APCM Notes

Filippo De Grandi
September, 2025

Contents

1. Hashing

One-way Function

Function that is easy to compute on every input, but hard to invert given the image of a random input.

The existence of such one-way functions is still an open conjecture.

1.1. Hash functions

A hash function is secure if:

- it is **strictly one-way**, it lets us compute the digest of a message, but does not let us figure out a message for a given digest (even very short).
- it is **computationallu infeasible to find collisions**, i.e. construct messages whose digest would equal a specified value.

1.2. MERKLE

The input message is partitioned into t number of bit blocks, each of size n bits.
If necessary, the final block is padded so that it is of the same length as others.

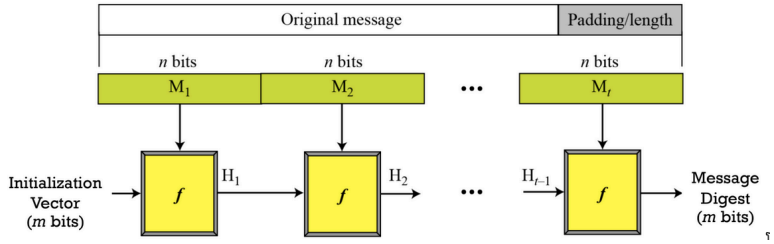

Figure 1: Merkle

Each stage of the Merkle structure takes two inputs

- the n-bit block of the input message meant for that stage
- the m-bit output of the previous stage

For the m-bit input, the first stage is supplied with a special m-bit pattern called the **Initialization Vector** (IV).
The function f that processes the two inputs, one n bits long and the other m bits long, to produce an m-*bit* output is usually called **compression function**.

- This is so since $n > m$, i.e. the output of the function f is shorter than the length of the input message segment

The compression function f may involve multiple rounds of processing of the two inputs to produce the output. The precise nature of f depends on what hash algorithm is being implemented

1.3. SHA (Secure Hash Algo)

SHA refers to a family of NIST-approved cryptographic hash functions.

	Message size (bits)	Block size (bits)	Word size (bits)	Message digest (size)	Security (bits)
SHA-1	<2^64	512	32	160	80
SHA-2 SHA-256	<2^64	512	32	256	128
SHA-2 SHA-384	<2^128	1,024	64	384	192
SHA-2 SHA-512	<2^128	1,024	64	512	256

Figure 2: SHA hash functions

The last column refers to how many messages would have to be generated before two can be found with the same digest with a probability of 0.5.

For a secure hash algorithm that has no security holes and produces n-bit digests, one would need to come up with $2^{\frac{n}{2}}$ messages to discover a collision with a probability of 0.5.
This is why the entries in the last column are half in size compared to the entries in the **Message Digest Size**.

1.3.1. SHA-1

SHA-1 is a successor to MD5 that was a widely used hash function

- There still exist many legacy applications that use MD5 for calculating digests
- This despite the fact that its usage has been deprecated as it is considered insecure

SHA-1 was cracked theoretically in the year 2005 by two different research groups. In one of these two demonstrations, it was shown how it is possible to come up with a collision for SHA-1 within a space of size 2^69 only.
This is much less than the security level of 2^80 associated with this hash function.

> **Note**
>
> A 2013 attack breaks MD5 collision resistance in 2^18 time.
> This attack runs in less than a second on a regular computer. In 2017, SHA-1 was broken in practice... Full details at https://shattered.io/

1.3.2. Compression function

- W_t contains a word of 4 bytes derived from the message block of 512 bits, i.e. 64 bytes.
- K_t contains words of 4 bytes from a defined array of 64 constants.

The blue components perform the operations:

$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$

$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\varepsilon_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$

$$\varepsilon_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

The red operator is addition modulo 2^{32}. The compression function is invoked in 64 rounds. In the first round, the registers A, B, ..., H are initialized with constants

> **Important**
>
> This function contributes to performing **diffusion**.

