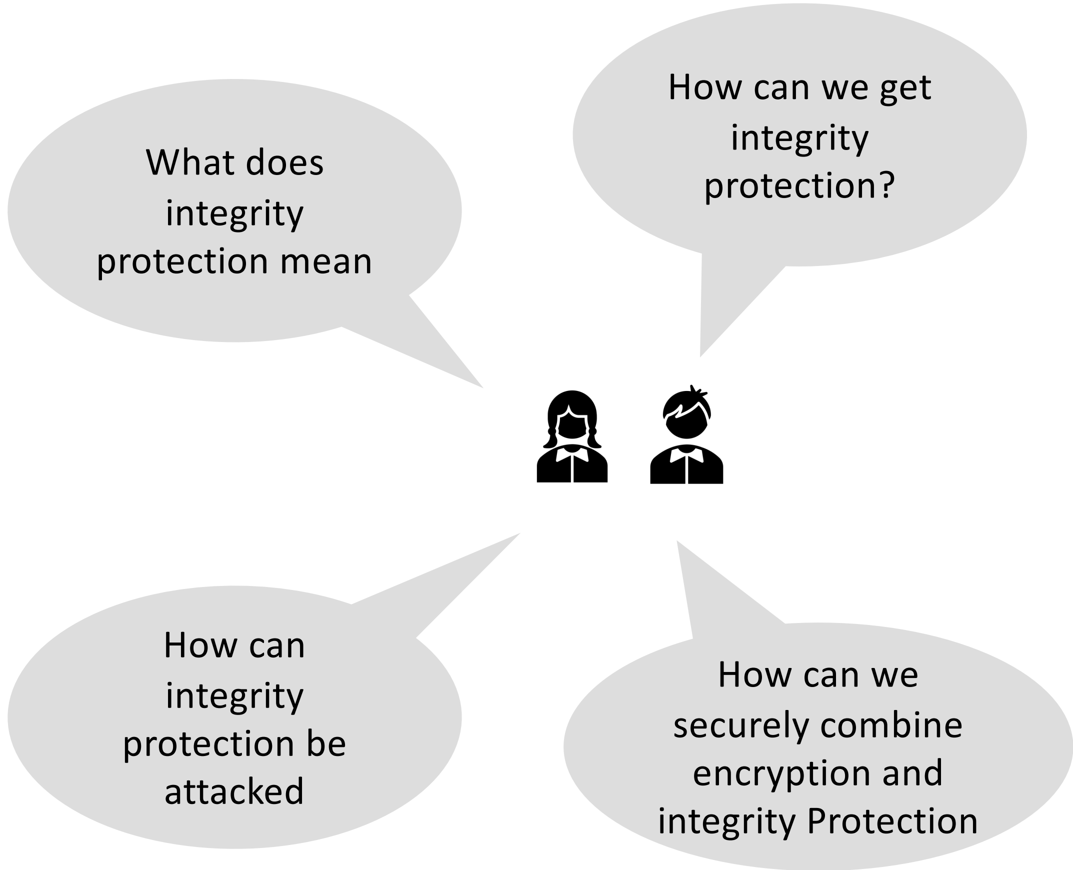
# IT-Security

## Chapter 3: Symmetric Integrity Protection

**Prof. Dr.-Ing. Ulrike Meyer**

# Overview

- **Definition and security of integrity protection**
  - Intuition
  - More formal definition
- **Message Authentication Codes**
  - Based on cryptographic hash functions
  - Based on symmetric ciphers
- **Combining Encryption and Integrity Protection**
- Based on cryptographic hash functions
  - Based on symmetric ciphers



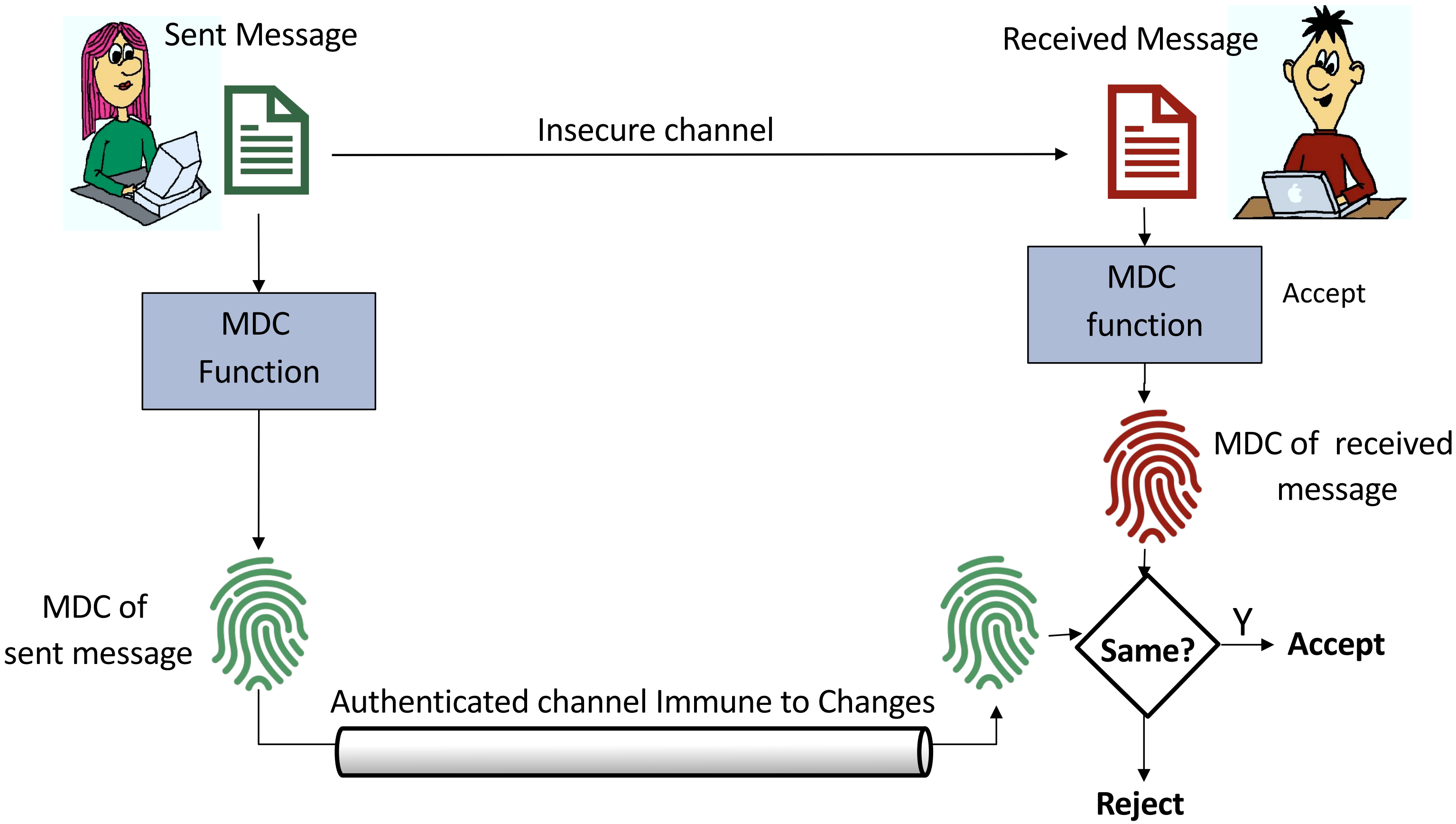
# Intuition for Data Integrity protection

- **Manipulation of messages sent over an insecure network cannot be prevented**
  - Anyone between the communicating entities can change the message
    - Flip bits, delete bits, replace messages with other ones
- **Encryption schemes do typically NOT enable detection of such manipulations**
  - See the many examples in the exercises
- **Data integrity protection mechanisms aim at detecting any message manipulation by unauthorized entities**
  - Can be realized in form of Modification Detection Codes (MDCs)
  - Can be realized in form of Message Authentication Codes (MACs)

