ELECS425F Computer and Network Security

Lec 05

Outline – Public Key Cryptography

- The Rabin cryptosystem.
  - Description.
  - An example.
  - Advantages and disadvantages.
- OAEP
- The Elgamal cryptosystem.
  - Description.
  - $\mathbf{Z}^*_p$: primitives/generators.
  - An example.

Outline – Digital Signature

- Digital signatures.
  - Required properties.
- Signature schemes.
  - Component algorithms and requirements on them.
- Digital signatures using PKC.
- A problem and a solution.
- The Elgamal Signature scheme.
  - Signing and verifying.
  - Example.
- The Digital Signature Standard (DSS).
  - Development history.
  - Digital Signature Algorithm.

The Rabin cryptosystem

- The security of this system is equivalent to the difficulty of factoring.
- **Key generation**:
  - Bob randomly chooses two large primes,
    - **p** and **q**, and
    - calculates **N=pq**.
  - The public key is **N** and
    - The secret key is **(p,q)**.

The Rabin cryptosystem

- **To Encrypt**: the ciphertext **Y** for a message (plaintext) **X:**

  $Y=X^2 \bmod N$

- **To decrypt** the received message:
  - Bob must find the square root of Y mod N.
  - Knowing (p,q) it is easy to find the square root, otherwise it is provably as difficult as factoring.

- **Special case:** **p** and **q** are 3 mod 4 (N: Blum integer).
  - We construct four intermediate factors.

    $\mathbf{x_1=Y^{(p+1)/4} \bmod p}$

    $\mathbf{x_2=p-x_1}$

    $\mathbf{x_3=Y^{(q+1)/4} \bmod q}$

    $\mathbf{x_4=q-x_3}$

The Rabin cryptosystem

- We define

  $a=q(q^{-1} \bmod p)$ **$b=p(p^{-1} \bmod q)$**
- This means, for example, that you calculate **$q^{-1} \bmod p$** and multiply the result by **q**.

- Then 4 **possible plaintexts** can be calculated.

$$X_1 = (ax_1+bx_3) \bmod N$$
$$X_2 = (ax_1+bx_4) \bmod N$$
$$X_3 = (ax_2+bx_3) \bmod N$$
$$X_4 = (ax_2+bx_4) \bmod N$$

An example

- We choose p=7, q=11 so N=77.
  - Note that p and q are 3 mod 4.

$$x_1 = y^{(p+1)/4} \bmod p = 9^{(7+1)/4} = 9^2 \bmod 7 \quad (p=7)$$

$$x_2 = p - x_1 = 7 - 4 = 3$$

$$x_3 = y^{(q+1)/4} \bmod q = 9^{(11+1)/4} \bmod 11 = 3$$

$$x_4 = q - x_3 = 11 - 3 = 8$$

- Bob's public key is 77, and his private key is (7,11).

- To encrypt **X=3** Alice calculates

  $Y = 3^2 \bmod N = 9 \bmod 77$.

  always

- To decrypt Bob calculates

  $x_1 = 9^2 \bmod 7 = 4$ $\quad x_2 = 7-4 = 3$

  $x_3 = 9^3 \bmod 11 = 3$ $\quad x_4 = 11-3 = 8$

If an integer mod 4 is 3, that number is blym integer.
For example 10 mod 4 =2, so 10 is not an blum integer
11 mod 4 =3, so 11 is a blum integer

If p,q are both integer
It is say to find square root

- Bob then finds **a** and **b**:

  $7(7^{-1} \bmod 11) = 7 \times 8 = 56$

  $11(11^{-1} \bmod 7) = 11 \times 2 = 22$

- … and then the four possible plaintexts.

