliptic curve $E: y^2 = x^3 + 4x + 20$ over $GF(29)$. For $P = (8,10)$, use the Double-and-add algorithm to find $9P$. (Answer: $9P = (4,10)$) . (You can use an EC point calculator to perform point computations.)

Exercise 6. Let $a = 2, b = 9, p = 17$ and $P = (4,8)$ be the domain parameters for the ECDH protocol. Alice and Bob will use the ECDH to agree on a joint session key T_{AB}. Their respective choices for private keys are $a = 10, b = 13$. Show the steps of the protocol. (You can use an EC point calculator to perform point computations.)

Digital Signatures



A digital signature is a digital equivalent of a handwritten signature, a mathematical technique used to verify the authenticity and integrity of a digital document or message. Digital signatures are widely used in digital transactions, such as electronic contracts, digital certificates, online banking, and e-commerce. Some popular DS protocols are:

- RSA Signature scheme: one of the most widely used digital signature protocols.
- DSA (Digital Signature Algorithm): this protocol is based on the discrete logarithm problem and is widely used in government and military applications.
- ECDSA: the elliptic curve variant of DSA is known for its efficiency and strong security.
- GPG (GNU Privacy Guard): an open-source implementation of the OpenPGP standard, which provides a way to encrypt and sign digital messages.

Digital signing provides the following cryptographic goals:

1. **Authentication**: Digital signatures provide a way to confirm the identity of the signer.
2. **Integrity**: Digital signatures that the content has not been modified since it was signed.
3. **Non-repudiation**: the signer cannot deny that they signed the message. The signature provides proof of the signer's intent to sign the content.

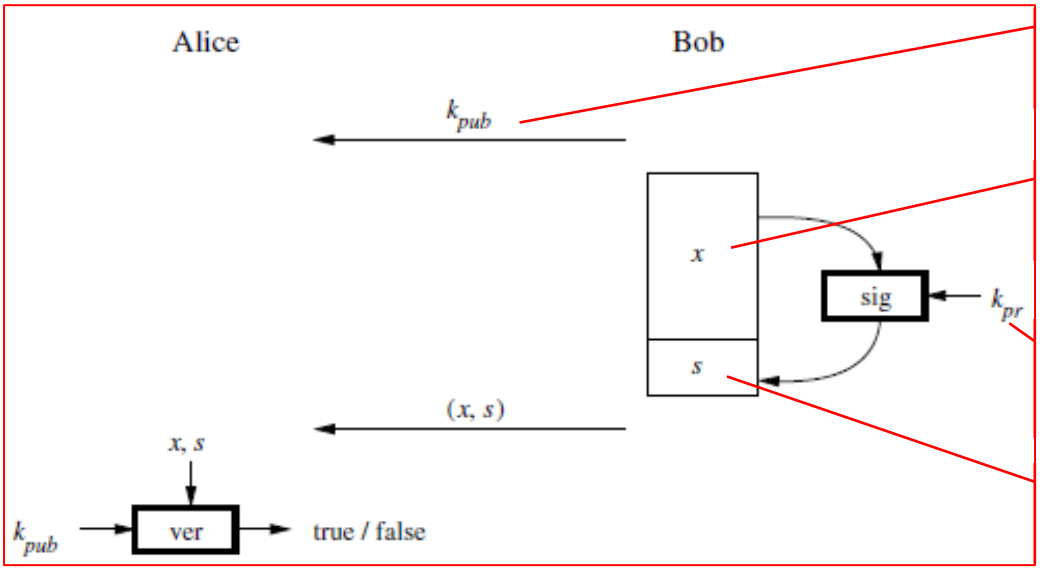
Note that digital signatures as such do not provide confidentiality. However, they can be used in conjunction with encryption to add confidentiality.

Principle of digital signature schemes

Digital signing can be realized with public key cryptography. The basic principle is the same, regardless of which digital signature scheme is in use.

- To *create* a digital signature, the signer uses a private key to encrypt the message being signed. The resulting encrypted data, or signature, is then attached to the message.

- Anyone who wants to *verify* the authenticity of the signature will use the signer's corresponding public key to decrypt the signature. If the value obtained by decryption matches the original message, then the signature is considered valid, and the content is deemed to be authentic and unaltered.



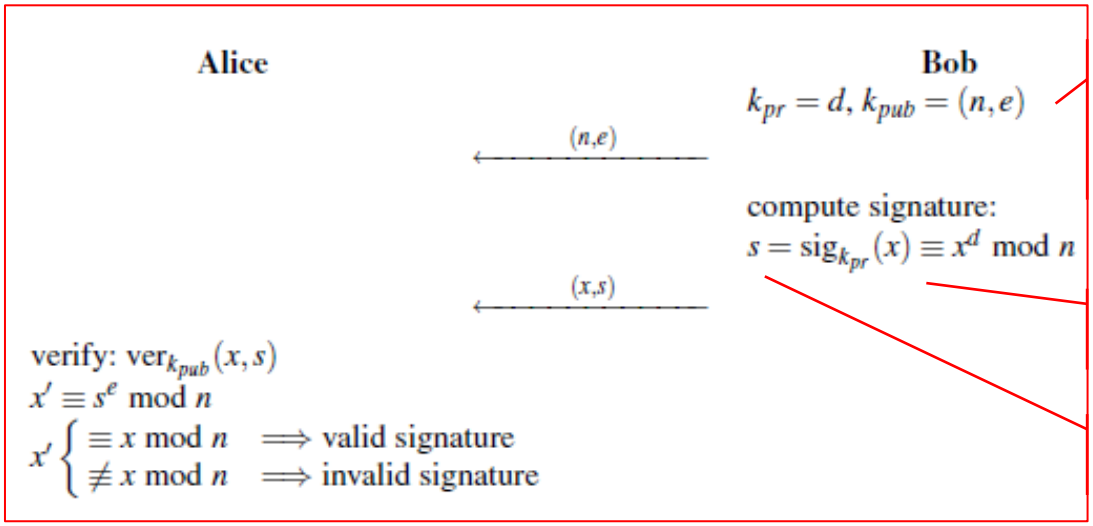
In practice, digital signature schemes are typically used in combination with *secure hash functions*: the original message is first hashed into a (short) fixed-length message, which is then used as the input for the signing algorithm to create the signature.

RSA Digital Signature Scheme

The key generation of RSA Digital signature scheme is identical to that of the RSA Encryption scheme. The protocol operates on the keys of the signer. The resulting signature s is as long as the modulus n (typically 1024 to 3072 bits).

As the operation to compute and verify s is modular exponentiation, the SQ-and-MUL algorithm is used to speed up the computations. Choosing a small public exponent e will make the verification part very fast. This comes handy in many applications, as contracts, certificates etc. are typically signed just once but verified multiple times.

RSA Signature Protocol:



Example. (RSA Digital signature scheme)

Bob is to sign the message $x = 234$ using his RSA keys $(n, e) = (253,7),\ (n, d) = (253,63)$. The protocol steps are as follows:

Alice		**Bob**
	$\xleftarrow{(n,e)=(253,7)}$	
		Compute signature: $s = 234^{63} \equiv 170\ (mod\ 253)$
	$\xleftarrow{(x,s)=(234,170)}$	

Verification:

$x' = 170^7 \equiv 234\ (mod\ 253)$

Compare if $x \equiv x'$?

→ a valid signature

Proof of Correctness

It is easy to show that for a valid signature s, the verification process yields the original message x:

$$\begin{aligned} s^e &= (x^d)^e \\ &= x^{de} && \mid de \equiv 1 \left(mod\ \varphi(n)\right) \\ &= x^{k\cdot\varphi(n)+1} \\ &= \left(x^{\varphi(n)}\right)^k \cdot x && \mid x^{\varphi(n)} \equiv 1\ (mod\ n) \\ &\equiv x\ (mod\ n) \end{aligned}$$

RSA Digital Signature with Encryption

To add *confidentiality* to the signing protocol, one can either use a symmetric algorithm to encrypt the message-signature pair (x, s), or use a DS scheme with augmented encryption, such as the RSA Digital Signature with Encryption scheme.

In the following, (n_A, e_A) is Alice's public key, and (n_A, d_A) and (n_B, d_B) are Alice's and Bob's private keys, respectively. After exchanging the public keys, Bob proceeds to perform signature generation, signature encryption, and finally message encryption.

After receiving the ciphertext y and the encrypted signature, Alice decrypts the message and the signature, then verifies the signature.

Alice		**Bob**
	$\xrightarrow{(n_A, e_A)}$	
	$\xleftarrow{(n_B, e_B)}$	$s = sig_{d_B}(x) = x^{d_B} \pmod{n_B}$
		$E_{e_A}(s) = s^{e_A} \pmod{n_A}$
	$\xleftarrow{(y, E_{e_A}(s))}$	$y = E_{e_A}(x) = x^{e_A} \pmod{n_A}$
$x = D_{d_A}(x) \equiv y^{d_A} \pmod{n_A}$		
$s = D_{d_A}(E_{e_A(s)})$		
$x' = s^{e_B} \pmod{n_B}$		
$x' \equiv x$?		
yes → valid signature		
no → invalid signature		

Some security aspects

Algorithmically, RSA Signature scheme is as safe as RSA Encryption Scheme. However, the textbook RSA Signature Scheme has some weaknesses such as being exposed to *existential forgery.* To prevent existential forgery and other attacks, it is customary to use padding schemes, which means specifying the form of the message, in order to be able to distinguish between valid and invalid messages.