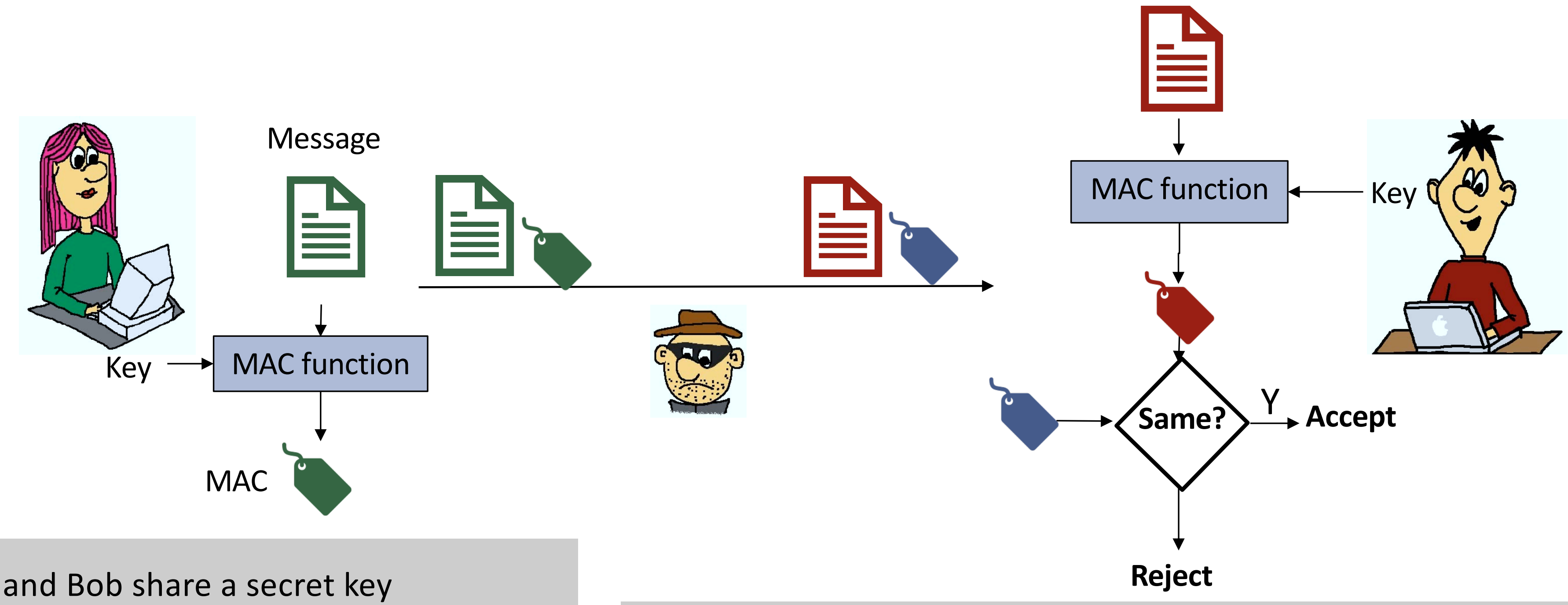
# Idea of Modification Detection Codes

**Problem with MDCs**

- Require a second channel immune to change
- Bob needs to be sure that the MDC he receives really comes from Alice



# Idea of Message Authentication Codes



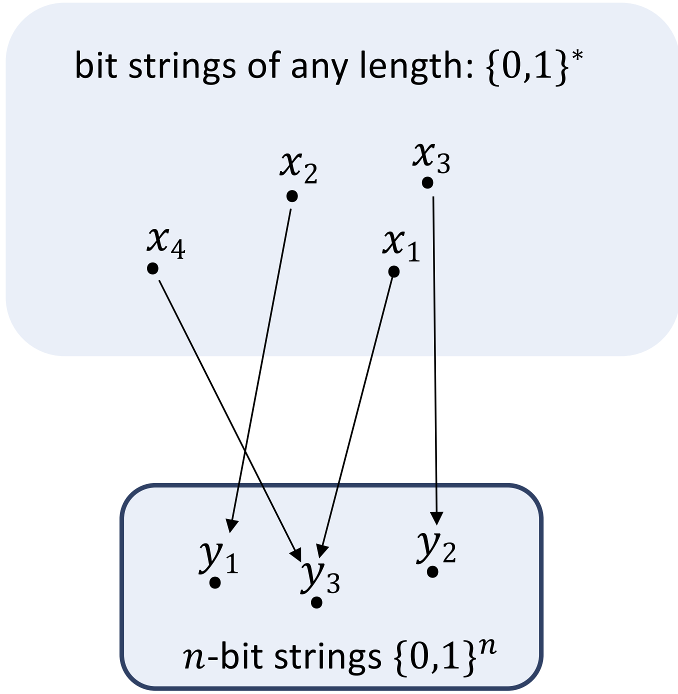

- Alice and Bob share a secret key
- Alice computes MAC of message using key
- Alice sends message and MAC to Bob
- Attacker may change message and/or MAC

- Bob computes MAC of received message using key
- Compares computed MAC to received MAC
- Decides that message was received as sent if both are the same

# Hash Function

- A **hash function** is a function $h$ with the properties
  - **compression**: $h$ maps an input $x$ of arbitrary bit-length to an output $h(x)$ of fixed bit-length $n$
  - **ease of computation**: given $h$ and $x$, $h(x)$ is easy to compute
    - there is a polynomial-time algorithm to compute h(x)

- A **collision** of a hash function is a
  - pair of inputs $x_1, x_2$, with $h(x_1) = h(x_2)$




**Any hash function $h$ has collisions!**

# Minimal Number of Collisions of a hash function

- **Basic pigeonhole principle**
  - If $n$ pigeonholes are occupied by $n + 1$ pigeons

    then at least one pigeonhole is occupied with more than one pigeons
- **Generalization**
  - If $n$ pigeonholes are occupied by $k \cdot n + 1$ pigeons

    then at lease one pigeonhole is occupied with more than $k$ pigeons

- **Consequence for the minimal number of collisions**
  - If a hash function maps $k \cdot n$ messages to $n$ hash values

    then there is at least one hash value to which $k$ or more messages hash
    - E.g., if n = 16, and $k \cdot n$ = 64, then there are 4 or more messages that hash to the same value

# Cryptographic Hash Function

- **A hash function is preimage resistant**
  - if given a randomly chosen $y = h(x)$ but not $x$ it is computationally infeasible to find any pre-image $x'$ with $h(x') = y$
- **A hash function is second preimage resistant**
  - if given x, $h(x)$ it is computationally infeasible to find a second pre-image $x' \neq x$ with $h(x') = h(x)$
- **A hash function is collision resistant**
  - if it is computationally infeasible to find a pair $x, x'$ with $x' \neq x$ and $h(x') = h(x)$

**Computationally infeasible** here means theoretically computable but impractical (**except with negligible probability**) as it takes too many resources and too much time to compute!