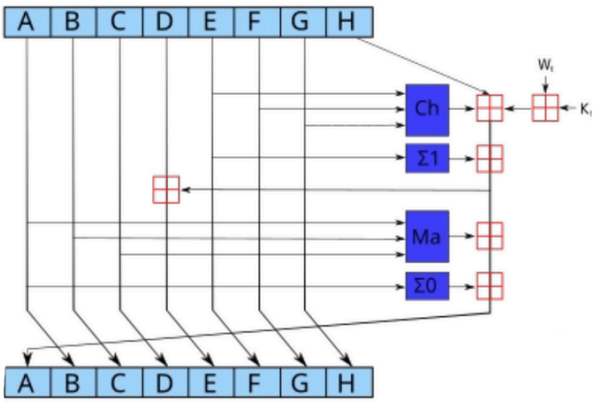

Figure 3: Compression Function in SHA-1

1.4. Application of Hash Functions

Compare possibly huge messages very efficiently by

- compute small digest
- compare digests in place of original messages

Under which assumptions this method can work?
How does this relate to one property of the CIA triad?
Is it enough to produce digests to check that the property is preserved or do you need to protect them in some way?
If you need protection, which kind of protection? With the protection, do you obtain something more if the protection is properly done?

2. Symmetric

An attacker does not extract any meaning from the ciphertext $\rightarrow$ random sequence of bits

complex to understand the plaintex that generated it, supposed to produce a digest of fixed size impossible to extract information from

- non invertibility property
- only assumed if one does not know a secret -> the key
- with the key -> very easy to invert the process and extract the plaintext

ciphertext must be inverted under the assumption that one has the key

- Key must be kept secret, cryptography is useless if the key is not protected
- anything should be public about the cipher -> NO security through obscurity

with several encryption techniques -> not only one point of failure (POF) do not use the same key for everything

Pros:

- same key for enc and dec
- much faster that public key
- can be efficiently implemented directly in hardware

Cons:

- less secure -> key exchange is difficult under certain use cases over the internet
- establishing a secure channel over the internet is difficult

Used to exchange large quantities of data over the internet Assumption: someone, magically distributed the symmetric keys between the participants -> `Delicate assumption`

$$E(m, k) = c$$

$$D(c, k) = m$$

It's a `One Way Function` under the assumption that you know a certain secret Requirements underlying this kind of symmetric key encryption algorithm

- Block Ciphers
- Stream Ciphers

exploit the same ideas they refer to the assumption under which the data that is taken as input is

2.1. Block Ciphers

Takes large sets of bits Widely used nowadays that requires heavy-duty enc and dec

Symmetric criptography is based on the assumption of *using a very fast bit operation* Design an operation that transforms one bit of the plaintext into a bit of the ciphertext in a very efficient way

XOR it's the fastest operation from the hardware point of view, and XOR can be reversed by just XORing again

I also need a stream of bits that represents the **key** This stream **must be random**

The ciphertext can become predictable if the key is not random enough

The plaintext we are interested in exchanging usually contains different structuring inside, e.g. encrypting a transaction with different fields:

$$\text{Key is reused no more than: } \frac{m}{n} > 1 \text{ times}$$

Moreover, if the same plaintext is encrypted again it should not generate the same ciphertext, distrucping regularities

Block Ciphers are a **random permutation** of bits
Can be generated using a `RNG`

One description of Block Cipher can be:
- **Encryption**: Daemon checks in the left-hand column to see if it has a record of plaintext. If not, it asks the random number generator to generate a ciphertext that does not yet appear in the right-hand column, and then writes down the plaintext/ciphertext pair in the codebook and returns the ciphertext. If it does find a record, it returns the corresponding ciphertext from the right hand column
- **Decryption**: same as above but with columns swapped

> **Note**
>
> The **key** here is the table itself (or *codebook*), with the inconvenience that the table is large.

Each possible input block is one of 2^N integers, and each integer can specify an output N-bit block. The codebook will be of size $N * 2^N$ -> the **encryption key** for the ideal block will be of this size (huge).

2.1.1. Actual implementation
The following structure is the same of basically every block cipher.

keyed family of pseudorandom permutations $\rightarrow$ for each key, a single permutation independent of all the others Choose 2^K permutations uniformly at random from set of all $(2^N)!$ permutations, with K the length of the keys under consideration

> **Important**
>
> For a block cipher to be good, an attacker should not be able to recover the key even using multiple plaintext-ciphertext pairs

Key scheduling and rounds aim to implement **diffusion** and **confusion**.

