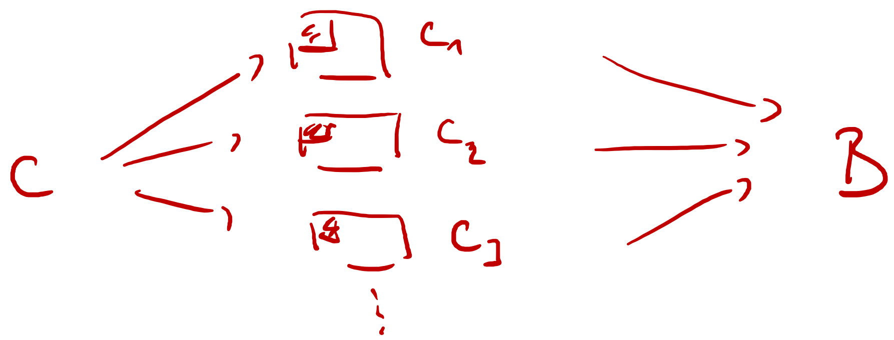
# Network Security

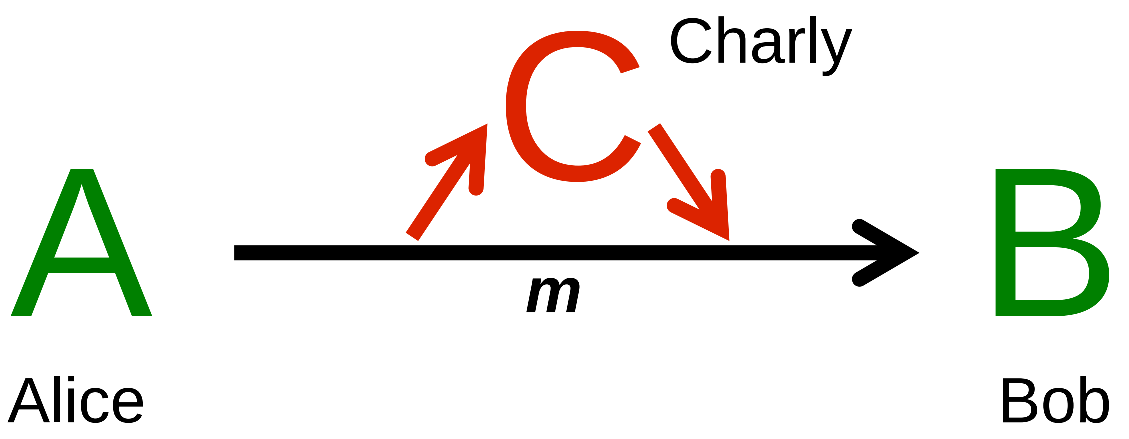
## Cryptography

Prof. Dr. Stefan Heiss

# IT-Security: Threats

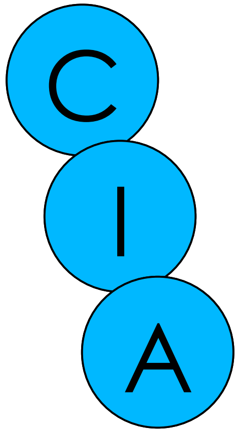
- **Eavesdropping, Sniffing**
- **Impersonation, Spoofing, Unauthorized Access**
- **Replaying attacks**
- **Denial of Services (DoS)**
- **Misuse of resources** → DDoS