**A cryptographic hash function is a preimage resistant and collision resistant hash function**

# Relations between the Properties

- Collision resistance $\Rightarrow$ 2nd pre-image resistance

$\Rightarrow$ A cryptographic hash function is always 2nd pre-image resistant as it is collision resistant

- 2nd pre-image resistance $\not\Rightarrow$ collision resistance
- Collision resistance $\not\Rightarrow$ pre-image resistance
- Pre-image resistance $\not\Rightarrow$ collision resistance
- 2nd pre-image resistance $\not\Rightarrow$ pre-image resistance
- Pre-image resistance $\not\Rightarrow$ 2nd pre-image resistance

Note that some of these implications do hold for a narrower definition of a hash function mapping long fixed length-messages to much shorter hashes

# Example Proof of the Relations

- **Collision resistance ⇒ 2nd pre-image resistance**

**Proof by contradiction**

- Assume h is collision resistant but not 2nd pre-image resistant, then given $x, h(x)$ we can find an $x'$ such that $h(x') = h(x)$.
- Thus, we have found the collision $(x, x')$
- This contradicts our assumption which thus cannot hold

# Example Proof of the Relations

**Collision resistance $\not\Rightarrow$ pre-image resistance**

**Constructive proof**

- Assume $g$ is collision resistant $n$-bit hash function
- Define $h(x) = \begin{cases} 1 \parallel x & \text{if the bitlength of } x \text{ is } n \\ 0 \parallel g(x) & \text{otherwiese} \end{cases}$
- Then $h(x)$ is a $(n+1)$-bit hash function that is collision resistant but not pre-image resistant

Note that $a \parallel b$ stands for the concatenation of two bit-strings $a$ and $b$

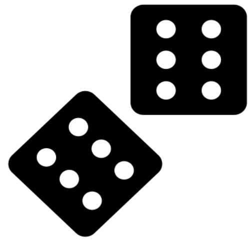
A similar proof can be used to proof that $2^{nd}$ -pre-image resistance does not imply pre-image resistance

# Related Terms and Synonyms

- **Cryptographic hash function = Secure hash function**
  - pre-image resistant + collision resistant
  - thereby also second-preimage resistant
- **One way hash function**
  - pre-image resistant
- **Second preimage resistant = weak collision resistant**
  - as it is implied by collision resistant
- **Collision resistant = strong collision resistant**
- **Output of hash function = hash value = message digest = hash**

# Ideal Hash Function through Random Oracle Model

- **An ideal $n$-bit hash function $h$ would operate as follows**
  - Upon receipt of a message $m$ it has not seen before
    - Pick an $n$-bit value uniformly at random from $\{0,1\}^n$ and return it as $h(m)$
  - Upon receipt of a message $m$ it has seen before
    - Return the same value $h(m)$, that was picked when the message was new



- **This ideal hash function is as pre-image and collision resistant as possible**
- **We can thus use it to determine an upper bound on**
  - how pre-image resistant a real-world hash function can be
  - how collision resistant a real-world hash function can be

# Complexity of Attacks against Ideal Hash Function

**Pre-image attack: Given a hash value y**

- Randomly select $x$ and compute $h(x)$
- Compare $h(x)$ to $y$
  - Stop if $h(x) = y$
  - Return to Step 1 otherwise
- **Requires** $0.69 \cdot \mathbf{2^n} = O(\mathbf{2^n})$ **hash computations to find a pre-image with probability ½**

**Collision attack:**

- Randomly select $x$ and compute $h(x)$, store result
- Compare each newly computed hash with the values already stored
  - Stop if $h(x) = h(x')$ and output $(x, x')$
  - Return to Step 1 otherwise
- **Requires** $\mathbf{1.18 \cdot 2^{n/2}} = O(\mathbf{2^{n/2}})$ **hash computations to find a collision with probability ½**

- Both statements on the complexities can be proven by the solution to flavors of the so-called Birthday Problem

# Example Proof of Complexity of Pre-image Attack

**The 1st birthday problem**

- Given $N$ different balls in a jar and one fixed ball $\hat{x}$
- How many times do we need to pull from the jar independently and uniformly at random with put back until with probability $P$ we pulled $\hat{x}$ at least once?

**Solution**

- If we chose one ball $x$, then the probability that $x \neq \hat{x}$ is $1 - \frac{1}{N}$
- The probability that we are unsuccessful $k$-times in a row is $(1 - \frac{1}{N})^k$
- The probability $P$ that we picked $\hat{x}$ at least once if we pick $k$-times is thus

$$P = 1 - (1 - \frac{1}{N})^k \sim 1 - e^{-\frac{k}{n}}$$ (using the approximation $1 - x \sim e^{-x}$ $(x \ll 1)$ )

- Thus $k \sim \ln[1/(1-P)]\, N$ and in particular for $\mathbf{P} = \frac{1}{2}$ **we get** $\mathbf{k \sim 0.69 \cdot N}$

1st birthday problem

Given **253** students, the **probability** that at least one of them has **February 2nd** as its birthday is **1/2**

# Similar but Omitted: Proof of Complexity of Collision Attack

**Birthday Paradoxon**

- Given $N$ different balls in a jar
- How many times do we need to pull independently and uniformly at random with put back from the jar until with probability $P$ we drew the same ball $\hat{x}$ twice?

**Solution**

- We need to draw $k \sim \sqrt{2\ln[1/(1-P)]\,N}$ times and in particular for $\mathbf{P} = \frac{1}{2}$ **we get** $\mathbf{k} \sim \mathbf{1.18} \cdot \sqrt{\boldsymbol{N}}$ $= \mathbf{1.18} \cdot \boldsymbol{N}^{\frac{1}{2}}$

Birthday paradox

Given **23** students, the **probability** that at least two of share the same birthday is **1/2**

# Examples for Hash Functions and their Properties

| Algorithm | Maximum Message Size in Bit | Block Size in Bit | Rounds | Size of Hash Value | Year |
|---|---|---|---|---|---|
| MD5 | $2^{64}$ | 512 | 64 | 128 | 1991 |
| SHA-1 | $2^{64}$ | 512 | 80 | 160 | 1993 |
| SHA-2-224 | $2^{64}$ | 512 | 64 | 224 | 2002 |
| SHA-2-256 | $2^{64}$ | 512 | 64 | 256 | |
| SHA-2-384 | $2^{128}$ | 1024 | 80 | 384 | |
| SHA-2-512 | $2^{128}$ | 1024 | 80 | 512 | |
| SHA-3-256 | unlimited | 1088 | 24 | 256 | 2015 |
| SHA-3-512 | unlimited | 576 | 24 | 512 | |

- **MD5 and SHA-1 are not considered collision resistant anymore and should no longer be used**
- **SHA-2 not broken yet, but break needs to be feared**

# Example Time-Lines of Breaks of MD5 and SHA-1

**MD5**

- 1993: Collision found by Boer and Bosselaers
- 1996: Attack that found a collision in a modified version of MD5
- 2004: Wang et al. found collisions in MD5 and others
- 2005: Further make collision finding feasible on a laptop (8 hours to find a collision)
- 2006: Black et al. implemented a toolkit for collisions in MD5
- 2007: Stevens et al. find collisions in less than 10 seconds on a on a 2.6Ghz Pentium 4
- 2009: MD5 attacks successfully used to fake certificates
- March 2011 IETF recommendation: MD5 should not be used any more where collision resistance is needed

