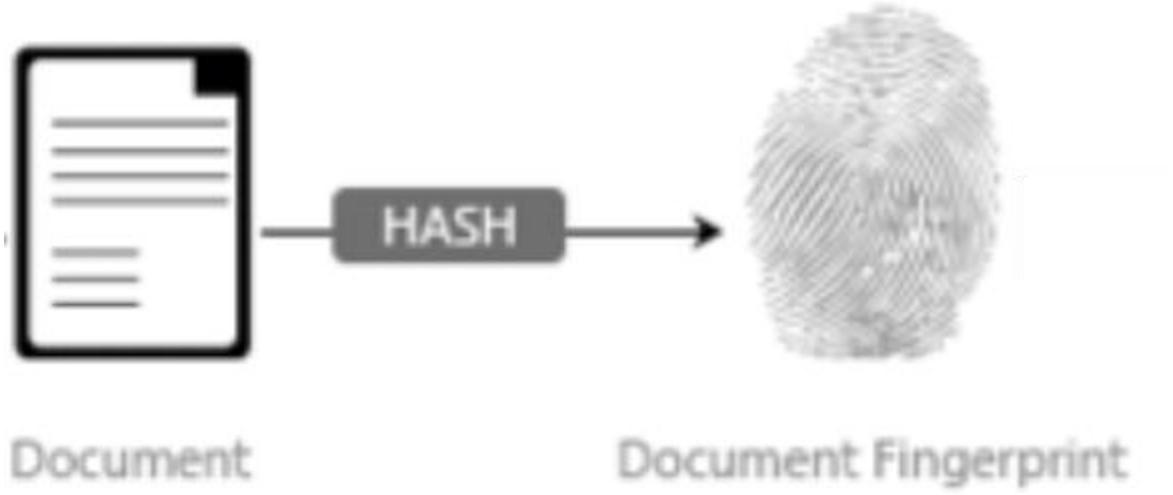
Hashing

110001
010101011101
1100101011011
0110101100010111
01101101110110000
11010101010111001
01010010101001010
1000111010101001
0101010101010

Outline

- Cryptographic Hash Function
- Digital Signature
- Message Authentication Code (MAC)