RSA Probabilistic Signature Scheme (RSA-PSS) combines signature and verification with *message encoding*, known as Encoding Method for Signature with Appendix (EMSA) in the RSA Cryptography standard PKCS#1.

Principle of EMSA-PSS

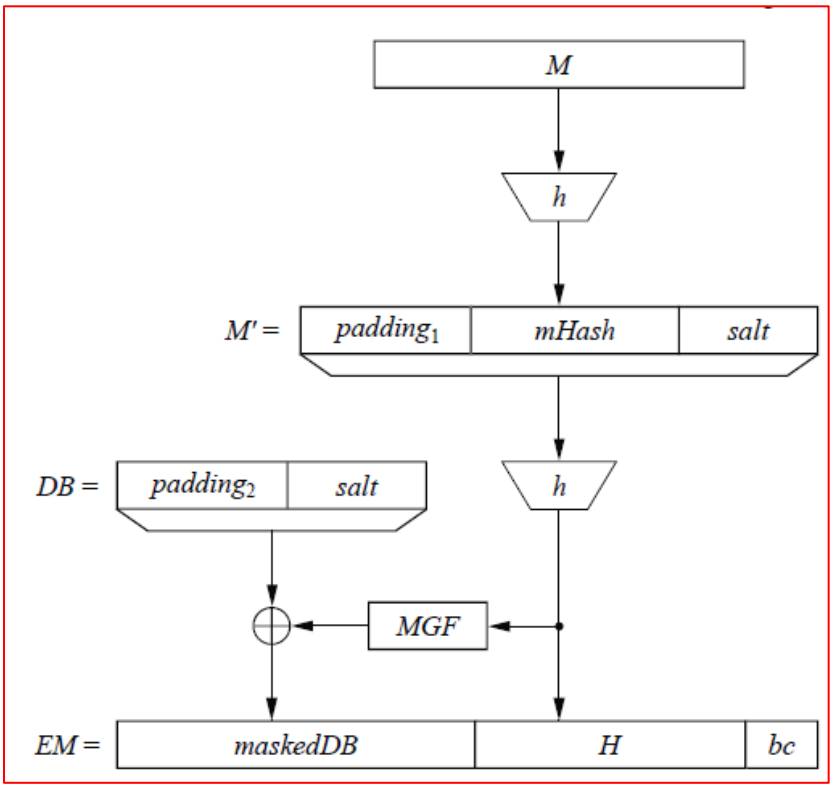
Message M
Hash function h

Fixed paddings $padding_i$
$mHash = h(M)$
Random value *salt*

Mask Generation function *MGF*
$H = h(M')$

Data Block *DB*
$maskedDB = DB \oplus MGF(H)$
Fixed padding *bc*

Encoded Message *EM*



After encoding the message, the actual signing operation is applied to EM:

$$s \equiv (EM)^d \ (mod\ n)$$

Values $padding_1, padding_2$ and bc are fixed in the standard but *salt* is a random value, which makes the algorithm probabilistic. Thereby encoding and signing the same message will result in a different signature each time.

Digital Signature Algorithm

The Digital Signature Algorithm (DSA) is a widely used digital signature algorithm that was introduced by the NIST in 1991. DSA has been standardized by several organizations, including NIST, and is widely used in digital certificates, secure email, and other applications where digital signatures are required.

One of the benefits of DSA is that it provides a high level of security while using smaller key sizes compared to other algorithms like RSA. Also, the signature length is smaller than RSA signatures of equivalent security. However, DSA can be slower than many other algorithms, and it is mainly used in government and military applications.

DSA Key Generation

1. Generate a prime p with $2^{1023} < p < 2^{1024}$.
2. Find a prime divisor q of $p-1$ with $2^{159} < q < 2^{160}$.
3. Find an element α with $\text{ord}(\alpha) = q$, i.e., α generates the subgroup with q elements.
4. Choose a random integer d with $0 < d < q$.
5. Compute $\beta \equiv \alpha^d \bmod p$.

Here, p, q and α are the domain parameters. The public key is $k_{pub} = \beta$ and the private key is $k_{pr} = d$. Mathematically, two cyclic groups are involved:

- Z_p^* with an order of bit length 1024
- A subgroup of Z_p^*, generated by α, with an order of bit length 160

DSA Signature Generation

1. Choose an integer as random ephemeral key k_E with $0 < k_E < q$.
2. Compute $r \equiv (\alpha^{k_E} \bmod p) \bmod q$.
3. Compute $s \equiv (SHA(x) + d \cdot r)\, {k_E}^{-1} \bmod q$.

Here, $SHA(x)$ represents the 160-bit fingerprint of message x computed using a hash function from SHA family. One could use another secure hash function instead. The signature is the pair (r, s) with the length $160 + 160 = 320$ bits.

For enhanced security level, one can opt for longer p and q, which will also affect the required hash output length. The security level here is in comparison to the time complexity of a brute force attack in a key space of size 2^{80} or 2^{112} or 2^{128} bits.

p	q	Hash output (min)	Security levels
1024	160	160	80
2048	224	224	112
3072	256	256	128

DSA Signature Verification

1. Compute auxiliary value $w \equiv s^{-1} \bmod q$.
2. Compute auxiliary value $u_1 \equiv w \cdot SHA(x) \bmod q$.
3. Compute auxiliary value $u_2 \equiv w \cdot r \bmod q$.
4. Compute $v \equiv (\alpha^{u_1} \cdot \beta^{u_2} \bmod p) \bmod q$.
5. The verification $ver_{k_{pub}}(x, (r, s))$ follows from:

$$v \begin{cases} \equiv r \bmod q \implies \text{valid signature} \\ \not\equiv r \bmod q \implies \text{invalid signature} \end{cases}$$

In the following example, we will go through the DSA key generation, signature and verification process using small numerical values.

Example.

Alice | **Bob**

Bob:
1. choose $p = 59$
2. choose $q = 29$
3. choose $\alpha = 3$
4. choose private key $d = 7$
5. $\beta = \alpha^d \equiv 4 \bmod 59$

$\xleftarrow{(p,q,\alpha,\beta)=(59,29,3,4)}$

Bob:
sign:
compute hash of message $h(x) = 26$
1. choose ephemeral key $k_E = 10$
2. $r = (3^{10} \bmod 59) \equiv 20 \bmod 29$
3. $s = (26 + 7 \cdot 20) \cdot 3 \equiv 5 \bmod 29$

$\xleftarrow{(x,(r,s))=(x,(20,5))}$

Alice:
verify:
1. $w = 5^{-1} \equiv 6 \bmod 29$
2. $u_1 = 6 \cdot 26 \equiv 11 \bmod 29$
3. $u_2 = 6 \cdot 20 \equiv 4 \bmod 29$
4. $v = (3^{11} \cdot 4^4 \bmod 59) \bmod 29 = 20$
5. $v \equiv r \bmod 29 \Longrightarrow$ valid signature

Some computational aspects

The key generation phase of DSA is quite complex, but it is only executed once at set-up. Instead of following the text-book algorithm steps, the general approach would be to first find the smaller prime q and then use it to construct the larger prime p. Primality testing such as the Miller-Rabin test is required at this phase.

In the signing phase, one can use SQ-and-MUL for modular exponentiation. Parameter r can be precomputed, as it is independent of message x. The costliest step is the inversion of k_E, for which Extended Euclidean Algorithm is used. Note that while a tempting option, the ephemeral key k_E must not be reused for security reasons.

The verification process of DSA is relatively fast.

Elliptic Curve Digital Signature Algorithm

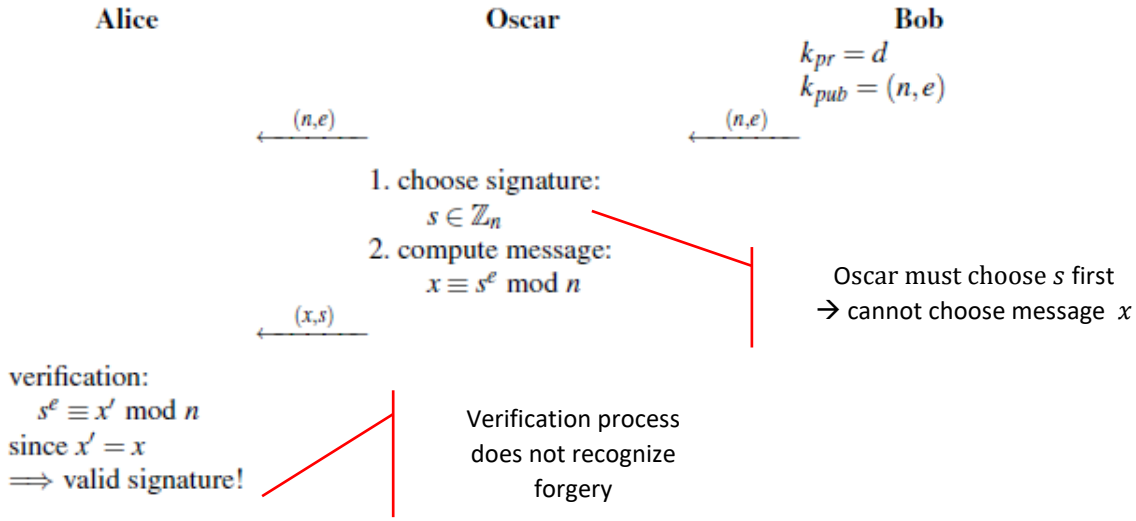
Elliptic Curve Digital Signature Algorithm (ECDSA) is a variant of DSA on elliptic curve cryptosystems. It was standardized by ANSI in 1998 and is a federal US government standard for digital signatures (DSS). ECDSA standard is defined for elliptic curves over prime fields Z_p and Galois extension fields $GF(2^m)$. With public key bit lengths 160-256, ECDSA provides security equivalent to 1024-3072 bit RSA or DL schemes. The signature size is comparable to that of DSA.

