# Table of contents



# Cryptographic Hash Functions and MACs

## One-Way Functions (OWF)

A function $f$ is one-way if $f(x) = y$ is easy to compute, but finding $x$ from $y$ is computationally impossible for the majority of values.

**Examples:**

- Squares modulo composite: $f(x) = x^2 \bmod n$ with $n = pq$
- DES construction: $y = E_k(x) \oplus x$ with $k$ fixed and known

**Note:** OWF  OWHF (hash functions impose compression and 2nd-preimage resistance).

> Original Text
>
> A **function f is called one-way** (one-way function or **OWF**) if for $x \in X$ we can easily compute $f(x) = y$ but for the vast majority of $y \in Y$ it is **computationally impossible** to find an $x$ such that $f(x) = y$.
> **Examples:**
>
> - computing squares modulo a composite: $f(x) = x^2 \bmod n$ with $n = pq$ (p and q

unknown) is a **one-way function** because the inverse is difficult (see the basic problem **SQROOTP**).
- we can construct a one-way function based on DES or any other block encryption system $E$ as follows: $y = f(x) = E_k(x) \oplus x$, $\forall x$, with $k$ a fixed and known key. We can consider that $E_k(x) \oplus x$ has (pseudo)random behavior by construction of $E$. Computing the inverse amounts to finding an $x$ such that: $x = E_k^{-1}(x \oplus y)$, which is considered difficult given the properties of $E$. Note that $f(x) = E_k(x)$ would not be sufficient to make an OWF because, with the key known, DES is reversible.

**OWF** $\neq$ **OWHF**: Note that an OWHF as a hash function imposes additional restrictions on the source and image domains as well as on 2nd-preimage resistance that are not necessarily satisfied by OWFs.
**Example**: $f(x) = x^2 \bmod n$ with $n = pq$ (p and q unknown) is not an OWHF because given $x$, $-x$ is a trivial collision.

Quick Revision

**OWF**: easy in one direction ($f(x) \to y$), impossible in the other ($y \to x$).
Examples: modular squares, $E_k(x) \oplus x$.
OWF $\neq$ OWHF (hash functions = more constraints).

## Hash Functions: Definitions

A hash function $h$ has two essential properties:

- **Compression**: transforms data of arbitrary length into fixed-length output
- **Ease of computation**: $h(x)$ is fast to compute

**Classification:**

- **Unkeyed** (no key): MDC (Manipulation Detection Code)
- **Keyed** (with key): MAC (Message Authentication Code)

**Security properties:**

1. **Preimage resistance**: given $y$, impossible to find $x$ such that $h(x) = y$
2. **2nd-preimage resistance** (weak collision): given $x$, impossible to find $x' \neq x$ such that $h(x) = h(x')$
3. **Collision resistance** (strong collision): impossible to find any $x \neq x'$ with $h(x) = h(x')$

**Terminology:**

- **OWHF** (weak one-way): satisfies (1) and (2)

- **CRHF** (strong one-way): satisfies (2) and (3)

Original Text

A **hash function** is a function $h$ having the following properties:

- **compression**: the function $h$ maps a set $X$ composed of bit strings of finite but arbitrary length to a set $Y$ composed of bit strings of finite and fixed length (and normally smaller than the size of $X$) with $h(x) = y$, and $x \in X$, $y \in Y$.
- **easy to compute**: given $h$ and $x \in X$, $h(x)$ is easy to compute.

A hash function is called **"keyed"** (keyed hash function) if a key is involved in the computation of the function ($h_k(x) = y$); otherwise it is called **"unkeyed"** (unkeyed hash function).
Hash functions have many computer applications including structured archiving facilitating search. On the security side we will study two main categories:

- **manipulation detection codes** (**MDC**) or message integrity codes (**MIC**): these are unkeyed functions allowing to provide an integrity service under certain conditions. The result of such a function is called **MDC-value** or simply **digest**.
- **message authentication codes** (**MAC**) which are keyed functions allowing to authenticate the source of the message and ensure its integrity without using additional (encryption) mechanisms.