**SHA-1**

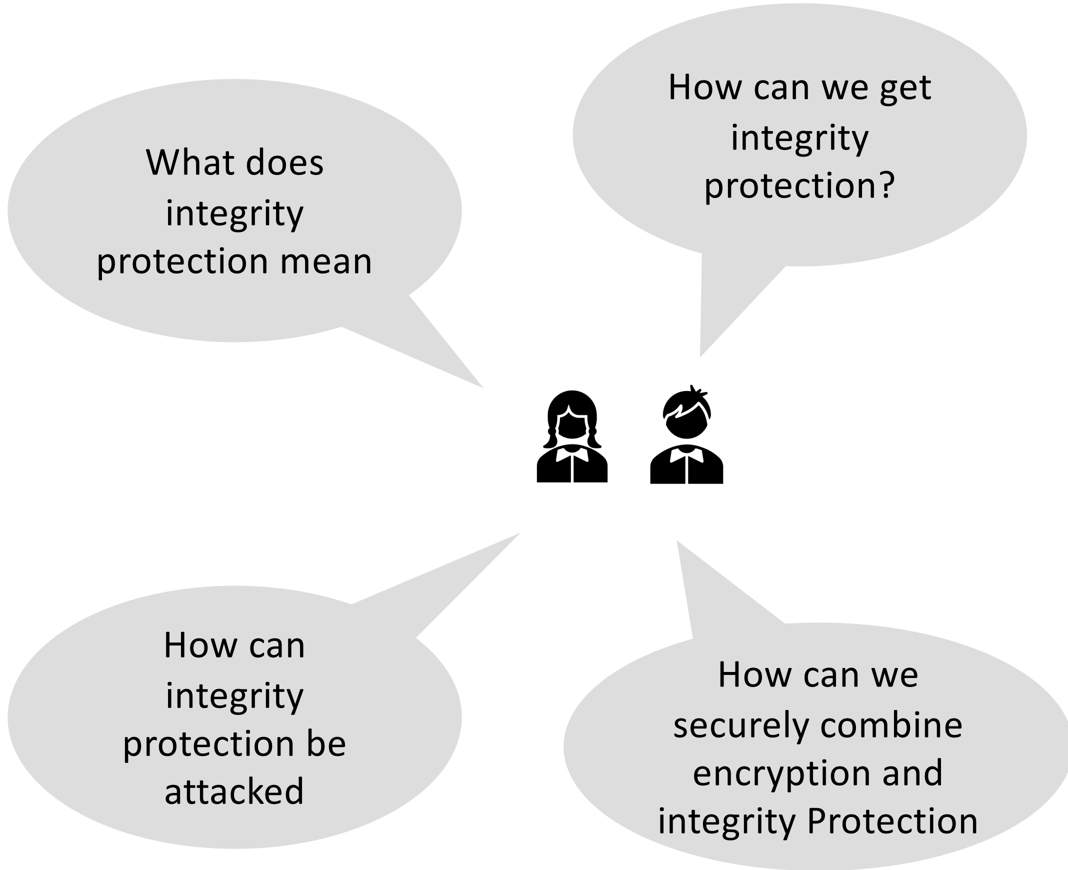
- 2004: 2nd preimage attack on SHA-1 in $2^{106}$
- 2005: Attack found by Wang et al. that finds a collision with $2^{69}$ hash operations
- 2013: Attack by Stevens et al. finds identical prefix collision in $2^{61}$ and chosen prefix collision in $2^{77.1}$
- 2015: Attack by Stevens et al. that finds a Free-Start Collision on 76-step SHA-1 in $2^{50}$ hash operations
- 2017: Collision on SHA-1 found
- 2016/2017 SHA1 was phased out starting from 2016/17 by all major browsers
- SHA-1 is not used anymore in the context of certificates

# Overview

- **Definition and security of integrity protection**
    - Intuition
    - More formal definition

- **Message Authentication Codes**
    - Based on cryptographic hash functions
    - Based on symmetric ciphers

- **Combining Encryption and Integrity Protection**
- Based on cryptographic hash functions
    - Based on symmetric ciphers