  $X_1 = 4 \times 22 + 3 \times 56 = 11+14 = \mathbf{25} \bmod 77$

  $X_2 = 4 \times 22 + 8 \times 56 = 11+(-14) = -3 = \mathbf{74} \bmod 77$

  $X_3 = 3 \times 22 + 3 \times 56 = -11+14 = \mathbf{3} \bmod 77$

  $X_4 = 3 \times 22 + 8 \times 56 = -11-14 = -25 = \mathbf{52} \bmod 77$

Advantages and disadvantages.

- Unless the RSA exponent *e* is small, Rabin's encryption is considerably faster than RSA.
  - Decryption requires roughly the same time as RSA.

- Decryption of a message generates 4 possible plaintexts. The receiver needs to be able to decide which one is the right message.
  - A simple solution: One can append messages with known patterns, for example 20 zeros, to allow easy recognition of the correct plaintext.

Agreement : pattern / hint in the correct one

Optimal Asymmetric Encryption Padding (OAEP)

why OAEP ?

① need to add pattern "0...0" in the figure.

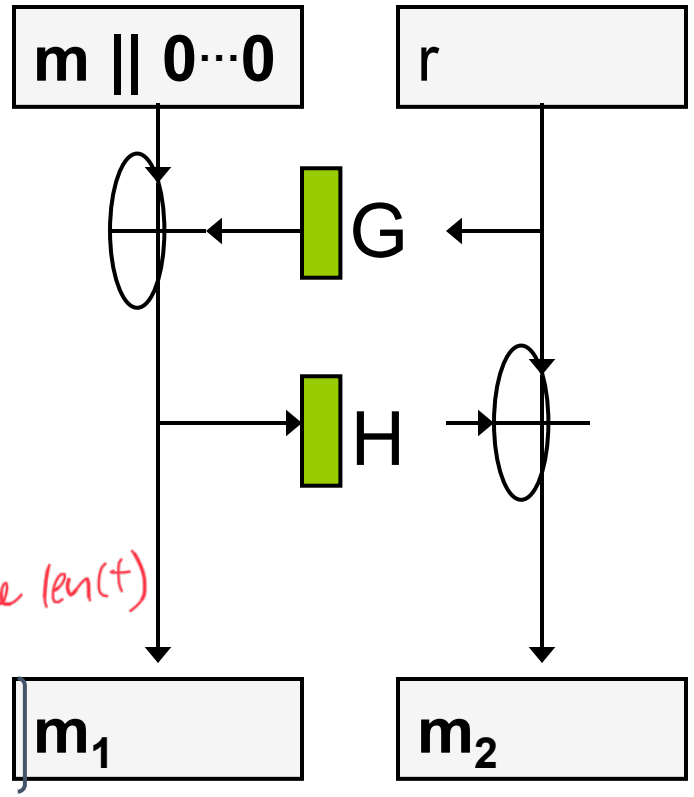
② Add "r", a random nums or digit

⇓

avalance effect

→ not easy in short plaintext, expand the length

⇒ OAEP implemented



Desirable feature:

→ avalance effect

If one bit in the plaintext change ⇒ ciphertext change a lot.

Add sth on plaintext, before Rabin/RSA encryption function, OAEP will process ☆

f: RSA or Rabin encryption function

The ElGamal cryptosystem

- The security of this system relies on another hard problem
  - Discrete log problem
- Operates on group Z_p^* = {1, 2, …, p-1} where p is a large prime number

★ Discrete log problem

→ $4^x = 9 \bmod 11$, what is x? $x = (\log_4 9) \bmod 11$

⇒ not easy to get the answer in the mod 11 system

★ LPN ⇒

$p = 11$, $Z_p^* = \{1, 2, 3, \cdots \omega\}$

Generator of Z_p^*

- An element α is a generator (or primitive root) of Z_p^* if $\alpha^i \pmod p$ for $0 < i \leq p-1$ generates all numbers *1,.. p-1*
- Finding a generator in general is a hard problem with no efficient algorithm known.
- If factorization of *p-1* is known it is not hard.
- In particular if $p=2p_1+1$ where p_1 is also a prime we have the following.