**Some basic properties of hash functions:**

- **1) preimage resistance**: given a $y \in Y$, it is computationally impossible to find a preimage $x \in X$ satisfying $h(x) = y$.
- **2) 2nd-preimage resistance**: given an $x \in X$ and its image $y \in Y$, with $h(x) = y$, it is computationally impossible to find an $x' \neq x$ such that $h(x) = h(x')$. Also called **weak collision resistance**.
- **3) collision resistance**: it is computationally impossible to find two distinct preimages $x, x' \in X$ for which $h(x) = h(x')$ (no restriction on the choice of values). Also called **strong collision resistance**.

A **one-way hash function** (**OWHF**) is an MDC satisfying 1) and 2). Also called: **weak one-way hash function**.
A **collision resistant hash function** (**CRHF**) is an MDC satisfying properties 2) and 3). (Note that 3) 2)). Also called: **strong one-way hash function**.
**OWF OWHF**: Note that an OWHF as a hash function imposes additional restrictions on the source and image domains as well as on 2nd-preimage resistance that are not necessarily satisfied by OWFs.
**Example**: $f(x) = x^2 \bmod n$ with $n = pq$ (p and q unknown) is not an OWHF because given $x$, $-x$ is a trivial collision.

Quick Revision

**Hash function**: compression + easy computation
**MDC** (unkeyed) for integrity
**MAC** (keyed) for authentication
**Properties**

1. preimage resistance
2. 2nd-preimage resistance
3. collision resistance

**OWHF** = (1)+(2)
**CRHF** = (2)+(3).

## Message Authentication Codes (MACs)

A MAC is a family of functions $h_k$ parameterized by a secret key $k$:

**Properties:**

1. **Compression**: arbitrary input → fixed output
2. **Easy to compute**: with known $k$, $h_k(x)$ is fast
3. **Computation-resistance**: without $k$, impossible to compute valid pairs $(x, h_k(x))$

**Implications:**

- Key non-recovery: impossible to recover $k$ from pairs $(x_i, h_k(x_i))$
- Preimage and collision resistance for anyone not possessing $k$

**Usage:** Source authentication + message integrity without directly revealing secrets.

Original Text

A **Message Authentication Code** (**MAC**) is a family of functions $h_k$ parameterized by a secret key $k$ having the following properties:

- **1) compression**: as for generic hash functions but applied to $h_k$.
- **2) easy to compute**: from a function $h_k$, and a known key $k$, we can easily compute $h_k(x)$. The result is called a **MAC-value** or simply a **MAC**.
- **3) computational resistance** (computation-resistance): without knowledge of the symmetric key $k$, it is (computationally) impossible to compute pairs $(x, h_k(x))$ from 0 or several known pairs $(x_i, h_k(x_i))$ for any $x \neq x_i$.

Property 3) implies that the pairs $(x_i, h_k(x_i))$ cannot be used to compute the key $k$ (**key**

**non-recovery**). However the key non-recovery property does not imply computation-resistance because chosen/known-plaintext attacks could lead to forged pairs $(x, h_k(x))$. The impossibility of computing pairs $(x, h_k(x))$ also translates to preimage and collision resistance (cf. previous slide) for any entity not possessing the key $k$.

Quick Revision

**MAC** = hash with key $k$
Without $k$: impossible to forge $(x, h_k(x))$ or recover $k$
Guarantees source authentication + integrity.

## Attacks on MDCs

### 2nd-Preimage Resistance Attack

**Problem:** Given $h(x) = y$, find $x'$ such that $h(x') = h(x)$.

**Probabilistic analysis:**

For an $m$-bit digest ($n = 2^m$ possible outputs), the probability of having at least one collision after $k$ attempts is:

$$P(\text{collision}) \approx 1 - (1 - 1/n)^k \approx k/n$$

For $P = 0.5$: $k = n/2 = 2^{m-1}$

**Conclusion:** For an $m$-bit digest, approximately $2^{m-1}$ attempts are needed to find a 2nd-preimage with probability 0.5.