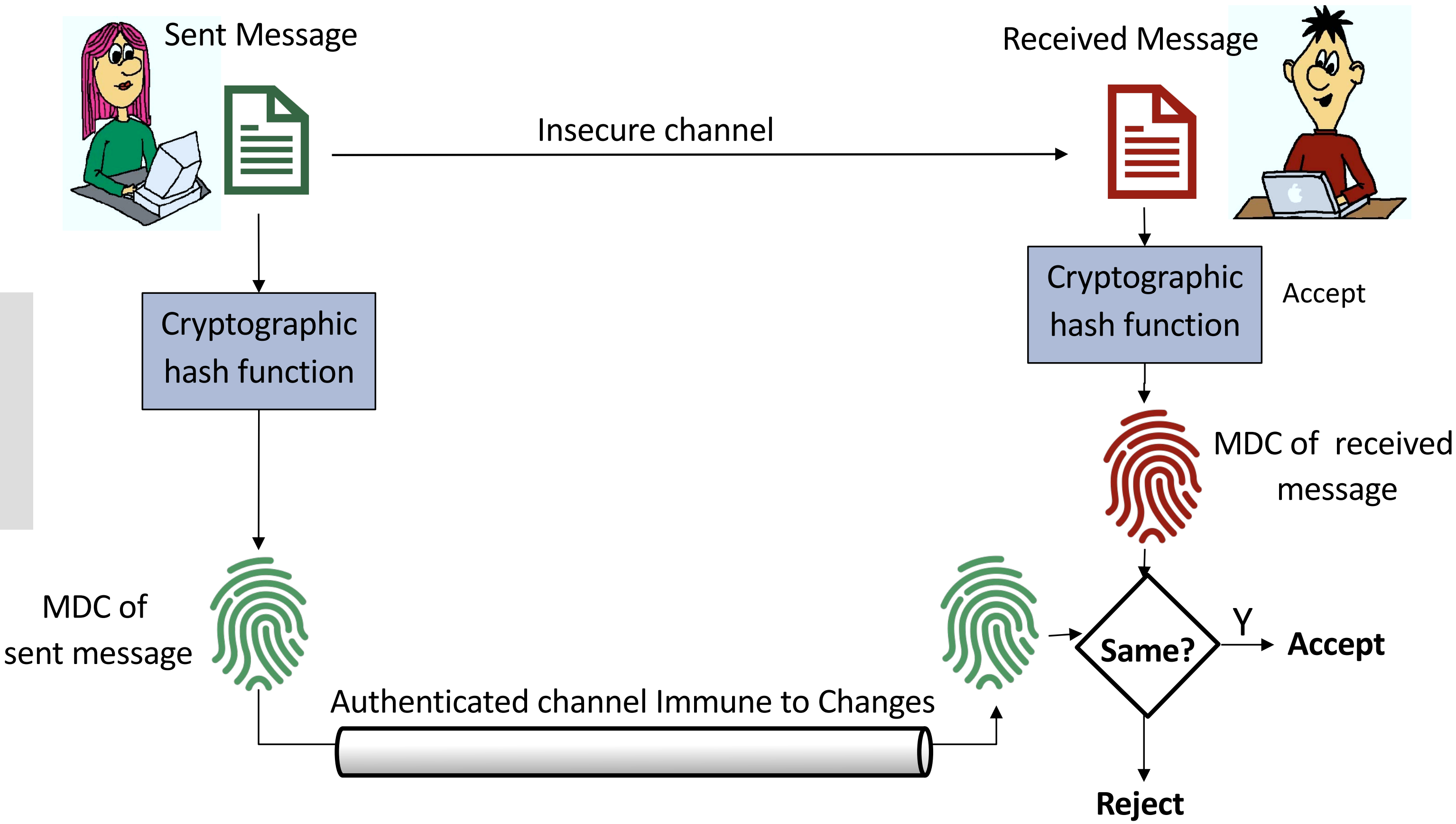


# Modification Detection Codes

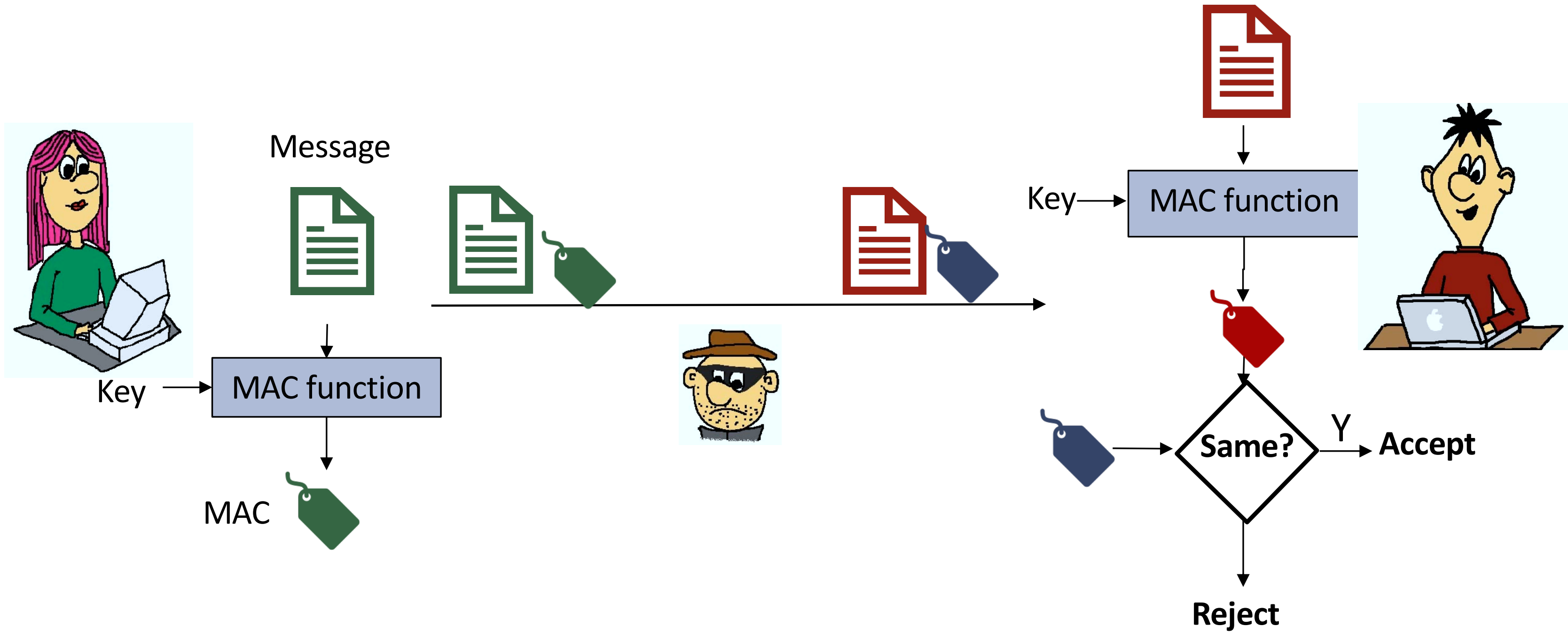
**Modification Detection Codes**

can be implemented by

**cryptographic hash functions**



# Message Authentication Codes




- MACs require a secret key as additional input
- MAC functions can be constructed from **cryptographic hash functions** or **block ciphers**

# Definition of a Message Authentication Code

- **A Message Authentication Code (MAC) is a family of functions $\mathrm{MAC}_{\boldsymbol{K}}$ parameterized by a secret key $\boldsymbol{K}$ with the following properties**
  - Ease of computation – given K and $\boldsymbol{x}$, $\mathrm{MAC}_{\boldsymbol{K}}(\boldsymbol{x})$ is easy to compute
  - Compression – $\mathrm{MAC}_{\boldsymbol{K}}$ maps an input $\boldsymbol{x}$ of arbitrary finite bit-length to an output $\mathrm{MAC}_{\boldsymbol{K}}(\boldsymbol{x})$ of fixed bit-length $\boldsymbol{n}$
  - **Computation resistance** – for every $\boldsymbol{K}$ and any given number of pairs $(x_i, \mathrm{MAC}_{\boldsymbol{K}}(x_i))$ it is without knowledge of $\boldsymbol{K}$ computationally infeasible to compute any pair $(\boldsymbol{x}, \mathrm{MAC}_{\boldsymbol{K}}(\boldsymbol{x}))$ with $\boldsymbol{x}$ different from all $\boldsymbol{x_i}$
    - Note that such pairs$(x_i, \mathrm{MAC}_{\boldsymbol{K}}(x_i))$ can typically be obtained by an attacker by eavesdropping

- **MACs can be constructed from cryptographic hash functions or block ciphers**

# HMAC: Bellare, Canetti, and Krawczyk 1996

- **Let h be a cryptographic hash function, then for a message $M$ and key $K$**

$$\mathrm{HMAC}_K(M) = h(K \oplus \mathrm{opad} \parallel h(K \oplus \mathrm{ipad} \parallel M))$$

  **where opad and ipad are constant values.**

  - ipad = 0x36….0x36
  - opad = 0x5C…0x5C

- **HMAC is computation resistant if h is cryptographic hash function**
  - HMAC construction does not introduce any new risk
  - ipad and opad guarantee that different keys are used in the inner and outer hash computation
    - The two keys will differ in half of the bits because of the choice of ipad and opad

# Can't we just use $h(K \parallel M)$ as MAC?

- **Unfortunately, no! Simple constructions like that are typically insecure**
- **Many hash functions (e.g., MD2, SHA-1, SHA-2) operate on blocks of $M$**
  - $M = M_0 \parallel M_1 \parallel \ldots . \parallel M_n$
  - $h$ operates on the first block $M_0$ which is then used as first state to operate on $M_1$,…
  - Thus, $h(M)$ is the initial state of $h(M \parallel X)$
    - I.e., from known hashes of shorter messages, we can construct hashes of longer messages
  - I.e., knowing $h(K \parallel M)$ we can compute $h(K \parallel M \parallel X)$ without knowing the key

# CMAC: Constructing a MAC from a Block Cipher

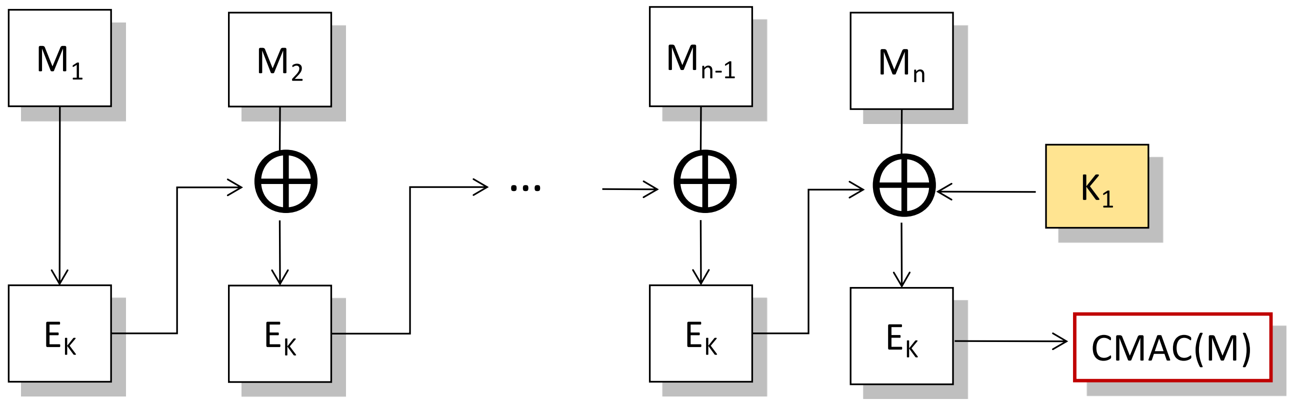
- **CMAC uses a block cipher $E_K$ of block length $b = 64$ or $b = 128$**
- **A message M is split into $n$ blocks of length $b$:**

  $M = M_1 \| M_2 \| \ldots . \| M_n$

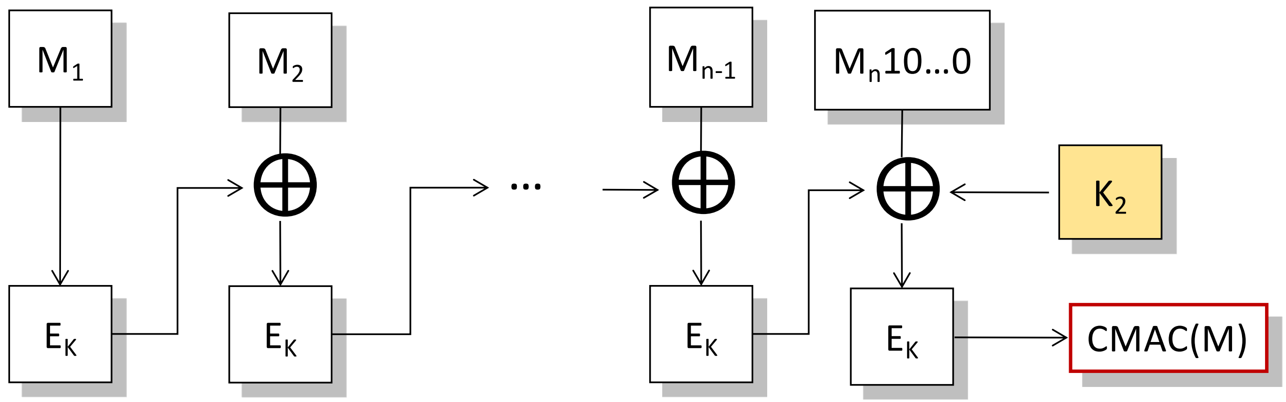
- **If the last block $M_n$ is not of length $b$ it is padded with $10 \ldots 0$ until it is $b$ bit long**
- **CMAC computation is equivalent to**
  - Applying CBC Mode of encryption to the message with an IV of all zeros
  - Except that the last block is additionally masked with
    - A sub-key $K_1$ if $M_n$ is of bit length b and with
    - A sub-key $K_2$ if $M_n$ was padded to be of full bit length b
  - The resulting last ciphertext block is the CMAC of the message