Exercise 1. Given an RSA signature scheme with the public key $(n = 9797, e = 131)$, which of the following signatures are valid?

a) $(x = 123, s = 6292)$
b) $(x = 4333, s = 4768)$
c) $(x = 4333, s = 1424)$

Exercise 2. Bob wants to sign message $x = 12345$ and send it to Alice. Bob's public key is $(n_B, e_B) = (86609{,}17)$ with set-up parameters $p = 257, q = 337$. Find Bob's private key and show the computations and steps of interaction between Alice and Bob in the RSA signature protocol.

Exercise 3. Given an RSA signature scheme with the public key $(n = 9797, e = 131)$, show how Oscar can perform an existential forgery attack by providing an example of one for the parameters of the RSA digital signature scheme.
Existential forgery for RSA signature protocol:

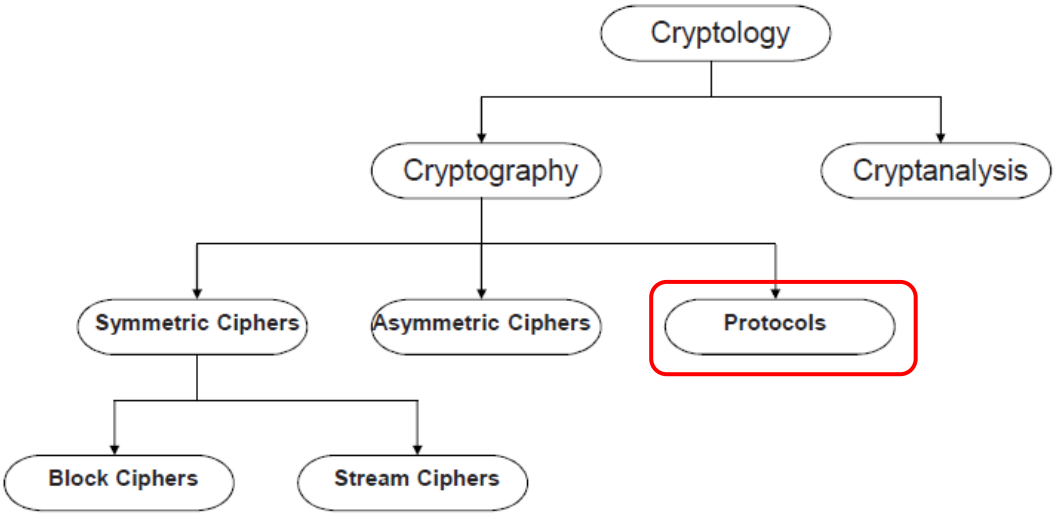


Exercise 4. What are the current recommendations for bit lengths for DSA parameters p and q? What are the corresponding signature lengths?

Exercise 5. In the example for DSA key generation, signature and verification process, verify that the chosen parameter $\alpha = 3$ has the proper order with respect to chosen parameters p and q.

Exercise 6. Given DSA parameters $p = 59, q = 29, \alpha = 3$ and Bob's private key $d = 23$, show the steps of the signing process (Bob) and verification (Alice) for hash value $h(x) = 17$ and ephemeral key $k_E = 25$.

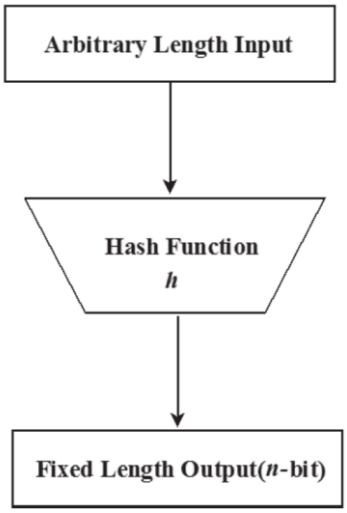
Hash Functions



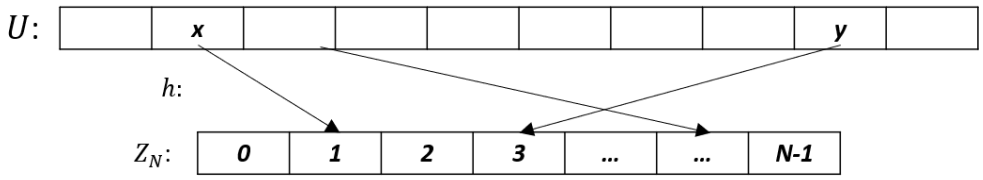
A hash function is a mathematical function that takes in an input (or message or key) and returns a fixed-size string of characters, known as a hash value (or digest or fingerprint). Hash functions are commonly used for a variety of purposes, such as

- indexing data in hash tables
- verifying data integrity
- encrypting passwords

Because the output of a hash function is deterministic, hash functions can be used to verify the authenticity of data without having to store the original data itself. For example, if two files have the same hash value, it is highly likely that they are identical.



Mathematically speaking, a hash function h is a function $h: U \to Z_N$, where U is the set of possible inputs and N is an integer. As there is an infinite number of possible inputs but just N possible outputs, we know by the *pigeonhole principle* that collisions are inevitable: $h(x) = h(y)$ for some input pairs x, y, for which $x \neq y$.



Birthday attack and collisions

The *Birthday paradox* gives an answer to how hard it is to find them: if hash output has length n, then the number of messages t we need to hash to find a collision is about

$$\sqrt{2^n} = 2^{\frac{n}{2}}$$

Hence, to achieve a security level of x bits, the hash function needs to have an output length of approximate $2x$ bits. A more accurate approximation is given by the following formula. The number of hash values t needed to find a collision with success probability λ is approximately

$$t \approx 2^{\frac{n+1}{2}} \sqrt{\ln\left(\frac{1}{1-\lambda}\right)}$$

Example. Suppose we want to find a collision for a hash function with an 80-bit output with success rate $\lambda = 80\ \%$. Then we expect to hash about

$$t \approx 2^{81/2} \sqrt{\ln\left(\frac{1}{1-0.8}\right)} \approx 2^{40.8}$$

messages.

Below are some further examples of hash values needed for different hash output lengths and two different collision probabilities.

	Hash output length				
λ	128 bit	160 bit	256 bit	384 bit	512 bit
0.5	2^{65}	2^{81}	2^{129}	2^{193}	2^{257}
0.9	2^{67}	2^{82}	2^{130}	2^{194}	2^{258}

Properties of hash functions

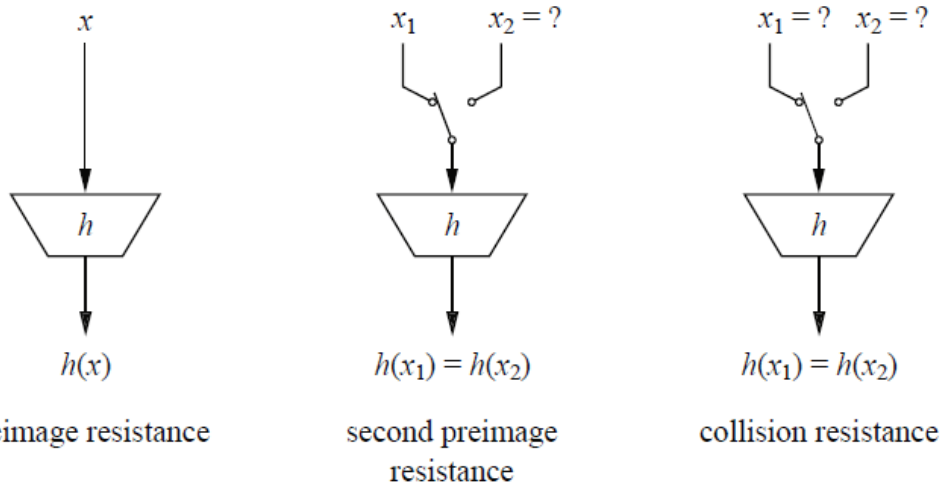
A good hash function should be designed to satisfy the following properties:

- **Determinism**. A hash function must be deterministic, i.e., for a given input, the function always produces the same output.
- **Uniformity**. A good hash function should distribute its output evenly across the range of possible hash values. This property is important for hash table implementations to ensure efficient access and retrieval of data.
- **Efficiency**. Hash functions should be efficient and fast to compute for practical use in real-world applications.