Original Text

**Problem**: given $h(x) = y$, find $x'$ such that $h(x') = h(x)$.
**Practical example**: we have a text with an associated digest bearing a digital signature; we want to create a fake text bearing the same signature (without control over the original text). What are our chances from a probabilistic point of view?
Let a hash function $h$ with $n$ possible outputs and a given value $h(x)$. If $h$ is applied to $k$ random values, what must be the value of $k$ so that the probability of having at least one $y$ such that $h(x) = h(y)$ is 0.5?
For the first value of $y$, the probability that $h(x) = h(y)$ is $1/n$. Conversely, the probability that $h(x) \neq h(y)$ is $1-1/n$. For $k$ values, the probability of having no collision is: $(1-1/n)^k$, i.e.:

$$\left(1-\frac{1}{n}\right)^k = 1 - \frac{k}{n} + \frac{1}{2!}\left(\frac{k}{n}\right)^2 - \frac{1}{3!}\left(\frac{k}{n}\right)^3 + ...$$

which for very large $n$ can be approximated by $1 - k/n$. Therefore, the complementary probability of having at least one collision is about $k/n$; which gives us $k = n/2$ for a probability of 0.5.
**Conclusion**: for an $m$-bit digest, the number of attempts needed to find a $y$ such that $h(x) = h(y)$ with a probability of 0.5 is $2^{m-1}$.

Quick Revision

To break 2nd-preimage resistance with $m$-bit digest: $2^{m-1}$ attempts (prob 0.5).

#### Collision Resistance Attack: Birthday Paradox

**Problem:** Find two distinct values $x, x'$ such that $h(x) = h(x')$.

**Birthday paradox:** In a group of 23 people, probability $> 0.5$ of having two people with the same birthday.

**Mathematical result:**

For $n$ possible outputs, the probability of collision after $k$ computations:

$$P(\text{at least 1 collision}) = 1 - e^{-k(k-1)/(2n)}$$

For $P \geq 0.5$: $k \approx 1.17\sqrt{n}$

**Cryptographic consequence:** For an $m$-bit digest ($n = 2^m$ outputs), approximately $2^{m/2}$ computations are needed to find a collision with probability $> 0.5$.

**Practical example:** Modification of a contract into 237 variations to find a fraudulent version having the same digest as the legitimate version.

Original Text

**Problem**: find two values $x, x'$ distinct such that $h(x) = h(x')$.
**Practical example**: We have to have someone sign a text and we want to apply this signature to a falsified text (we control the original text). What are our chances of finding two original texts satisfying this criterion?
The **birthday paradox** is a classic probabilistic problem that shows that in a gathering of only 23 people, there is already a 50% chance of having two people with the same birthday.

Let $y_1, y_2, ..., y_n$ all the possible outputs of a hash function. How many $h(x_i)$: $h(x_1), h(x_2), ..., h(x_k)$ must we compute to have a probability of collision equal to or greater than 0.5?
The first choice for $h(x_1)$ is arbitrary (prob = 1), the second $h(x_2) \neq h(x_1)$ has a probability of $1 - 1/n$, the third of $1 - 2/n$, etc. This gives us a probability of having no collisions equal to:

$$P_{\text{no collision}} = \prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right)$$

We easily prove (series expansion of $e^{-x}$) that for $0 \leq x \leq 1$: $1 - x \leq e^{-x}$ and therefore:

$$P_{\text{no coll}} \leq \prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right) \leq \prod_{i=1}^{k-1} e^{-i/n} = e^{-k(k-1)/(2n)}$$

The probability of having at least one collision is $P_{\text{at least1}} = 1 - P_{\text{no-coll}}$. To know the value of $k$ for which $P_{\text{at least1}}$ is greater than 0.5, it suffices to calculate:

$$\frac{1}{2} \leq 1 - e^{-k(k-1)/(2n)}$$

If $k$ is large, we replace $k(k-1)$ by $k^2$ and we obtain after simple calculations:

$$k \geq \sqrt{2\ln(2) \cdot n} \approx 1.17\sqrt{n}$$

Taking $n = 365$ for the birthday, we get $k = 22.3$, which confirms the statement of the problem.
**Consequence for hash functions**: Let a hash function with $2^m$ possible outputs. If $h$ is applied to $k = 2^{m/2}$ inputs we have a probability greater than 0.5 of obtaining $h(x_i) = h(x_j)$.