# Illustration of the CMAC Computation

**If $M_n$ has block length b**




**If $M_n$ is padded to b bits**




**K1 and K2 are derived from K**

- $L = E_K(0^b)$, where $0^b$ is the bitstrings of b zeros
- $R_{128} = 0^{120}10000111$
- $R_{64} = 0^{59}11011$

- **Then $K_1$ is computed by**
  - If $MSB1(L) = 0$, $K_1 = L\ll1$
  - Else $K_1 = L \oplus R_b$
- **$K_2$ is computed by**
  - If $MSB1(K_1) = 0$, $K_2 = K_1\ll1$
  - Else $K_2 = (K_1\ll1) \oplus R_b$

# Rational for the Two Different Keys

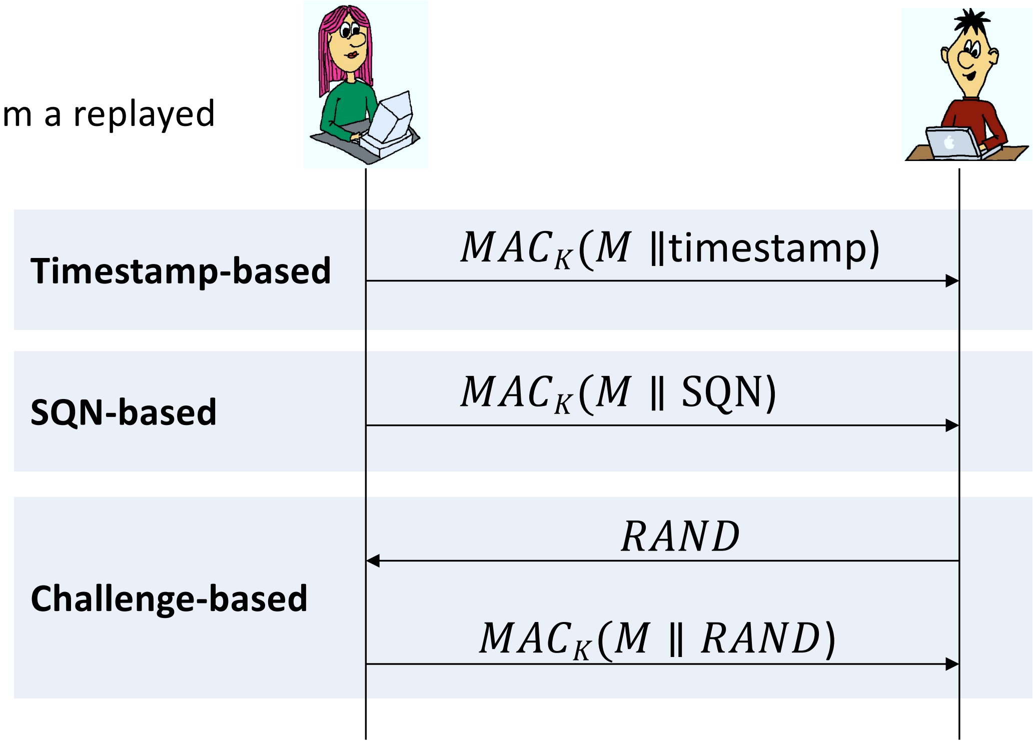
- **Let's assume we have a one block message** $M = 011$
  - then $\mathrm{CMAC}_K(M) = E_K(01110 \ldots 0 \oplus K_2)$
- **The one block message** $M' = 01110 \ldots 0$ **has** $\mathrm{CMAC}_K(M') = E_K(01110 \ldots 0 \oplus K_1)$
- **So, if** $K_1$ **and** $K_2$ **were the same,**
  - then $\mathrm{CMAC}_K(M)$ would be the same as $\mathrm{CMAC}_K(M')$
  - Thus, an attacker could replace $M$ with $M'$ without the receiver noticing it

# Why Do we need the Masking with $K_1$ and $K_2$

- **Using a "pure" CBC-MAC is insecure!**
  - I.e., without the masking by $K_1$ or $K_2$ in the last step
- **A CBC-MAC allows for forgery in some specific settings**
  - For example, let $M$ and $P$ be two one-block messages and $MAC_K$ be a CBC-MAC
    - $MAC_K(M) = E_K(M)$
    - $MAC_K(P) = E_K(P)$
  - If an attacker observes $M, MAC_K(M)$ and $P, MAC_K(P)$
    - he can forge a valid CBC-MAC on $M \parallel (P \oplus MAC_K(M))$ without knowing $K$ because:
    - $\boldsymbol{MAC_K(M \parallel (P \oplus MAC_K(M)))} = E_K(E_K(M) \oplus P \oplus MAC_K(M)) = E_K(P \oplus MAC_K(M) \oplus MAC_K(M)) = E_K(P) = \boldsymbol{MAC_K(P)}$
- **The masking with $K_1$ and $K_2$ solves this problem**

# Replay Protection

- **A MAC computed over a message alone**
  - does not protect against replay of the protected message
- **Replay protection requires additional input**
  - Make a message sent twice distinguishable from a replayed message
- **Additional input**
  - Counters
    - Time stamps
    - Sequence numbers ($SQN$)
  - Random numbers as challenges ($RAND$)



# Replay Protection

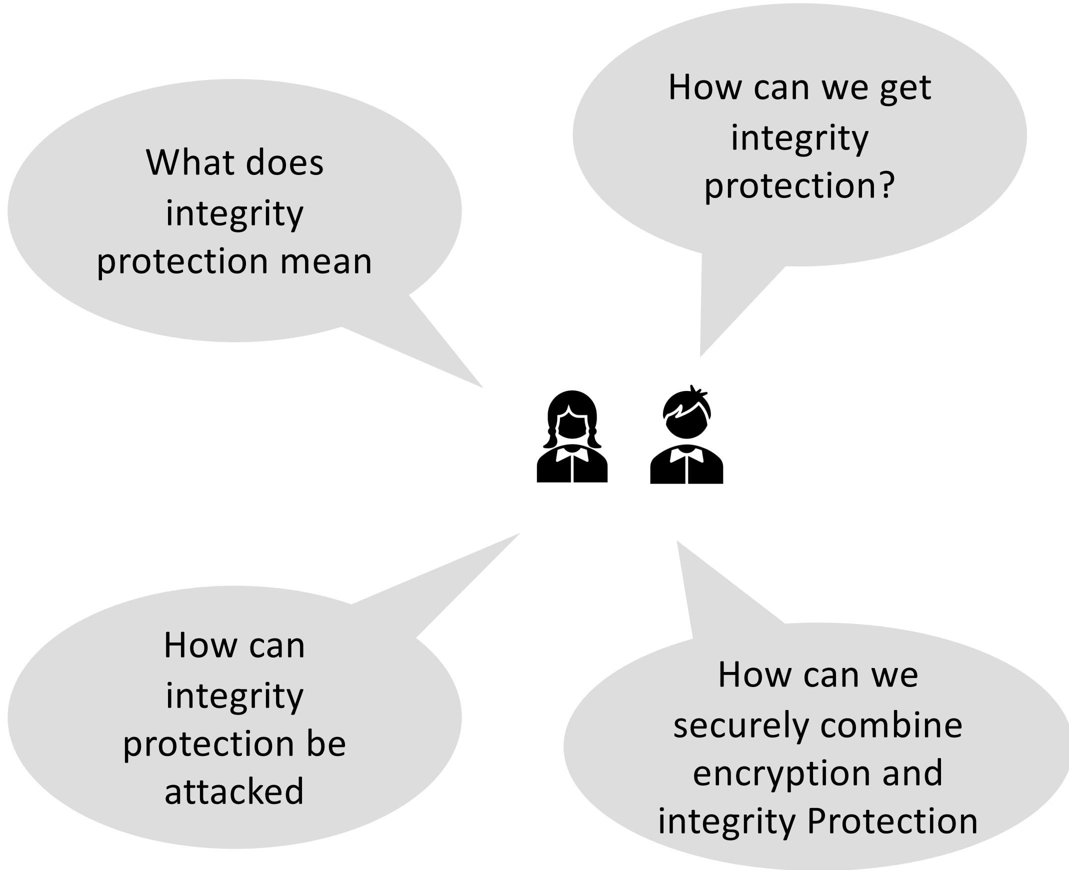
| | Advantage | Disadvantage | Main Use |
|---|---|---|---|
| **Timestamps** | No explicit initial value needs to be known by sender and receiver | Require time synchronization between sender and receiver | Whenever sender and receiver are time-synchronized anyway |
| **SQNs** | Simple, no time-synchronization required | Requires (re-)synchronization of SQN, Agreement on initial value, Window of acceptable SQNs if in-order delivery of messages cannot be guaranteed | Protect all traffic between two entities once keys are established |
| **RAND** | Does not need synchronization, requires random number generator | Requires receiver to challenge the sender and thus adds communication overhead | Mainly used as part of authentication and key agreement protocols, where single messages need to be protected against replay |

