MSc - Cybersecurity

CMT310: Developing Secure Systems and Applications

Digital Signature Algorithms

Dr Neetesh Saxena
saxenan4@cardiff.ac.uk

Learning Outcomes

- Hash – salt and iteration count
- ECDH Key Exchange
- Digital Signature Algorithms
  - RSA
  - DSA
  - ECDSA

Passwords: How can Passwords be Stored?

- Filing System
  - Clear text

- Dedicated Authentication Server
  - Clear text

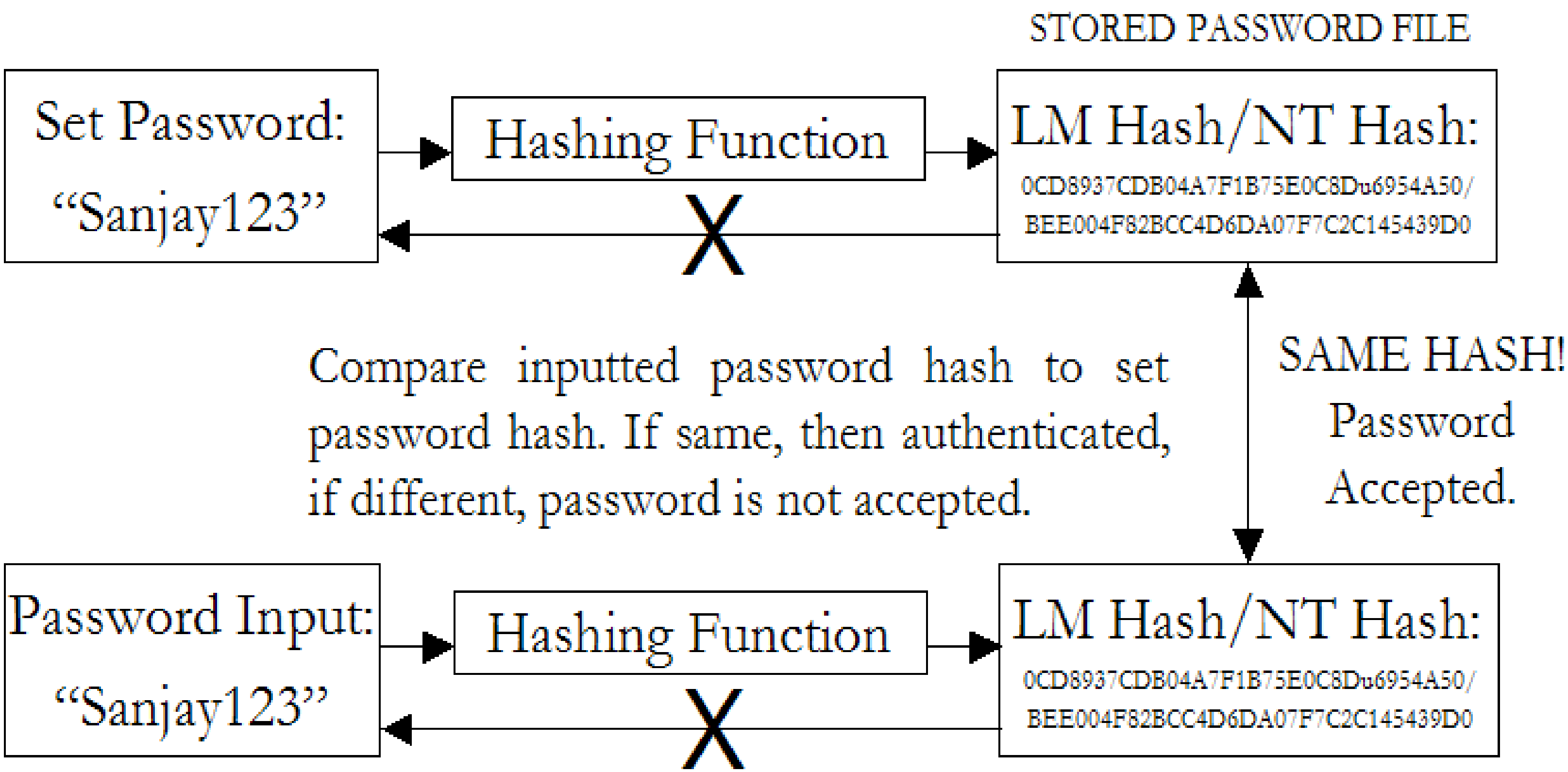
- Encrypted
  - Password + Encryption = bf4ee8HjaQkbw

- Hashed
  - Password + Hash function = aad3b435b51404eeaad3b435b51404ee

- Salted Hash
  - (Username + Salt + Password) + Hash function = e3ed2cb1f5e0162199be16b12419c012

Passwords: Hashing

- Usually stored as hashes (not plaintext)
  - Plaintext is converted into a message digest through use of a hashing algorithm (i.e., MD5, SHA)