Cryptographic hash functions

Cryptographic hash functions are designed with security and integrity in mind and are used in various applications such as digital signatures, password storage, and message authentication. Some additional properties are required from hash functions that are used in cryptography:

- **Avalanche effect (sensitivity to input changes)**. Even small changes in the input should produce a significantly different output. This property is important for ensuring the integrity of data that may be subject to accidental or intentional modifications.
- **Preimage resistance (one-wayness)**. Given a hash value, it should be computationally infeasible to determine the original input. This property is important for cryptographic applications where data confidentiality and integrity are critical.
- **Second preimage resistance (weak collision resistance)**. Given an input value, it should be computationally infeasible to find another input value that would produce the same hash value.
- **Collision resistance (strong collision resistance)**. It should be computationally infeasible to find two distinct inputs that produce the same hash value. This property is important for ensuring data integrity and preventing malicious attacks on hash-based systems.



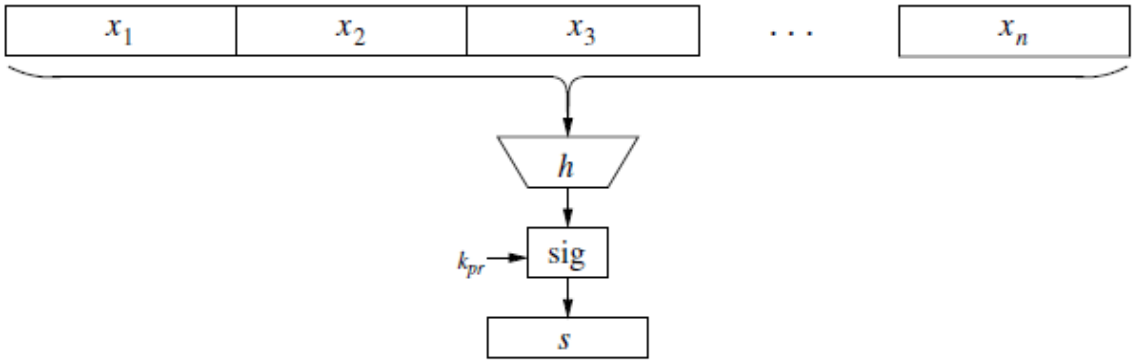
preimage resistance | second preimage resistance | collision resistance

Overall, a good hash function should be designed to satisfy as many of these properties as possible. In practice the design requires a trade-off between different properties such as security, efficiency, and ease of use. Also, the requirements depend on the specific use case and threat model. For example, some password hashing algorithms such as bcrypt or scrypt are designed to be slow and memory-intensive in order to make brute-force attacks more difficult.

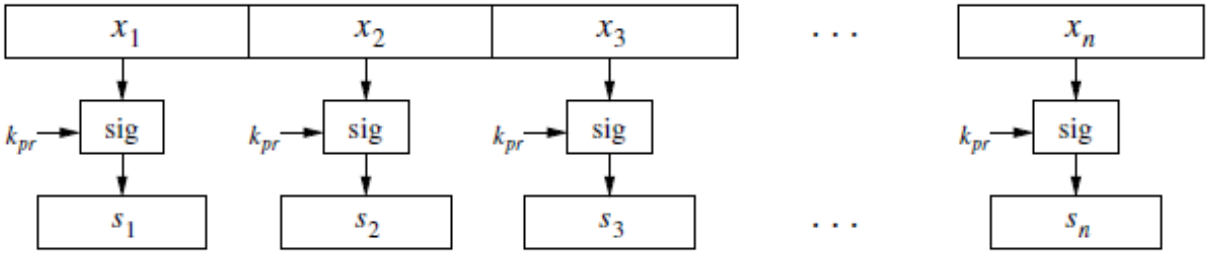
Hash functions in digital signatures

Digital signatures are typically designed to work with fixed-size messages, which can pose a challenge when dealing with long messages. There are several techniques that can be used to sign long messages, including:

1. Hash-and-sign. The simplest approach is to compute a hash value of the long message using a cryptographic hash function, and then sign the hash value using a digital signature algorithm. This approach is widely used in practice and is relatively efficient since the hash function can compress a long message into a fixed-length hash value.



2. Divide-and-conquer. Another approach is to divide the long message into smaller chunks and sign each chunk individually using the hash-and-sign approach. The resulting signatures can be concatenated to produce a signature for the entire message. This approach can be useful for very large messages or in situations where different parts of the message need to be signed by different signers.

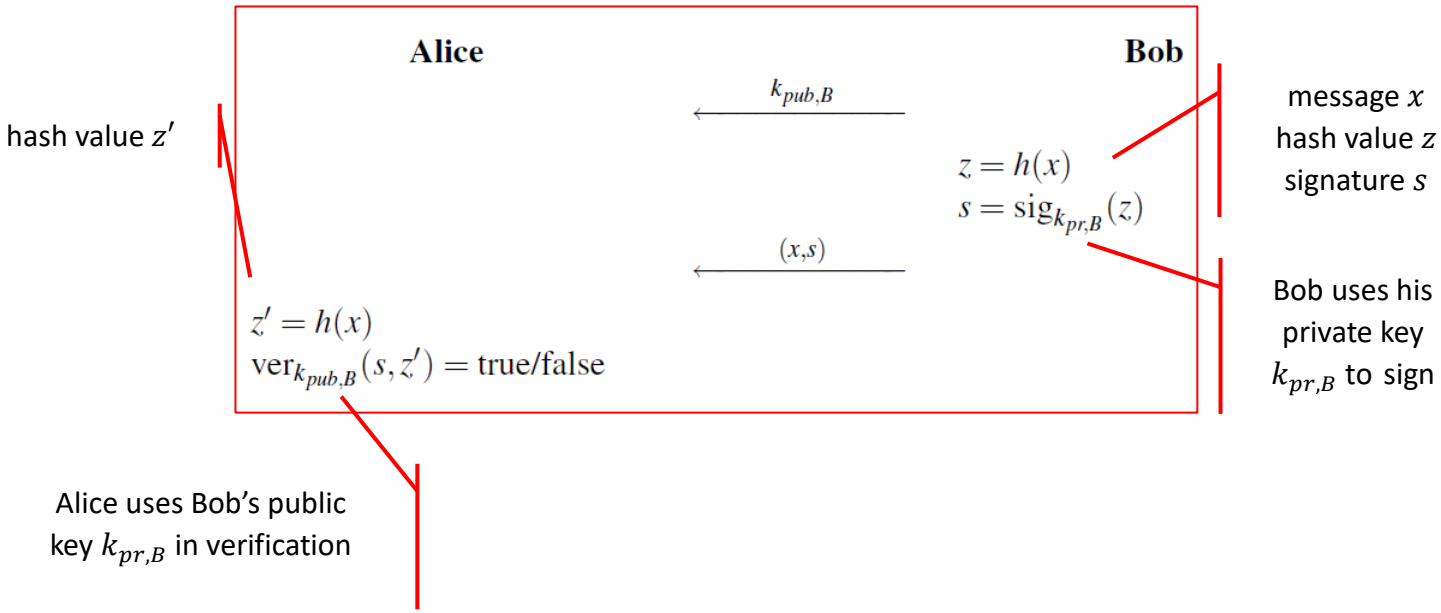


Benefits of using hash functions in digital signing

Digital signing with hash function provides same security services as digital signatures in general: message authentication, integrity, and nonrepudiation. However, using hash functions in digital signing offers several benefits such as:

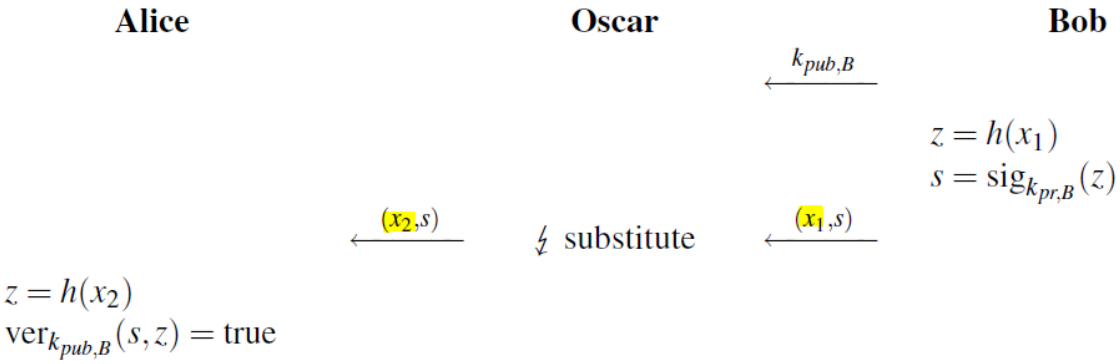
- **Enhanced efficiency**. Hash functions are designed to be computationally efficient and can quickly produce fixed-length message digests from arbitrary-length messages. This allows for fast signing and verification operations, even for very large messages.
- **Enhanced security**. Hash functions provide a one-way function that makes it infeasible to recover the original message from the message digest. This prevents attackers from forging digital signatures or tampering with signed messages.
- **Flexibility**. Hash functions can be used in a variety of digital signing schemes, including RSA, DSA, and ECDSA. This allows for compatibility with a wide range of cryptographic systems and protocols.
- **Standardization**. Hash functions are widely standardized and have undergone extensive cryptanalysis and testing to ensure their security and effectiveness. This provides a high level of confidence in the security of digital signing operations that use hash functions.
- **Versatility**. Hash functions can be used for a variety of cryptographic operations beyond digital signing, including message authentication, key derivation, and password storage. This allows for a unified approach to cryptographic operations and reduces the complexity of cryptographic systems.

