Cryptography

Corso di Laurea Magistrale in Informatica

Constructing Pseudorandom Objects and Hash Functions, in Practice

Ugo Dal Lago

ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

Inria informatiques mathématiques

Academic Year 2023-2024

Constructing Pseudorandom Objects and Hash Functions

- In the last two chapters, we have seen how pseudorandom objects and hash functions are the **primitives** from which we can construct (and prove secure) ciphers and MACs.

Constructing Pseudorandom Objects and Hash Functions

- In the last two chapters, we have seen how pseudorandom objects and hash functions are the **primitives** from which we can construct (and prove secure) ciphers and MACs.
- We have not discussed yet how to *construct* pseudorandom generators and functions, nor collision-resistant hash functions.

Constructing Pseudorandom Objects and Hash Functions

- In the last two chapters, we have seen how pseudorandom objects and hash functions are the **primitives** from which we can construct (and prove secure) ciphers and MACs.
- We have not discussed yet how to *construct* pseudorandom generators and functions, nor collision-resistant hash functions.
- There are two ways of doing this:
  1. **Proving** how such objects exist in a theoretical sense on the basis of assumptions about the difficulty of some problems. We will deal with this in the next chapter.
  2. **Describing** primitives that, although not provably pseudorandom (or collision-resistant), seem to have good features. We will deal with this in this chapter.

So What?

- If the concrete primitives we will discuss in this chapter *cannot* be shown to have the pseudorandomness (or collision-resistance) properties we are looking for, **what is the point** of studying them?
  - Obviously, we can consider the possibility that they satisfy these properties *as an assumption*!

So What?

- If the concrete primitives we will discuss in this chapter *cannot* be shown to have the pseudorandomness (or collision-resistance) properties we are looking for, **what is the point** of studying them?
  - Obviously, we can consider the possibility that they satisfy these properties *as an assumption*!
- Obviously, assuming that (for example) AES is secure is **essentially different** from assuming that factorization is a difficult problem.
  - The latter is a problem that has been studied for hundreds of years, while the former is a scheme that is only 20 years old.

So What?

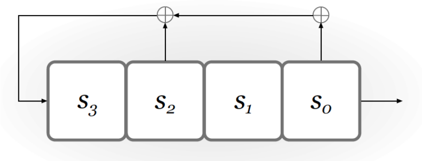
- If the concrete primitives we will discuss in this chapter *cannot* be shown to have the pseudorandomness (or collision-resistance) properties we are looking for, **what is the point** of studying them?
  - Obviously, we can consider the possibility that they satisfy these properties *as an assumption*!
- Obviously, assuming that (for example) AES is secure is **essentially different** from assuming that factorization is a difficult problem.
  - The latter is a problem that has been studied for hundreds of years, while the former is a scheme that is only 20 years old.
- Along the way, we will deal not only with concrete primitives, but also with *models* for such primitives.
  - This will allow us to give **necessary** (but not sufficient) conditions for their security.

Constructing Stream Ciphers

- Practically speaking, stream ciphers are made up of a pair of algorithms ($Init$, $GetBits$) where:
  - The $Init$ algorithm initializes the internal state, starting from a key (and optionally from an initialization vector IV)
  - The $GetBits$ algorithm outputs a single bit while simultaneously modifying the internal state.
- By iteratively applying $GetBits$ to the state obtained by $Init$ we can obtain arbitrarily long streams of bits.
  - We would like the algorithms obtained to be as similar as possible to (variable-length) pseudorandom generators.

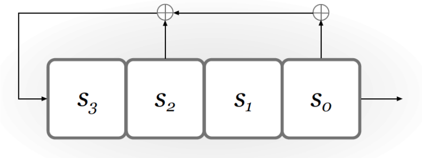
Linear Feedback Shift Registers

- They are a particular type of stream cipher, in which:
  - The state consists of n bits $s_{n-1}s_{n-2}\cdots s_1 s_0$.
  - *GetBits* computes a bit b as the XOR of *some* of the bits in the state, outputs the value of the least significant bit s_0, shifts the other bits to the right, and initializes s_{n-1} to b.



Linear Feedback Shift Registers

- They are a particular type of stream cipher, in which:
  - The state consists of n bits $s_{n-1}s_{n-2}\cdots s_1s_0$.
  - *GetBits* computes a bit b as the XOR of *some* of the bits in the state, outputs the value of the least significant bit s_0, shifts the other bits to the right, and initializes s_{n-1} to b.