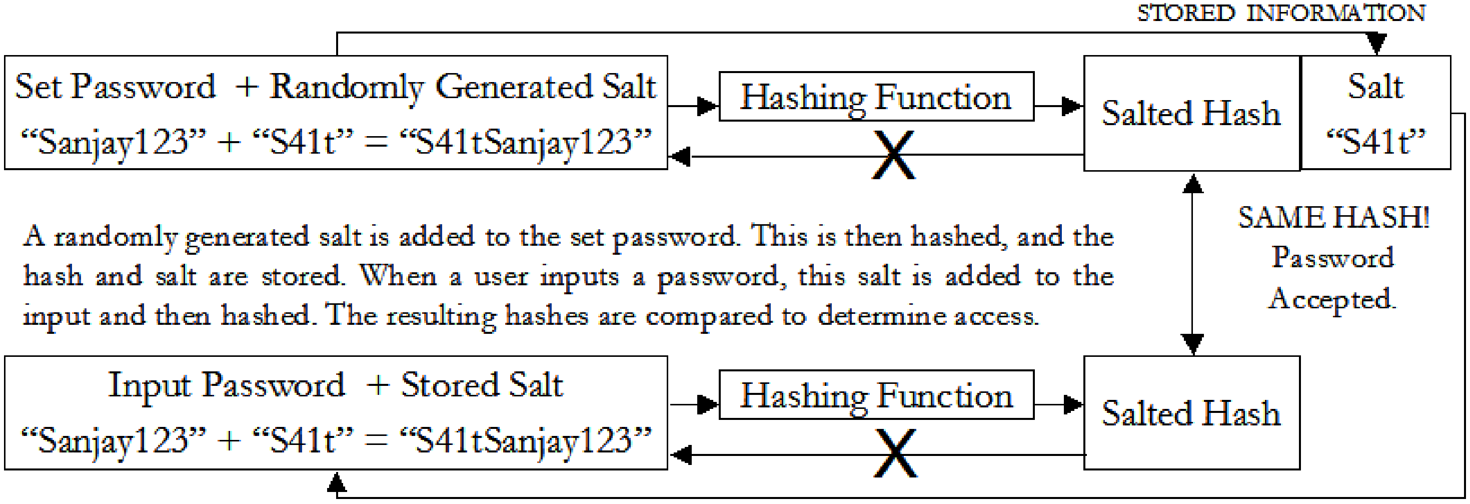


*LM Hash - LAN Manager hash, NT hash - NT LAN Manager

Passwords: Cracking Protection - Salting

- Salting requires adding a random piece of data and to the password before hashing it.
  - The same string will hash to different values at different times
  - Users with same password have different entries in the password file
  - Salt is stored with the other data as a complete hash

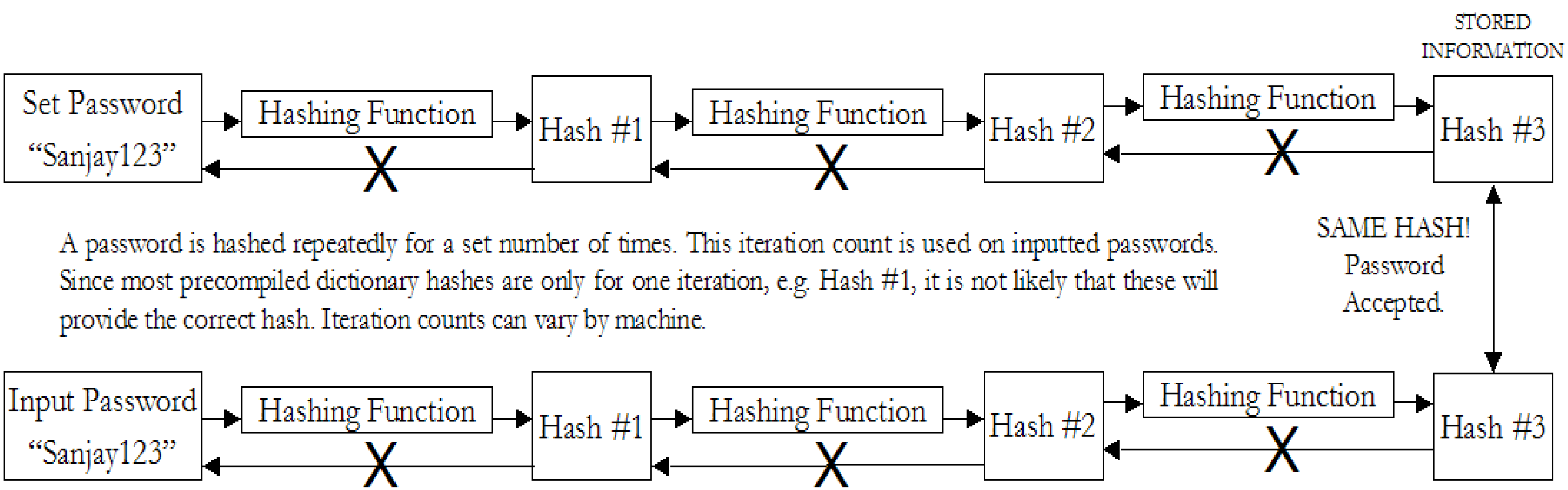
- Hacker has to get the salt add it to each possible word and then rehash the data prior to comparing with the stored password.