Cryptographic Hash Function



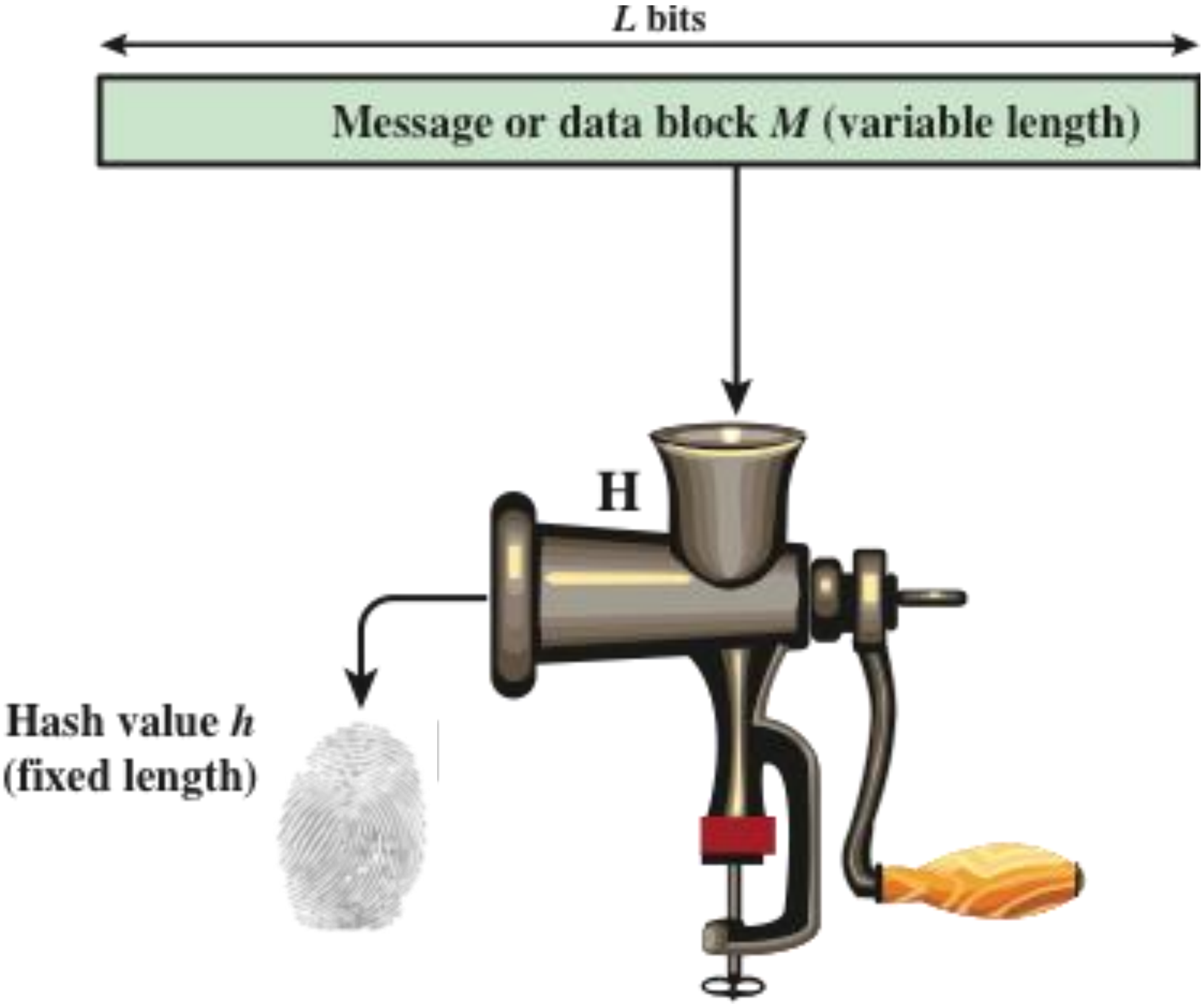
Back

Properties of a Cryptographic Hash Function

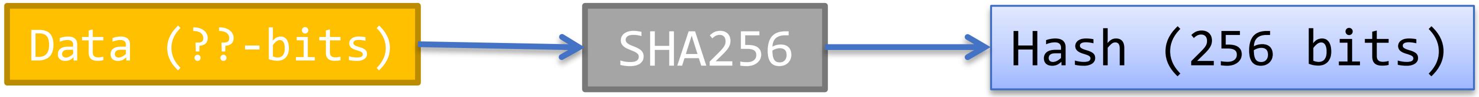
- h = H(m). Hash(Variable size message m) to produce a **fixed size hash value** (sometimes called a *message digest*)
- Efficient computation
- Pseudorandom (small change of ***m*** yields a big change of *h*)
- Cryptographic hash function has 2 properties:

1. **Pre-image Resistant** (the *one-way property*): Infeasible to determine ***m*** from ***H(m)***
2. **Collision Resistant** (the *collision-free property*): Infeasible to find any two messages m_1 and m_2 such that $m_1 \neq m_2$ and $H(m_1) = H(m_2)$

Analogy


L bits
Message or data block *M* (variable length)
H
Hash value *h*
(fixed length)

Sample in Python



Data (??-bits)

SHA256

Hash (256 bits)

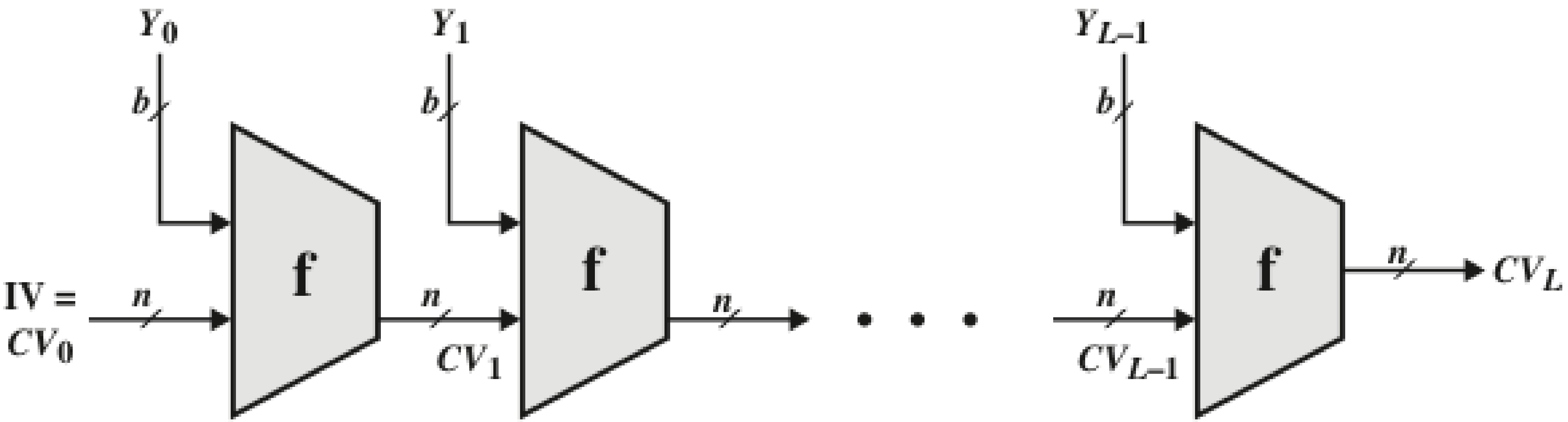
```
import hashlib
md = hashlib.sha256(b"The quick brown fox jumps over the lazy dog").hexdigest()
print (md)
```

```
d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592
```

Examples of Real Hash Functions

- MD5 (Message Digest 5)
    - Produces a 128-bit hash
    - Collisions can be found. An attacker can use them to substitute an authorized message with an unauthorized one.
- SHA1 (Secure Hash Algorithm 1)
    - 160-bit hash
    - Collisions can be found
- SHA2
    - Actually 4 different hash functions: SHA-224, SHA-256, SHA-384, SHA-512
    - Minor attacks, but still good
- SHA3
    - New NIST standard
    - No known attacks