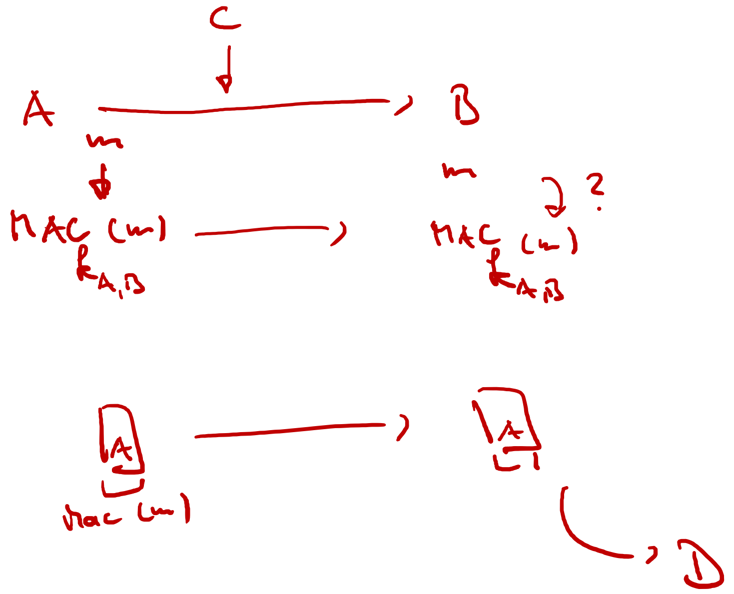


# IT-Security: Aims

- **Confidentiality**, Privacy, Secrecy
- **Integrity** — CRC is not enough
- **Availability**
- **Authenticity** — Message-Authi
- **Non-Repudiation**
- **Access Control**




A declines to be sender of the document

# IT-Security: Mechanisms

| | |
|---|---|
| ▪ Confidentiality | ▪ Encryption |
| ▪ Integrity | ▪ Message Digest, MAC, Digital Signature = Digital Fingerprint (opposite to a CRC) |
| ▪ Availability | ▪ Network Filter, Firewall, Robust Impl. → Secure Programming, Secure Configuration |
| ▪ Authenticity | ▪ MAC, Digital Sig., Key (physical token), Biometric identification |
| ▪ Non-Repudiation | ▪ Digital Signature |
| ▪ Access Control | ▪ Secure Configurations, Best Security Practices, Security awareness of users, Policies |

- Cryptographic secure Pseudo Random Number Generators (PRNGs)
- Message Digests (Cryptographic Hash Functions)

- Symmetric Ciphers
- MACs (Message Authentication Codes)

- Asymmetric Ciphers
- Digital Signatures
- Key derivation algorithms / schemes

Key exchange algorithms

generate

$A$: $k$, $m$ $\xrightarrow{c,\,t \;/\; c'}$ $B$: $k$, $c$, $t$

$c = Enc_k(m)$

$t = Mac_k(m)$

$c' = Enc_k(m|t)$

$Dec_k(c) = m$

$Mac_k(m) \stackrel{?}{=} t$

# Cryptographic Algorithms

- Cryptographic secure Pseudo Random Number Generators (PRNGs)
- Message Digests (Cryptographic Hash Functions)

- Symmetric Ciphers
- MACs (Message Authentication Codes)

- Asymmetric Ciphers
- Digital Signatures
- Key derivation algorithms / schemes

Asym. Enc.:

A: $k_{A,pub}, k_{A,priv}$ $\qquad$ B $\qquad$ $k_{A,pub} \rightarrow$ C, D

$A \xleftarrow{c} B$

B: $m$, $c = Enc_{k_{A,pub}}(m)$

A: $m = Dec_{k_{A,priv}}(c)$

Dig. Sig.:

$m \xrightarrow{m,s} m$

$s = Sig_{k_{A,priv}}(m)$

$Verify_{k_{A,pub}}(m,s) = 0/1$

# Kerkhoff's Principle

- **Kerckhoffs von Nieuwenhof (1835-1903):**
  - The security of a cryptographic algorithm should not depend on its nondisclosure.
  - Today's best practice: Only use and implement well-known algorithms that have been thoroughly investigated by the community of international distinguished cryptographers. (E.g.: Contest for election of AES)
  - Do not rely on “Security by obscurity” !