Passwords: Cracking Protection - Iteration Count

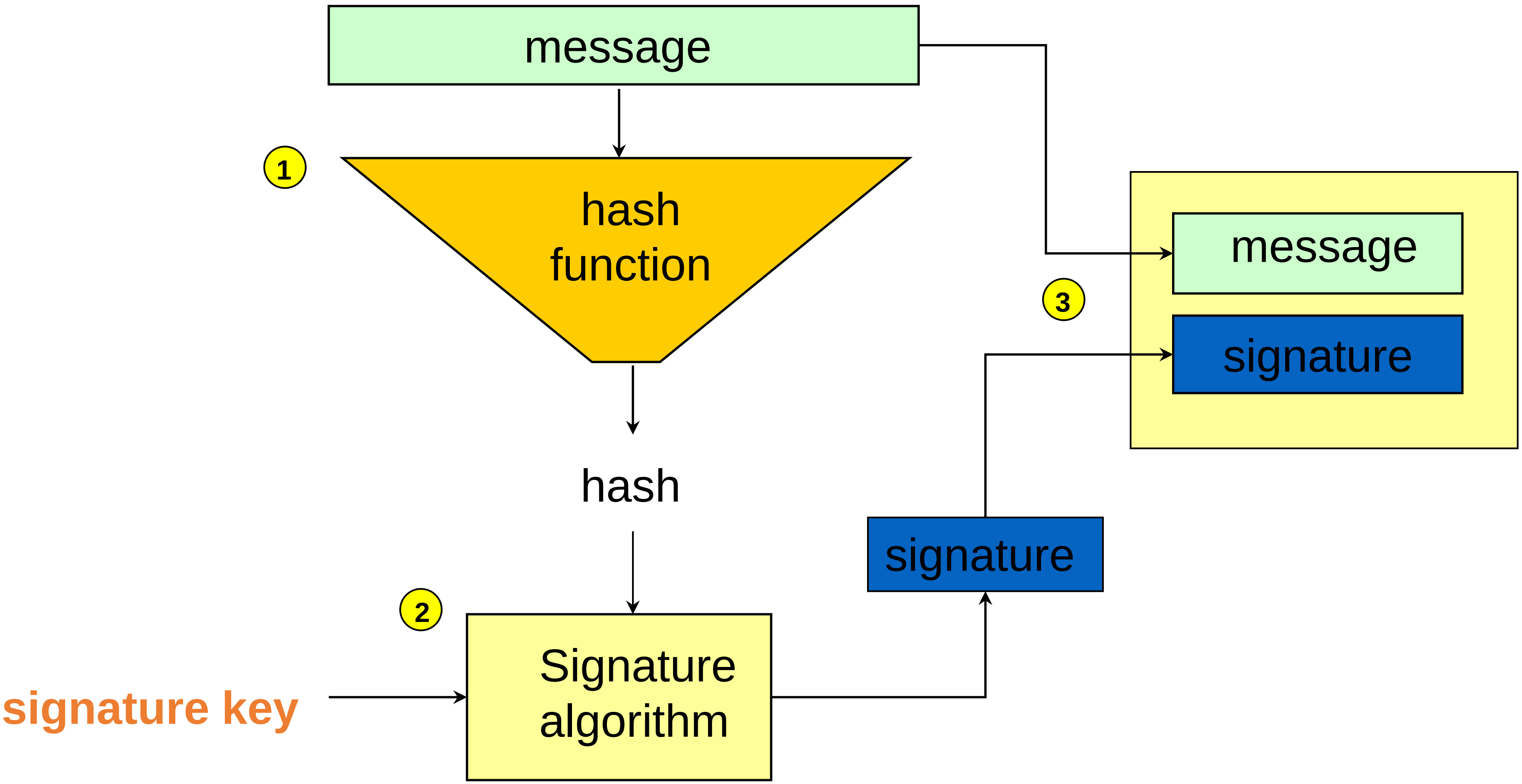
- The same password can be rehashed many times over to make it more difficult for the hacker to crack the password.

- The precompiled dictionary hashes are not useful since the iteration count is different for different systems.