General Structure of Secure Hash Function



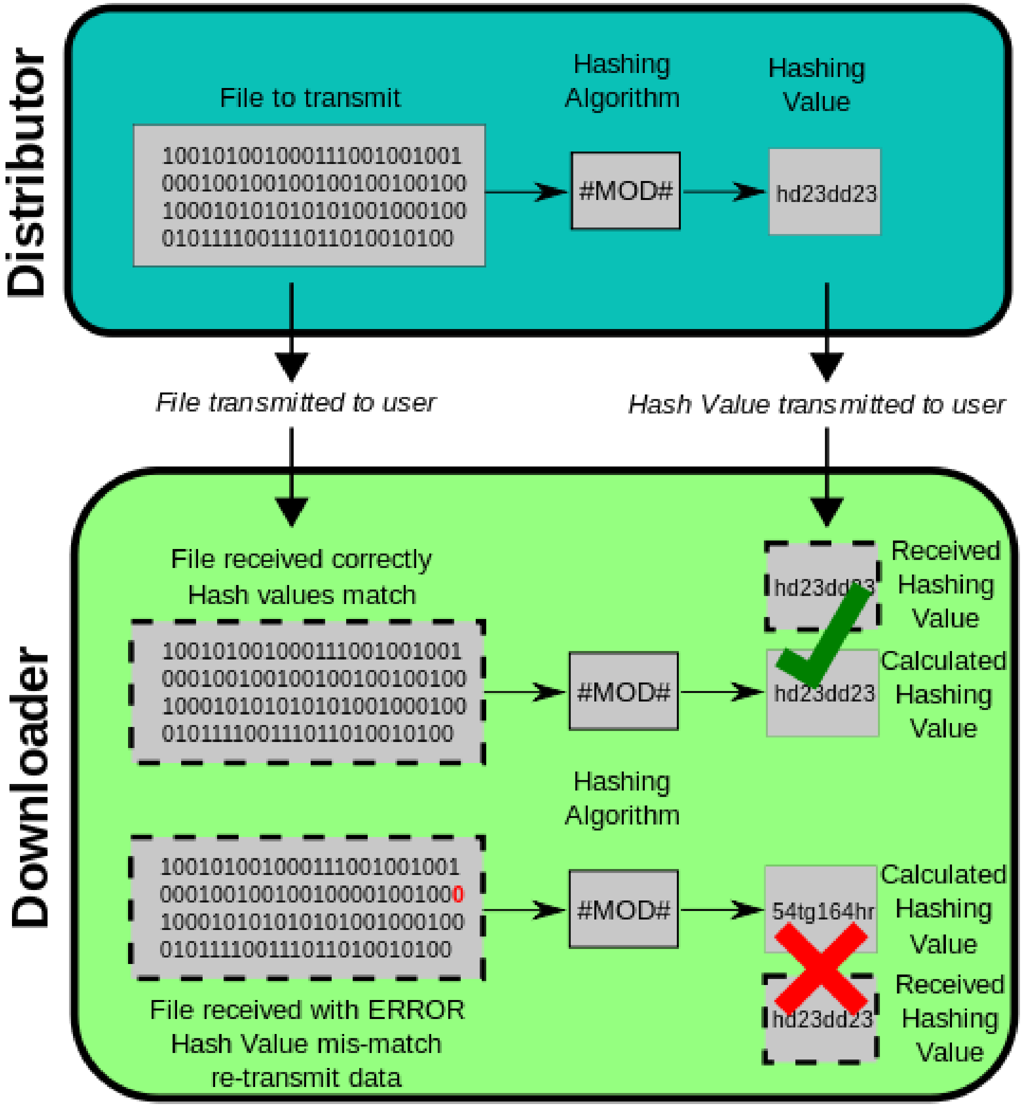
IV	=	Initial value	L	=	number of input blocks
CV_i	=	chaining variable	n	=	length of hash code
Y_i	=	*i*th input block	b	=	length of input block
f	=	compression algorithm			

- Break input message into equal-sized blocks
- Apply a compression function *f* iteratively to blocks Y_i
- Use output of previous stage CV_i as input to the next

Applications of Hash Functions

- **Message Authentication: Integrity + Source Authentication**
  - Integrity to ensure that the message has not been modified in transit
  - Source Authentication: the receiver is assured of the origin of the message
  - Encrypt hash using a shared secret key
- **Digital Signatures**: Encrypt hash with private key to ensure Integrity + Source Authentication + Non-repudiation
- **Password storage**: Stored hashed password with a salt.
  - When a user enters a password, the hash of that password is compared to the stored hash value for verification
  - Hackers can not get password from storage.
- More!
  - Detect errors in file transfers.
  - **Pseudorandom number generation**: Hash an IV, Hash the hash, …, repeat

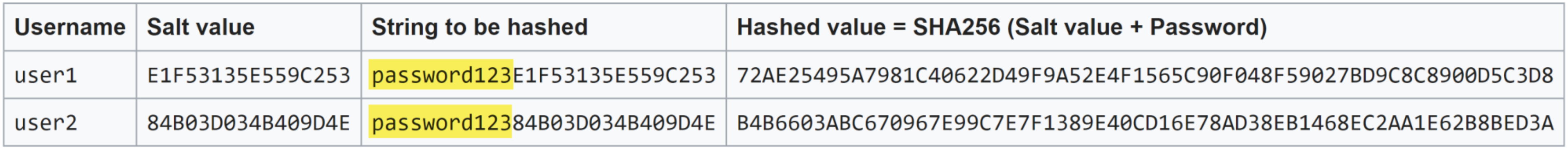
Application: File Transmission



https://en.wikipedia.org/wiki/MD5

Application: Password Storage

- When designing an application that stores passwords, don't store them in plaintext
  - If someone steals your password file, then they have all the user passwords!
  - Store **slated hashes** instead
  - A **salt** is random data (similar to nonce) that is concatenated with the password then hashed.
  - The primary function of salts is to defend against dictionary attacks.