- α in Z_p^* and α ≠ ± 1 mod p.
- Then α is a primitive element if and only if

$$\alpha^{\frac{p-1}{2}} \neq 1 \bmod p$$

Example:

$p = 11$, $\frac{p-1}{2} = 5$

$3^5 = 1 \bmod 11$

→ 3 is not primitive

- Suppose α in Z_p^* and α is not a primitive element.
- Then $-\alpha$ is a primitive element.

- This gives an efficient algorithm to find a primitive elements.

- Example: Z^*_{11}
  - $3^5=1 \bmod 11$
  - 3 is not primitive, -3 = 8 is primitive

An example : Z_{11}^*

	1	2	3	4	5	6	7	8	9	10
1	1	1								1
2	**2**	**4**	**8**	**5**	**10**	**9**	**7**	**3**	**6**	1
3	3	9	5	4	1	3				1
4	4	5	9	3	1	4				1
5	5	3	4	9	1	5				1
6	**6**	**3**	**7**	**9**	**10**	**5**	**8**	**4**	**2**	1
7	**7**	**5**	**2**	**3**	**10**	**4**	**6**	**9**	**8**	1
8	**8**	**9**	**6**	**4**	**10**	**3**	**2**	**5**	**7**	1
9	9	4	3	5	1	9				1
10	10	1	10							1

$\alpha^i \to Z_i^*$

α

i

Z_{11}^*

generator = 2, 6, 7, 8

Format's little theorem

- For any **prime** number p and integer a, $a^{p-1} = 1 \bmod p$

11

10

$10^{10} = 1 \bmod 11$

always 1

Discrete Logarithm Problem

INPUT:

- Z_p^* ✓
- g *in* Z_p^* , a generator of Z_p^* ✓
- h in Z_p^* ✓

$4^{(8)} = 9 \bmod 11$

Find the unique number $a<p$ such that

$h=g^a \bmod p$

- **DL Assumption:** There is no efficient algorithm to solve DL problem.
- It is widely believed that this assumption holds.

DLP

- When p is small discrete log can be found by exhaustive search
- The size of prime for which DL can be computed is approximately the same as the size of integers that can be factored
  - In 2001: 120 digit DL, 156 digit factorization
- DL is an example of one-way function:
- A function f is **one-way** if
  - Given x, finding f(x) is easy
  - Given y, finding x such that f(x)=y is hard

The El Gamal Cryptosystem

- **Key generation**:
  - Alice chooses a prime **p** and two random numbers **g** and **u**, both less than **p**, where **g is** a generator of $\mathbf{Z_p}^*$.
  - Then she finds:

$$y = g^u \bmod p$$

Alice's public key is **(p, g, y)**, her secret key is **u**.

generator (must)

Relationship between public and private key

private key

The security point of ELGamal

you know y, g, p → hard to get u

The El Gamal Cryptosystem

- To encrypt a message **X** for Alice, Bob chooses a random number **k** < p - 1. Then he calculates:

$$a = g^k \bmod p$$
$$b = y^k \times X \bmod p$$

- The cryptogram is **(a,b)**
- The length is twice the length of the plaintext.

- To decrypt **(a,b)** Alice calculates

$$X = \frac{b}{a^u} \bmod p$$

Division means calculating the inverse

ElGamal is malleable

part of public key

message

$$Given\ (a,b) = (g^k, y^k \times m) \mod p$$

Eve: – *chooses a random number* r *in* Z_p^*, (generator)

– *computes* rb

– *sends* (a, rb) *to Alice*

gf *Alice returns the decryption:* $r \times m \mod p$

Eve will find: $$\frac{r \times m}{r} = m \mod p$$

Exercise

Alice's public key (p, g, y) and secret key is u

Given that $p = 11$, $g = 2$, if $u = 3$, $y = 8$, $m = 10$ → random number $k = 7$

① Encrypted message: $a = g^k \bmod p$, $b = y^k \cdot m \bmod p$

$a = 128 \bmod 11$, $a = 7$

$b = 20971520 \bmod 11$, $b = 9$

cipher = $(7, 9)$

???