Digital Signature Algorithms

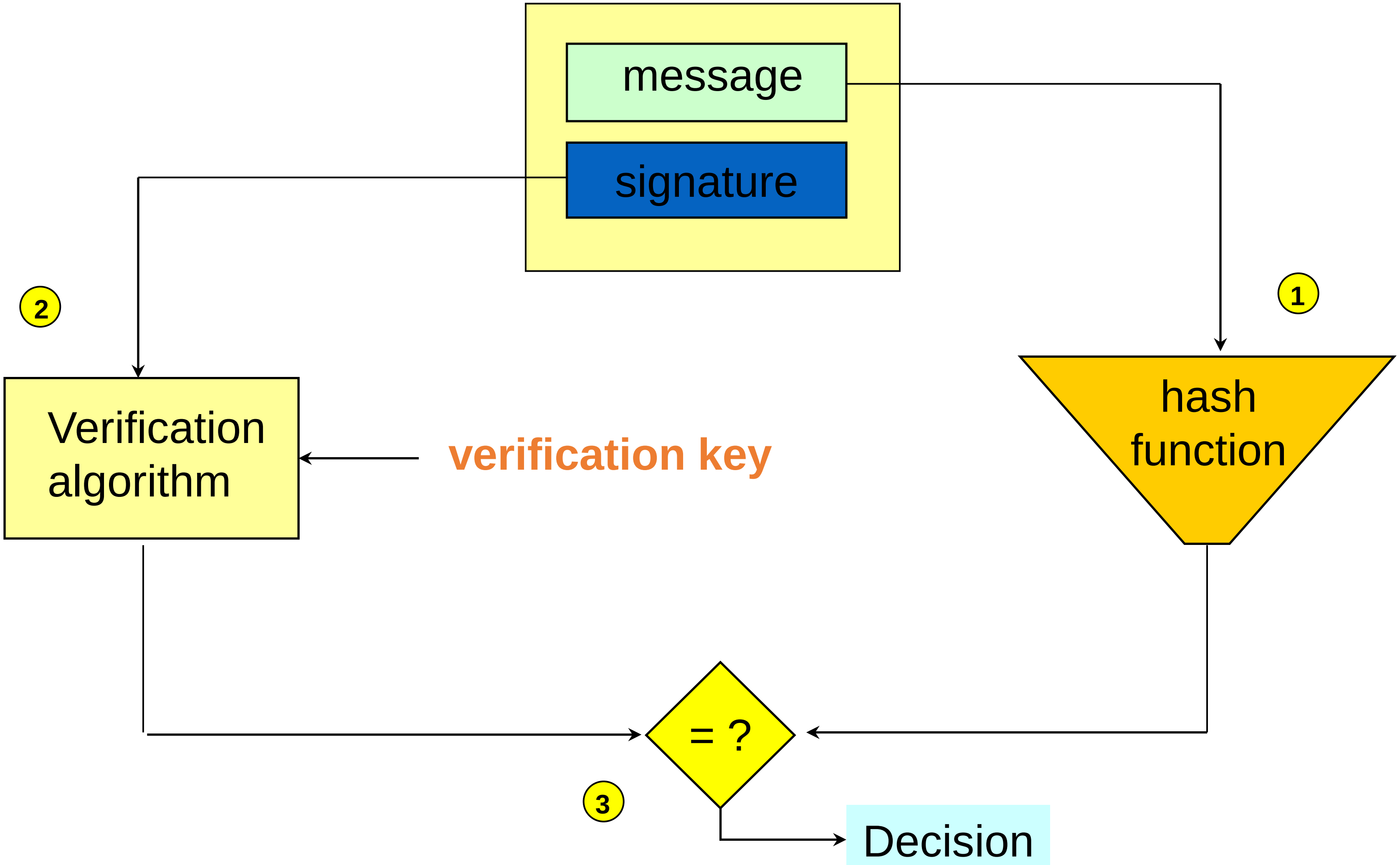
Creating an RSA Signature



There are two reasons why a message is hashed before it is signed using RSA.

What are they?

Verifying an RSA Signature



RSA is Special

You cannot obtain a digital signature scheme by swapping the roles of the private and public keys of any public key cipher system.

You cannot obtain a public key cipher system by swapping the roles of the signature and verification keys of any digital signature scheme.

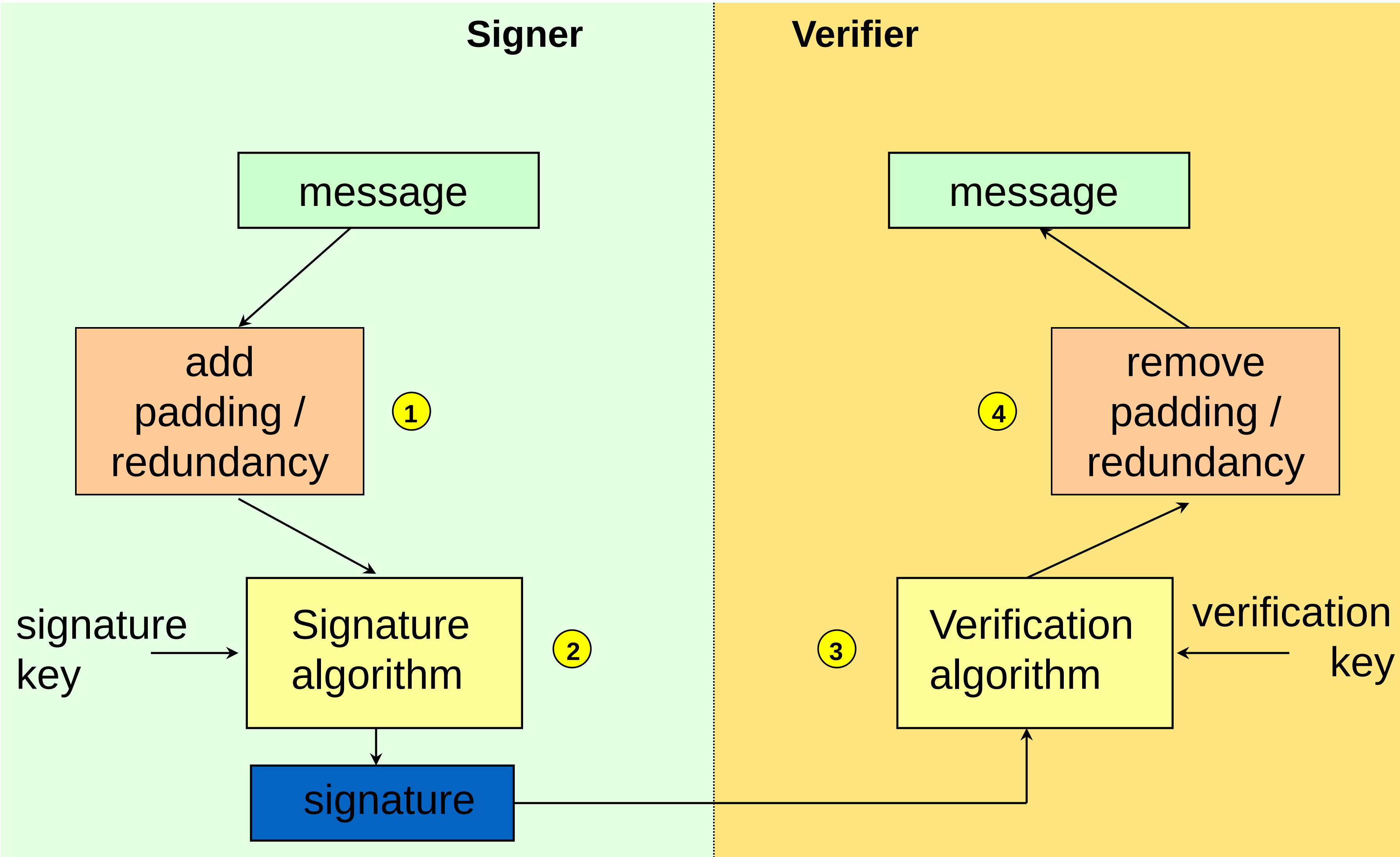
Task!