Quick Revision

**Birthday paradox**: to break collision resistance with $m$-bit digest: $2^{m/2}$ attempts (prob $> 0.5$).
Example: 23 people suffice for identical birthdays.

## Computational Resistance: Recap

For a hash function with $n$-bit digest and MAC key of $t$ bits:

| Type | Property | Difficulty | Recommended Size |
|---|---|---|---|
| **OWHF** | Preimage | $2^n$ | $n \geq 128$ bits |
| | 2nd-preimage | $2^{n-1}$ | |
| **CRHF** | Collision | $2^{n/2}$ | $n \geq 256$ bits |
| **MAC** | Key recovery | $2^t$ | $t \geq 256$ bits |
| | Computation | $\min(2^t, 2^n)$ | $n \geq 128$ bits |

**Practical implications:**

- For integrity only (OWHF): 128 bits sufficient
- For collision resistance (CRHF): minimum 256 bits
- MACs: 256-bit key, 128-bit digest minimum

Original Text

$n$: size of the MDC-value or MAC-value resulting from the application of the hash function
$t$: size of the MAC key

| Hash Fct. Type | Characteristic | Computational Difficulty | Attack Goal | Recommended digest/key size |
|---|---|---|---|---|
| **OWHF** | preimage resistance | $2^n$ | find a preimage | $n \geq 128$ bits |
| | 2nd-preimage resistance | $2^{n-1}$ | find $x'$ with $h(x') = h(x)$ | |
| **CRHF** | collision resistance | $2^{n/2}$ | find a collision | $n \geq 256$ bits |
| **MAC** | key non-recovery | $2^t$ | find the key | $n \geq 128$ |
| | computation resistance | $\min(2^t, 2^n)$ | produce a $(x, h_k(x))$ | $t \geq 256$ |

Quick Revision

**Efforts**: preimage $2^n$, 2nd-preimage $2^{n-1}$, collision $2^{n/2}$.
**Sizes**: OWHF  128 bits, CRHF  256 bits, MAC key  256 bits.

### MDCs Based on Encryption Systems

**Principle:** Use a symmetric encryption algorithm (DES, AES) to construct an MDC.

**Challenges to solve:**

- Break the reversibility of symmetric algorithms
- Increase the nominal width (DES = 64 bits insufficient for CRHF)

**Operation:**

- Sequential processing of blocks
- Chaining operations with XOR
- Combination of $n$ boxes for digests of size $n\times$ nominal width

**Classical models:**

1. **Matyas-Meyer-Oseas**: $H_i = E_{m_i}(H_{i-1}) \oplus H_{i-1}$
2. **Davies-Meyer**: $H_i = E_{m_i}(H_{i-1}) \oplus m_i$
3. **Miyaguchi-Preneel**: $H_i = E_{m_i}(H_{i-1}) \oplus H_{i-1} \oplus m_i$

**Practical examples:**

- **MDC-2**: uses 2 DES boxes → 128-bit digest
- **MDC-4**: uses 4 DES boxes → 128-bit digest

**Limitation:** Security strongly dependent on the underlying algorithm.

Original Text

**Idea**: use a known symmetric encryption system to construct an MDC.
**Problems to solve**:

- we must "break" the reversibility of symmetric algorithms to make them OWHF or CRHF.
- The "nominal width" of some encryption systems (eg. DES) is 64 bits, which is not sufficient to build CRHF.

**Operating principle**:

- the text blocks are sequentially processed by the encryption "box".
- compression is based on chaining operations with the blocks resulting from previous iterations and logical functions (fundamentally XOR). This also makes the process irreversible.
- If necessary, $n$ encryption boxes will be combined to obtain digest lengths $n$ times greater than the nominal width of the boxes used.

**Attention**: the security of these algorithms is strongly dependent on the properties of the underlying encryption boxes.
**Examples**:

- The models of **Matyas-Meyer-Oseas**, **Davies-Meyer** and **Miyaguchi-Preneel**.
- **MDC-2** and **MDC-4** using respectively 2 and 4 DES boxes. Digest = 128 bits.