② Decrypted message: $m = \frac{b}{a^u} \bmod p$

$= \frac{9}{343} \bmod 11$

$= 9(6) \bmod 11 = 10$

Outline – Digital Signature

- Digital signatures.
  - Required properties.
- Signature schemes.
  - Component algorithms and requirements on them.
- Digital signatures using PKC.
- A problem and a solution.
- The Elgamal Signature scheme.
  - Signing and verifying.
  - Example.
- The Digital Signature Standard (DSS).
  - Development history.
  - Digital Signature Algorithm.

Digital Signatures

- Introduced by Diffie-Hellman (1976).
- A digital signature is the electronic analogue of handwritten signature. It ensures integrity of the message and authenticity of the sender. That is, if Bob gets a message which supposedly comes from Alice, he is able to check that…
  - The message has not been modified.
  - The sender is truly Alice.

- Digital signatures require a few properties. They should be:
  - Easy to generate.
  - Easy to verify by a receiver.
  - Hard to forge.

- A digital signature is generally represented as a **string of bits** attached to a message.
- This is similar to the mechanism for **M**essage **A**uthentication **C**odes but digital signature is publicly verifiable.

Signature schemes

- A **signature scheme** consists of two algorithms (possibly three if you consider key generation too):
  1. A **signature generation algorithm** denoted **S**, takes a message **X** and produces a signature **S(X)**.
  2. A **signature verification algorithm** denoted **V**, takes a message **X** and the associated signature **S(X)** and produces a **True** or **False** value.
     - True indicates the message should be accepted while False indicates the message should be rejected.

Algorithm requirements

- The algorithms defining a signature scheme need to satisfy some requirements:
  - **V(X, S(X)) = True**
  - Without knowledge of the key, it is computationally infeasible to construct a valid pair **(X, S')** such that **V(X,S') = True**.

Very hard to make a fake signature.

Digital Signatures using PKC

- **A**lice signs **X** by creating **Y=D_{z_A}(X)**. The signed message is **(X,Y)**.
- **B**ob validates **A**lice's signature by computing **X'=E_{z_A}(Y)** and comparing it with **X**.

- Everyone has access to E_{z_A} and hence anyone can verify the signature by computing $E_{z_A}(Y)$.

- Since D_{z_A} is a trapdoor one-way function, it is not possible for an intruder to find z_A.
- Hence **A**lice's signature cannot be forged.

Other integrity services

- Digital signatures can provide more than just explicit authentication and integrity.
- For example, they can provide **non-repudiation**. This prevents the sender from denying he/she has sent the message.

- If Alice signs a message, since she is the only one who knows z_A, she will be held accountable for it.
  → The system therefore provides **non-repudiation**.

Some problems …

- As described we would implement a PKC based digital signature by breaking a message into blocks and signing each block independently.

  $X_1, X_2, \ldots, X_t$

  $S(X_1), S(X_2), \ldots, S(X_t)$

- This means, however, the signed message is susceptible to *block re-ordering, block deletion*, and *block replication*.

- Other problems:
  - PKC is slow. ☹
  - The signature is the same size as the message. ☹

  → It is desirable to produce a **short** signature that is a function of the whole message.

… and a solution

- The solution is to use hash function in conjunction with PKC.

- We will look at hash functions after we have looked at some examples of digital signatures.