Identify the special property of RSA that allows it to be used as both an encryption and a signature algorithm.

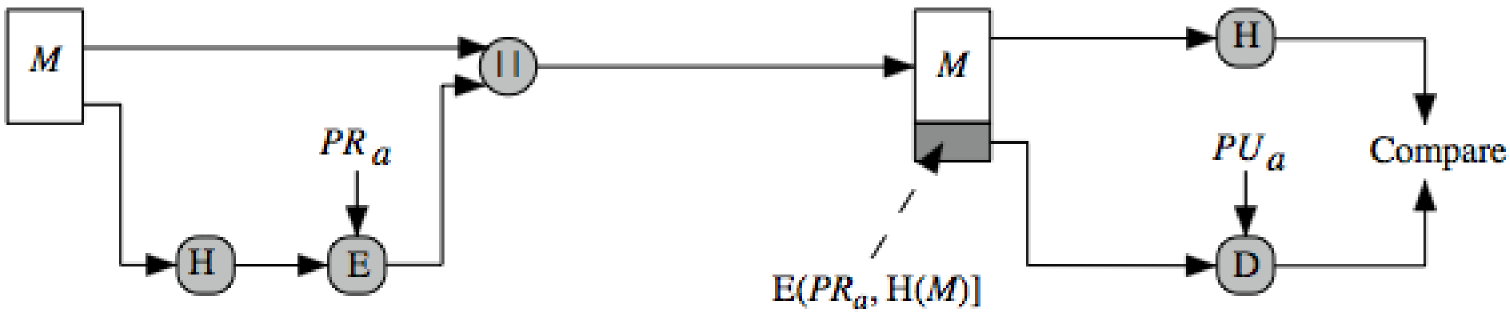
In real applications you should avoid using the same RSA key pair for both encryption and for digital signatures.

Why?

RSA Signatures with Message Recovery



DSS vs RSA Signatures

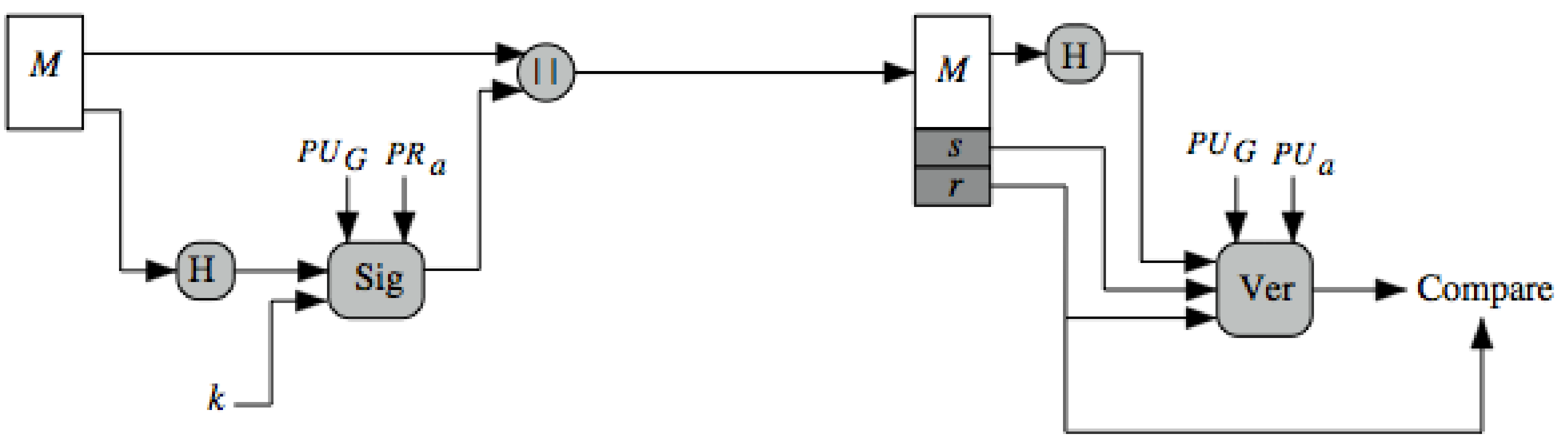


(a) RSA Approach

k - random number

PU_G - global public key

Output of Ver = r



(b) DSS Approach

Digital Signature Algorithm (DSA)