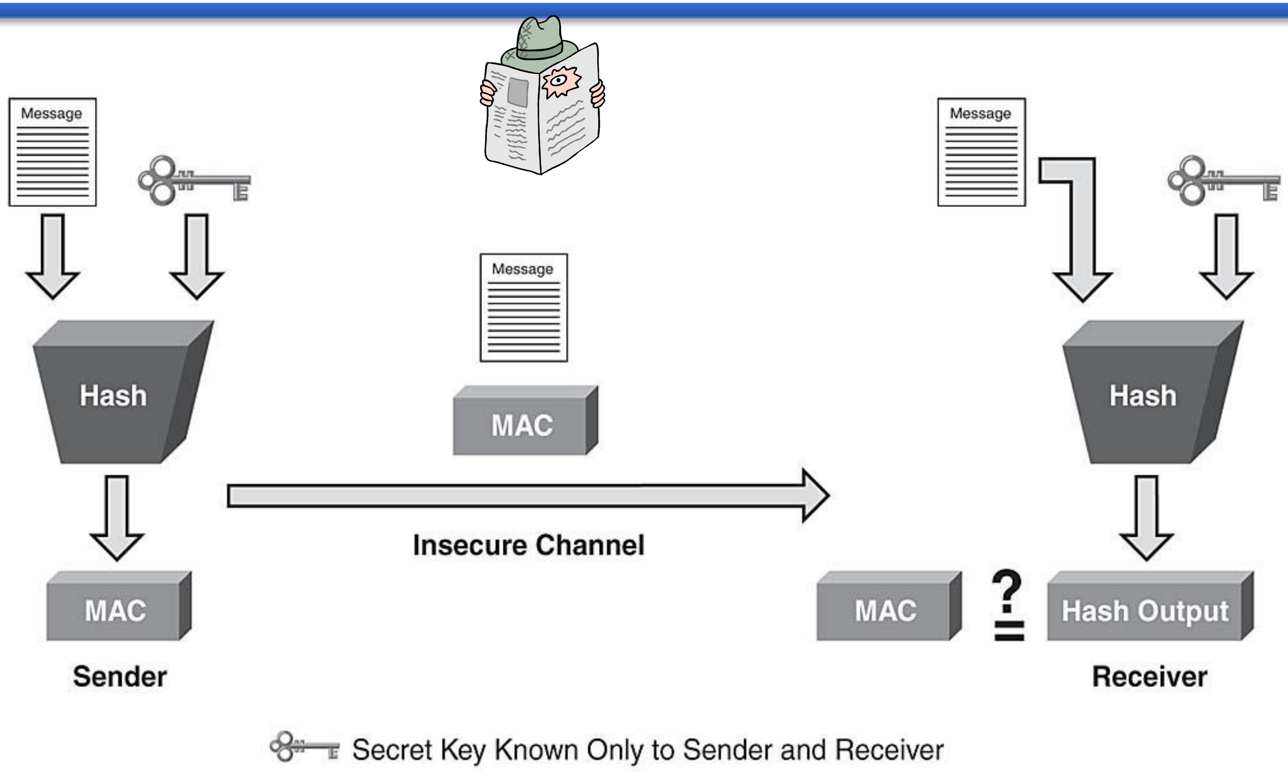
Username	Salt value	String to be hashed	Hashed value = SHA256 (Salt value + Password)
user1	E1F53135E559C253	password123E1F53135E559C253	72AE25495A7981C40622D49F9A52E4F1565C90F048F59027BD9C8C8900D5C3D8
user2	84B03D034B409D4E	password12384B03D034B409D4E	B4B6603ABC670967E99C7E7F1389E40CD16E78AD38EB1468EC2AA1E62B8BED3A

=> Using a different slat produces a different hash

Message Authentication Code (MAC)

✓ AUTHENTICITY

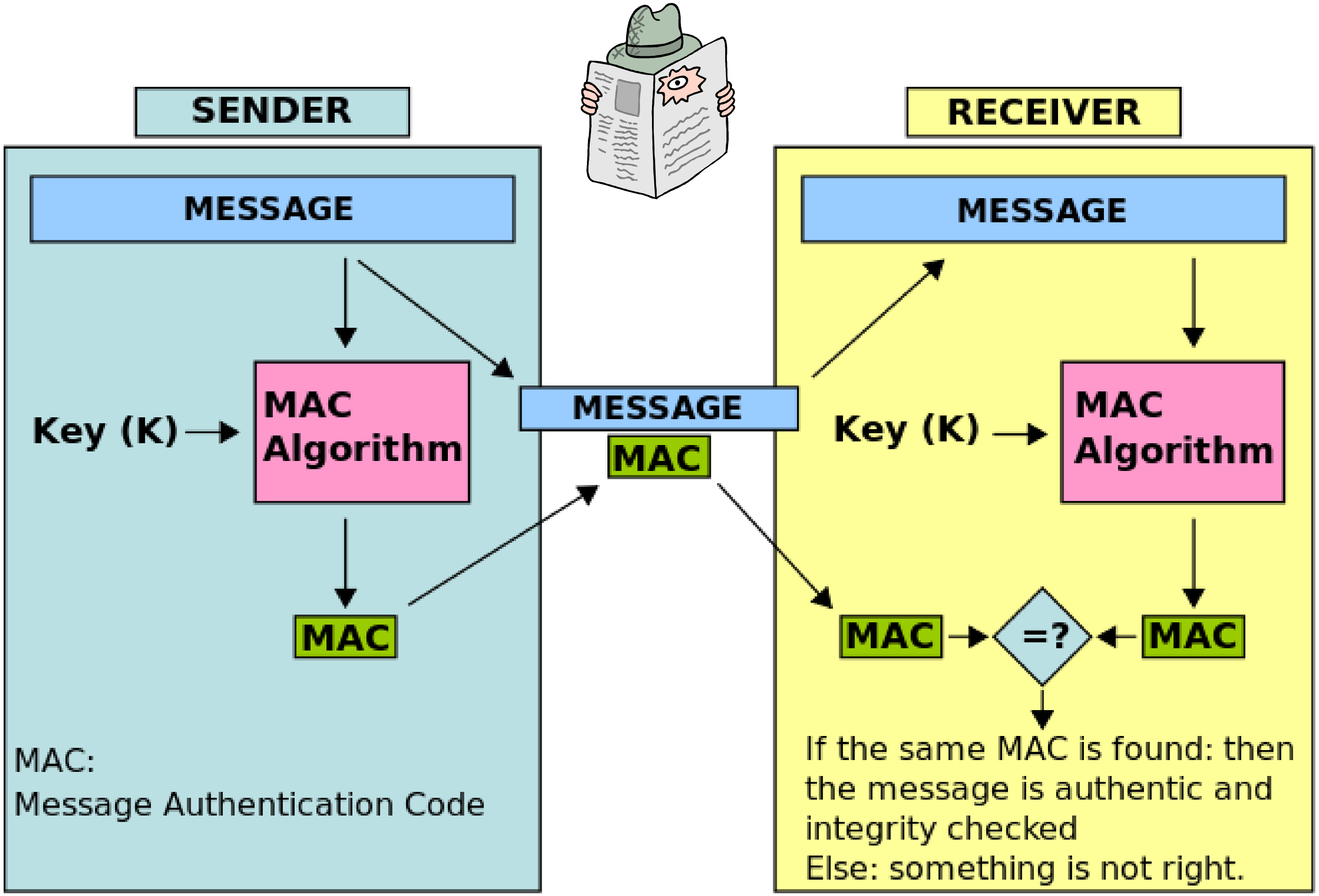
✓ INTEGRITY



< Back

Principle of Message Authentication Code (MAC)

- MACs append an authentication tag to a message
- MACs use a symmetric key ***k*** for generation and verification



MAC:
Message Authentication Code

If the same MAC is found: then the message is authentic and integrity checked
Else: something is not right.