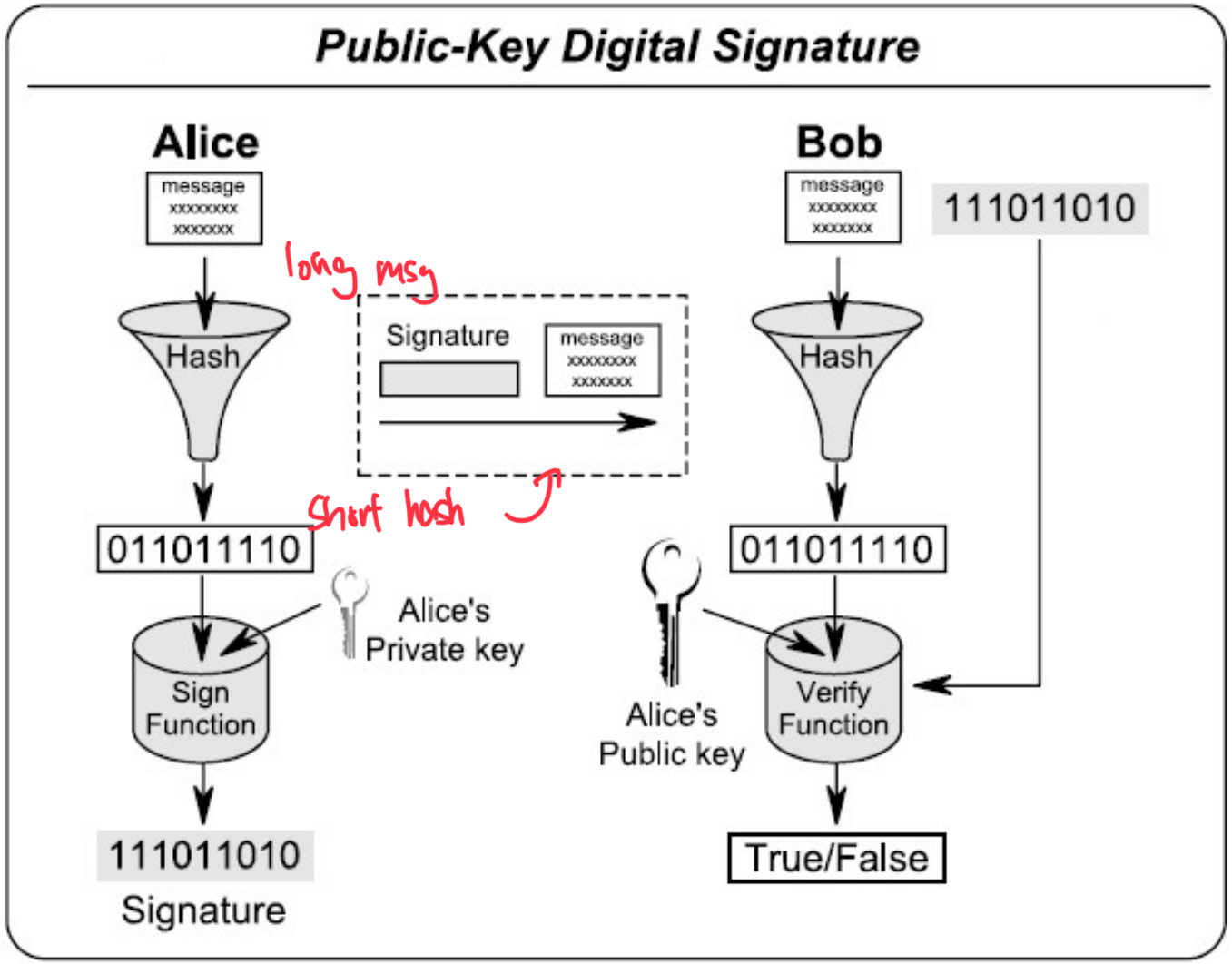
Hash-then-Sign

① The original msg may be long

② After hashed by hash f(x)
⇒ shorter than original msg

③ sign on hashed msg instead of msg

★ message content ⇒ hash (Different)



The ElGamal signature scheme

- This is based on the difficulty of computing discrete logarithms over $\mathbf{Z_p^*}$, where **p** is a prime.