- Creates a 320 bit signature
- With 512-1024 bit security (e.g., authentication key)
- Smaller and faster than RSA
- A digital signature scheme only
- Security depends on difficulty of computing discrete logarithms

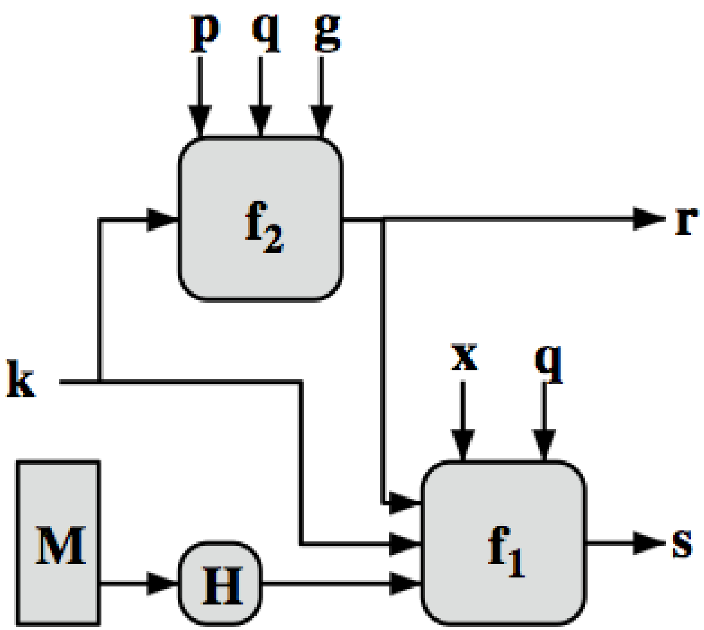
DSA Overview

global values (p,q,g):

- 160-bit prime number q
- a large prime p with $\mathbf{2^{L-1} < p < 2^{L}}$ where L= 512 to 1024 bits
- $\mathbf{g = h^{(p-1)/q}}$

- Private & public keys:
  - choose random private key: **x<q**
  - compute public key: $\mathbf{y = g^{x} \bmod p}$

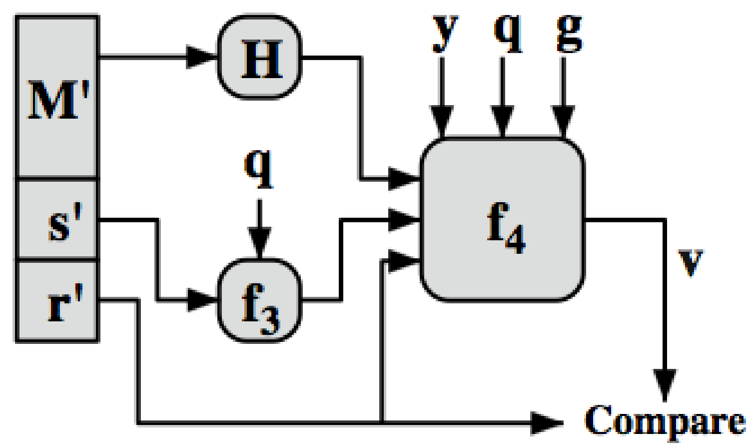
random signature key **k, k<q**



$$s = f_1(H(M), k, x, r, q) = (k^{-1}(H(M) + xr)) \bmod q$$

$$r = f_2(k, p, q, g) = (g^{k} \bmod p) \bmod q$$

(a) Signing



$$w = f_3(s', q) = (s')^{-1} \bmod q$$

$$v = f_4(y, q, g, H(M'), w, r')$$

$$= ((g^{(H(M')w) \bmod q}\, y^{r'w \bmod q}) \bmod p) \bmod q$$

(b) Verifying

What is Elliptic Curve Cryptography (ECC)?

- Elliptic curve cryptography (ECC) is a **public-key** cryptosystem just like RSA and El Gamal.