# Cryptographic Secure PRNGs

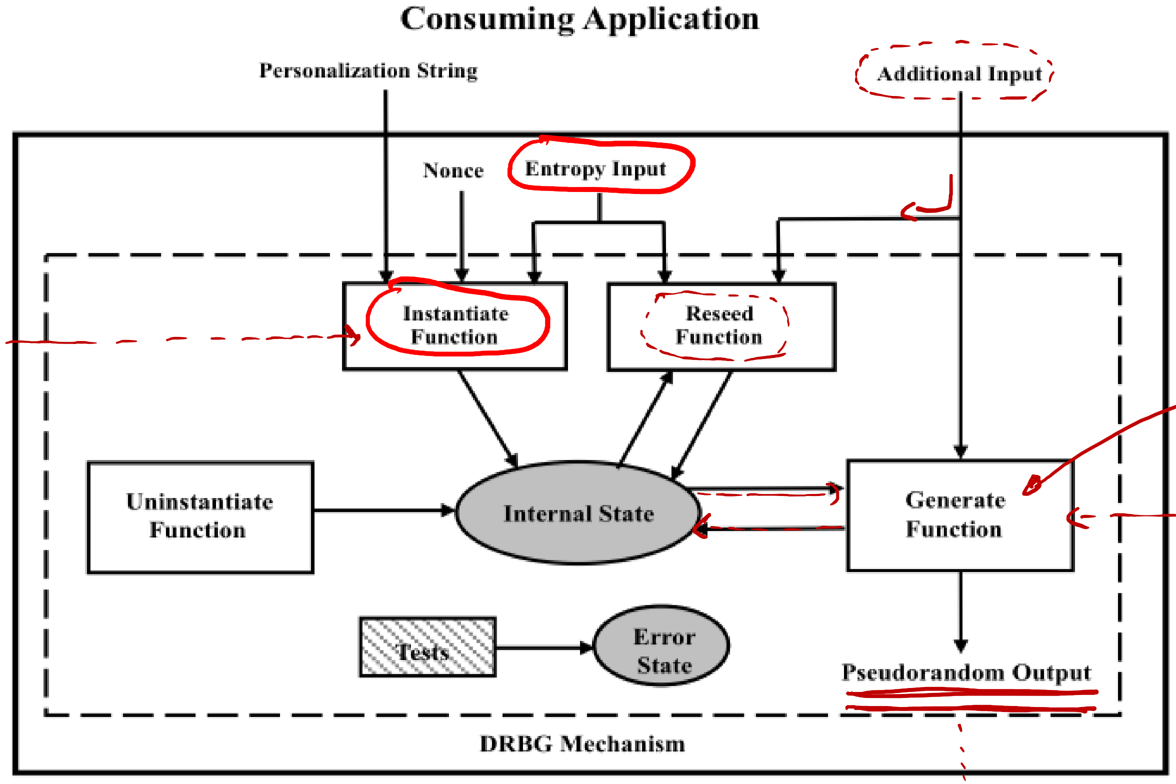
- JCA implementations: **SecureRandom**
- Recommendation for Random Number Generation Using Deterministic Random Bit Generators, NIST Special Publication 800-90A, June 2015

  http://dx.doi.org/10.6028/NIST.SP.800-90Ar1

  - DRBG Functional Model:

SecureRandom - Implementation:

- called with first getXXX function call
- takes very long ~ 0.5 s




often uses some Hash function repeatedly

getXXX()

# SecureRandom_PerformanceDemo

```java
public class SecureRandom_PerformanceDemo {

  static Random rng = new SecureRandom();
  static byte[] b = new byte[1];

  public static void main(String[] args) throws Exception {
    for( int i = 0; i < 100; i++ ) {
      long t1 = System.nanoTime();
      rng.nextBytes(b);
      rng.nextLong();
      System.out.println(System.nanoTime() - t1);
    }
  }
}
```

# Standard Random() implementations are NO CSPRNGs !

- Example: Java's [Random](#) class implements the PRNG based on the following linear congruential formula:

```java
public static long nextSeed(long seed) {
  return (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);
}
```

$a, b, n \in \mathbb{N}$ constants

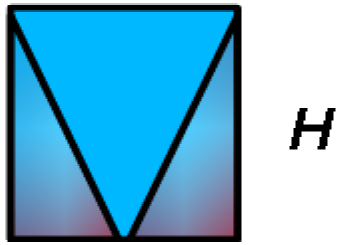
$x_{n+1} = a \cdot x_n + b \mod n$

  - $x_{n+1} = (25214903917 \cdot x_n + 11) \mod 2^{48}$
  - (Only the bits from (int)(seed >>> 16) are return via the Random API.)

```java
public static void findNextIntValue(long r1, long r2) {
  long seed = (r1 << 16);
  while( (nextSeed(seed) >>> 16) != r2 ) {
    ++seed;
  }
  System.out.println("Next value: " +
   Long.toHexString( nextSeed(nextSeed(seed)) >>> 16) );
}
```

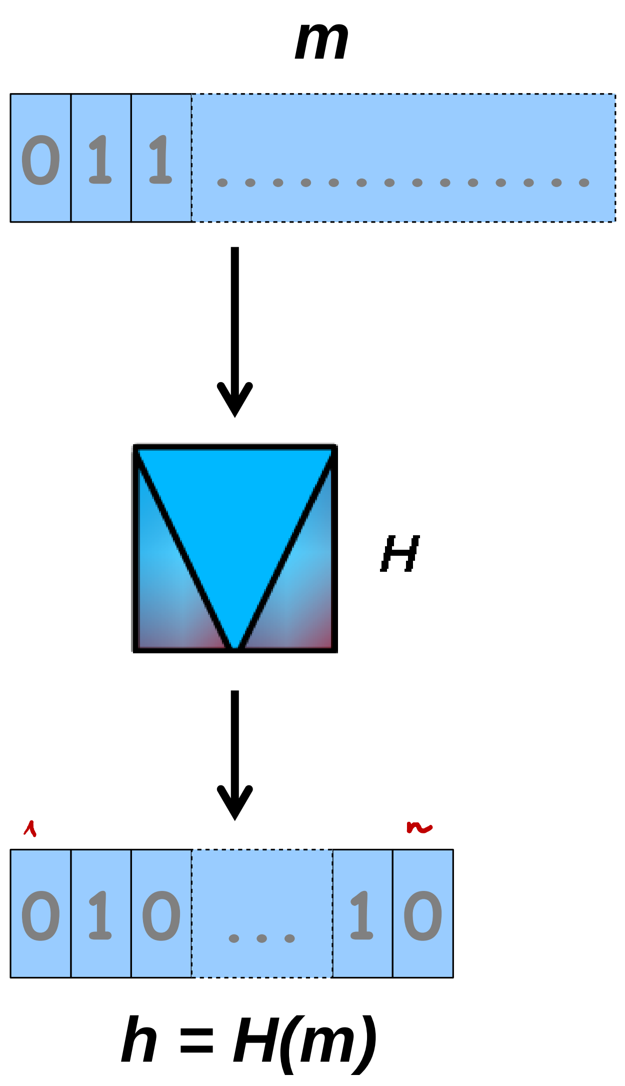
# Message Digests

- ***Cryptographic Hash Functions***
- ***Digital Fingerprints***
- JCA implementations: MessageDigest



A Message Digest (Cryptographic Hash Function $H$ is a mapping of the set of all binary sequences of finite length $m = (m_1, m_2, m_3, \ldots)$ to the set of binary sequences of some fixed length $n$:

$$H(m) = (h_1, h_2, \ldots, h_n) \in (F_2)^n$$



- **Preimage resistence**

  Given a sequence $(h_1,h_2,\ldots,h_n) \in (F_2)^n$, it is practically impossible to find a sequence $(s_1,s_2,s_3,\ldots)$ with $H(s_1,s_2,s_3,\ldots) = (h_1,h_2,\ldots,h_n)$.

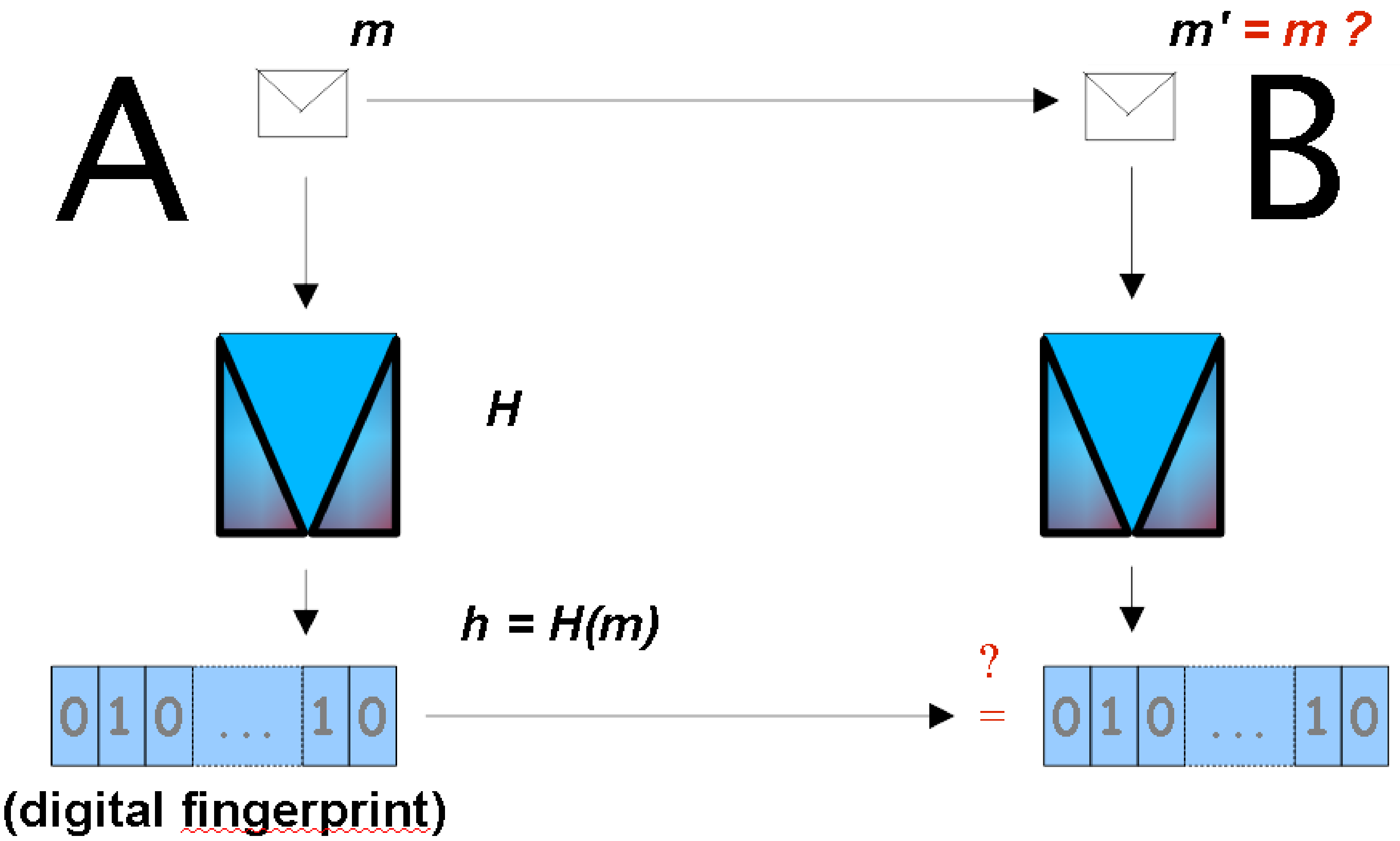
→ n has to sufficiently large to prevent brute force attacks ($\sim 2^n$)

- **Collision resistence**

  It is practically not possible to find two sequences $(s_1,s_2,s_3,\ldots)$ and $(t_1,t_2,t_3,\ldots)$ with $H(s_1,s_2,s_3,\ldots) = H(t_1,t_2,t_3,\ldots)$.

→ brute force attack takes about $\sqrt{2^n} = 2^{n/2}$ runs

(Birthday paradoxon)


m
m' = m ?
A
B
H
h = H(m)
?
=
0 1 0 ... 1 0
0 1 0 ... 1 0
(digital fingerprint)

- **Integrity checks**

  Example: Check of MD5 message digest after some file download

- **Protection of secrets**

  Example: Password files

  → nowadays good practice: random value (CSPRNG) → salt

  pwd.file

| user | salt | |
| --- | --- | --- |
| Alice | $s_A$ | $H(pwd_A \| s_A)$ |
| ⋮ | ⋮ | ⋮ |

- **Construction of PRNGs and stream ciphers**

- **Construction of MAC's (keyed hash)** → HMAC

- **Usage in Signature Schemes**

  m |______ - - - ]

  ↓ H

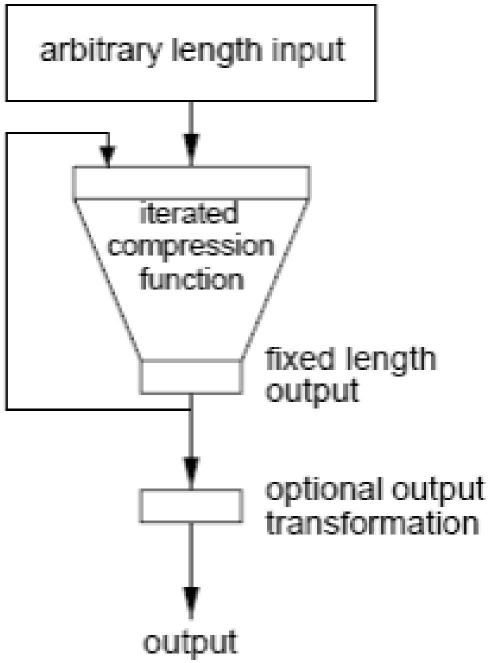
  $t = h(m)$

  $s = Sig_{k_{A,priv}}(t)$

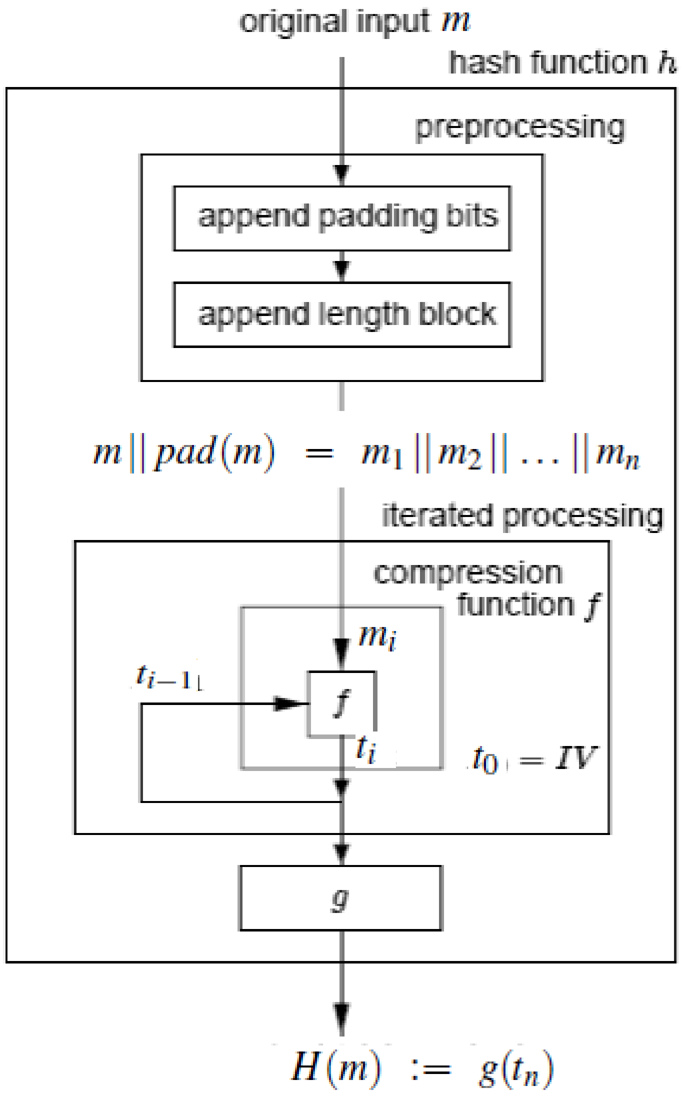
  preimage resistance and collision ... are very important

# Iterated Hash Functions – General Construction

(a) high-level view




(b) detailed view




$$l(m_1) = \cdots = l(m_n) = r$$

$$f : \mathbb{Z}_2{}^{r+s} \rightarrow \mathbb{Z}_2{}^{s}$$

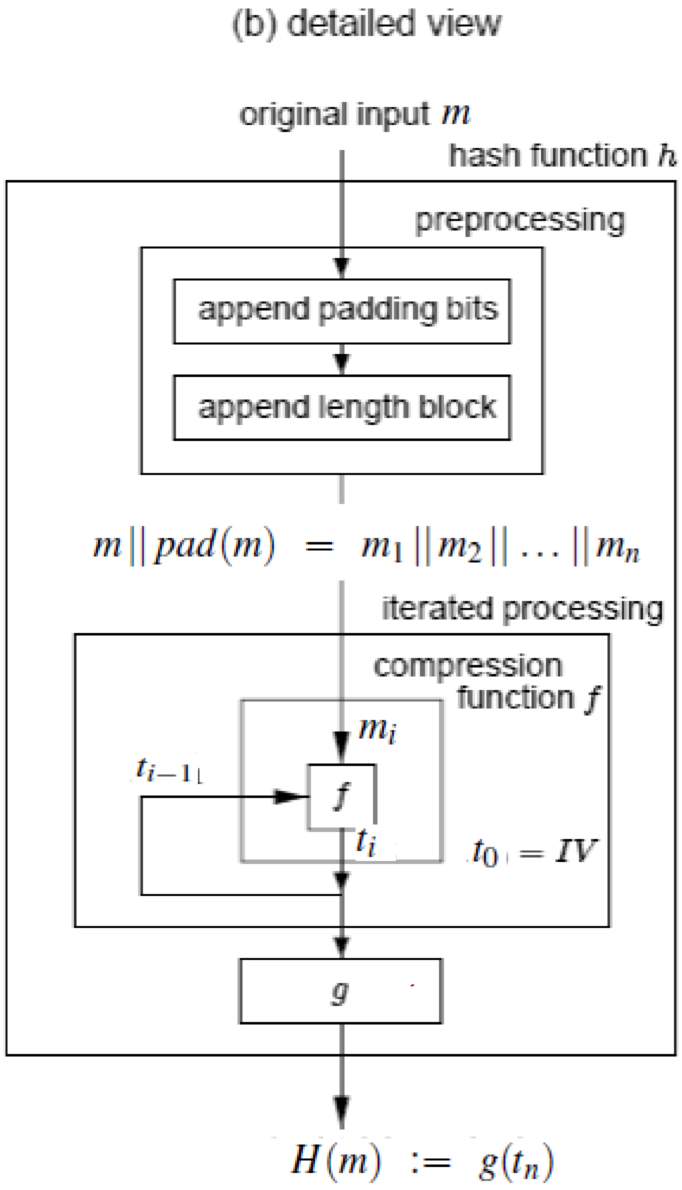
$$t_i \;:=\; f(m_i \,||\, t_{i-1})$$

$$g : \mathbb{Z}_2{}^{s} \rightarrow \mathbb{Z}_2{}^{l}$$

See: Handbook of Applied Cryptography, Chapter 9

# Hash Functions – Important Examples

| $H$ | $l = l(H(m))$ | $r = l(m_i)$ | $s = l(t_i)$ | $min\{l(pad)\}$ |
|---|---|---|---|---|
| MD5 | 128 | 512 | 128 | 65 |
| SHA-1 | 160 | 512 | 160 | 65 |
| SHA-224 | 224 | 512 | 256 | 65 |
| SHA-256 | 256 | 512 | 256 | 65 |
| SHA-512/224 | 224 | 1024 | 512 | 129 |
| SHA-512/256 | 256 | 1024 | 512 | 129 |
| SHA-384 | 384 | 1024 | 512 | 129 |
| SHA-512 | 512 | 1024 | 512 | 129 |
| SHA3-224 | 224 | 1152 | 1600 | 4 |
| SHA3-256 | 256 | 1088 | 1600 | 4 |
| SHA3-384 | 384 | 832 | 1600 | 4 |
| SHA3-512 | 512 | 576 | 1600 | 4 |




$l(m_1) = \cdots = l(m_n) = r$

$f : \mathbb{Z}_2{}^{r+s} \to \mathbb{Z}_2{}^{s}$

$t_i := f(m_i||t_{i-1})$

$g : \mathbb{Z}_2{}^{s} \to \mathbb{Z}_2{}^{l}$

- **Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD**

  *Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu,* August 2004

  http://eprint.iacr.org/2004/199.pdf

- **The first collision for full SHA-1**

  Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, Yarik Markov, 2017
  https://shattered.io/

# MessageDigest_Demo

```java
public class MessageDigest_Demo {

    public static void main(String[] args) throws Exception {

        FileInputStream fis
         = new FileInputStream("shattered-1.pdf");
        byte[] m = fis.readAllBytes();

        MessageDigest md = MessageDigest.getInstance("SHA-1");
        byte[] hashValue = md.digest(m);

//      System.out.println("Data:");
//      System.out.println(Dump.dump(m));

//      System.out.println("Hash value of shattered-1.pdf:");
//      System.out.println(Dump.dump(hashValue));
    }
}
```