- **Setup and Key generation**:
  - **A**lice chooses a large prime **p**.
  - Let **g** denote a primitive element of $\mathbf{Z_p^*}$. **A**lice chooses **x** as her private key and calculates $\mathbf{y=g^x \bmod p}$.

- The public key is **(p, g, y)**.
- The private key is **x.**
- This is the same as for encryption.

Signing and verifying

- To sign a message X:
  - **A**lice chooses a random integer *k, 1≤ k ≤ p-1,* such that **gcd(k,p-1)=1**.
  - **A**lice calculates
    - **$r=g^k \bmod p$.**
    - **$s=k^{-1}(X-xr) \bmod (p-1)$**
  - The signed message is **(X, (r,s))**.
- To verify the message; compare g^x and $y^r r^s$.

Using the received values

$$g^X = g^{xr+ks} = g^{xr} * g^{ks} = y^r * r^s \bmod p$$

Example

- **Example**: p=11, g=2. (Remember in previous notes we showed that 2 is a generator of Z_{11}^*.

- **Private key**: x=3.
- **Public key**: $y=2^3=8 \bmod 11$.

- **Signing X=9**.
  - Choose **k=7**. → Check that **gcd(7,10)=1**. ✓
  - Calculate **$k^{-1}=3 \bmod 10$**. → Calculated with above!
  - Calculate **$r=g^k=2^7=7 \bmod 11$**.
  - Calculate **$s=3(9-7*3) \bmod 10 = 4$**.
- The signed message is **(9,(7,4))**.

- **Verification** is by comparison of g^x and $y^r r^s$.
  $2^9 = 6 = 8^7 7^4 \bmod 11$

$k \in \{1-10\}$

$p=11, g=6$

private key: $x=5$

public key: $y=6^5 \bmod 11$

$= 10$

sign: $x=3$

Choose k-value, $k \in \{1-10\}$

$\rightarrow k=7$

(r,s)

$=(8,4)$

find r,s

$r=g^k \bmod p$, $r=6^7 \bmod 11 = 8$

$s=k^{-1}(x-xr) \bmod (p-1) = 9$

The Digital Signature Standard (DSS)

- Proposed by NIST in 1991, finally issued in 1994 (Federal Information Processing Standard FIPS 186).
- Various modifications were made.
- We are just going to talk about the original Digital Signature Algorithm in FIPS 186.

Digital Signature Algorithm (DSA)

- **Key generation and setup**:
  - **q** a 160-bit prime is chosen
  - **p** a prime. 512-1024 bits long with the length a multiple of 64. **q** is a factor of **p-1**.
  - **h** a number less than **p-1** such that

    $g = h^{p-1/q} \bmod p > 1$
  - **u < q**
  - **$y = g^u \bmod p$**

Fact: $g^q = 1 \bmod p$

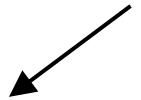
- **Public parameters**: p, q, g. (Multiple people could use these).
- **Private key**: u.
- **Public key**: y.

Generating and verifying a signature

- To sign a message …
  - **A**lice generates a random **k**, such that **k<q**.
  - **A**lice computes
    $r = (g^k \bmod p) \bmod q$
    $s = (k^{-1}(H(X) + ur))) \bmod q$
- To verify a message Bob calculates…
    $w=s^{-1} \bmod q$
    $t_1=(H(X)*w) \bmod q$
    $t_2=r*w \bmod q$
    $v=((g^{t_1}*y^{t_2}) \bmod p) \bmod q$

  … and accepts the signature if **v=r**.

Using the received values



H(X) is the hash of X. We will get to these soon.