- The key is given by the initial state and the feedback coefficients.
- Linear feedback shift registers should *not* be considered as secure, but have inspired the development of concrete stream ciphers.

Linear Feedback Shift Registers

- But *why* are linear feedback registers (also called LFSRs) to be considered insecure?

Linear Feedback Shift Registers

- But *why* are linear feedback registers (also called LFSRs) to be considered insecure?
- First of all, it should be noted that:
  - The "future" behaviour of an LFSR is completely determined by its state.
  - There are 2^n states, and an LSFR that cycles through the $2^n - 1$ states other than $0 \cdots 0$ is called *maximum length LSFR*.
  - Maximum length LSFRs have very good statistical properties, and are the only interesting ones from a cryptographic perspective.

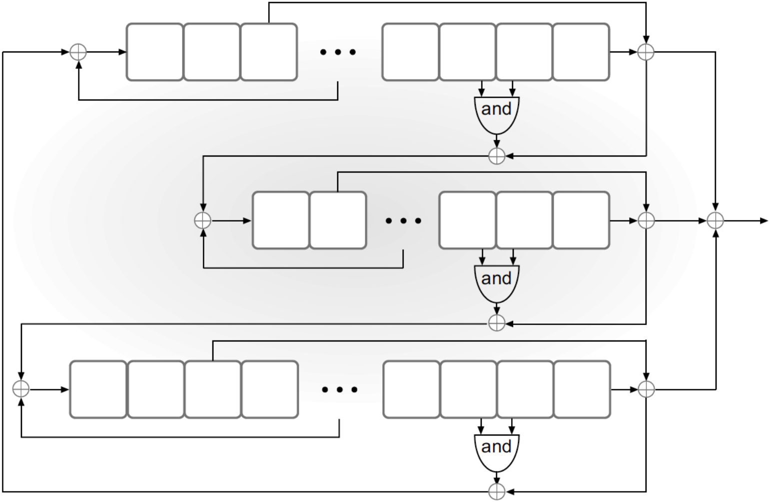
Linear Feedback Shift Registers

- But *why* are linear feedback registers (also called LFSRs) to be considered insecure?
- First of all, it should be noted that:
  - The "future" behaviour of an LFSR is completely determined by its state.
  - There are 2^n states, and an LSFR that cycles through the $2^n - 1$ states other than $0 \cdots 0$ is called *maximum length LSFR*.
  - Maximum length LSFRs have very good statistical properties, and are the only interesting ones from a cryptographic perspective.
- The **weakness** of LSFRs comes from the following attack:
  - From the first $2n$ outputs $y_1 \cdots y_{2n}$ it is possible to totally reconstruct the initial state (i.e. $y_1 \cdots y_n$) and the feedback coefficients (by means of a system of linear equations, which can be solved efficiently).

Linear Feedback Shift Registers

- But *why* are linear feedback registers (also called LFSRs) to be considered insecure?
- First of all, it should be noted that:
  - The "future" behaviour of an LFSR is completely determined by its state.
  - There are 2^n states, and an LSFR that cycles through the $2^n - 1$ states other than $0 \cdots 0$ is called *maximum length LSFR*.
  - Maximum length LSFRs have very good statistical properties, and are the only interesting ones from a cryptographic perspective.
- The **weakness** of LSFRs comes from the following attack:
  - From the first $2n$ outputs $y_1 \cdots y_{2n}$ it is possible to totally reconstruct the initial state (i.e. $y_1 \cdots y_n$) and the feedback coefficients (by means of a system of linear equations, which can be solved efficiently).
- The weakness of LSFRs can be overcome by introducing nonlinearity on the feedback and/or output side.

Trivium



RC4

- RC4 is a stream cipher introduced by Ronald Rivest in 1987 and designed to be implemented by SW rather than HW.
- Its internal state consists of a bytes vector of length equal to 256, which represents a permutation of the set $\{0, \ldots, 255\}$, together with two values $i, j \in \{0, \ldots, 255\}$.

RC4

- RC4 is a stream cipher introduced by Ronald Rivest in 1987 and designed to be implemented by SW rather than HW.
- Its internal state consists of a bytes vector of length equal to 256, which represents a permutation of the set $\{0, \ldots, 255\}$, together with two values $i, j \in \{0, \ldots, 255\}$.