# Overview

- **Definition and security of integrity protection**
  - Intuition
  - More formal definition

- **Message Authentication Codes**
  - Based on cryptographic hash functions
  - Based on symmetric ciphers

- **Combining Encryption and Integrity Protection**
- Based on cryptographic hash functions
  - Based on symmetric ciphers



# Combining Integrity Protection and Encryption

**Encrypt, then MAC:** $E_{K_1}(M) \parallel MAC_{K_2}(E_{K_1}(M))$

- Encrypt plaintext with $K_1$
- Compute MAC on encrypted plaintext with $K_2$

**MAC, then Encrypt:** $E_{K_1}(M \parallel MAC_{K_2}(M))$

- Encrypt plaintext with $K_1$
- Compute MAC on encrypted plaintext with $K_2$
- MAC can only be checked AFTER decryption

**Encrypt and MAC:** $E_{K_1}(M) \parallel MAC_{K_2}(M)$

- Encrypt plaintext with $K_1$
- Compute MAC on plaintext with $K_2$
- MAC may reveal information on M
- MAC can only be checked AFTER decryption

**Special authenticated modes of encryption**

- E.g., Galois Counter Mode (GCM)
- E.g., Counter mode with CBC MAC (CCM)
- Typically take an **encrypt then MAC** approach

# Example: Galois Counter Mode of Encryption (GCM)

- **Mode of encryption that also provides integrity protection**
  - Authenticated Encryption with Associated Data (AEAD) Mode
    - Allows for additional data to be integrity protected but not encrypted
- **Based on a block cipher with 128-bit blocklength**
- **GCM can be used as MAC alone**
  - called **GMAC** then
- **Properties**
  - Can use IVs of arbitrary length
  - Easy to implement very efficiently in hardware
  - Very good software performance

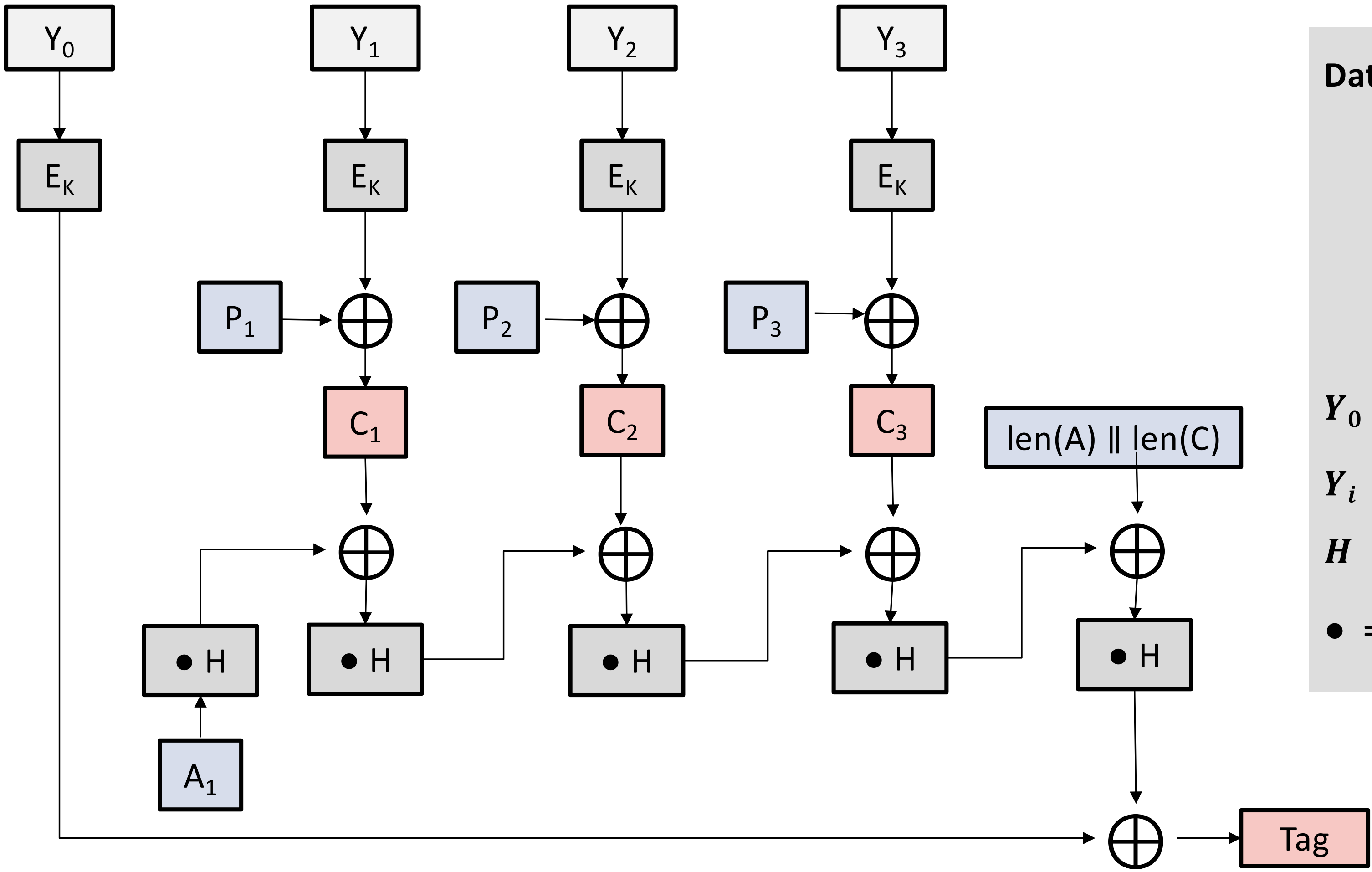
**Data blocks to protect**

$$\boldsymbol{A_1} \parallel \dots \parallel \boldsymbol{A_m} \parallel \boldsymbol{P_1} \parallel \dots \parallel \boldsymbol{P_n}$$

$\boldsymbol{A_i}\ (i = 1, \dots, m)$ are to be integrity protected only

$\boldsymbol{P_i}\ (i = 1, \dots, n)$ are to be integrity protected and encrypted