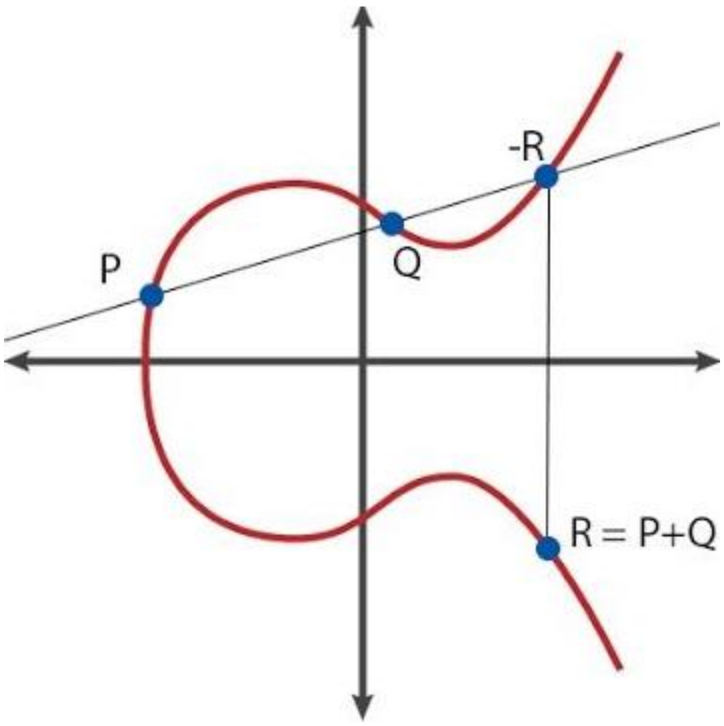
- Every user has a **public** and a **private** key.
  - Public key is used for encryption/signature verification.
  - Private key is used for decryption/signature generation.

- Elliptic curves are used as an extension to other current cryptosystems.
  - Elliptic Curve Diffie-Hellman Key Exchange
  - Elliptic Curve Digital Signature Algorithm

Using Elliptic Curves in Cryptography

- The central part of any cryptosystem involving elliptic curves is the **elliptic group**.

- All public-key cryptosystems have some underlying mathematical operation.
  - RSA has exponentiation (raising the message or ciphertext to the public or private values)
  - ECC has point multiplication (repeated addition of two points).



Generic Procedures of ECC

- Both parties agree to some publicly-known data items
  - The **elliptic curve equation**
    - values of ***a*** and ***b***
    - prime, ***p***
  - The **elliptic group** computed from the elliptic curve equation
  - A **base point**, G (or B) = (X, Y), taken from the elliptic group
    - Similar to the generator used in current cryptosystems

- Each user generates their public/private key pair
  - Private Key = an integer, x, selected from the interval [1, p-1]
  - Public Key = product, Q, of private key and base point
    - (Q = x*G)

elliptic curve prime field - p
elliptic curve binary field -2^m

Elliptic Curve Cryptography

- Components

Private Key	Public Key	Set of Operations	Domain Parameters (Predefined constants)
A random number	Point on a curve = Private Key * G	These are defined over the curve $\mathbf{y^2 = x^3 + ax + b}$**,** where $\mathbf{4a^3 + 27b^2 \neq 0}$	G, a, b

Discrete Logarithm Problem (DLP)

- Let P and Q be two points on the elliptic curve
  - Such that Q = kP, where k is a scalar value
- DLP: Given P and Q, find k?
  - If k is very large, it becomes computationally infeasible
- The security of ECC depends on the difficulty of DLP - ECDLP
- Main operation in ECC is Point Multiplication

Point Multiplication

- Point Multiplication is achieved by two basic curve operations:

1. Point Addition, R = P + Q
2. Point Doubling, P = 2P

Example:

If k = 23; then, kP = 23*P

= 2(2(2(2P) + P) + P) + P

ECC Diffie-Hellman

- **Public:** Elliptic curve and point G=(X,Y) on curve
- **Secret:** Alice's a and Bob's b

a(X,Y) →

← b(X,Y)

Alice

Bob

- Alice computes a(b(X,Y))
- Bob computes b(a(X,Y))
- These are the same since ab(X,Y) = ba(X,Y)

Elliptic Curve Diffie-Hellman Exchange

- Alice and Bob want to agree on a shared key.
  - Alice and Bob compute their public and private keys.
    - Alice
      - Private Key = a
      - Public Key = P_A = a * G
    - Bob
      - Private Key = b
      - Public Key = P_B = b * G
  - Alice and Bob send each other their public keys.
  - Both take the product of their private key and the other user's public key.
    - Alice → K_{AB} = a(bG)
    - Bob → K_{AB} = b(aG)
    - **Shared Secret Key = K_{AB} = abG**

Why Use ECC?

- How do we analyze Cryptosystems?
  - How difficult is the underlying problem that it is based upon
    - RSA – Integer Factorization
    - DH – Discrete Logarithms
    - ECC - Elliptic Curve Discrete Logarithm problem

  - How do we measure difficulty?
    - We examine the algorithms used to solve these problems

Security of ECC

- To **protect** a 128-bit AES key it would take:
  - RSA Key Size: 3072 bits
  - ECC Key Size: 256 bits

- How do we strengthen RSA?
  - Increase the key length

NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	AES KEY SIZE (Bits)
163	1024	
256	3072	128
384	7680	192
512	15 360	256

Supplied by NIST to ANSI X9F1

Applications of ECC

- Many devices are small and have limited storage and computational power

- Where can we apply ECC?
  - Wireless communication devices
  - Smart cards
  - Web servers that need to handle many encryption sessions
  - **Any application where security is needed but lacks the power, storage and computational power that is necessary for current cryptosystems**

Benefits of ECC

- Same benefits of the other cryptosystems: confidentiality, integrity, authentication and non-repudiation but…

- Shorter key lengths
  - Encryption, Decryption and Signature Verification speed up
  - Storage and bandwidth savings

Summary of ECC

- "**Hard problem**" analogous to discrete log
  - Q=kP, where Q,P belong to a prime curve

    given k,P → "easy" to compute Q

    given Q,P → "hard" to find k
  - known as the elliptic curve logarithm problem
    - k must be large enough