Organize in rounds of computations the transformation of the plaintext. This transformation is the same in each round.
As output, a candidate ciphertext is produced. The more rounds, the better it is because it increases confusion and diffusion.

The idea is to use a key much shorter than the number of bits in the block cipher, by using a different key at each round.

2.1.2. Confusion and Diffusion
Key design principles:

- **Diffusion**: if a plaintext bit changes, several ciphertext bits should change.
  Basic demand on a block cipher, typically achieved by using linear transformations
- **Confusion**: Every bit of the ciphertext should depend on several bits in the key

2.1.3. Substitution-Permutation Network

The simplest way to achieve both diffusion and confusion.
Takes a block of plaintext and key as inputs and produces ciphertext block by applying several alternating **rounds** of:

- **S-boxes** and **P-boxes** that transform blocks of input bits into output bits
- commond for the transformations to be efficient operations in hardware (xor and bitwise rotation)

Key introduced in each round derived from **key schedule algorithm**

Decryption done by reversing the process

S-Box

Substitutes small block of bits by another block of bits.
Substitution should be one-to-one → ensure invertibility (hence decryption).

Not just a permutation of the bits:
A good S-Box will have the property that changing one input bit will change half of the output bits, and each output bit will depend on every input bit

P-Box

It's a permutation of all the bits.
Takes the outputs of all the S-boxes of one round, permutes the bits and feeds them into the S-boxes of the next round

A good P-box has the property that **the output bits of any S-box are distributed to as many S-box inputs as possible**

The result of this S-box - P-box transformation is **XORed** with a Key derived from the original encryption key

2.1.4. DES

Adopted by NIST in 1977 → now considered broken.
Uses a **56-bit encryption key**.
Uses the **Feistel cipher structure** with 16 rounds of processing.

Feistel structure

Uses same basic algorithm for bot henc and dec

Consists of multiple rounds of processing of the plaintext, with each round consisting of a **substitution** step **followed by a permutation** step.

In each round:

- right half of the block, R, goes through unchanged
- left half, L, goes through an operation that depends on R and enc key.
- operation carried out on the left L is referred to as the **Fesitel function F**
- the permutation step consists of **swapping** the modified **L and R**

This permutation is one of the easiest to implement in hardware -> **Circular Shift Register**