Basic Protocol for Digital Signatures using Hash Function



Example (substitution attack)

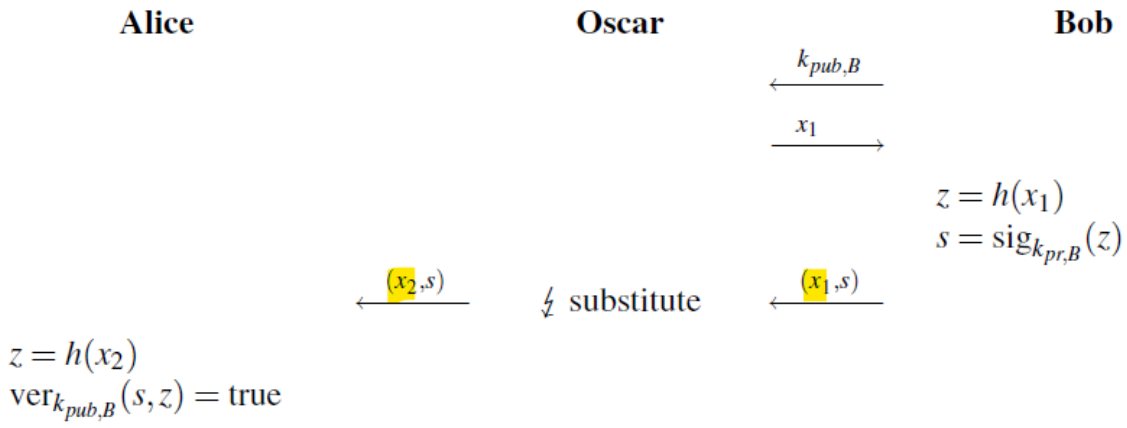
Suppose Bob is applying hash-and-sign using a hash function without second preimage resistance. It means that given message x_1, the adversary Oscar can find another message x_2 such that $h(x_1) = h(x_2)$. Then he can launch the following substitution attack:



Effectively, Oscar can replace Bob's original message x_1 by his own message x_2. He cannot choose x_2 freely, but the signature s will be valid.

Example (collision attack)

Suppose Bob is applying hash-and-sign using a hash function strong collision resistance. Then Oscar can find two messages x_1, x_2 for which $h(x_1) = h(x_2)$ and can launch the following substitution attack:



In this scenario, Oscar gets Bob to sign message x_1, but Alice will think he signed x_2.

Types of hash functions

We can categorize hash functions based on their purpose or their construction.

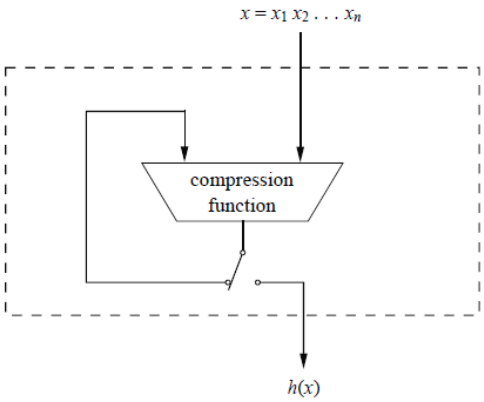
Some types of hash functions by purpose include:

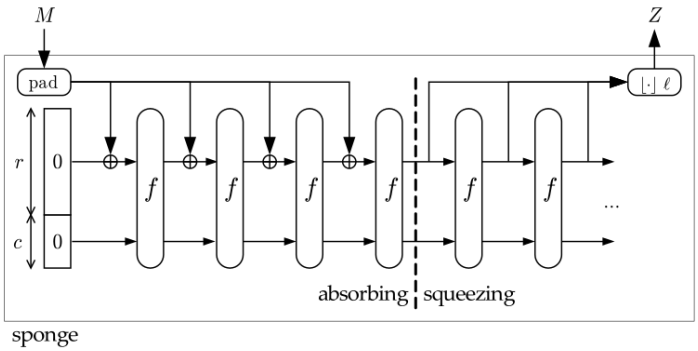
- Cryptographic hash functions: dedicated hash functions designed for use in cryptography.
- Non-cryptographic hash functions: dedicated hash functions that are not designed for use in cryptography and are typically used for data storage, indexing, and searching. Non-cryptographic hash functions are often faster than cryptographic hash functions but do not provide the same level of security guarantees.
- Message authentication code (MAC) functions: hash functions that are used to produce a message authentication code, which is a short piece of data that is used to authenticate a message. MAC functions typically use a secret key are used in applications such as network

Sometimes a cryptographic hash function loses its status as a secure hash function, as vulnerabilities are found (SHA-1, MD-5). Typically, these hash functions continue to be used for other purposes.

Some categories of hash functions by construction include:

- Merkle-Damgård construction: a construction used to build hash functions by iterating a compression function over blocks of the input message. Many popular hash functions, such as SHA-1 and SHA-2, use the Merkle-Damgård construction.
- Sponge construction: a more recent construction used to build hash functions that is based on a sponge function. The sponge construction is used in the SHA-3 family of hash functions.



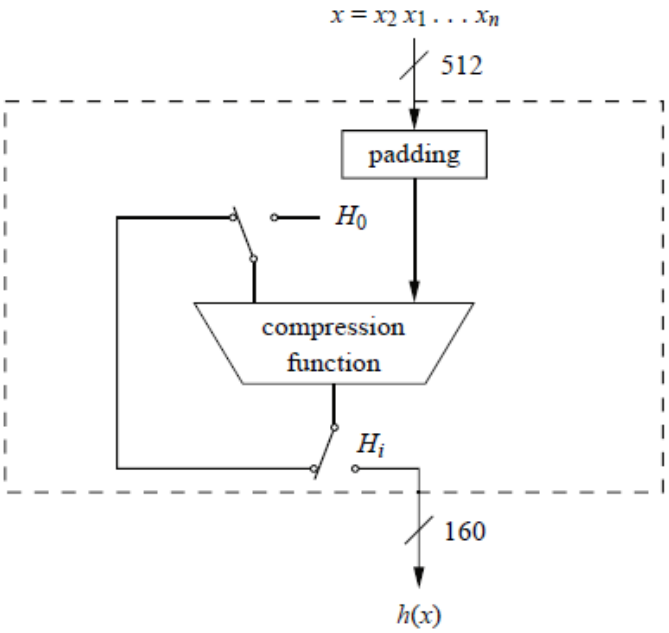


Some examples of widely used hash functions

- SHA-1 (Secure Hash Algorithm 1 is a widely used hash function that produces a 160-bit hash value. It is no longer considered secure and has been in largely deprecated.
- SHA-2 (Secure Hash Algorithm 2 is a family of hash functions that includes SHA-224, SHA-256, SHA-384, and SHA-512. SHA-256 and SHA-512 are the most commonly used members of the family and produce 256-bit and 512-bit hash values, respectively.
- SHA-3 (Secure Hash Algorithm 3) is a relatively new family of hash functions that was selected as the winner of the NIST competition in 2012. It includes SHA3-224, SHA3-256, SHA3-384, and SHA3-512 and is based on sponge construction.
- BLAKE2: a family of hash functions that is optimized for performance and security, based on the ChaCha stream cipher.
- MD5 (Message-Digest Algorithm 5): a widely used hash function that produces a 128-bit message digest. However, it is no longer considered secure.
- Whirlpool: a hash function that produces a 512-bit message digest and is designed to be resistant to both collision attacks and preimage attacks.

Example. (SHA-1)

SHA-1 (Secure Hash Algorithm 1) was published by the NIST in 1995. It uses the Merkle-Damgård construction and produces a 160-bit digest. The compression function works like a block cipher, where the input is the previous hash value H_{i-1} and the key is the message block x_i:



SHA-1 quickly became one of the most widely used hash functions in the world and was used in a variety of applications, including digital signatures, message authentication, and password storage.

A first practical collision attack on SHA-1 was demonstrated in 2005, but it continued to be widely used until in 2017 Google announced to have successfully performed a collision attack on SHA-1. After that, SHA-1 is no longer considered secure and has been replaced by more secure hash functions, such as SHA-256 or SHA-3.

Exercise 1. True or false?

a) A hash function is a mathematical function that converts a variable-sized message into a fixed-sized output.
b) A hash function can be used to encrypt a message so that it cannot be read by unauthorized parties.
c) Hash functions can be used for digital signatures, message authentication codes, and password storage.
d) A hash function can be used to generate random numbers.
e) A hash function can be used to check the integrity of a file by comparing the hash value of the original file with the hash value of the downloaded file.