Verification

- $s = (k^{-1}(H(X) + ur)) \bmod q$
- $s^{-1} = (k(H(X) + ur)^{-1}) \bmod q$
- $k = s^{-1}*(H(X) + ur) \bmod q$
- Let $w = s^{-1} \bmod q$
- $g^k = g^{w*(H(X)+ur)}$
  $= g^{w*H(X)}\, g^{w*ur} = g^{w*H(X)}\, y^{w*r} \bmod p$

Example

- **Setup**:
  - p=29, q=7 (is a factor of 28).
  - h=3, $g=h^{(29-1)/7}=3^4=23 \bmod 29$
  - u=2 < q.
  - $y=g^2 \bmod 29 = 23^2 \bmod 29$

$$= (-6)^2 \bmod 29$$

$$= 36 \bmod 29 = 7$$

- **Signing:**
  - To sign **H(X)=5**, Alice chooses **k=4** (<q) and calculates $k^{-1}=2 \bmod 7$.

    $r = (g^4 \bmod 29) \bmod 7$

    $= (49 \bmod 29) \bmod 7 = 20 \bmod 7 =$ **6**

    $s = k^{-1}(H(X) + ur) \bmod q$

    $= 2(5+2*6) \bmod 7$

    $= 34 \bmod 7$

    = **6**

- **Verifying**:
  $w = s^{-1} \bmod q = 6^{-1} \bmod 7 =$ **6**
  $t_1 = (H(X)*w) \bmod q$
  $= 5*6 \bmod 7 =$ **2**
  $t_2 = r*w \bmod q$
  $= 6*6 \bmod 7 =$ **1**
  $v = ((g^{t_1}*y^{t_2}) \bmod p) \bmod q$
  $= ((23^{2}*7^{1}) \bmod 29) \bmod 7$
  $= ((7*7) \bmod 29) \bmod 7$
  $=$ **6**
- Since **v=r** the message and signature are accepted as authentic.

Notes

- The recommended hash function for DSA is **SHA-1**.
  - We will discuss this later.

- There were several criticisms against DSA.
  - DSA cannot be used for encryption or key distribution.
  - It was developed by NSA and so cannot be trusted.
  - RSA is the de facto standard.
  - The key size is too small.

Signature scheme variants

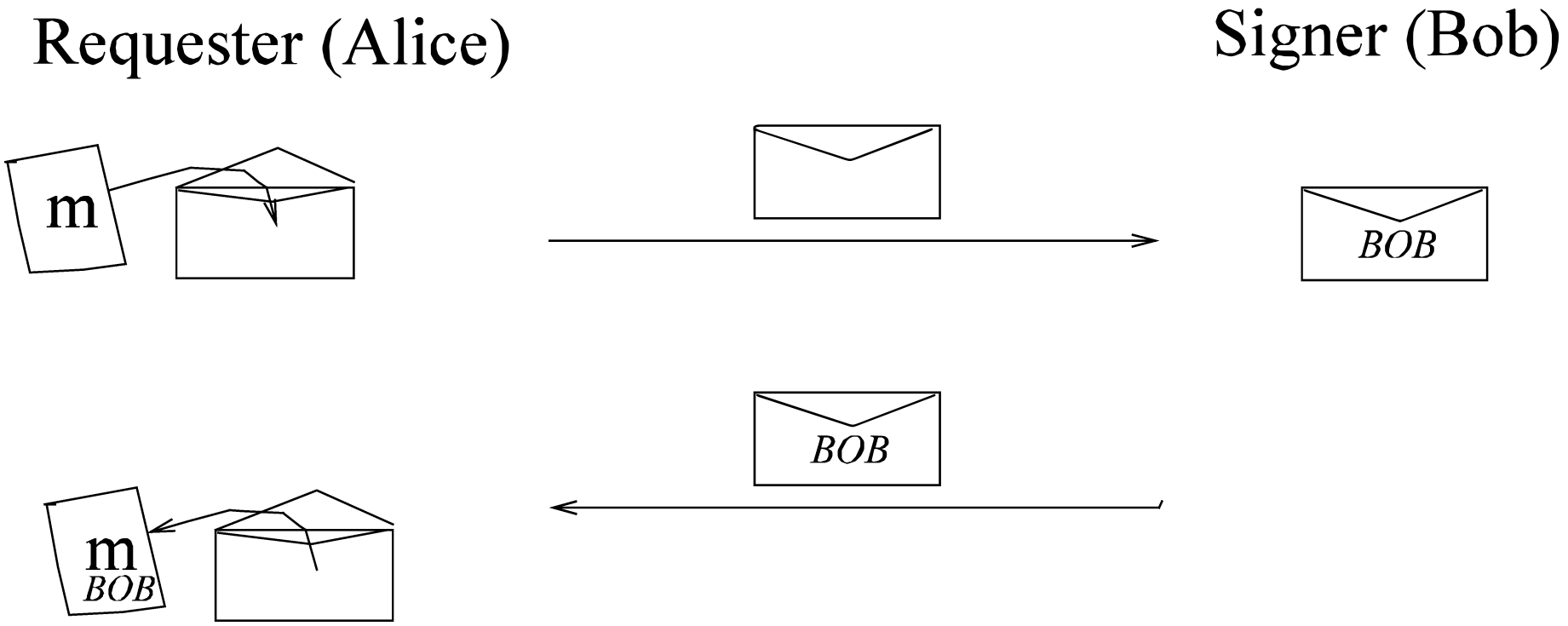
- There are many different types of signature schemes.
- They are designed to meet the requirements of different scenarios.
- The cost of adding properties is efficiency, both in terms of computation and storage.
- We are going to look briefly at a few types of signature schemes.