> **Note**
>
> For DES, number *n* of rounds is 16

2.1.5. Fesitel Function

The 32-bit right half of the 64-bit input data is expanded into a 48-bit block

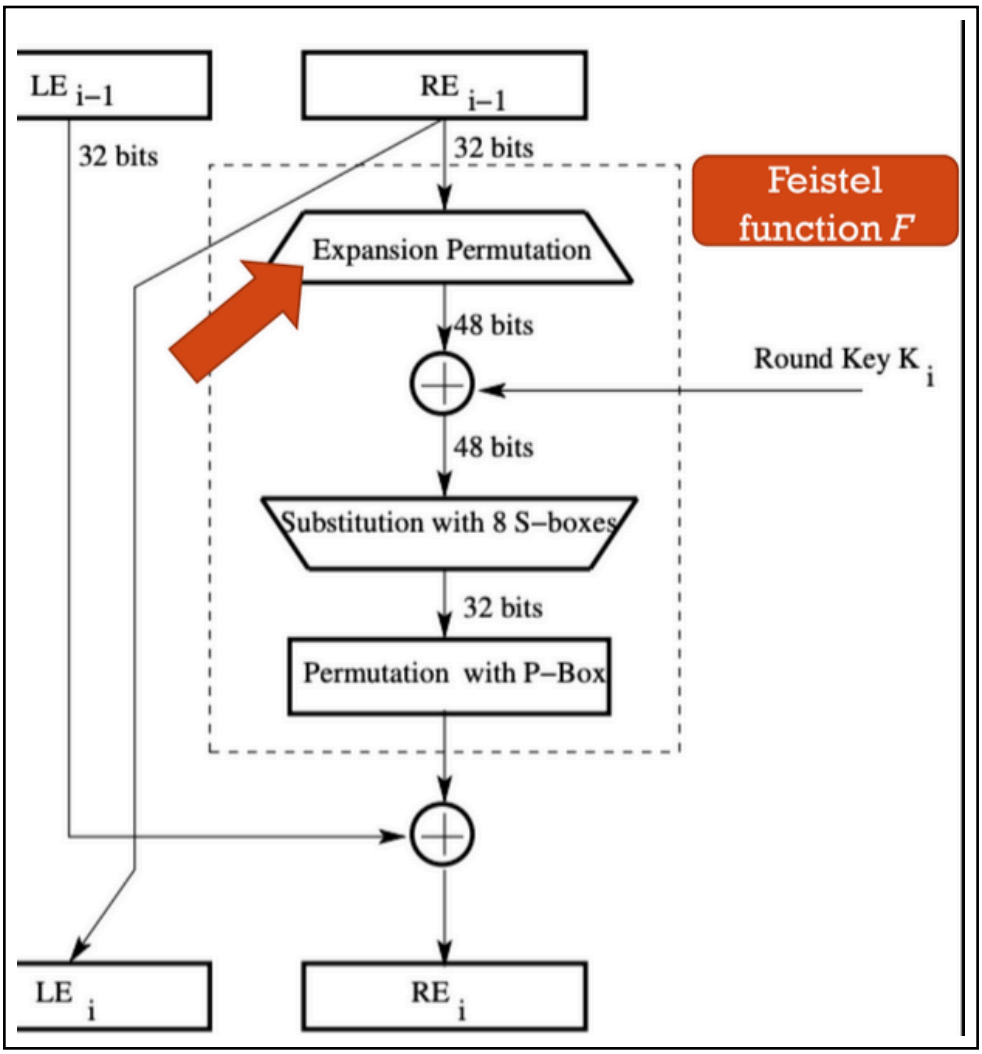

Figure 4: Feistel Function schema

Expansion permutation step

1. divide the 32-bit block into eight 4-bit words
2. attach an additional bit on the left to each 4-bit word that is the last bit of the previous 4-bit word
3. attach an additional bit to the right of each 4-bit word that is the beginning bit of the next 4-bit word
4. The 56-bit key is divided into two halves, each half shifted separately, and the combined 56-bit key permuted/contracted
5. The 48 bits of the expanded output produced by the Expansion permutation step are xor-ed with the round key
   - This is called key mixing
6. The output produced is broken into eight 6- bit words

7. Each six-bit word goes through a substitution step; its replacement is a 4-bit word
   - The substitution is carried out with an S-box
   - So after all the substitutions, we again end up with a 32-bit word

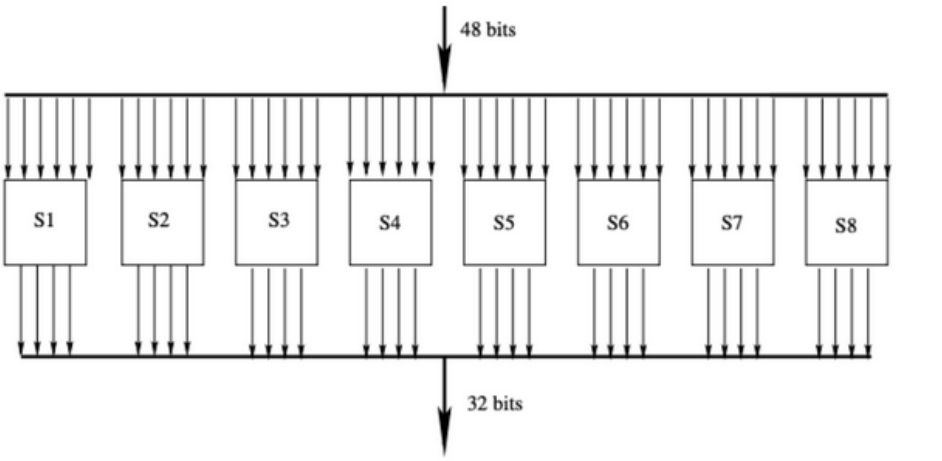

Figure 5: Substitution with S-boxes

> **Tip**
>
> Why exactly those numbers in the **Feistel S-Boxes**?
> It was an iterative process that ends up in those specific numbers, experience of the designers.