# Illustration of GCM Encryption and Integrity Protection Operation




**Data blocks to protect**

$A_1 \parallel P_1 \parallel P_2 \parallel P_3$

$A_1$ integrity protected

$P_i\ (i = 1, \ldots, 3)$ integrity protected and encrypted

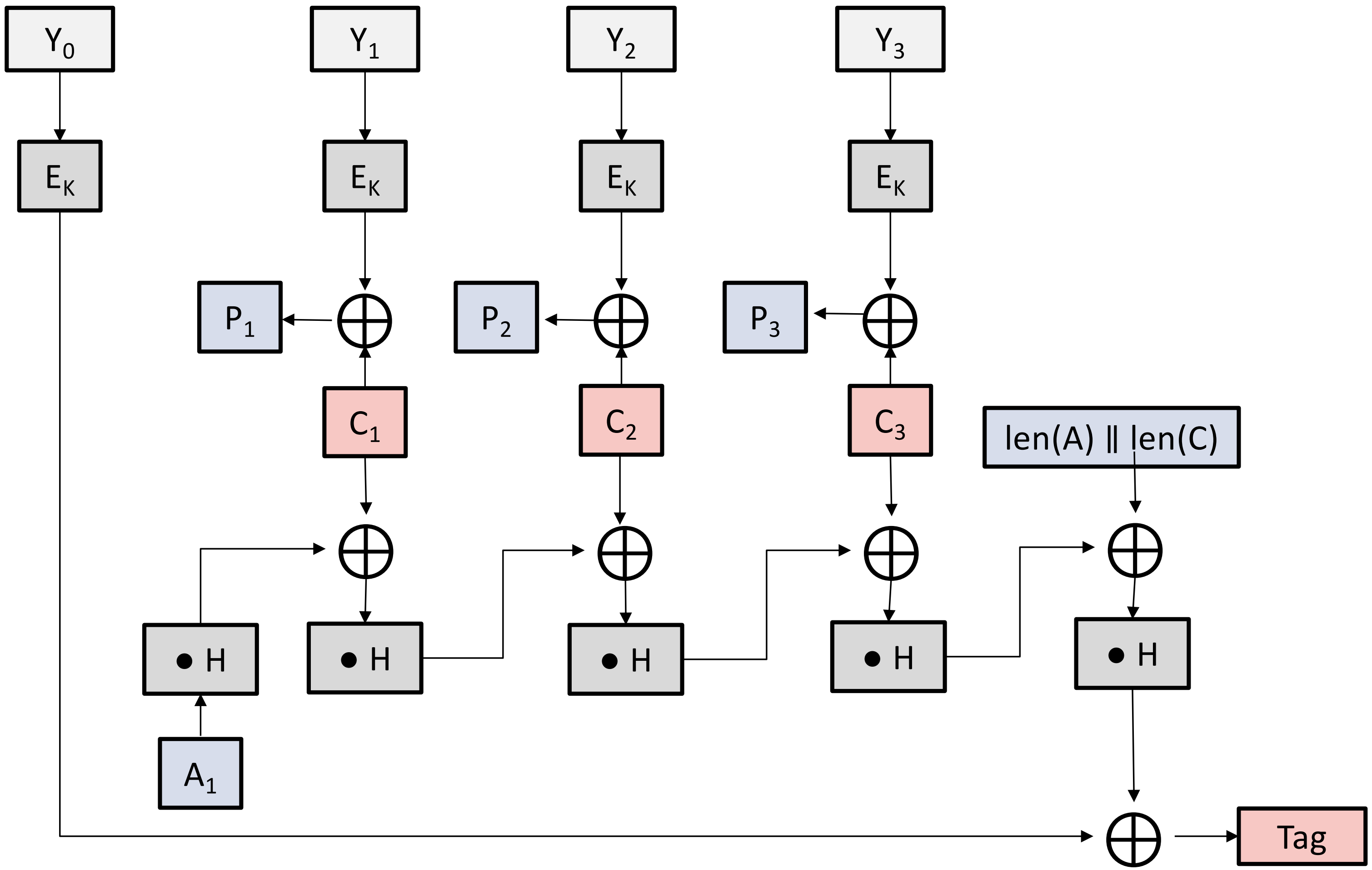
$Y_0$ Initial counter value

$Y_i = Y_{i-1} + 1$

$H = E_K(0^{128})$

$\bullet$ = Multiplication in $GF(2^{128})$

# Illustration of GCM Decryption and Integrity Verification Operation



# GCM in Formulars

**Data to be protected**

$$\boldsymbol{M = A_1 \parallel \ldots \parallel A_m \parallel P_1 \parallel \ldots \parallel P_n}$$

**Initialization:**

$Y_0$ Initial counter value

$Y_i = Y_{i-1} + 1$

$H = E_K(0^{128})$ wheer $0^{128} = \overbrace{0 \cdots 0}^{128}$

$\bullet$ = Multiplication in GF($2^{128}$)

**Encryption:** $C_i = E_K(Y_i) \oplus P_i$ for $(i = 1, \ldots, n)$

**Integrity Protection:** $T_0 = 0$

$$T_i = (T_{i-1} \oplus A_i) \bullet H \text{ for } i = 1, \ldots, m$$

$$T_{m+i} = (T_{m+i-1} \oplus C_i) \bullet H \text{ for } i = 1, \ldots, n$$

$$T_{m+n+1} = (T_{m+n} \oplus (len(A) \parallel len(C))) \bullet H$$

$$GMAC_K(M) = T_{m+n+1} \oplus E_K(Y_0)$$

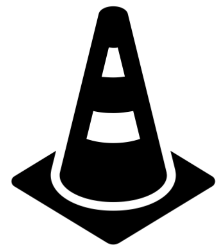
Note: if $P_n$ is not of full block length, then $C_n$ is not of full block length
If $A_m$ or $C_n$ are not of full block length, they are padded with zeros in the $GMAC$ computation

# Reminder: Multiplication in GF($2^{128}$)

- GF($2^{128}$) is the finite field with $2^{128}$ elements
  - It is unique up to isomorphism
- GCM uses the irreducible polynomial $f(x) = 1 + x + x^2 + x^7 + x^{128}$
- Identify each 128-bit string $a = a_0 \ldots a_{127}$ with the polynomial $a(x) = \sum_{i=0}^{127} a_i x^i$
- Multiplication of $a$ and $b$ in GF($2^{128}$) is then defined as
  - bit string representation of $a(x) \cdot b(x)$ mod f:

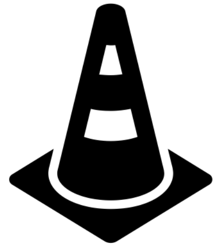
    $(\sum_{i=0}^{127} a_i x^i) \cdot (\sum_{i=0}^{127} b_i x^i) \mod f$

# Summary



- **Message Authentication Codes provide integrity protection**
  - MACs can be constructed from cryptographic hash functions: HMAC
  - MACs can be constructed from block ciphers: CMAC
  - Simple constructions like $h(M \parallel K)$ or CBC-MAC are insecure
- **Cryptographic hash functions**
  - Are pre-image resistant and collision resistant
  - Finding a pre-image with probability ½ requires at most $O(\mathbf{2}^{\boldsymbol{n}})$ hash computations for an ideal hash function
  - Finding a second pre-image with prob. ½ requires at most $O(\mathbf{2}^{\boldsymbol{n}})$ hash computations
  - Finding a collision with prob. ½ requires at most at most $O(\mathbf{2}^{\boldsymbol{n}/2})$ hash computations
- **Replay protection requires additional input to an integrity protection mechanism**
  - E.g., a counter, a time stamp, or a random number selected by the receiver

# Summary



- **Securely combining encryption and integrity protection**
  - ▶ Requires an encrypt-then-MAC type of an approach
    - Special modes of encryption which also provide integrity protection use this as well
  - ▶ Other approaches are insecure or unnecessarily expensive
- **The GCM Mode of encryption is an example for an AEAD cipher**
  - ▶ Provides encryption and integrity protection
  - ▶ Makes use of CTR mode for encryption
  - ▶ Can additionally protect the integrity of data which is not encrypted

# References

- **Johannes Buchmann, Einführung in die Kryptographie, Springer Verlag 2016**
  - Chapter 11 on Hash Functions and Message Authentication Codes
- **W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022**
  - Chapters 12: Message Authentication Codes

- **Specifications**
  - HMAC: NIST Specification FIPS 198-1
    - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf
  - CMAC:
  - GCM and GMAC NIST Special Publication 800-38D
    - https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf