Introduction to applied cryptography

– Lecture 2

Omar Hasan

Books

- The content in the lectures is drawn from the following books:
  - **Applied Cryptography**, Second Edition – Bruce Schneier
  - **An Introduction to Cryptography**, version 8.0 – PGP Corporation

RSA

- A **public-key cryptosystem** named after its three inventors—Ron Rivest, Adi Shamir, and Leonard Adleman.
- RSA **supports authentication** of the participants (through digital signatures) as opposed to Diffie-Hellman, which does not.
- RSA gets its security from the **difficulty of factoring large numbers**.
- **Recovering the plaintext** from the public key and the ciphertext is conjectured to be equivalent to **factoring the product of two primes**.

RSA

- Let **e**, **d**, **n** be positive integers, and **p**, **q** be prime numbers, where **n = pq**, then:
  - **Public key:** (e, n)
  - **Private key:** (d, n)
- **Encryption:**

  $$C = E(M) = M^{e} \pmod{n}$$

- **Decryption:**

  $$M = D(C) = C^{d} \pmod{n}$$

RSA

Key Generation: Public key

- Choose two random large prime numbers, p and q, of equal length. **p = 47** and **q = 71**.
- Compute n: $n = pq$ ***n = 47 * 71 = 3337***
- Compute φ: $\varphi = (p - 1)(q - 1)$ ***φ = 46 * 70 = 3220***
- Choose a random integer e, such that e and φ are relatively prime. **e = 79**
  - In other words: $gcd(e, \varphi) = 1$ ***gcd(79, 3220) = 1***
  - Where, gcd: greatest common divisor
- Public key: **(e = 79, n = 3337)**

GCD calculator: https://planetcalc.com/323/

RSA

Key Generation: Private key

- Use the extended Euclidean algorithm to compute d, such that: $ed = 1 \pmod{\varphi}$
  - In other words: $d = e^{-1} \pmod{\varphi}$

$$\boldsymbol{d = 79^{-1} \pmod{3220} = 1019}$$

- Private key: **(d = 1019, n = 3337)**

Extended Euclidean algorithm, modular multiplicative inverse calculator: https://planetcalc.com/3298/
Modular exponentiation: https://planetcalc.com/8979/

RSA

Encryption

- The encryption function:

  $C = E(M) = M^{e} \pmod{n}$

Example

- **M = 688**
- Public key: **(e = 79, n = 3337)**
- Encryption:

  $\boldsymbol{C = E(688) = 688^{79} \pmod{3337} = 1570}$

RSA

Decryption

- The decryption function:

$$M = D(C) = C^{d} \ (mod\ n)$$

Example

- **C = 1570**
- Private key: **(d = 1019, n = 3337)**
- Decryption:

$$\boldsymbol{M = D(1570) = 1570^{1019} \ (mod\ 3337) = 688}$$

RSA

Encryption of larger messages

- To encrypt a large message m, first break it into numerical **blocks smaller than n**.
- If **n has 100 digits**, then each message block, $\mathbf{m_i}$, should be **under 100 digits long**.
- The encrypted message, c, will be made up of similarly sized message blocks, $\mathbf{c_i}$, of about the same length.

RSA

Example

- To encrypt the message:

  $m = 6882326879666683$

- First break it into small blocks. The message is split into six blocks, m_i :

  $m_1 = 688$

  $m_2 = 232$

  $m_3 = 687$

  $m_4 = 966$

  $m_5 = 668$

  $m_6 = 003$

RSA

Example

- The first block is encrypted as

  $688^{79} \bmod 3337 = 1570 = c_1$

- Performing the same operation on the subsequent blocks generates an encrypted message:

  c = 1570 2756 2091 2276 2423 158

RSA

Decryption

- To decrypt the entire message, decrypt each encrypted block c_i.

Example

- Ciphertext:

  c = 1570 2756 2091 2276 2423 158

- Decrypting the message requires performing exponentiation using the decryption key of 1019, so

  $1570^{1019} \bmod 3337 = 688 = m_1$

- The rest of the message can be recovered in this manner.

RSA

Exercise

- Ciphertext:

  c = 564 648 324 243 696 562 331 689

- Public key: **(e = 11, n = 703)**
- Private key: **(d = 59, n = 703)**
- The decryption function:

$$M = D(C) = C^{d} \ (\mathrm{mod}\ n)$$

- ASCII:

65	A	66	B	67	C	68	D	69	E
70	F	71	G	72	H	73	I	74	J
75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T
85	U	86	V	87	W	88	X	89	Y
90	Z								

https://www.cs.drexel.edu/~jpopyack/IntroCS/HW/RSAWorksheet.html

RSA

Solution

564	648	324	243	696	562	331	689
$564^{59} \pmod{703}$ ….							
71	89	77	78	65	83	69	66
G	Y	M	N	A	S	E	B

Security of RSA

- The security of RSA depends on the problem of **factoring a large number**, i.e. **n**, the **product of p and q**.
  - Any **adversary** will have the **public key, e**, and the **modulus, n**. To find the **decryption key, d**, he has to **factor n**.
  - $n = pq;\ d = e^{-1} \bmod ((p - 1)(q - 1))$

Security of RSA

- It is also possible to attack RSA by **guessing the value of (p - 1)(q - 1)**. This attack is no easier than factoring n.
- A cryptanalyst can also try **every possible d** until he stumbles on the correct one. This brute-force attack is even less efficient than trying to factor n.
  - $m = c^d \bmod n$

Security of RSA

Chosen Ciphertext Attack against RSA, Scenario 1

- Eve, listening in on Alice's communications, manages to collect a ciphertext message, c, encrypted with RSA in her public key.

  $$m = c^{d} \mod n$$

- To recover m, she first chooses a random number, r, such that r is less than n. She gets Alice's public key, e. Then she computes
  - $x = r^{e} \mod n$
  - $y = x^{c} \mod n$
  - $t = r^{-1} \mod n$
- If $x = r^{e} \mod n$, then $r = x^{d} \mod n$.

Security of RSA

- Now, Eve gets Alice to sign y with her private key, thereby decrypting y.
- Alice sends Eve $u = y^{d} \mod n$
- Now, Eve computes:
  - $tu \mod n$

    $= r^{-1} y^{d} \mod n$

    $= r^{-1} x^{d} c^{d} \mod n$

    $= c^{d} \mod n$

    $= m$
- Eve now has m.

Security of RSA

Chosen Ciphertext Attack against RSA, Scenario 2

- Eve wants Alice to sign m_3.
- She generates two messages, m_1 and m_2, such that:
  - $m_3 = m_1 m_2 \pmod{n}$
- If Eve can get Alice to sign m_1 and m_2, she can calculate m_3 :
  - $m_3^{\,d} = (m_1^{\,d} \bmod n)(m_2^{\,d} \bmod n)$

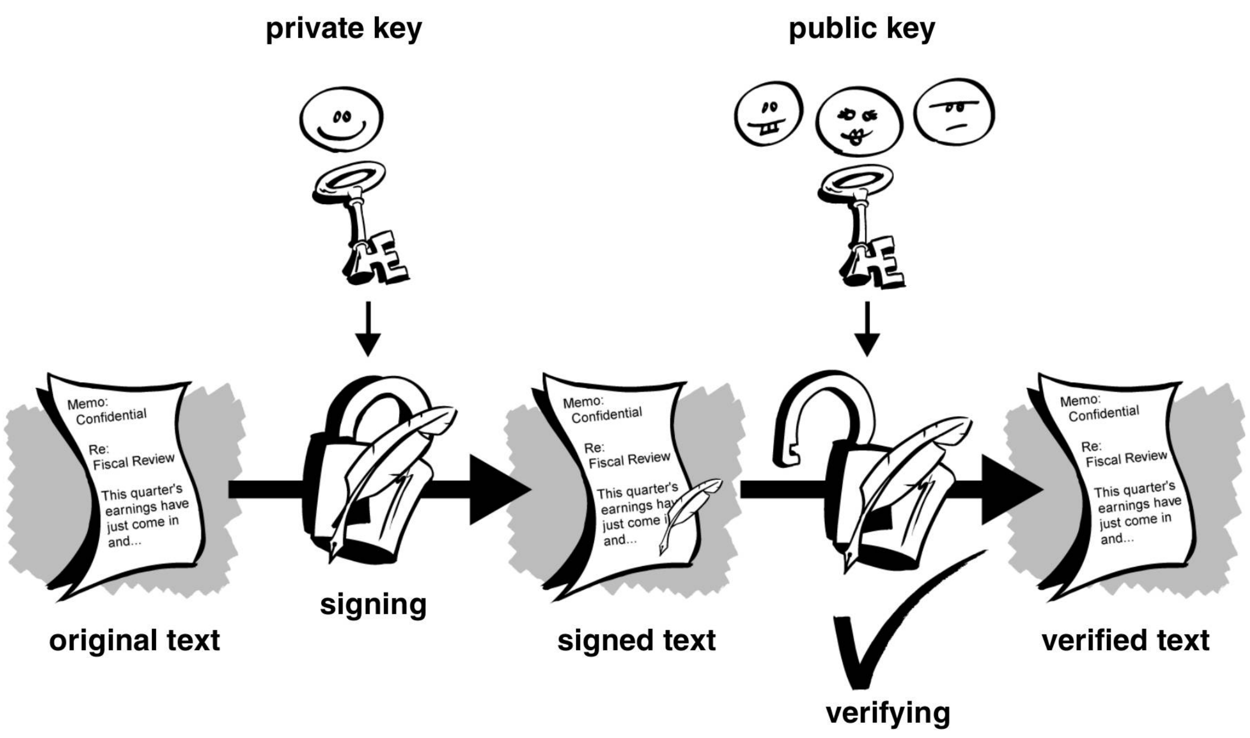
- **Moral:** Never use RSA to sign a random document presented to you by a stranger.

Digital signatures

- A digital signature serves the same purpose as a **handwritten signature**.
- However, a **handwritten** signature is **easy to counterfeit**.
- A **digital** signature is superior to a handwritten signature in that it is very **hard to counterfeit**.
- Moreover, a digital signature **attests to the contents** of the information as well as to the **identity of the signer**.

Digital signatures

- Instead of encrypting information using someone else's public key, you **encrypt** it with your **private key**.
- If the information can be **decrypted** with your **public key**, then it must have originated with you.



Digital signatures

- Public key digital signatures provide **authentication**, data **integrity**, and **non-repudiation**.
- **Authentication:** Digital signatures let the recipient of information verify the authenticity of the information's origin.
- **Integrity:** Digital signatures also allow verification that the information was not altered while in transit.
- **Non-repudiation:** Digital signatures can prevent the sender from claiming that he or she did not actually send the information.

Digital signatures

Exercise

- Plaintext:

  M = “MESSAGE” or just **“M”**

- RSA public key: **(e = 11, n = 703)**
- RSA private key: **(d = 59, n = 703)**
- Signing (encryption) and verifying (decryption) functions:

$$C = E(M) = M^{d} \ (\mathrm{mod}\ n); \quad M = D(C) = C^{e} \ (\mathrm{mod}\ n)$$

- ASCII:

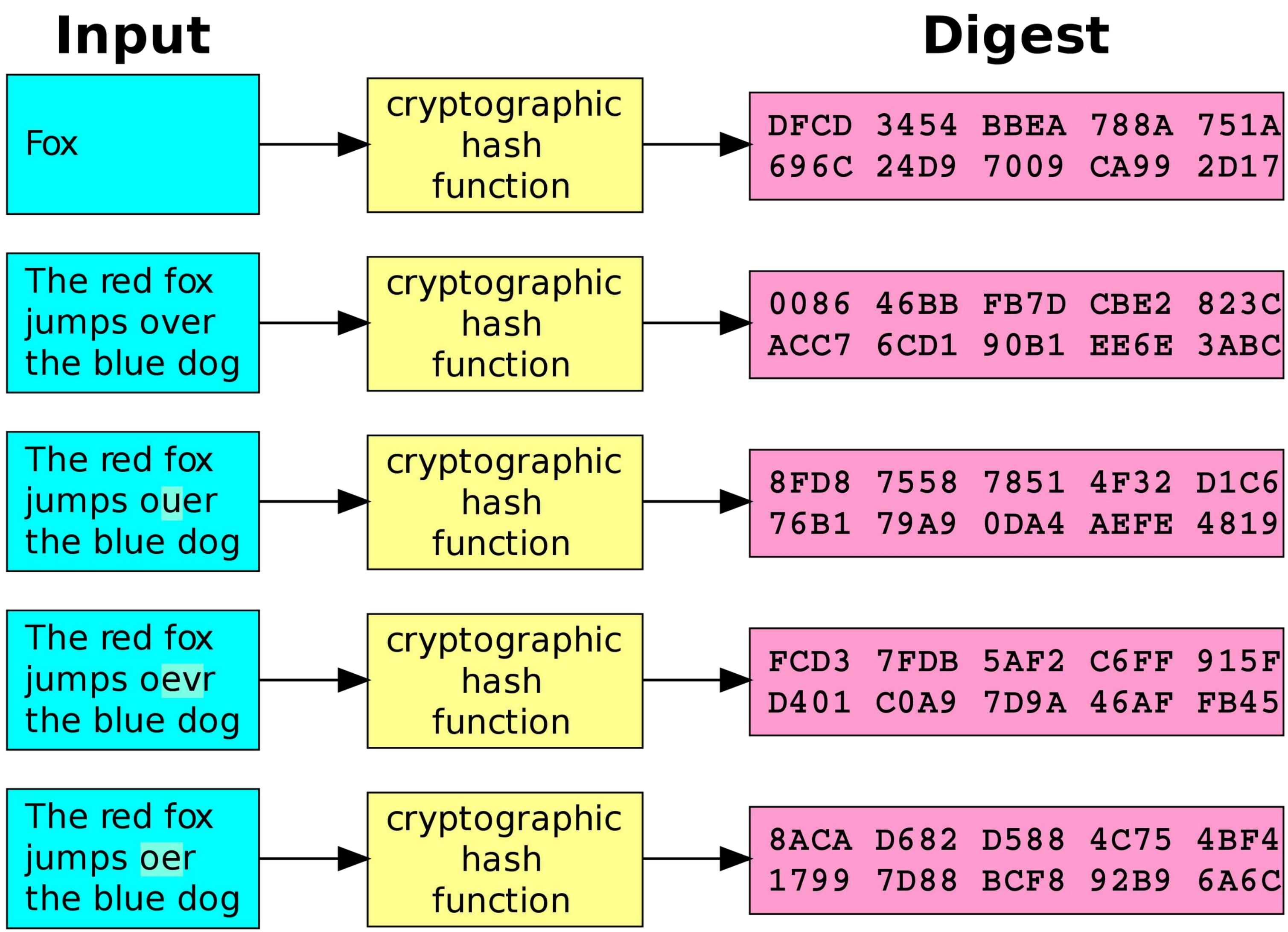
65	A	66	B	67	C	68	D	69	E
70	F	71	G	72	H	73	I	74	J
75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T
85	U	86	V	87	W	88	X	89	Y
90	Z								

https://www.cs.drexel.edu/~jpopyack/IntroCS/HW/RSAWorksheet.html

Hash functions

- A **one-way hash function** takes **variable-length input** — a message of any length, even thousands or millions of bits.
- It then produces a **fixed-length output**, e.g., 256 bits.
- The hash function ensures that, if the **input is changed** in any way — even by just one bit — an **entirely different output** value is produced **(avalanche effect)**.
- A hash function is **efficiently computable**. Computing the hash of an n-bit string should have a running time that is O(n).
- An application: hash tables

Hash functions



By User:Jorge Stolfi based on Image:Hash_function.svg by Helix84 - Original work for Wikipedia, Public Domain, https://commons.wikimedia.org/w/index.php?curid=5290240

Hash functions

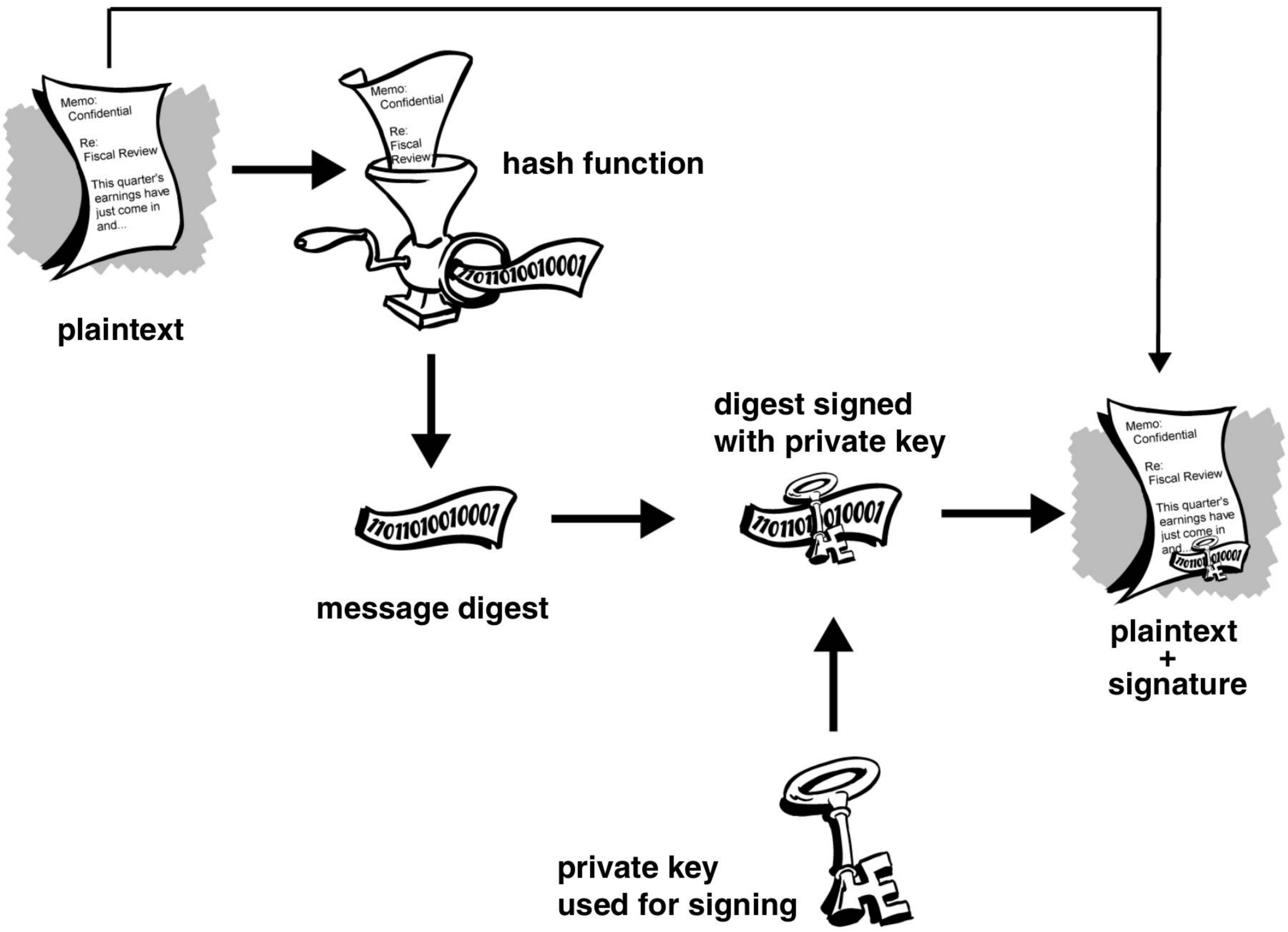
An application:

- Using **digital signatures** without hash functions is **slow**, and it produces an enormous volume of data—at least double the size of the original information.
- **PGP** uses a cryptographically strong **hash function** on the **plaintext** the user is signing. This generates a fixed-length data item known as a **message digest**.

Hash functions

- Then PGP uses the **digest** and the **private key** to create the **digital signature**.
- PGP transmits the signature and the plaintext together.
- Upon receipt of the message, the recipient uses PGP to **recompute the digest**, thus verifying the signature.

Hash functions



Cryptographic hash function

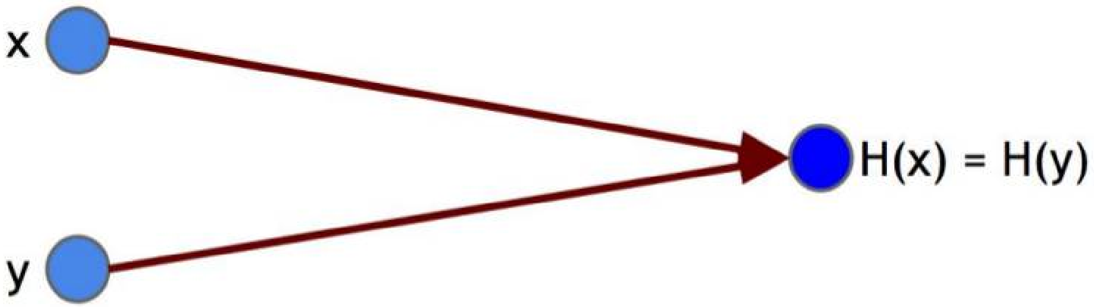
- For a hash function to be **cryptographically secure**, it must have the following additional properties:

  1) **Collision resistance**

  2) **Hiding**

Collision resistance

- A **collision** occurs when two **distinct inputs** produce the **same output**.



- A **hash collision**. x and y are distinct values, yet when input into hash function H, they produce the same output.
- Collision resistance: A **hash function H** is said to be **collision resistant** if it is infeasible to find two values, x and y, such that $x \neq y$, yet $H(x) = H(y)$.

Hiding

- If we are given the output of the hash function **y = H(x)**, there is **no feasible way to determine the input, x**.
- In order to be able to achieve the **hiding property**:
  - No value of x should be particularly likely.
  - That is, x has to be chosen from a **large set**.

Homomorphic Cryptosystems

Additive Homomorphic Cryptosystems:

- **Product** of **ciphertexts** ⇒ **Sum** of **plaintexts**

$$E(x) * E(y) = E(x + y)$$

$$E(3) * E(4) = E(3 + 4) = E(7)$$

Zero Knowledge Proofs (ZKP)

- A **Prover** convinces a **Verifier** that a **statement is true**
- **No** additional **information** is **revealed**

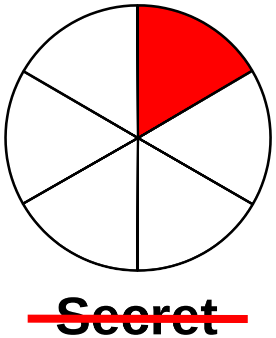
ZKP: Set Membership

- Given a ciphertext $E_u(x)$ and a public set S
- *User* ***u*** proves: $x \in S$
- ***x*** is not revealed

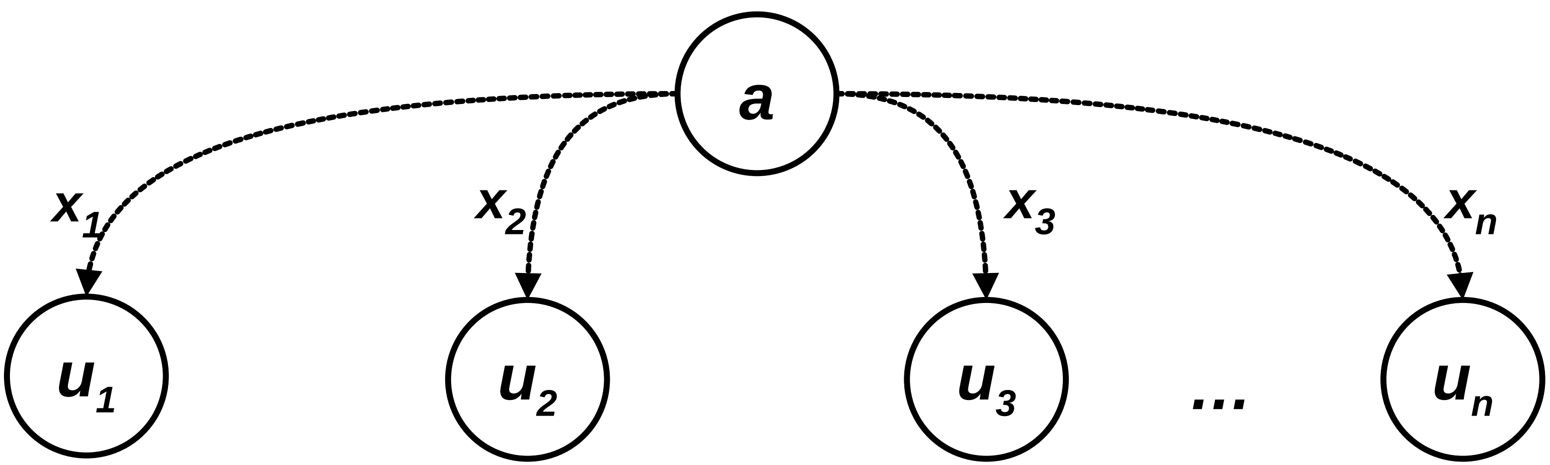
ZKP: Plaintext Equality

- Given two ciphertexts $E_u(x)$ and $E_v(x)$
- *User* ***u*** proves: Both $E_u(x)$ and $E_v(x)$ encrypt ***x***
- ***x*** is not revealed

Secret Sharing



- Split a secret into ***n*** shares: $\boldsymbol{x_1, x_2, \ldots, x_n}$
- Send the shares to ***n*** users



- $\boldsymbol{m \leq n}$ users required to unlock the secret

Secure Multiparty Computation

- There are **n parties** in a network. Each party has **an input** which is **private**.
- The n parties wish to **compute some joint function** over their inputs such that **all inputs remain private**, i.e., known only to their owners.
- **Security must be preserved** in the face of adversarial behavior by some of the participants, or by an external party.

Examples

Yao’s Millionaire Problem:

- **Two millionaires**, Alice and Bob, wish to learn **which of them is richer** without revealing their individual wealth.
- **Two private numbers** x and y each belonging to a **different party**, and the goal is to **solve the inequality** while preserving the **privacy of the inputs**.

Private Sum, Product, etc.:

- **n parties** wish to compute some function such as **sum** or **product** over their inputs such that all inputs remain **private**.