Properties of MAC

1. **Cryptographic hash**

   A MAC generates a cryptographically secure authentication tag for a given message. MAC accept a message of arbitrary length and generate fixed-size authentication tag.

2. **Symmetric**

   MACs are based on secret symmetric keys. The signing and verifying parties must share a secret key.

3. **Message integrity**

   MACs providemessage integrity: Any manipulations of a message during transit will be detected by the receiver.

(An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key)

4. **Message authentication**

   The receiving party is assured of the origin of the message.

5. **No nonrepudiation**

   Since MACs are based on symmetric principles, they do not provide nonrepudiation.

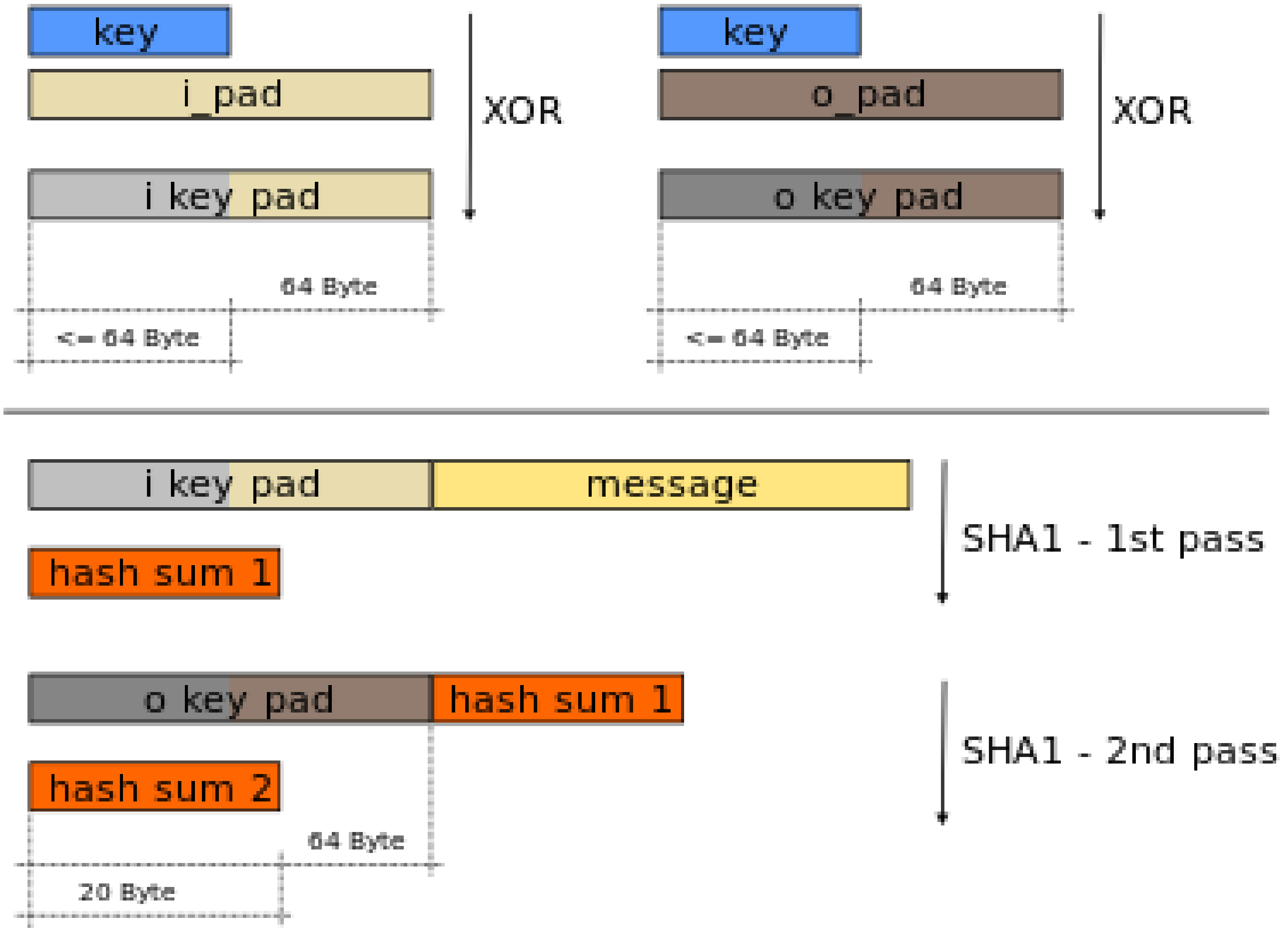
MACs from Hash Functions

- MAC can be realized with cryptographic hash functions (e.g., SHA-1)
- Basic idea: Key is hashed together with the message
- Two possible constructions:
  - secret prefix H(K | m)
  - secret suffix H(m | K)
- Both are vunerable to attacks
- For better security combine secret prefix and suffix as done by HMAC (see next slide)

HMAC

- Proposed by Mihir Bellare, Ran Canetti and Hugo Krawczyk in 1996
- **HMAC** (**keyed-hash message authentication code**) uses a *cryptographic hash function* and a *secret key*.
- It may be used to simultaneously verify both the *data integrity* and the *authentication* of a message.
- Any cryptographic hash function, such as SHA256 or SHA-3, can be used to compute the HMAC (e.g. HMAC-SHA256 or HMAC-SHA3)