Quick Revision

MDCs from symmetric crypto: break reversibility + chaining XOR.
Models: Matyas-Meyer-Oseas, Davies-Meyer, Miyaguchi-Preneel.
MDC-2/4 with DES → 128 bits.

## Customized MDCs

Functions specifically designed for digest generation, optimized for speed and security.

**Construction elements:**

- Padding + adding the message length
- Predefined constants to increase dispersion
- Successive rounds with logical operations and rotations
- Chaining of outputs between rounds
- Every bit of the digest depends on every input bit

**Main algorithms:**

| Algorithm | Year | Digest | Status |
|---|---|---|---|
| **MD5** | 1992 | 128 bits | Broken |
| **SHA-0** | 1993 | 160 bits | Collisions in $2^{39}$ |
| **SHA-1** | 1995 | 160 bits | Collisions in $2^{63}$ |
| **SHA-2** | - | 224-512 bits | Currently secure |
| **SHA-3** (Keccak) | 2012 | 224-512 bits | Current standard |

**Attack evolution:**

- 2004: Full collisions on MD5 (X. Wang)
- 2005: SHA-1 theoretically broken ($2^{63}$ operations)
- 2008: Creation of fraudulent CA certificates via MD5
- 2012: SHA-3 adopted as new standard

Original Text

These are functions designed exclusively to generate integrity codes (digests) with a main concern for speed and security.
Their operation is based on the following elements:

- initialization operations (**padding** + adding the length).
- a set of **predefined constants** chosen specifically to increase dispersion.
- a set of "steps" (**rounds**) that will sequentially apply to all the original data blocks. These rounds will perform a combination of logical operations and rotations on the data and constants.
- **chaining** operations involving the outputs of previous rounds.

In these functions, every bit of the digest is a function of every bit of the inputs.
The most famous are:

- **MD5**: R. Rivest, 1992; RFC 1321. Digest = 128 bits. **Broken!**
- **SHA-0**: NIST, 1993. Digest = 160 bits. Collisions in $2^{39}$ operations instead of $2^{80}$
- **SHA-1**: NIST, 1995. Digest = 160 bits. Revision of SHA-0 with additional bit rotation. Collisions in $2^{63}$ operations (instead of $2^{80}$).
- **SHA-2**: NIST (FIPS 190-3). Includes: SHA-224, SHA-256, SHA-384 and SHA-512. Digest sizes range from 224 to 512 bits.
- **SHA-3**: Keccak Algorithm (digest size variable from 224 to 512 bits)

**Latest Developments**:

- X.Wang et al. culminated in 2004 a long work aiming to find collisions in the MD5 algorithm. They publish two pairs of collisions for 1024-bit messages.
- In 2005, X.Wang et al. prove at the CRYPTO'05 conference that the number of operations needed to find collisions on SHA-1 (current standard for secure hash functions) is only $2^{63}$.
- These attacks target the search for arbitrary collisions but during CRYPTO'06 researchers from the University of Graz in Austria propose a method to partially control the content of collisions.
- In December 2008 it is shown that controlled collisions on MD5 can be generated and thus create an illicit Certification Authority allowing to forge certificates accepted by any browser.
- These results rely on **analytical** approaches (as opposed to brute force!)
- The selection process for SHA-1's successor is similar to the one that designated AES as a block encryption standard. NIST decided (October 2012) that **Keccak** would be the base algorithm for **SHA-3**.

Quick Revision

**Customized MDCs**

- MD5 (broken)
- SHA-0 (broken)
- SHA-1 (weak)
- SHA-2 (secure)
- SHA-3/Keccak (current standard).

Construction: padding + constants + rounds + chaining.

## MACs Based on Encryption Systems

**CBC-MAC:** Uses a block cipher algorithm in CBC mode.

**Operation:**

- CBC mode with IV = 0
- Elimination of intermediate ciphertexts
- Only the last encrypted block is kept as MAC

**With DES:**

- Key length: 56 bits (112 in optional Triple-DES)
- MAC length: 64 bits

**Advantages:**

- Reuse of existing encryption infrastructure
- Acceptable performance

**Limitations:**

- Security limited by block size (64 bits for DES)
- Vulnerable if used incorrectly (ex: without variable IV)

Original Text

**CBC-MAC algorithm based on DES-CBC with IV = 0 and elimination of intermediate ciphertexts**

- key length = 56 bits (112 in case of using the optional part)
- MAC-value length = 64 bits

The diagram shows the sequential processing of message blocks $M_1, M_2, M_3$ with the encryption algorithm $E$ and the key $k$. The intermediate ciphertexts $C_1, C_2$ are eliminated. Only the last block $C_3$ constitutes the MAC.

Quick Revision

**CBC-MAC**: CBC mode + IV=0, only last block kept. DES: key 56/112 bits, MAC 64 bits.

## Nested MACs and HMACs

**Nested MAC (NMAC):** Composition of two MAC families $G$ and $H$:

$$\text{NMAC}_{k,l}(x) = g_k(h_l(x))$$

**Security:** Depends on two criteria:

- $G$ resistant to collisions
- $H$ resistant to specific MAC attacks

**HMAC (FIPS 198 standard, 2002):** Nested MAC using unkeyed MDCs (SHA-1, SHA-256).

**Construction:**

- Constants: `ipad = 0x363636...36` and `opad = 0x5C5C5C...5C` (512 bits)
- Key $k$ of 512 bits

$$\text{HMAC-256}_k(x) = \text{SHA-256}((k \oplus \text{opad}) \parallel \text{SHA-256}((k \oplus \text{ipad}) \parallel x))$$

**Advantages:**

- Most widely used MACs in practice
- Attacks on SHA more difficult with secret key
- Excellent performance
- Standardized and widely supported

Original Text

A **Nested MAC** or **NMAC** is a composition of 2 families of MAC functions $G$ and $H$ parameterized by keys $k$ and $l$ such that:

$$G \circ H = \{g \circ h \text{ with } g \in G \text{ and } h \in H\} \text{ with } g \circ h_{(k,l)}(x) = g_k(h_l(x))$$

The security of an NMAC depends on two criteria:

- The family of functions $G$ is collision resistant.
- The family of functions $H$ is resistant to specific attacks for MACs, i.e.: It is impossible to find a pair $(x, y)$ and a fixed but unknown key $m$, such that: $\text{MAC}_m(x) = y$.

An **HMAC** (FIPS 198, 2002) is a Nested MAC using at its base dedicated unkeyed MDCs like SHA-1 or SHA-256.
An HMAC uses two 512-bit constants called `ipad` and `opad` such that:

- `opad := 363636 ... 36`
- `ipad := 5C5C5C ... 5C`

and a key $k$ of 512 bits.
The operating scheme of HMAC-256 (based on SHA-256) is as follows:

$$\text{HMAC-256}_k(x) := \text{SHA-256}((k \oplus \text{opad}) \parallel \text{SHA-256}((k \oplus \text{ipad}) \parallel x))$$

**HMACs** are the most used MACs. The attacks mentioned on the functions of the SHA family are more difficult to carry out on an HMAC because of the key $k$.

Quick Revision

**HMAC**: double hash with derived keys (`ipad`/`opad`). $\text{HMAC}_k(x) = H((k \oplus \text{opad}) \parallel H((k \oplus \text{ipad}) \parallel x))$. Standard, secure, performant.

## Hash Functions Applications

### Data Integrity

**Three main approaches:**

1. **MAC only:**
   - $A \to B : X, \text{MAC}_k(X)$
   - Authentication + integrity guaranteed
   - Requires shared key
2. **MDC + Encryption:**
   - $A \to B : E_k(X, \text{MDC}(X))$

- Confidentiality + integrity
- Shared symmetric key

3. **MDC + Authentic channel:**
   - $A \to B : X$ (normal channel)
   - $A \to B : \text{MDC}(X)$ (authentic channel)
   - Channel separation

**Limitations:** These simple protocols offer no protection against replay attacks.

**Solution:** add timestamps or sequence numbers.

Original Text

**MAC Only**:

$$A \to B : X, \text{MAC}_k(X)$$

If $B$ computes on its side $\text{MAC}_k(X)$ and obtains the same value the message comes from $A$.
**MDC + symmetric encryption** (key $k$ known to $A$ and $B$)

$$A \to B : X, E_k(\text{MDC}(X))$$

$B$ computes $\text{MDC}(X)$ and then $E_k(\text{MDC}(X))$. If equal message comes from $A$.
**As 2) with confidentiality of $X$ added**:

$$A \to B : E_k(X, \text{MDC}(X))$$

**MDC + digital signature**:

$$A \to B : X, \text{Sig}_{\text{priv-A}}(\text{MDC}(X))$$

$B$ computes $\text{MDC}(X)$ and verifies $\text{Sig}_{\text{priv-A}}(\text{MDC}(X))$ with an authentic copy of `pub-A`. If equality $A$ is the origin of the message. This solution additionally offers **origin non-repudiation**.
These simple protocols offer no support for uniqueness nor for the timeliness of received messages and are exposed to **replay attacks**! They require mechanisms taking into account time or the transaction context (cf. entity authentication).

Quick Revision

**Integrity**: MAC only, MDC+crypto, MDC+signature.
Vulnerable to replay without timestamps/nonces.

### Blockchains and Proof of Work

**Bitcoin and blockchains:** Use of hash functions to chain transaction blocks.

**Characteristics:**

- Public and visible transactions
- Blocks chained via cryptographic hash functions
- Mining = solving a cryptographic puzzle (proof of work)

**Proof of Work:**

- Find a nonce such that hash(block $\|$ nonce) $<$ target
- Computationally expensive puzzle, rapid validation
- First miner to solve receives bitcoin reward

**Security:**

- Blockchain = public, decentralized, immutable ledger
- Falsification would require effort $>$ all honest miners
- Protection based on CRHF properties

**Bitcoin statistics (October 2025):**

- Difficulty: 150.84 T
- Target: $\approx 2^{177}$ (pseudo-collision on 79 bits)
- Hashrate: ~1.1 ZH/sec ($1.1 \times 10^{21}$ hash/sec)
- Average block generation time: 10 minutes

Original Text

Bitcoin transactions are published and visible by all participants. They are encapsulated in blocks chained using cryptographic hash functions.
**Mining** consists of iteratively adding new blocks containing current transactions.
Generating a valid block requires solving a **cryptographic puzzle** (proof of work) very costly in computation time (finding pseudo-collisions in cryptographic hash functions).
Validation remains very efficient.
The first miner able to generate a valid block will receive a monetary reward (in bitcoins).
The mining process is open to all miners but only the first is rewarded.
The resulting chain of blocks (**blockchain**) then becomes a public ledger, decentralized and **immutable** protecting all past transactions. Falsification/modification of data protected by the blockchain would require computational effort greater than that performed by all honest miners.
**Bitcoin Statistics 13/10/2025**:

- **Difficulty**: 150.84 T
- **Target**: $2^{224}$/Difficulty $=\approx 2^{177}$. The valid digest to generate a block must be less than $2^{177}$, which means a pseudo-collision on the 79 most significant bits. The variation on the inputs depends on the **nonce**.
- **Hashrate**: ~ 1.1 ZH/sec ($1.1 \times 10^{21}$ hashes /sec)
- **Hash functions executed to obtain a block**: $\sim 660 \times 10^{21}$
- **Average block generation time**: 10 min

Quick Revision

**Blockchain**: chaining of blocks via hash.
**Proof of Work**: find nonce for hash $<$ target.
Security = effort $>$ all miners.
Bitcoin: ~10 min/block, $10^{21}$ hash/sec.

### Other Applications

1. **Authentication:**
   - Data origin authentication (DOA)
   - Transaction authentication (DOA + temporal parameters)
2. **Virus checking:**
   - Creator publishes digest = $h$(software) via secure channel
   - Users verify integrity by recalculating the digest
3. **Public key distribution:**
   - Publish $h$(public key) instead of the complete key
   - Simplified authenticity verification
4. **Document timestamping:**
   - Timestamp applied to digest rather than complete document
   - Reduction of data to sign
5. **One-time password (S-Key):**
   - Hash chain: $x_1 = h(x_0), x_2 = h(x_1), \ldots, x_n = h(x_{n-1})$
   - System stores $x_n$, user provides $x_{n-1}$
   - Verification: $h(x_{n-1}) = x_n$
   - After validation, system stores $x_{n-1}$ for next time

Original Text

**Authentication**:

- data origin authentication (DOA)
- transaction authentication (= DOA + time-variant parameters)

**Virus checking**:

- The creator of software creates a digest = $h(x)$ with $x$ being the original and distributes it via a secure channel (eg. CD-ROM).

**Distribution of public keys**:

- Allows controlling the authenticity of a public key.

**Timestamp on a document**:

- The document on which we want to perform the timestamp is first submitted to a hash function. The timestamp (with the signature of the corresponding entity) then applies only to the digest.

**One-time password (S-Key)** (identification mechanism):

- From a secret seed $x_0$, we create a chain of hash-values: $x_1 = h(x_0)$, $x_2 = h(x_1)$, ... $x_n = h(x_{n-1})$.
- The system stores $x_n$ and the user enters $x_{n-1}$. If $h(x_{n-1}) == x_n$ OK.
- The system then stores $x_{n-1}$ and so on.

Quick Revision

**Applications**

- authentication
- virus checking
- public key distribution
- timestamping
- one-time passwords (hash chain)

### Randomized Hash Functions: UNIX Example

**Problem:** Deterministic hash functions always produce the same result for the same password.

**Risks:**

- Detection of identical passwords
- Offline dictionary attacks (pre-computed codebooks)
- Rainbow tables

**UNIX solution: Salt**

- Addition of a 12-bit pseudo-random element (salt) before hashing
- Different salt for each user
- 4096 possibilities ($2^{12}$) for each password

**Advantages:**

- Prevents detection of duplicates
- Pre-computed codebooks become ineffective
- Each password requires 4096 dictionary entries

**UNIX implementation:**

- File `/etc/passwd` globally accessible
- Format: `username:hash(salt+password):uid:gid:...`
- Hash based on modified DES (25 iterations)
- Salt stored in clear (first 2 characters of hash)

**Example:**

```
root:Jw87u9bebeb9i:0:1:Operator:/:/bin/csh
pp:1Qhw.oihEtHK6:359:355:PP:/net/spp_telecom/pp:/bin/cs
```

**Limitations:**

- Effective protection against pre-computed dictionaries
- Online attacks limited by the system (number of attempts)
- Offline attacks possible if file compromised

> **Original Text**
>
> UNIX keeps its passwords in a globally accessible file (or possibly distributed by NIS). The stored information corresponds to the result produced by a hash function.

**Example (fictional)**:

```
root:Jw87u9bebeb9i:0:1:Operator:/:/bin/csh
pp:1Qhw.oihEtHK6:359:355:PP:/net/spp_telecom/pp:/bin/cs
```

**Problems**:

- the hash function being deterministic, it produces the same result for identical passwords.
- one could create "books" (codebooks) containing the result of applying the hash function to given inputs (eg. a dictionary) and easily compare them (off-line) with the strings stored by UNIX (**brute force dictionary attack**).

**Solution**:

- Add a (pseudo) random element of **12 bits** different for each password (called **salt**) before computing the hash function and during verification.
- This element allows adding a random factor of **4096 possibilities** for each password and prevents detection of duplicates.

The operating scheme uses DES with 25 iterations, the password as key, and the salt to modify the E-boxes. The final 64-bit result is converted to 11 ASCII characters.
User awareness (not visiting dubious sites) decreases the effectiveness of this technique in malware transmission.
Dictionary attacks are normally less effective **online** because operating systems limit the number of unsuccessful authentication attempts.

Quick Revision

**UNIX salt**: 12 random bits added to password before hash.
4096 possible variations.
Prevents pre-computed codebooks and duplicate detection.