Exercise 2. Shortly explain the meaning of the following terms with respect to hashing:

a) a collision
b) clustering
c) resolving collisions

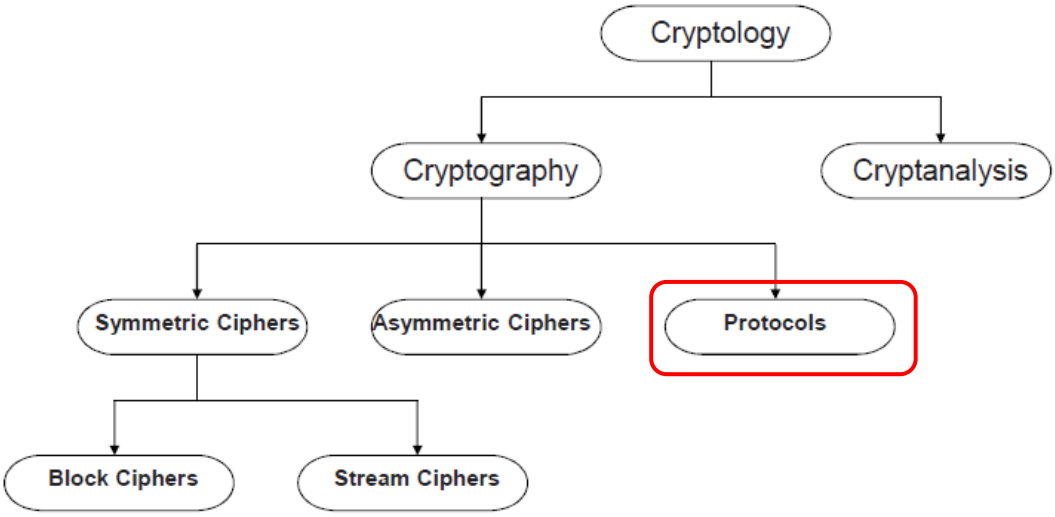
Exercise 3. There are two main approaches to resolving collisions: *chaining* and *open addressing*. Shortly explain the basic principle of each approach.

Exercise 4. Consider a hypothetical hash function with output length 160 bits. Find the approximate number of hash values needed to find a collision with success rate

a) $\lambda = 0.5$ b) $\lambda = 0.9$

Exercise 5. What are (one-way) compression functions and what is their role in building hash functions? Name some popular compression functions.

Message Authentication Codes



Message Authentication Code or MAC is a type of cryptographic primitive used to provide message integrity and authenticity. Essentially, a MAC is a piece of data that is generated using a secret key and a message, and it is appended to the message to form a new, authenticated message. The MAC serves as a cryptographic checksum of the message, allowing the recipient to verify that the message has not been tampered with or modified in transit.

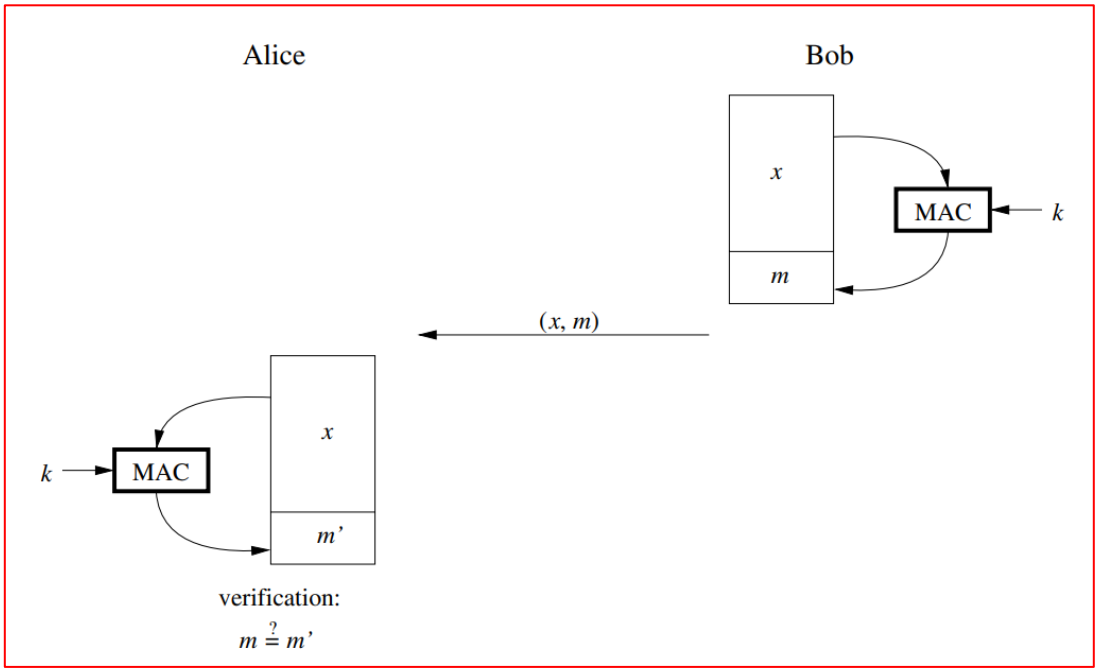
MACs are commonly used in a variety of applications, including network security, secure messaging, and digital signatures. Some common algorithms used for MAC generation are:

- HMAC (Hash-based Message Authentication Code)
- CMAC (Cipher-based Message Authentication Code)
- CBC-MAC ((Cipher Block Chaining Message Authentication Code)
- GMAC (Galois Message Authentication Code)

Properties of Message Authentication Codes

- **Cryptographic checksum:** A MAC generates a cryptographically secure authentication tag for a given message.
- **Symmetric**: MACs are based on secret symmetric keys. The signing and verifying parties must share a secret key.
- **Arbitrary message size:** MACs accept messages of arbitrary length.
- **Fixed output length:** MACs generate fixed-size authentication tags.
- **Message integrity:** MACs provide message integrity: Any manipulations of a message during transit will be detected by the receiver.
- **Message authentication:** the receiving party is assured of the origin of the message.
- **No nonrepudiation:** MACs are based on symmetric principles, hence they do not provide nonrepudiation.

Principle of the MAC calculation and verification



Alice and Bob must agree on a shared secret key k beforehand. Bob will compute the MAC m from the message x and send both over to Alice. As the set-up is symmetric, the verification means simply that Alice will repeat the computation of the MAC and compare the result m' to the original m.

MACs versus digital signatures

It's worth noting that MACs serve a similar purpose as digital signatures but are not quite the same. A MAC is generated by applying a *secret key* and a message to a cryptographic algorithm to produce a fixed-size output, which is then appended to the message. When the recipient receives the message, they can *re-compute* the MAC using the same secret key and message.

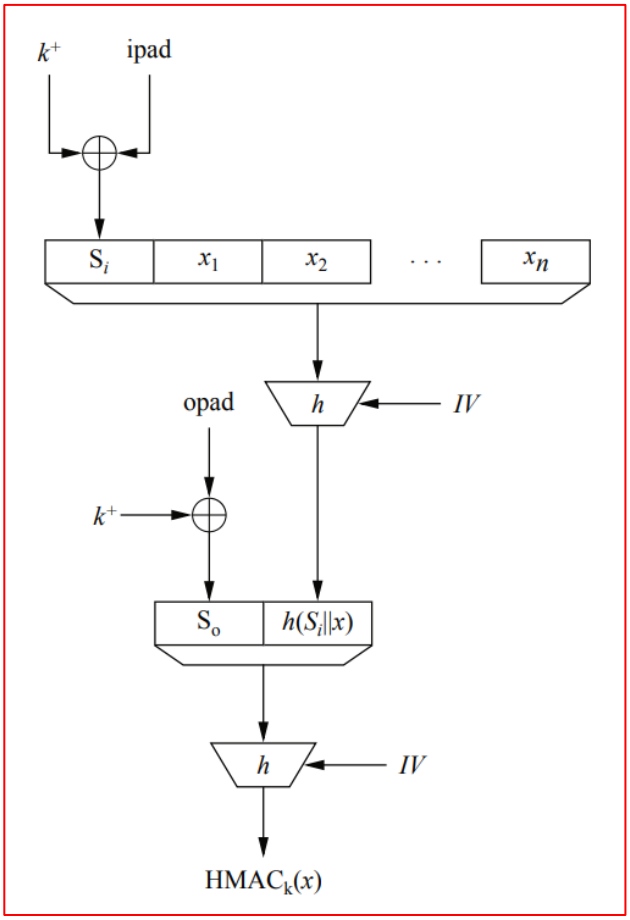
A digital signature is generated using a *private key* and a message. The private key is used to sign the message, producing a digital signature that is unique to the signer and the message. When the recipient receives the message and the digital signature, they can use the signer's public key to *verify* the signature and confirm that the message was signed by the person claiming to be the signer.

While both MACs and digital signatures provide message integrity and authenticity, digital signatures also provide non-repudiation, as the signer cannot deny having signed the message. MACs do not provide non-repudiation, because anyone who knows the secret key can generate a valid MAC for any message.

MACs are often used as a *component* of digital signature schemes. For example, the HMAC algorithm can be used to generate a MAC for a message, which can then be signed using a digital signature scheme like RSA or DSA.

Hash-based Message Authentication Codes

The term HMAC can be used generally to refer to a type of MAC that uses a cryptographic hash function to provide message authentication. However, in the following we refer specifically to the HMAC construction proposed in 1996 by Mihir Bellare, Ran Canetti and Hugo Krawczyk. This construction is the standard definition of HMAC that is considered to be a secure and efficient method for message authentication. It is widely used in various cryptographic libraries and protocols, such as SSL/TLS, IPsec, and SSH.



k^+: expansion of the key k with zeros to match the input block width b of the hash function

ipad = $00110110,\ldots,00110110$ with length b

$x_1, x_2, \ldots, x_n$: message blocks

h: hash function

IV: initial value (random but public)

opad = $01011100,\ldots,01011100$ with length b

The whole HMAC construction can be expressed as: $HMAC_k(x) = h[(k^x \oplus opad)||h[(k^+ \oplus ipad)||x]]$

Note that message x, which can be very long, is only hashed once in the first (inner) hash function. The second (outer) hash consists of only two blocks, namely the padded key and the output of the first hash. This construction gives great computational efficiency.

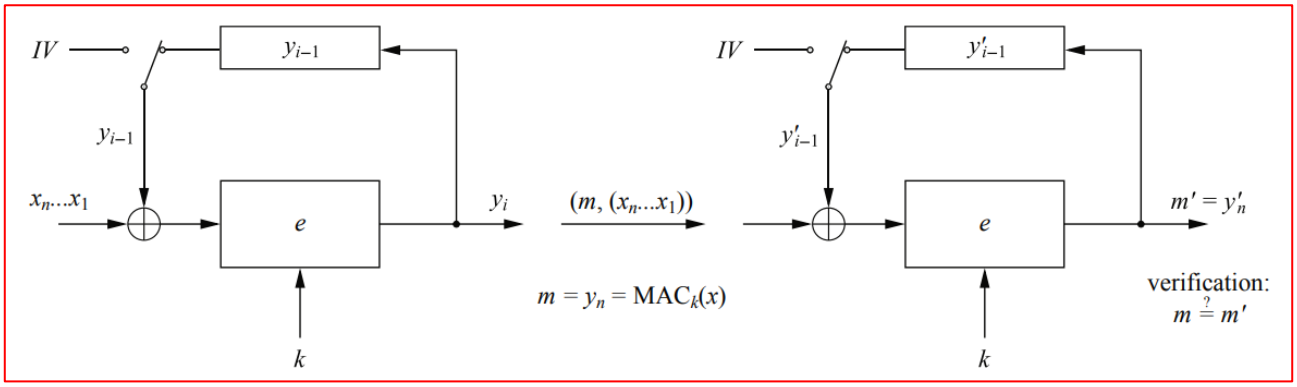
Security of the HMAC construction

There is a security proof for HMAC under certain assumptions. The proof was first presented in the original paper by Bellare et al. and has since been refined and extended in subsequent research. It shows that if the hash function used in HMAC satisfies certain properties (e.g., collision resistance, preimage resistance, and output unpredictability), then HMAC is also secure under a chosen message attack.

The security of HMAC is reduced to the security of the underlying pseudorandom function, which is then shown to be equivalent to the security of the hash function. There have been no known attacks against HMAC that can break its security under the assumptions made in the security proof.

MACs from Block Ciphers

An alternative method to construct MACs is to use block ciphers. The most popular approach is to use a block cipher such as AES in cipher block chaining (CBC) mode. This scheme is referred to as CBC-MAC.



Again, the message x is divided into blocks $x_1, \ldots, x_n$. The first iteration of the MAC algorithm is

$$y_1 = e_k(x_1 \oplus IV)$$

The subsequent rounds yield output blocks

$$y_i = e_k(x_i \oplus y_{i-1})$$

for $i = 2, \ldots, n$. The final MAC of the message x is the output y_n of the last round:

$$m = MAC_k(x) = y_n$$

Note that unlike in CBC mode encryption, the encrypted output blocks $y_1, \ldots, y_{n-1}$ are not transmitted to the receiver but only used for computing the final MAC value. Also, the verification is different from CBC decryption, as the verifier does not actually decrypt anything but just recompute the MAC value.

Security of the CBC-MAC construction

CBC-MAC is relatively simple to implement and efficient in terms of processing time and memory usage. However, CBC-MAC is vulnerable to length extension attacks, which can allow an attacker to append additional data to a valid message and still generate a valid MAC. For this reason, CBC-MAC should not be used for messages of arbitrary length but rather for messages of a fixed length (or for messages that are padded to a fixed length).

Exercise 1. True or false?

a) MACs are used to provide message authentication and integrity.

b) MACs can be used to provide confidentiality to a message.

c) MACs can be used to verify the identity of the sender of a message to the receiver.

d) MACs can be used to verify the identity of the sender of a message to a third party.

Exercise 2. Explain shortly:

a) the difference between a MAC and a digital signature

b) the difference between a hash function and a MAC

Exercise 3. Consider a MAC protocol where the sender performs the following operation:

$$y = e_{k_1}[x||h(k_2||x)]$$

here x is the message, h is the hash function, e is a symmetric encryption algorithm, and "||" means concatenation. k_1 and k_2 are secret keys shared by the sender and the receiver. Sender will transmit y to receiver. Give a step-by-step description of what the receiver will do upon receiving y to verify the MAC.

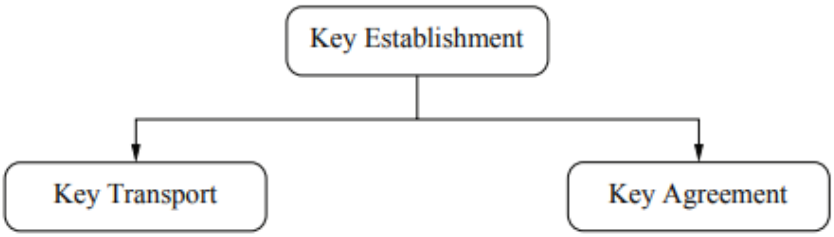
Exercise 4. How is CMAC similar to CBC-MAC, and how do they differ from each other?

Exercise 5. Explain shortly, what is GMAC and how does it compare to other types of MACs in terms of efficiency and security.

Key Establishment

Key establishment in cryptology refers to the process of securely and reliably establishing cryptographic keys between two or more parties in a communication system. This process often involves the use of mathematical algorithms and protocols that ensure that the keys are generated and exchanged securely. Key establishment is a critical step in establishing a secure communication channel between parties.

The methods of key establishment can be classified in key agreement (or *key exchange*) and key transport. Both methods have their advantages and disadvantages, and the choice of which method to use depends on the specific requirements of the communication system.



Key agreement

In key agreement, the parties collaborate to agree on a shared secret key, without exchanging the key directly. Instead, they exchange information that enables them to derive the same secret key. One example of a key agreement protocol is the Diffie-Hellman protocol.

Some examples of situations where key agreement is preferred over key transport because it does not require the transmission of the secret key over an insecure communication channel:

- Virtual private networks (VPNs) that allow remote users to securely connect to a corporate network over the internet.
- Secure web browsing using the Transport Layer Security (TLS) protocol, where the client and the server need to agree on a secret key to encrypt and decrypt data exchanged during the session.
- Secure messaging applications that require end-to-end encryption, where two or more users need to agree on a shared secret key to encrypt and decrypt messages.

Key transport

In key transport, one party generates and securely transmits the secret key to the other party. Key transport is often preferred over key agreement in scenarios where the two parties do not have a pre-existing relationship or need to exchange messages infrequently.

Public key encryption is often used in combination with symmetric-key encryption to provide secure key transport. The sender can use the recipient's public key to encrypt a randomly generated symmetric key, and then use that symmetric key to encrypt the message. The recipient would then use their private key to decrypt the symmetric key and use it to decrypt the message.

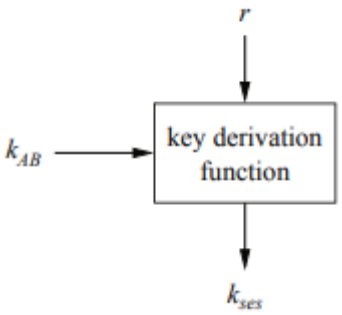
Key freshness and key derivation

Key freshness refers to the property of a cryptographic key that ensures that the key has not been used before and is not vulnerable to attacks due to previous use. Key freshness is critical to the security of cryptographic systems. For example, if a key is reused or compromised, an attacker can use the key to decrypt previously encrypted messages or impersonate one of the parties in a communication.

There are various techniques to ensure key freshness:

- *Key rotation* involves periodically replacing the old key with a new one to ensure that the key is not used for an extended period.
- *Key revocation* is the process of invalidating a compromised or suspected key to prevent its further use.
- *Session key establishment* involves creating a new key for each communication session to ensure that the key is fresh and has not been used before. Such keys are called *session keys* or *ephemeral keys*.

Generating fresh keys all over again from scratch is computationally intensive, especially in the case of public-key algorithms. Another approach to update keys is key derivation: the process of deriving one or more cryptographic keys from a shared secret or a master key already established between the parties. The principal idea is to use a key derivation function (KDF) to derive multiple keys.



Typically, the main components of a KDF are

1. Key material: The input to the KDF, which can be a shared secret, a password etc.
2. Salt: A random value that is used to prevent pre-computation attacks and to increase entropy.
3. Extraction function: A function that converts the input key material into a uniform random value. This is typically achieved using a cryptographic hash function or a pseudorandom function.
4. Expansion function: A function that expands the uniform random value obtained from the extraction function into a key of the desired length. This is typically achieved using a pseudorandom function or a block cipher.
5. Key Generation: The final step in the KDF, which generates the key from the expanded value.

There are several different types of KDFs:

- Hash-based KDFs use a cryptographic hash function to derive the key,
- Cipher-based KDFs use a block cipher to derive the key, and
- HMAC-based KDFs use the Hash-based Message Authentication Code (HMAC) construction to derive the key.