Init **algorithm for RC4**

Input: 16-byte key k
Output: Initial state (S, i, j)
(Note: All addition is done modulo 256)

for $i = 0$ to 255:
  $S[i] := i$
  $k[i] := k[i \bmod 16]$
$j := 0$
for $i = 0$ to 255:
  $j := j + S[i] + k[i]$
  Swap $S[i]$ and $S[j]$
$i := 0, \quad j := 0$
return (S, i, j)

GetBits **algorithm for RC4**

Input: Current state (S, i, j)
Output: Output byte y; updated state (S, i, j)
(Note: All addition is done modulo 256)

$i := i + 1$
$j := j + S[i]$
Swap $S[i]$ and $S[j]$
$t := S[i] + S[j]$
$y := S[t]$
return $(S, i, j), y$

RC4

- RC4 is a stream cipher introduced by Ronald Rivest in 1987 and designed to be implemented by SW rather than HW.
- Its internal state consists of a bytes vector of length equal to 256, which represents a permutation of the set $\{0, \ldots, 255\}$, together with two values $i, j \in \{0, \ldots, 255\}$.

Init **algorithm for RC4**

Input: 16-byte key k
Output: Initial state (S, i, j)
(Note: All addition is done modulo 256)

for $i = 0$ to 255:
  $S[i] := i$
  $k[i] := k[i \bmod 16]$
$j := 0$
for $i = 0$ to 255:
  $j := j + S[i] + k[i]$
  Swap $S[i]$ and $S[j]$
$i := 0, \quad j := 0$
return (S, i, j)

GetBits **algorithm for RC4**

Input: Current state (S, i, j)
Output: Output byte y; updated state (S, i, j)
(Note: All addition is done modulo 256)

$i := i + 1$
$j := j + S[i]$
Swap $S[i]$ and $S[j]$
$t := S[i] + S[j]$
$y := S[t]$
return $(S, i, j), y$

- There are **statistical attacks**, that do not allow RC4 to be considered secure.

Block Ciphers, in Practice

- In this part of the course, since we are interested in primitives and algorithms, we will deal with functions of the form $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$, and we will try to construct pseudorandom ones. However, we observe that:
  - n **is not necessarily equal** to ℓ.
  - n and ℓ are **constants**, so it is necessary to use the concrete approach.

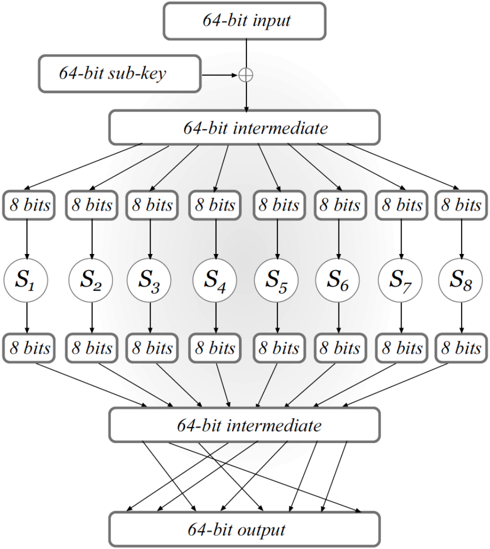
Block Ciphers, in Practice

- In this part of the course, since we are interested in primitives and algorithms, we will deal with functions of the form $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$, and we will try to construct pseudorandom ones. However, we observe that:
  - n **is not necessarily equal** to ℓ.
  - n and ℓ are **constants**, so it is necessary to use the concrete approach.
- Attacks against block ciphers are among the *four types* we already know about (ciphertext-only, known-ciphertext, chosen-plaintext, chosen-ciphertext), and most of the times are *key-recovery* attacks.
  - For the Kerchoffs' principle, determining the key is sufficient to force the cipher: the cipher becomes easily distinguishable from a random function.

Substitution-Permutation Networks

- The substitution-permutation networks (SPN) are simply one of many models from which concrete block ciphers can be constructed.
- To the input message is applied, iteratively, a transformation, that is obtained as a composition of:
  - **The mixing with the key** (or part of it);
  - **The so-called S-BOX**;
  - **A permutation**.
- The model is due to Shannon, whose many contributions to cryptography include this one.

Substitution-Permutation Networks



Two Guidelines

- In the construction of SPNs, there are no principles that guarantee the security of the cipher obtained.

Two Guidelines

- In the construction of SPNs, there are no principles that guarantee the security of the cipher obtained.
- On the other hand, there are guidelines that should always be followed.

1. The S-BOXes must be **invertible**.
   - If this were not the case, there would be no hope of constructing permutations, which is very often necessary.
2. The **avalanche effect** must be guaranteed.
   - A change of one bit of an S-BOX input must propagate to at least two bits of the output.
   - In this way, if the number of rounds is high enough, a change of one bit in the message will potentially affect *all bits* of the output.
   - The latter is a feature that is not only useful, but necessary for pseudorandomness.

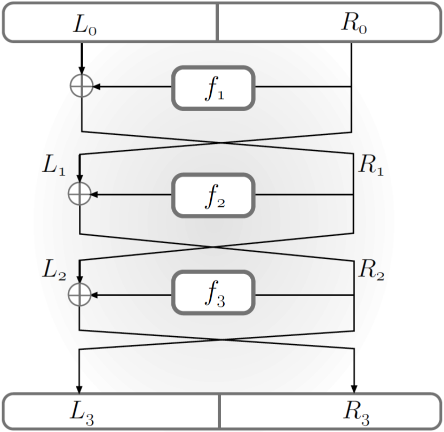
Feistel Networks

- Feistel Networks are a model very similar to SPNs, but with some interesting peculiarities.
- In each round, the underlying m message is split into two sub-messages of equal length m_L and m_R. After the round, the new message $p = p_L \cdot p_R$ will be such that

  $$p_L = m_R \qquad\qquad p_R = f(m_R) \oplus m_L$$

  where f is a function that typically depends on the key, and it is called **Mengler function**.
- This way of constructing the rounds has the interesting effect of allowing f to be invertible.
- Obviously, the Mengler function must somehow depend on the key, otherwise there is no hope of having any form of security.

Feistel Networks



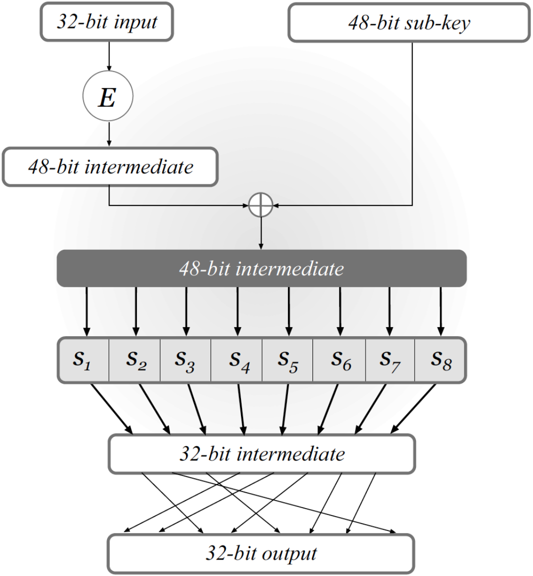
DES

- DES is one of the most important block ciphers, especially from a historical point of view.
- Keys are 56 bits long, while messages are 64 bits long, in other words DES can be seen as a function

$$F_{DES} : \{0,1\}^{56} \times \{0,1\}^{64} \rightarrow \{0,1\}^{64}.$$

- DES is structured as a 16-round Feistel network, in which a particular Mengler function f takes in input different portions of the key depending on the round.
  - More precisely, for each $i \in \{1, \ldots, 16\}$ there exists a subset $KS_i \subseteq \{1, \ldots, 16\}$ of bits of the key that will contribute to the computation of f at round i.
  - By convention, we call this "Mengler function" f_i, where i is the round.
- The function f itself has a structure very similar to that of an SPN.

DES Mengler Functions



DES S-BOXes

- The core of DES are its eight S-BOXes.

DES S-BOXes

- The core of DES are its eight S-BOXes.
- Each of them is a function $\{0,1\}^6 \to \{0,1\}^4$
  - There is therefore no hope that these functions can be invertible.