HMAC: HMAC-SHA1 generation example



opad - the outer padding: 0x5c5c5c…5c5c (one-block-long constant)
ipad - the inner padding: 0x363636…3636 (one-block-long constant)

HMAC Algorithm

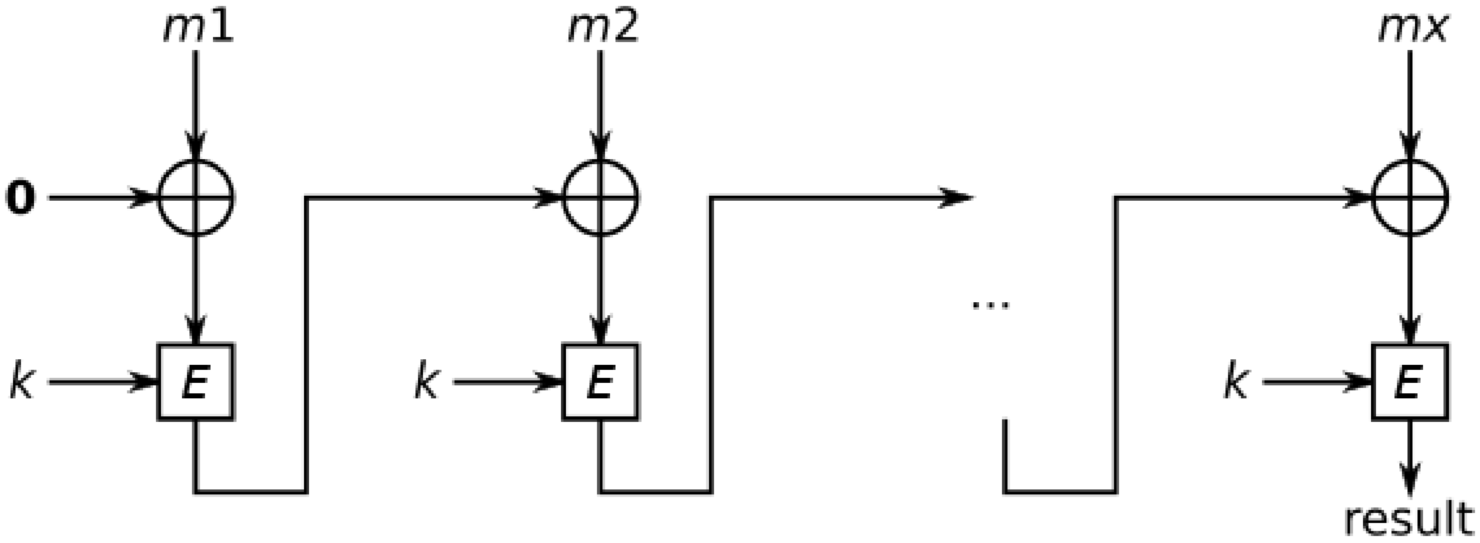
- HMAC uses two passes of hash computation:
  - The secret key is first used to derive two keys:
    - ***opad*** is the outer padding, consisting of repeated bytes, valued 0x**5c**, up to the block size
    - ***ipad*** is the inner padding, consisting of repeated bytes, valued 0x**36**, up to the block size.
  - The first pass of the algorithm produces an internal hash derived from the message and the inner key.
  - The second pass produces the final HMAC code derived from the inner hash result and the outer key.

$$\mathrm{HMAC}(K, m) = \mathrm{H}\Big(\big(K' \oplus opad\big) \parallel \mathrm{H}\big(\big(K' \oplus ipad\big) \parallel m\big)\Big)$$

$$K' = \begin{cases} \mathrm{H}(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

MAC from Block Ciphers: CBC-MAC

- Cipher Block Chaining Message Authentication Code (CBC-MAC) constructs a MAC using a block cipher in CBC mode.
  - Create a chain of blocks such that each block depends on the encryption of the previous block.
  - This interdependence ensures that a change to any of the plaintext bits will cause the final encrypted block to change in a way that cannot be predicted.
- The IV is initialized with zeros. MAC is the encrypted last block.



CBC-MAC

- MAC Generation
  - Divide the message *m* into blocks m_i
  - Compute first iteration $c_1 = e_k(m_1 \oplus IV)$. IV is initialized with zeros
  - Compute $c_i = e_k(m_i \oplus c_{i-1})$ for the next blocks
  - Final block is the MAC value: *m* =$\mathrm{MAC}_k(m) = c_n$