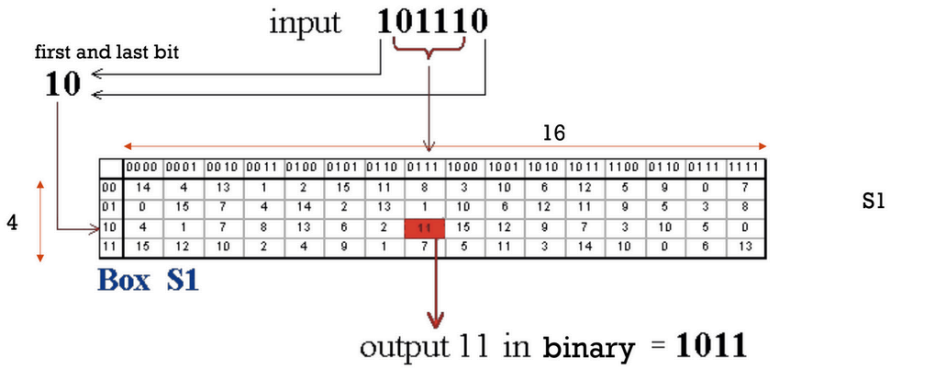

Figure 6: Indexing the S-boxes

2.1.5.1. Key Scheduling

The 56-bit encryption key is represented by 8 bytes, with the least significant bit of each byte used as a parity bit

- The relevant 56 bits are subject to a permutation before any round keys are generated (see Figure 7)
- This is called Key Permutation 1

The bit indexing is based on using the range 0-63 for addressing the bit positions in an 8-byte bit pattern in which the last bit of each byte is used as a parity bit.

- Each row has only 7 positions

The table specifies that:

- the 0th bit of the output will be the 56th bit of the input (in a 64 bit representation of the 56-bit encryption key)
- the 1st bit of the output will be the 48th bit of the input,
- ... and so on, until we have for the 55th bit of the output the 3rd bit of the input

Key Permutation 1						
56	48	40	32	24	16	8
0	57	49	41	33	25	17
9	1	58	50	42	34	26
18	10	2	59	51	43	35
62	54	46	38	30	22	14
6	61	53	45	37	29	21
13	5	60	52	44	36	28
20	12	4	27	19	11	3

Figure 7: Initial permutation

At the beginning of each round:
- divide the 56 key bits into two 28 bit halves
- circularly shift to the left each half by one or two bits, depending on the round, according to a table.

> **Note**
>
> This is to ensure that each bit of the original encryption key is used in 14 of 16 rounds

For generating the round key, we glue together the two halves and apply a **56 bit to 48 bit contracting permutation** (this is referred to as Permutation Choice 2) to the joined bit pattern
- The resulting 48 bits constitute the round key

Key permutation 2
- The bit addressing now spans the 0 through 55 index values for the 56 bit key. Out of this index range, the permutation shown above retains only 48 bits for the round key. Since there are only six rows and there are 8 positions in each row, the output will consist of 48 bits.
- As for the permutation tables above, what is shown on Figure 8 is not a table, in the sense that the rows and the columns do not carry any special and separate meanings
- The permutation order for the bits is given by reading the entries shown from the upper left corner to the lower right corner

Key Permutation 2							
13	16	10	23	0	4	2	27
14	5	20	9	22	18	11	3
25	7	15	6	26	19	12	1
40	51	30	36	46	54	29	39
50	44	32	47	43	48	38	55
33	52	45	41	49	35	28	31

Figure 8: Key Permutation 2

Diffusion & Confusion in DES

Avalanche effect

- if one changes one of the 64 bits in the input data block, it affects **34 bits of the ciphertext block**
- if one changes one bit of the encryption key, on the average **35 bits of the ciphertext are affected**

> **Note**
>
> 56-bit encryption key means **key space** of size $2^{56} \approx 7.2 * 10^{16}$.
>
> DES was broken in 1999.

2.1.6. After DES: AES

2.1.7. Trusted Computing Base

The trusted computing base (TCB) of a computer system is the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system. By contrast, parts of a computer system that lie outside the TCB must not be able to misbehave in a way that would leak any more privileges than are granted to them in accordance to the system's security policy.

2.2. Stream Ciphers

Takes bits or bytes as input
Were used for VOIP or Video