DES S-BOXes

- The core of DES are its eight S-BOXes.
- Each of them is a function $\{0,1\}^6 \rightarrow \{0,1\}^4$
  - There is therefore no hope that these functions can be invertible.
- Each possible configuration $s \in \{0,1\}^4$ is the image of *exactly* four input configurations.
- Is that all? Absolutely not.
  - S-BOXes are also designed to ensure the **avalanche effect**, as they are *the only ones responsible.*
  - The people who designed DES, a team of cryptographers later succeeded by the NSA, were concerned with making S-BOXes resistant to **differential cryptography**.

DES Security

- When considering versions of DES in which the number of rounds is small (up to 3), very simple cryptanalytic attacks are possible.
  - In a single round, from each pair (m, c) we go back to the input and the output of f_1, and from these to very few values for each portion of the key.
  - The way Feistel networks are constructed make it very easy to attack DES even when the number of rounds is 2.

DES Security

- When considering versions of DES in which the number of rounds is small (up to 3), very simple cryptanalytic attacks are possible.
  - In a single round, from each pair (m, c) we go back to the input and the output of f_1, and from these to very few values for each portion of the key.
  - The way Feistel networks are constructed make it very easy to attack DES even when the number of rounds is 2.
- If, on the other hand, we consider DES on 16 rounds, the best performing attack *from a concrete point of view* is still the brute-force attack.
  - Moreover, the number of operations required to carry out this key-recovery attack (of the order of 2^{56}) is within the reach of HPC systems.

DES Security

- When considering versions of DES in which the number of rounds is small (up to 3), very simple cryptanalytic attacks are possible.
  - In a single round, from each pair (m, c) we go back to the input and the output of f_1, and from these to very few values for each portion of the key.
  - The way Feistel networks are constructed make it very easy to attack DES even when the number of rounds is 2.
- If, on the other hand, we consider DES on 16 rounds, the best performing attack *from a concrete point of view* is still the brute-force attack.
  - Moreover, the number of operations required to carry out this key-recovery attack (of the order of 2^{56}) is within the reach of HPC systems.
- The techniques of **linear cryptanalysis** and **differential cryptanalysis** are both applicable, but they are experimentally less efficient than brute-force techniques.

Increasing the Key Length

- We could consider the so-called *double encryption*, i.e. $DDES_{k_1,k_2}(m) = DES_{k_2}(DES_{k_1}(m))$.
  - However, this block cipher is vulnerable to "meet-in-the-middle" attacks, which have complexity of the order of 2^{56}.

Increasing the Key Length

- We could consider the so-called *double encryption*, i.e. $DDES_{k_1,k_2}(m) = DES_{k_2}(DES_{k_1}(m))$.
  - However, this block cipher is vulnerable to "meet-in-the-middle" attacks, which have complexity of the order of 2^{56}.
- Similarly, we could consider the *triple* encryption, i.e.

$$
\begin{aligned}
TDES^1_{k_1,k_2,k_3}(m) &= DES_{k_3}(DES^{-1}_{k_2}(DES_{k_1}(m))) \\
TDES^2_{k_1,k_2}(m) &= DES_{k_1}(DES^{-1}_{k_2}(DES_{k_1}(m)))
\end{aligned}
$$

  Both in $TDES^1$ and in $TDES^2$, the brute-force attack has complexity 2^{112}.

Increasing the Key Length

- We could consider the so-called *double encryption*, i.e. $DDES_{k_1,k_2}(m) = DES_{k_2}(DES_{k_1}(m))$.
  - However, this block cipher is vulnerable to "meet-in-the-middle" attacks, which have complexity of the order of 2^{56}.
- Similarly, we could consider the *triple* encryption, i.e.

$$TDES^1_{k_1,k_2,k_3}(m) = DES_{k_3}(DES^{-1}_{k_2}(DES_{k_1}(m)))$$
$$TDES^2_{k_1,k_2}(m) = DES_{k_1}(DES^{-1}_{k_2}(DES_{k_1}(m)))$$

  Both in $TDES^1$ and in $TDES^2$, the brute-force attack has complexity 2^{112}.
- Then there is the possibility of constructing **Triple DES**, which is a standard, but which, despite its name, has keys that are 112 bits long.

AES

- AES has been selected among the 15 participants in a selection process (organised by NIST) for replacing DES.

AES

- AES has been selected among the 15 participants in a selection process (organised by NIST) for replacing DES.
- This is an SPN with a message length of 128 bits and a key length of 128, 192 or 256 bits.