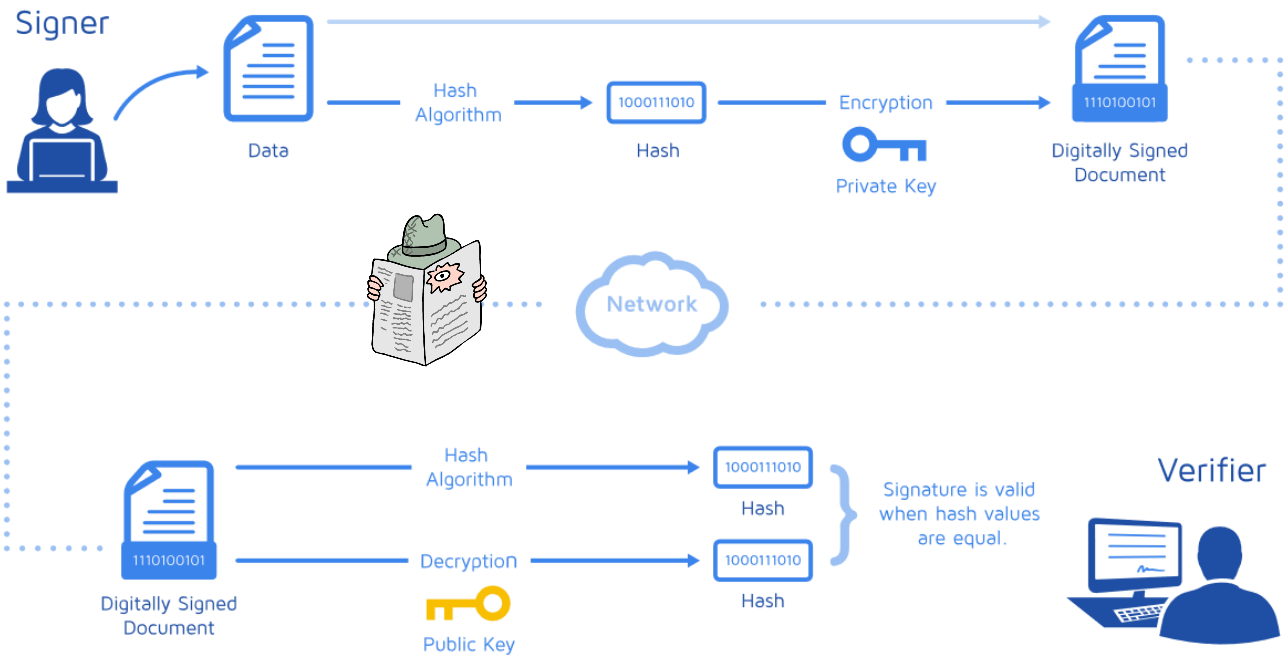
- MAC Verification
  - Repeat MAC computation (*m'*)
  - Compare results:In case *m'*= *m*, the message is verified as correct
  - In case $m' \neq m$, the message and/or the MAC value *m* have been altered during transmission

MAC Summary

- MACs provide two security services, *message integrity and message authentication,* using symmetric techniques. MACs are widely used in protocols.
- Both of these services also provided by digital signatures, but MACs are much faster as they are based on symmetric algorithms.
- MACs do not provide nonrepudiation.
- In practice, MACs are either based on block ciphers or on hash functions.
- HMAC is a popular and very secure MAC, used in many practical protocols such as TLS.

Digital Signature

- ✓ AUTHENTICITY
- ✓ INTEGRITY
- ✓ Non-Repudiation



< Back

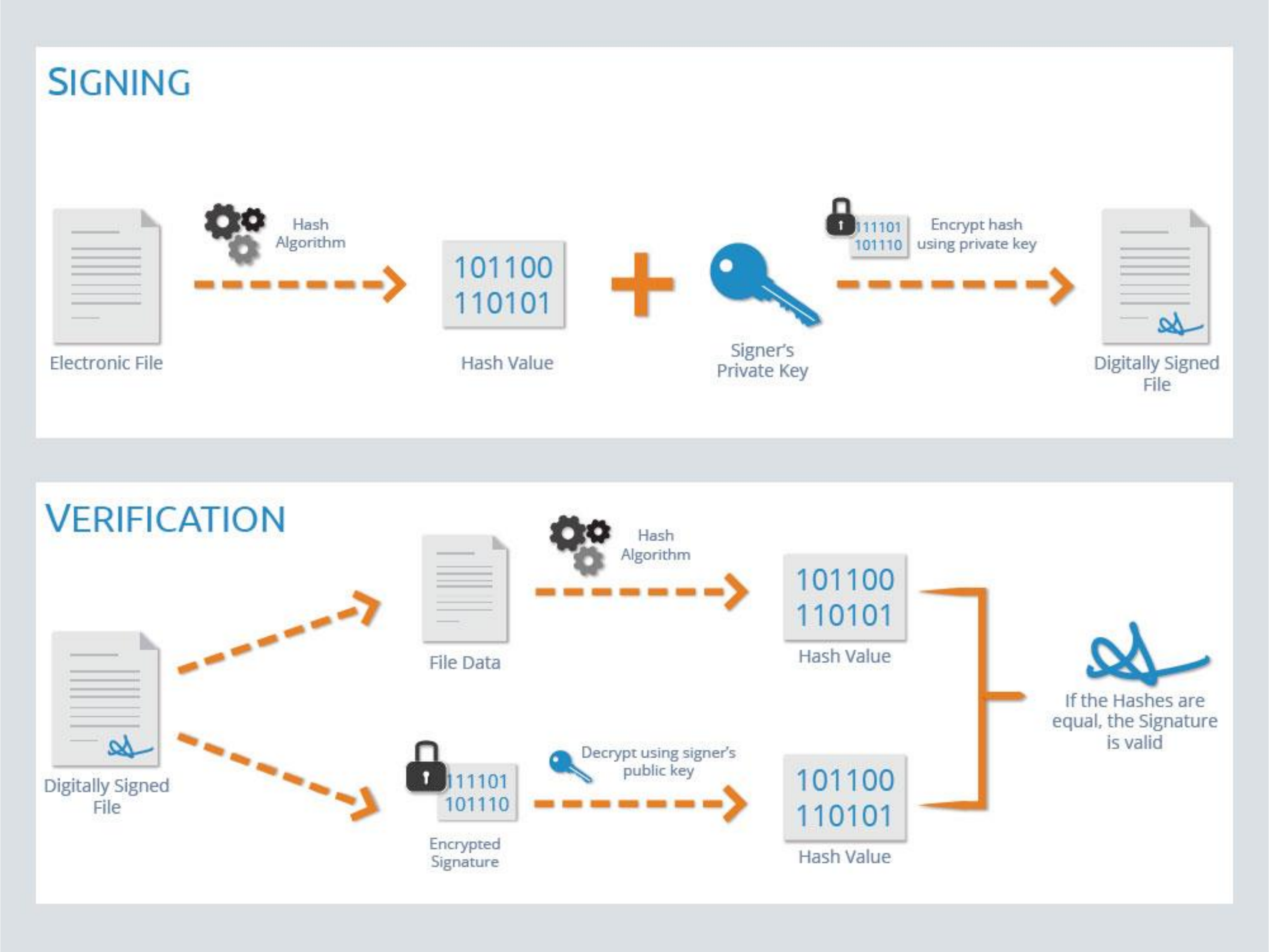
Motivation

- Alice orders a pink car from the car saler Bob
- After seeing the pink car, Alice states that she has never ordered it
- How can Bob prove towards a judge that Alice has ordered a pink car? (And that he did not fabricate the order himself)

$\Longrightarrow$ Symmetric cryptography fails because both Alice and Bob can be malicious

$\Longrightarrow$ Can be achieved with public-key cryptography

Basic Principle of Digital Signatures



Main idea

- For a given message *m,* a digital signature is appended to the message (just like a conventional signature).
- Only the person with the private key should be able to generate the signature.