The signer cannot see the content/message

Blind signatures

- In some cases it is necessary to get the signature of a party without allowing them to see the message. For example, in electronic cash (proposed by David Chaum), a coin is the means of payment. A coin has a serial number and must be signed by the bank to be authentic.
  - To generate a coin, a customer randomly generates a number. The customer needs the bank signature on the coin before being able to spend it. To ensure privacy of the coin, that is to ensure that the bank cannot link a coin with a particular customer, the customer uses a **blind signature scheme**.
- In general:
  - The requester wants to obtain the signer's signature of message *m*.
  - The requester doesn't want to reveal *m* to anyone, including the signer.
  - The signer signs *m* blindly, not knowing what they are signing.
  - The requester can retrieve the signature.

The requester's message is put in the envelope, then the envelope is sent yo signer. The signer signs on the envelope. The requester gets the signature of the envelope. But the requester can retrieve the signature of the message.



Blind signatures using RSA

- **Setup**: **B**ob has (d,e), a (private,public) key pair.
  - *N=pq*, where *p, q* are large primes, associated with **B**ob.
- For a message **m**, that **A**lice wants **B**ob to sign, **A**lice constructs $\mu = mr^e \bmod N$, where $\mathbf{r} \in_R Z_N^*$, and sends μ to **B**ob.
- **B**ob signs μ and sends his signature $\sigma = \mu^d \bmod N$ back to **A**lice.
- **A**lice retrieves the signature *s* of *m* by computing:

$$s = \frac{\sigma}{r} = \frac{\mu^d}{r} = \frac{m^d r^{ed}}{r} = \frac{m^d r}{r} = m^d \bmod N$$

How to verify the signature?
- Use the public key of the signer, e value
>> S^e = (m^d)^e = m

So anyone receive the (m,S) can use the signer's public key to verify the S is signed by the signer or not

Summary – Public Key Cryptography

- The Rabin cryptosystem.
  - Description.
  - An example.
  - Advantages and disadvantages.
- OAEP
- The Elgamal cryptosystem.
  - Description.
  - $\mathbf{Z^*_p}$: primitives/generators.
  - An example.

Summary – Digital Signature

- Digital signatures.
  - Required properties.
- Signature schemes.
  - Component algorithms and requirements on them.
- Digital signatures using PKC.
- A problem and a solution.
- The Elgamal Signature scheme.
  - Signing and verifying.
  - Example.
- The Digital Signature Standard (DSS).
  - Development history.
  - Digital Signature Algorithm.