AES

- AES has been selected among the 15 participants in a selection process (organised by NIST) for replacing DES.
- This is an SPN with a message length of 128 bits and a key length of 128, 192 or 256 bits.
- In each round, the state is seen as a 4×4 matrix of bytes that is initially equal to the message ($4 \cdot 4 \cdot 8 = 128$).

AES

- AES has been selected among the 15 participants in a selection process (organised by NIST) for replacing DES.
- This is an SPN with a message length of 128 bits and a key length of 128, 192 or 256 bits.
- In each round, the state is seen as a 4×4 matrix of bytes that is initially equal to the message ($4 \cdot 4 \cdot 8 = 128$).
- Each round involves the application of four transformations:
  1. **AddRoundKey**: a 128-bit long sub-key is put in XOR with the state.
  2. **SubBytes**: each byte in the matrix is replaced with the byte obtained by applying a fixed S-BOX.
  3. **ShiftRows**: each row of the matrix is shifted to the left of a variable number of positions.
  4. **MixColumns**: an invertible linear transformation is applied to each column of the matrix.
- The best performing concrete attack remains the brute-force one.

Constructing Hash Functions

- From a practical point of view, hash functions can be seen as functions of the form $H : \{0,1\}^* \to \{0,1\}^\ell$.
- In most cases, such functions are constructed from a **compression function**. $C : \{0,1\}^{n+\ell} \to \{0,1\}^\ell$ and applying to the latter a transformation similar to the Merkle-Damgård transformation.
- How to construct F? Are there standard models for constructing compression functions?

The Davies-Meyer Construction

- Given a block cipher $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$, a natural way to construct a compression function from F is to define $C : \{0,1\}^{\ell+n} \to \{0,1\}^\ell$ as follows

$$C(k||m) = F_k(m) \oplus m$$

The Davies-Meyer Construction

- Given a block cipher $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$, a natural way to construct a compression function from F is to define $C : \{0,1\}^{\ell+n} \to \{0,1\}^\ell$ as follows

$$C(k||m) = F_k(m) \oplus m$$

- But what can we say about the security of this compression functions' construction?

Theorem

If F is modelled as an ideal cipher, then the Davies-Meyer construction induces a collision-resistant hash function in a concrete sense: each attacker of complexity q cannot find collisions with probability greater than $\frac{q^2}{2^\ell}$

The Davies-Meyer Construction

- Given a block cipher $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$, a natural way to construct a compression function from F is to define $C : \{0,1\}^{\ell+n} \to \{0,1\}^\ell$ as follows

$$C(k||m) = F_k(m) \oplus m$$

- But what can we say about the security of this compression functions' construction?

Theorem

If F is modelled as an ideal cipher, then the Davies-Meyer construction induces a collision-resistant hash function in a concrete sense: each attacker of complexity q cannot find collisions with probability greater than $\frac{q^2}{2^\ell}$

The requirement of being an ideal cipher is even stronger than that of being a strong pseudorandom permutation.

Some Concrete Hash Functions

- **MD5**
  - The output is 128 bits long.
  - To be considered as *not secure*: finding a collision takes a few minutes.

Some Concrete Hash Functions

- **MD5**
  - The output is 128 bits long.
  - To be considered as *not secure*: finding a collision takes a few minutes.
- **SHA1** e **SHA2**
  - In *SHA1*, the output is 160 bits long, but there are attacks that require less than 2^{80} function calls.
  - In *SHA2*, the output can be 256 or 512 bits long (we speak of *SHA256* and *SHA512*, respectively).
  - All these functions are obtained through the Davies-Meyer construction from suitably defined block ciphers.

Some Concrete Hash Functions

- **MD5**
  - The output is 128 bits long.
  - To be considered as *not secure*: finding a collision takes a few minutes.
- **SHA1** e **SHA2**
  - In *SHA1*, the output is 160 bits long, but there are attacks that require less than 2^{80} function calls.
  - In *SHA2*, the output can be 256 or 512 bits long (we speak of *SHA256* and *SHA512*, respectively).
  - All these functions are obtained through the Davies-Meyer construction from suitably defined block ciphers.
- **SHA3**
  - Similarly to what happened a few years earlier with AES, NIST selected a new hash function in 2012, which was called *SHA3*
  - *SHA3* supports outputs of 256 and 512 bit length and is based on radically different principles from the constructions previously mentioned.