⇒The signature is realized as a function with the message *m* and the private key as input

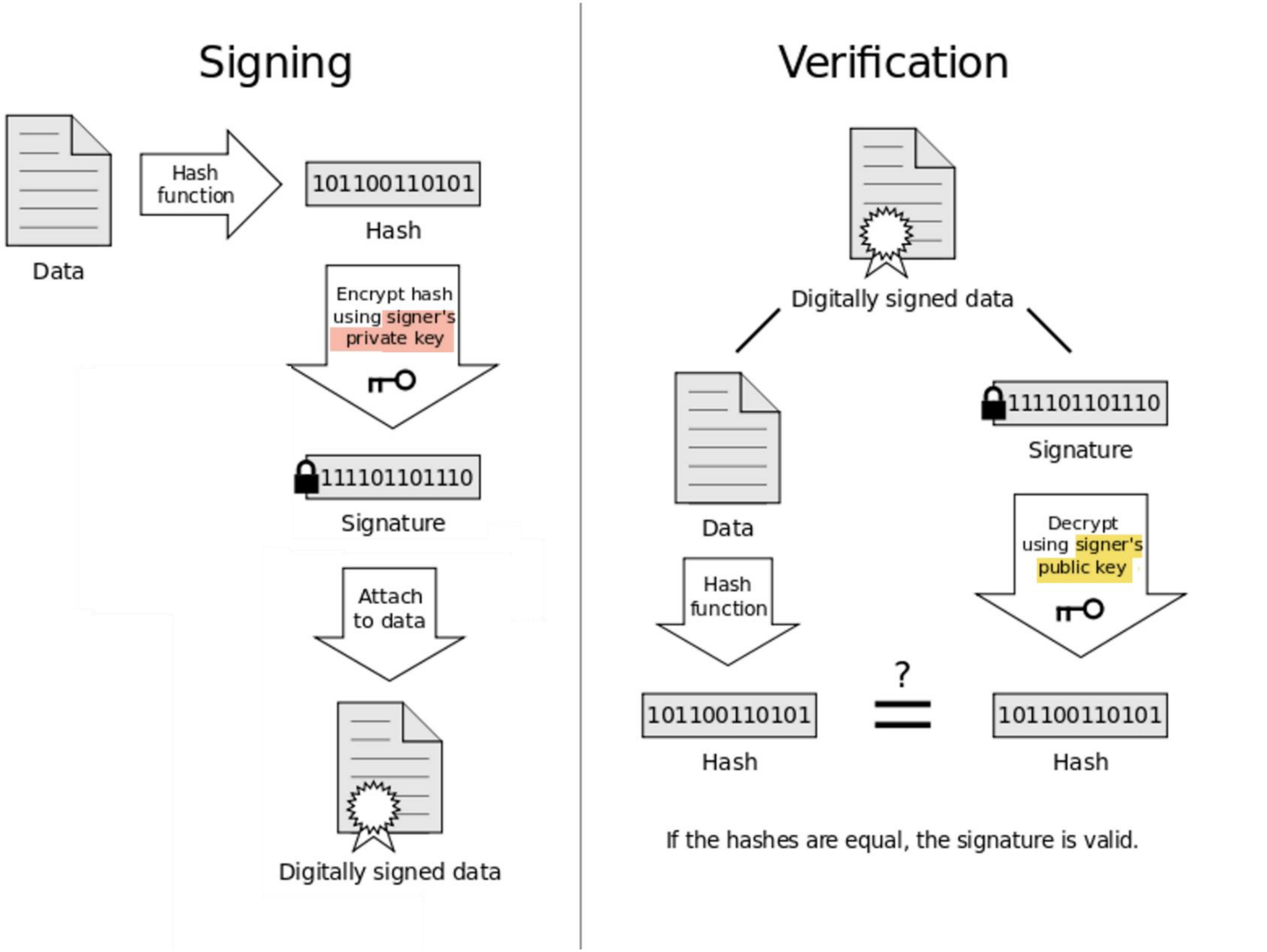
⇒The public key and the message *m* are the inputs to the verification function

- An attacker who wishes to alter the message would need to know the user’s private key

DC Core Security Services

1. **Integrity:** Ensures that a message has not been modified in transit.
2. **Message Authentication:** Ensures that the sender of a message is authentic. An alternative term is *data origin authentication*.
3. **Non-repudiation:** Ensures that the sender of a message can not deny the creation of the message. (e.g. order of a pink car)

RSA signature process



Bob can verify that Alice sent the message (i.e., **non-repudiation**) and that the message has not been modified (i.e., **integrity**)

Summary

- Hash functions are used to compute a digest of a message. Must take variable size input, produce fixed size pseudorandom output, be efficient to compute
- Cryptographic hash functions should be one-way and collision resistant
- Cryptographic hashes are used for message authentication, digital signatures, password storage, detecting errors in file transfers, … etc.

Resources

- Cryptographic hash function

https://simple.wikipedia.org/wiki/Cryptographic_hash_function

- HMAC

https://en.wikipedia.org/wiki/HMAC

- Digital signature

https://en.wikipedia.org/wiki/Digital_signature