- ECC security relies on elliptic curve logarithm problem
  - compared to factoring, can use much smaller key sizes than with RSA

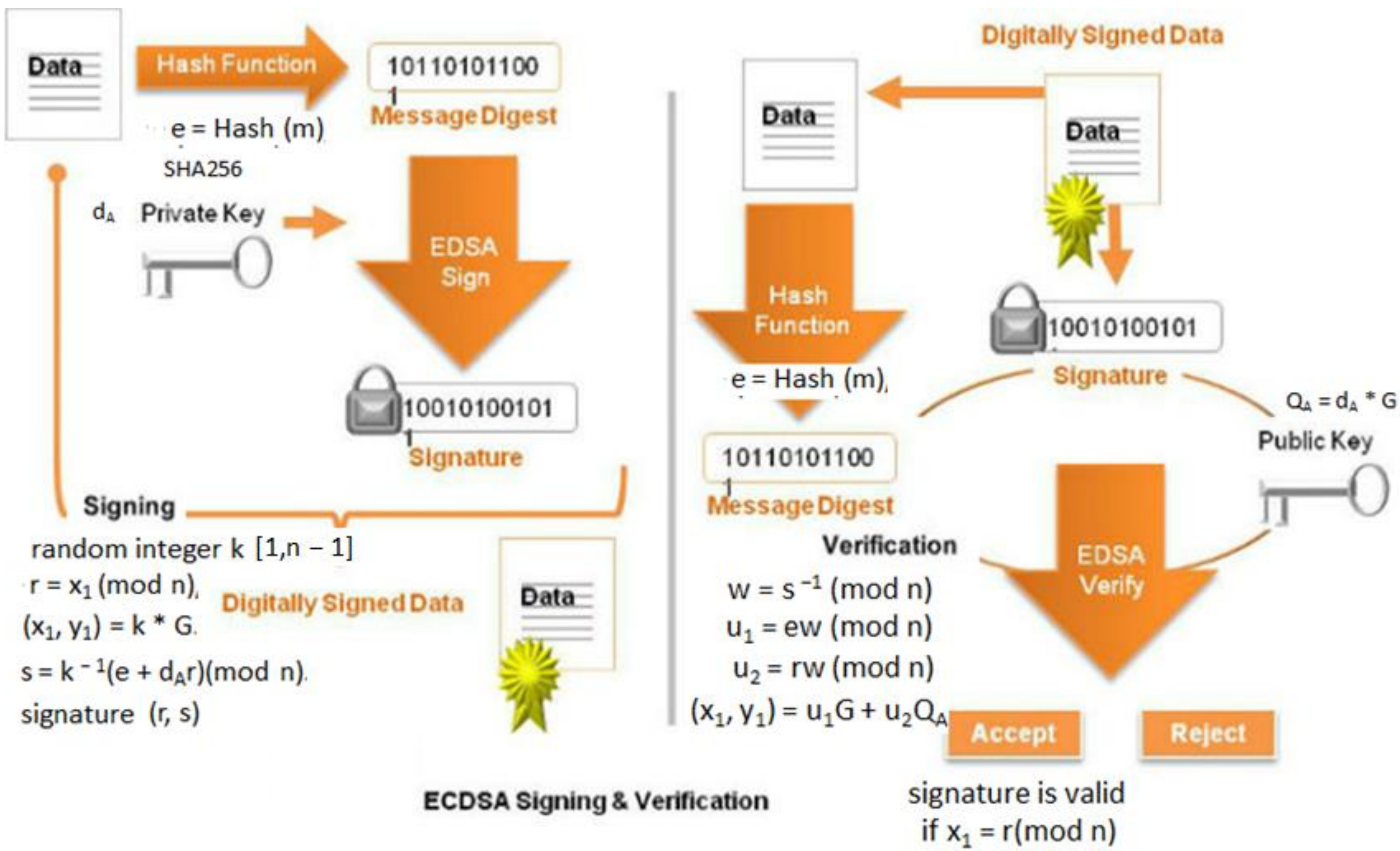
ECDSA 224 and 256

- Elliptic Curve Digital Signature (ECDSA) is introduced in the current IEEE 802.21d in two options:
    - ECDSA 224
    - ECDSA 256

- ECDSA 224 implies
    - ECDSA uses a curve with 224-bit group size with any hash function (in SHA-2) using SHA-224.

Curves with a size of less than 224 bits should not be used.
You should strongly consider using curves of at least 224 bits.

ECDSA - Elliptic Curve Digital Signature Algorithm



DSA vs. ECDSA vs. RSA

- Bit size of the public key for ECDSA is about twice the size of the security level, in bits.
- Example:
  - at a security level of 80 bits (requires a max 2^{80} operations to find the private key) the size of an ECDSA public key would be 160 bits, whereas the size of a DSA public key is at least 1024 bits.

- Signature size is same for DSA and ECDSA: approx. 4t bits,
  - where t is the security level measured in bits, that is, about 320 bits for a security level of 80 bits.

- In PKCS#1, the RSA standard describes a signature should always of the size of the modulus.
  - This means that for a 2048-bit modulus, all signatures have length exactly 256 bytes, never more, never less.