The most common KDFs are the hash-based *Key Derivation Function 1* (KDF1) and *Key Derivation Function 2* (KDF2), which are both defined in the ISO/IEC 18033-2 standard. Another widely used KDF is the HMAC-based *Password-Based Key Derivation Function 2* (PBKDF2), used in many security protocols, and specified in the PKCS#5 and RFC 2898 standards. However, PBKDF2 has been shown to be vulnerable to certain types of attacks and more advanced KDFs, such as *the Argon2* KDF, are recommended for password-based key derivation in modern cryptographic applications. Another alternative for PBKDF2 is *Scrypt* which is a *memory-hard KDF*, meaning that it is resistant to brute force attacks. Scrypt is commonly used in cryptocurrency applications.

Key establishment using symmetric-key techniques

Symmetric-key techniques for key establishment typically involve a centralized key distribution center that is responsible for generating and distributing shared secret keys to multiple users. The basic process of symmetric-key key establishment for multiple users is as follows:

1. Key Generation: The key distribution center generates a secret key that will be shared among the users.
2. Key Distribution: The key distribution center securely sends a copy of the secret key to each user. This can be done using a secure communication channel or by physically delivering the key.
3. Key Verification: Each user verifies that the received key matches the expected key. This can be done using a cryptographic hash function or by exchanging messages to confirm the key.
4. Key Use: The users can now use the shared secret key to encrypt and decrypt messages sent between them.

In another symmetric-key technique for key establishment, the KDC manages *unique* secret keys for each user. In this approach, the KDC is responsible for generating and distributing secret keys to users on an as-needed basis. This approach is often used in Kerberos, a widely used network authentication protocol. The basic process of key establishment is as follows:

1. User Authentication: The user authenticates to the KDC, typically by providing a username and password.
2. Session Key Request: The user requests a session key from the KDC, providing the identity of the intended communication partner.
3. Key Generation: The KDC generates a new secret key for the user, specifically for the requested communication partner.
4. Key Distribution: The KDC securely sends the session key to the user.
5. Key Use: The user can now use the session key to encrypt and decrypt messages sent to and from the intended communication partner.

This approach enables the KDC to control and monitor the use of secret keys, which can help detect and prevent unauthorized access. On the other hand, if the KDC is compromised, all secret keys managed by the KDC may be compromised as well.

Symmetric-key key establishment can be efficient and scalable for a small number of users, but it becomes impractical as the number of users increases. As a result, in large-scale systems, asymmetric-key techniques are typically used for secure key establishment.

Key establishment using asymmetric-key techniques

Asymmetric-key techniques for key establishment involve the use of public key cryptography and key agreement protocols to enable secure communication between two or more parties without the need for a shared secret key. The basic process of asymmetric-key key establishment is as follows:

1. Public Key Exchange: Each party generates a public-private key pair, and securely exchanges their public keys with each other.
2. Key Agreement: Using the exchanged public keys, the parties run a key agreement protocol to derive a shared secret key that will be used for encryption and decryption of messages.
3. Key Verification: Each party verifies that the shared secret key has been derived correctly and securely. This can be done using a cryptographic hash function or by exchanging messages to confirm the key.
4. Key Use: The parties can now use the shared secret key to encrypt and decrypt messages sent between them.

Two examples of widely used asymmetric-key key agreement protocols are the Diffie-Hellman key exchange and the RSA key exchange. In Diffie-Hellman, each party generates a random number and uses it to derive a public key, which is exchanged with the other party. Each party generates a secret key and uses it together with the other party's public key to derive a shared secret key. This key can then be used for encryption and decryption of messages. In RSA protocol, one party generates a random secret key and encrypts it using the other party's public key. The encrypted key is then sent to the other party, who uses their private key to decrypt the key and obtain the shared secret key.

Asymmetric-key techniques provide several advantages over symmetric-key techniques for key establishment. First, they do not require a pre-existing shared secret key, which can make them more flexible and scalable for large-scale systems. Second, they can enable secure communication even if one party's secret key is compromised, because the public keys can still be used to establish a new shared secret key. However, asymmetric-key techniques are typically slower and more computationally intensive than symmetric-key techniques.

Public-Key Infrastructures and Certificate Authorities

The secure exchange of public keys in Step 1 of asymmetric-key key establishment is crucial to ensure the security of the subsequent key agreement process. It is important to ensure the authenticity and integrity of the exchanged public keys to prevent any *Man-in-the-Middle attacks*, where the attacker replaces the public keys sent out by the participants with their own malicious keys. So, even though public-key schemes do not require a secure channel, they do require key authentication for the distribution of the public keys.

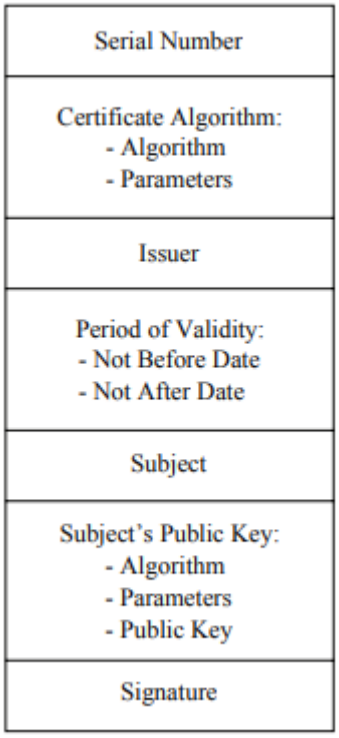
A Public Key Infrastructure (PKI) is a system of digital certificates, certificate authorities (CA), and other registration authorities that verify and authenticate the identity of parties and bind public keys to those identities. After verifying the identity of the certificate holder and their ownership of the corresponding private key, the CA issues a digital certificate and signs it with its own private key. When a user receives a digital certificate, they can use the CA's public key to verify the signature and authenticate the identity of the certificate holder. They can then use the public key contained in the certificate to establish a secure connection or exchange encrypted data with the certificate holder.

Example: X.509 certificates

X.509 is a widely recognized and trusted standard for digital certificates, and many certificate authorities and PKI systems use X.509 certificates to establish secure communication and data exchange between parties. X.509 is commonly used in conjunction with other security protocols and standards, such as

- SSL/TLS for secure web communication,
- S/MIME for secure email communication, and
- IPSec for secure network communication.

X.509 digital certificates are also used in many other security applications, including digital signatures, secure code signing, and secure remote access.



Certificate Algorithm: which signature algorithm is being used, e.g., RSA with SHA-1 or ECDSA with SHA-2

Issuer: Which company / organization generated the certificate

Period of Validity: the certificates, and thus the public keys, are only valid for limited time

Subject: This field contains identifying information such as names of people or organizations.

Subject's Public Key: The public key that is to be protected by the certificate. In addition to the public key, the algorithm (e.g., Diffie–Hellman) and the algorithm parameters are stored.

Signature: The signature over all other fields of the certificate.

There are many different entities that act as certificate authorities. *National CAs* are typically established by governments to provide digital certificates for government services and applications, as well as to regulate the digital certification processes within their jurisdiction.

The most widely recognized and trusted CAs (DigiCert, GlobalSign, Symantec etc.) are *commercial*. They are private companies that operate as for-profit; however they are subject to regulation and oversight by various industry and government bodies. It means that they are required to follow certain standards and best practices to ensure that they are trustworthy and reliable.

Chain of certificate authorities

A chain of certificate authorities (CA) is a hierarchical structure of trusted third-party organizations in a PKI system. The chain of trust is established by linking the digital certificates issued by one CA to those issued by another CA in a hierarchical structure.

At the top of the hierarchy is the *root CA*, a well-known and trusted entity that is responsible for issuing and managing digital certificates for the entire chain of CAs. The root CA's digital certificate is self-signed and contains its public key, which is used to verify the digital signatures of all other certificates in the chain.

Beneath the root CA are *intermediate CAs*, which are responsible for issuing digital certificates to other organizations or entities. The certificate issued by an intermediate CA contains its own public key, as well as the public key of the root CA, which is used to verify the signature of the intermediate CA's digital certificate.

The final link in the chain is the end-entity digital certificate, which is issued by the intermediate CA to an organization or individual. The end-entity digital certificate contains the public key of the organization or individual, as well as the public key of the intermediate CA and the root CA, which are used to verify the digital signature of the end-entity digital certificate.

Exercise 1. True or false?

a) In symmetric-key cryptography, key establishment typically involves the use of a key distribution center (KDC) to manage the distribution of secret keys to users.

b) In asymmetric-key cryptography, key establishment typically involves the use of public-key cryptography to securely exchange public keys between two or more parties.

c) Key establishment is not necessary for secure communication in a cryptographic system.

d) The key derivation function (KDF) is used to generate new cryptographic keys from a shared secret or other input.

Exercise 2. True or false?

a) Password-Based Key Derivation Function 2 (PBKDF2) is a type of KDF that is commonly used to derive cryptographic keys from passwords.

b) In a public key infrastructure (PKI), the root CA issues certificates directly to end-users.

c) The Diffie-Hellman key exchange is an example of a key agreement protocol used in asymmetric-key cryptography.

d) In a public-key infrastructure (PKI), a digital certificate contains a public key and other identifying information about the owner of the certificate.

Exercise 3. Explain the Man-in-the-Middle attack against the Diffie-Hellman Key Exchange protocol, if the public keys are not authenticated.

Exercise 4. Give some basic information about the Finnish national CA.

Exercise 5. Find the meaning of the following terms and shortly explain their purpose in key establishment:

a) key wrap algorithm
b) perfect forward secrecy