CSCI471/971 Modern Cryptography

Public Key Cryptography III

Rupeng Yang

SCIT UOW

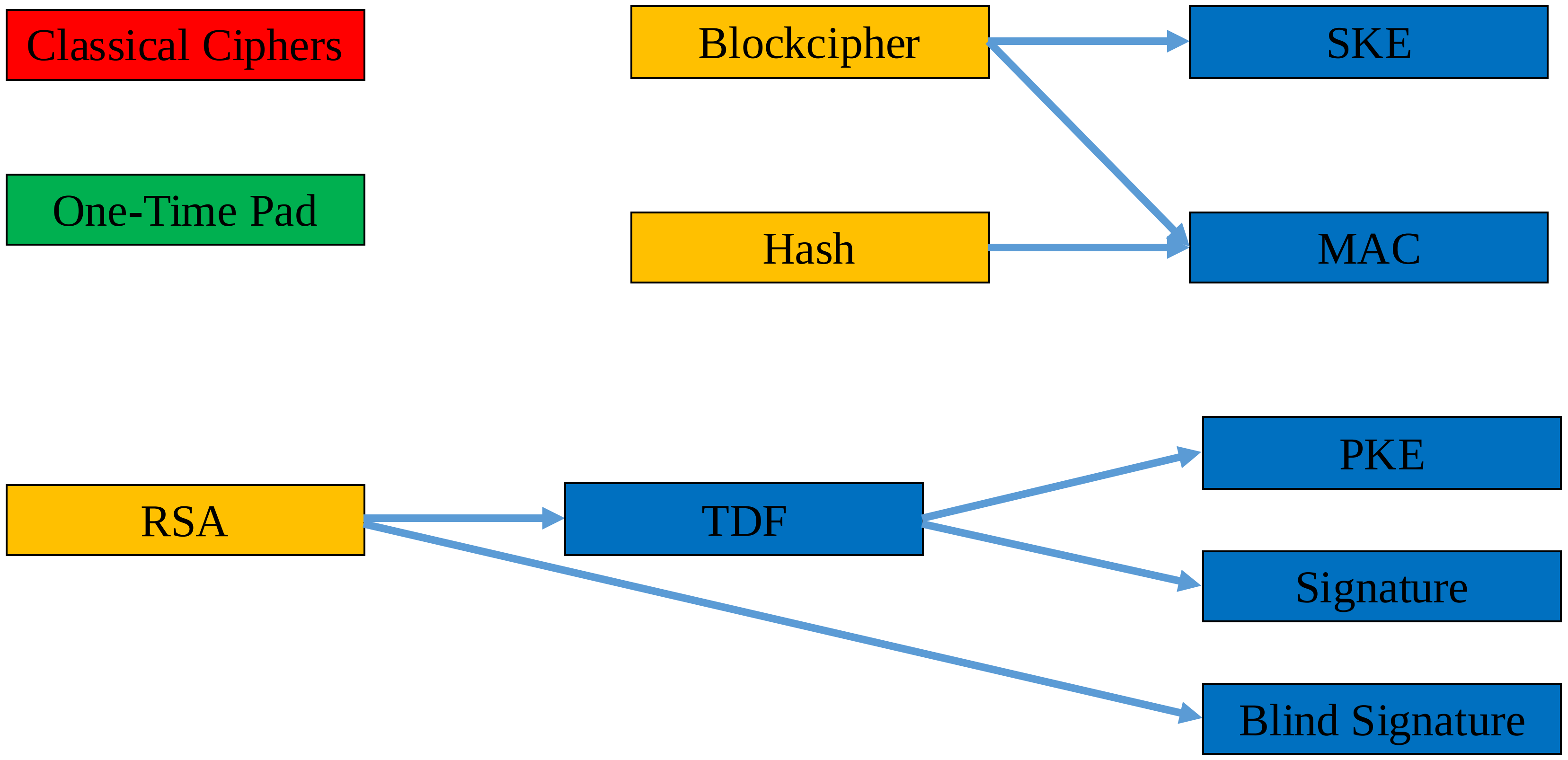
RoadMap

- Week 1-2: Preliminaries

- Week 3-4: Symmetric-Key Cryptography

- Week 5-6: PKC from factoring
- Week 7: PKC from DL

Roadmap


Classical Ciphers
One-Time Pad
Blockcipher
Hash
SKE
MAC
RSA
TDF
PKE
Signature
Blind Signature

Definition of an Abelian Group

- A Group is a set of objects together with an operation defined between any two objects in the set.
- Let G denote a set and $\bullet$ denote an operation
- $(G, \bullet)$ is an Abelian group if all the following conditions are met:
  - Closure: $a \bullet b \in G$ for all $a \in G$ and $b \in G$
  - Associative: for all $a, b, c \in G$, $(a \bullet b) \bullet c = a \bullet (b \bullet c)$
  - Commutative: for all $a, b \in G$, $a \bullet b = b \bullet a$
  - Identity: there exists an element $e \in G$ such that $\forall a \in G$, $a \bullet e = e \bullet a = a$
  - Inverse: $\forall a \in G$, there exists an element b in G, such that $a \bullet b = b \bullet a = e$

Definition of an Abelian Group

- If |G| is finite, we said (G, •) is a finite group and let |G| denote the order of the group.

- If (G, •) is a group, (H, •) is a subgroup of (G, •) if
  - $H \subseteq G$
  - (H, •) is a group

- We usually use multiplicative notation to describe the group:
  - the group operation applied to g, h is denoted by $g \cdot h$ or simply gh
  - the identity is denoted by 1 (The identity element in a group G is unique)
  - the inverse of an element g is denoted by g^{-1} (Each element has a unique inverse)

The Group Z_5^*

- Let Z_5^* be the set {1, 2, 3, 4}
- Consider the operator x mod 5
- Is (Z_5^* , x mod 5) a group?

* mod 5	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

Group Exponentiation

- we use g^n to denote $\overbrace{g \cdot g \; \dots \cdot g}^{\text{n times}}$

- It is easy to compute g^n given g and n.

- Let (G, •) be a finite Abelian group with q = | G | , the order of the group. Then for any element g ∈ G, it holds that $g^q = 1$

- Let (G, •) be a finite Abelian group with q = | G | > 1. Then for any g ∈ G and any integer x, we have

$$g^x = g^{[x \bmod q]}$$

Cyclic Group

- Let (G, ·) be a finite group of order q. For arbitrary $g \in G$, consider the set

$$\langle g \rangle = \{ g^0, g^1, \dots \}.$$

- $\langle g \rangle$ is a finite set.
- $(\langle g \rangle, \cdot)$ is a subgroup of (G, ·) .
- Let $p=|<g>|$, then $<g>= \{ g^0, g^1, \dots, g^{p-1}\}$

- $(\langle g \rangle, \cdot)$ is a cyclic group with order $p=| \langle g \rangle |$ and g is a generator of $(\langle g \rangle, \cdot)$.

An example: Z_{11}^*

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

Hard Problems in Cyclic Groups

- **The Discrete Logarithm Problem (DLP)**

Given a cyclic group G of order q, a generator *g* *in* G, and a group element *h* in G, find the unique non-negative number $a<q$ such that $h = g^a$.

- **The Computational Diffie-Hellman Problem**

Given a cyclic group G of order q, a generator *g* *in* G, and two group elements g^a and g^b ,find $g^{ab.}$

- **The Decisional Diffie-Hellman Problem**

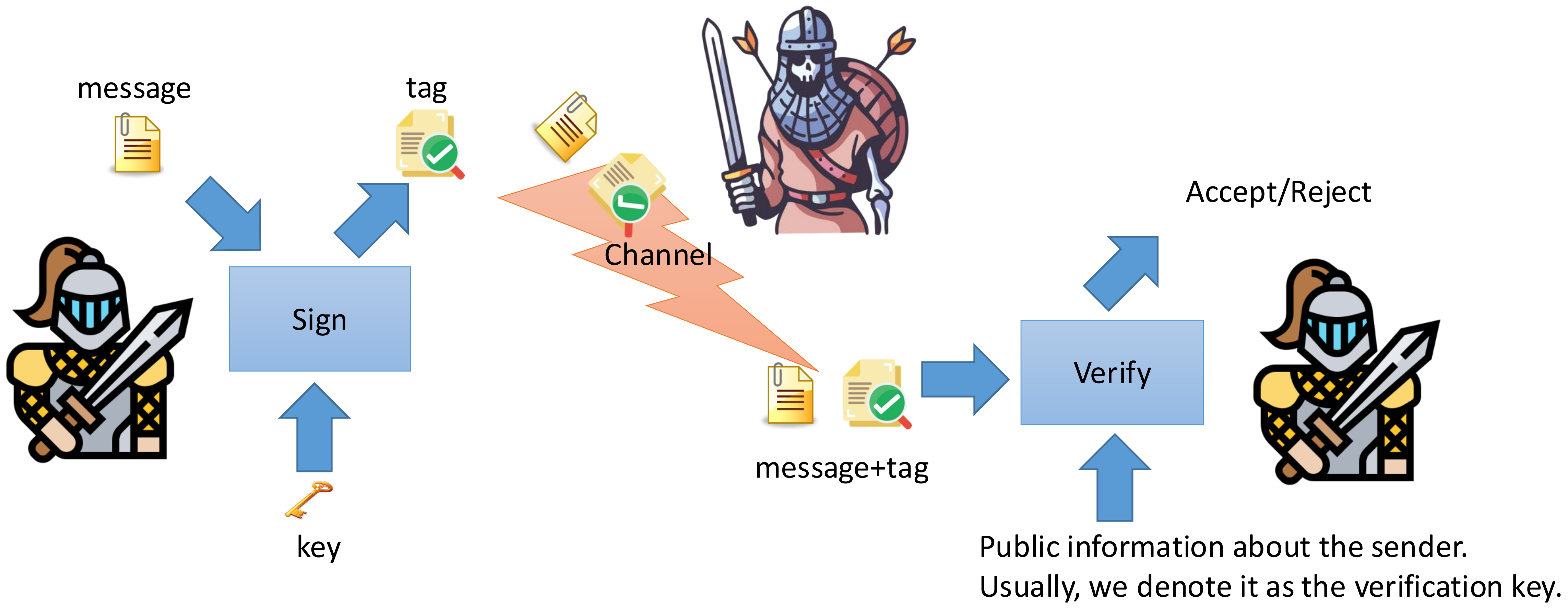
Given a cyclic group G of order q, a generator *g* *in* G, and two group elements g^a and g^b , distinguish g^{ab} from a random group element.

How to instantiate a cyclic group

- The group Z_p^* = {1, 2, …, p-1} where p is a large prime number
  - The DDH problem is easy in Z_p^*.
  - The DLP and the CDH problem are assumed to be hard in Z_p^* for large enough p.
- The prime order subgroup of Z_p^*
  - The DLP, the CDH problem, and the DDH problem are assumed to be hard in this subgroup if the group order is large enough.
- Elliptic curve groups
  - The DLP, the CDH problem, and the DDH problem are assumed to be hard in this subgroup if the group order is large enough.

Digital Signature from Cyclic Groups

Digital Signatures


message
tag
Sign
key
Channel
Accept/Reject
message+tag
Verify
Public information about the sender.
Usually, we denote it as the verification key.

The data that Alice sent to Bob cannot be modified by the adversary (even by Bob).
Bob only needs to know that pk belongs to Alice (no need to share a secret key!)

Digital Signatures

- KeyGen(λ): Taking as input a security parameter λ, the key generation algorithms returns (pk,sk)

- Sign(sk, M): Taking as input a message M and a secret key sk, the signing algorithm returns a signature denoted by S.

$$S \leftarrow \mathrm{Sign}(sk, M)$$

- Verify(S,M,pk): Taking as input signed message (S,M) and the public key pk, the verification algorithm returns 1 or 0.

Digital Signatures

- **Correctness**: For all generated (pk,sk) and all S←Sign(sk, M), we have

Pr[Verify(S,M,pk)=1]=1

Verify(S,M,pk)=1: Here 1 means that the signature is valid

Security Model for Signature: Unforgeability under Chosen Message Attacks

Algorithms KeyGen, Sign, Verify are public.

1. Run KeyGen to get (pk,sk)

2. Send pk to the Attacker

3.1 Attacker sends m to challenger

3.2 Run Sign(sk, m) to get t

3.3 Send t to the Attacker

4. Attacker returns (m*,t*)

Attacker wins if Verify(pk,m*,t*)=1 and m* is not queried in Step 3.

A message authentication code Scheme is Secure if ***NO*** efficient attacker can win with a probability of 1/poly(λ).

Preliminaries on Cyclic Group

- Let (G,g,p) be a cyclic group, where G is the set of group element, g is the generator, and p is the group order:
  - $G=\{ g^0, g^1 , . . . , g^{p-1}\}$
  - $g^p=1$
- The following operations are easy in the group (G,g,p):
  - Given any h_1, h_2 in G, it is easy to compute $h_1 \cdot h_2$
    - For any h in G and for any x,y in [0,p-1], given h^x and h^y, it is easy to compute $h^{x+y} =h^x \cdot h^y$
    - For any h_1, h_2 in G and for any x in[0,p-1], given h_1^x and h_2^x , we can compute $(h_1 \cdot h_2)^x= h_1^x \cdot h_2^x$
  - Given any h in G and any x in [0,p-1], it is easy to compute h^x
- The following operations are hard in the group (G,g,p):
  - Given g^x, it is hard to compute x (The DL problem)
  - Given g^x and g^y, it is hard to compute g^{xy} (The CDH problem)
  - Given g^x and g^y, it is hard to distinguish g^{xy} from a random group element in G (The DDH problem)

Schnorr Signature

- KeyGen(λ): Taking as input a security parameter λ, the P.P.T. algorithm
  1. Chooses a cyclic group G of order q and a generator g of G.
  2. Specifies a secure hash function H: {0,1}* $\rightarrow Z_q$.
  3. Chooses a uniform $x \in Z_q$ and compute $h = g^x$.
  4. The public key is (G, q, g, h, H) and the private key is (G, q, g, x, H).
- Sign(sk, M): Taking as input a message M and a secret key sk=(G, q, g, x, H), the P.P.T. algorithm
  1. <u>Choose a random number r and computes $R=g^r$</u>
  2. <u>Compute c=H(R, M)</u>
  3. <u>Compute z=r+ c*x mod q</u>
  4. <u>The signature is (R,z)</u>
- Verify(S,M,pk): Taking as input a signed message M, the public key pk=(G, q, g, h, H), and a signature (R,z), the P.P.T. algorithm
  1. <u>Compute c'=H(R,M) and Accept the signature if $g^z=R\cdot h^{c'}$</u>

- Correctness.

Schnorr Signature

- KeyGen(λ): Taking as input a security parameter λ, the P.P.T. algorithm
  1. Chooses a cyclic group G of order q and a generator g of G.
  2. Specifies a secure hash function H: {0,1}* $\rightarrow Z_q$.
  3. Chooses a uniform x ∈ Z_q and compute h = g^x .
  4. The public key is (G, q, g, h, H) and the private key is (G, q, g, x, H).
- Sign(sk, M): Taking as input a message M and a secret key sk=(G, q, g, x, H), the P.P.T. algorithm
  1. Choose a random number r and computes $R=g^r$
  2. Compute c=H(R, M)
  3. Compute z=r+ c*x mod q
  4. The signature is (R,z)
- Verify(S,M,pk): Taking as input a signed message M, the public key pk=(G, q, g, h, H), and a signature (R,z), the P.P.T. algorithm
  1. Compute c'=H(R,M) and Accept the signature if $g^z=R\cdot h^{c'}$
- Security.
  - The scheme is secure assuming the hardness of the Discrete Logarithm Problem and H is modeled as an idealized hash function (random oracle).
  - Notice: The random number r must be kept secret and never repeat. (The reasons are given in the workshop)

The parameters of schemes based on DLP

- If we use the group Z_p^*. The modulus p should have the same size as that of the RSA modulus N for the same security level
  - 80-bit security: p is a 1024-bit prime number
  - 112-bit security: p is a 2048-bit prime number
  - 128-bit security: p is a 3072-bit prime number
- If we use the subgroup G of Z_p^* that has prime order q
  - 80-bit security: p is a 1024-bit prime number and q is a 160-bit prime number
  - 112-bit security: p is a 2048-bit prime number and q is a 224-bit prime number
  - 128-bit security: p is a 3072-bit prime number and q is a 256-bit prime number

Implementing Signature

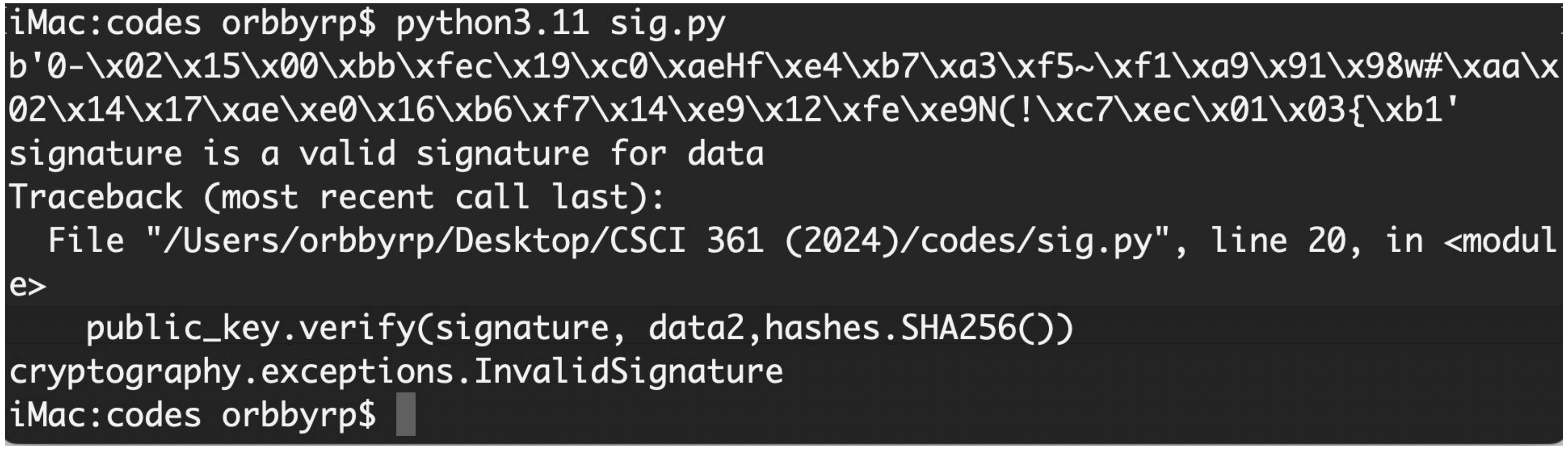
```
import os
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import dsa

# Key Generation
private_key = dsa.generate_private_key(key_size=1024)
public_key=private_key.public_key()


# Sign
data = b"signed data"
signature = private_key.sign(data,hashes.SHA256())
print(signature)
```

```
# Verify
data2 = b"signed data2"
public_key.verify(signature, data,hashes.SHA256())
print("signature is a valid signature for data")

public_key.verify(signature, data2,hashes.SHA256())
```



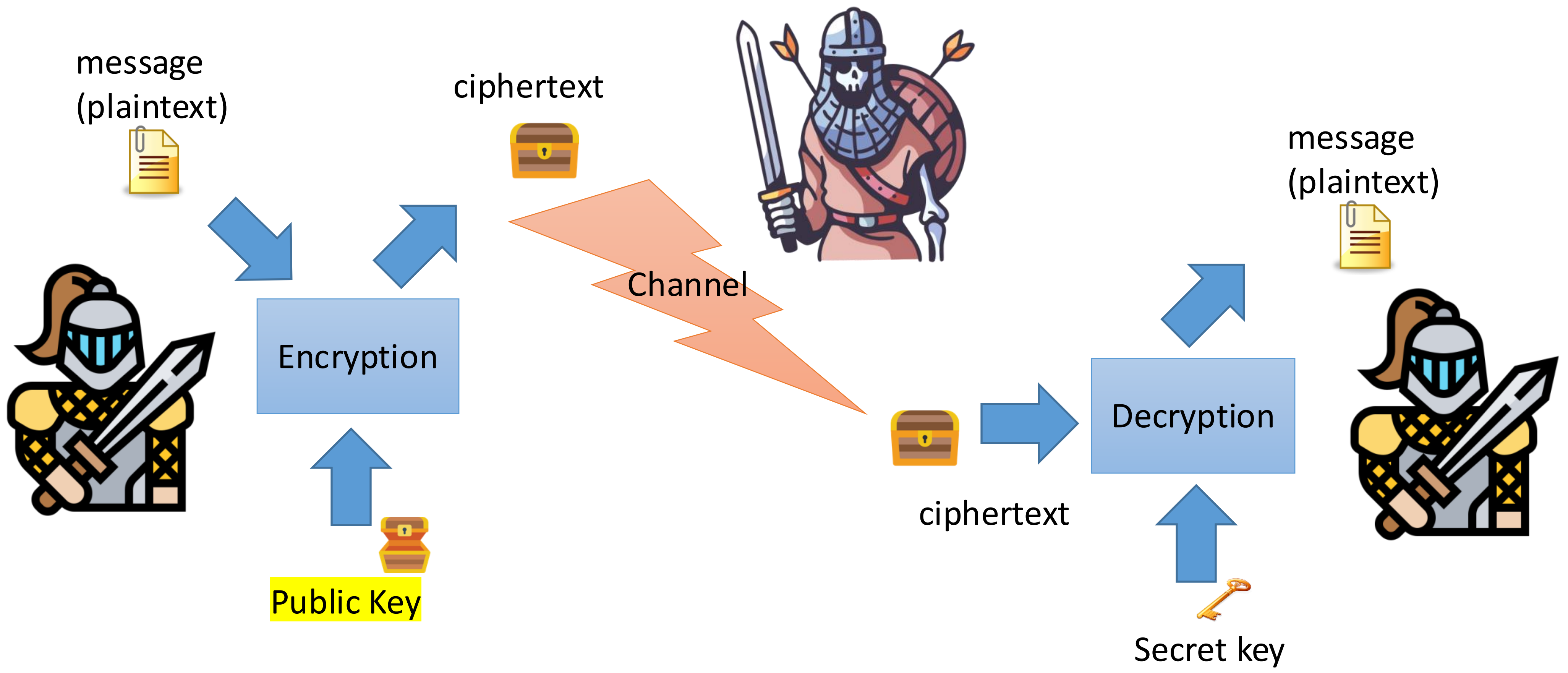
```
iMac:codes orbbyrp$ python3.11 sig.py
b'0-\x02\x15\x00\xbb\xfec\x19\xc0\xaeHf\xe4\xb7\xa3\xf5~\xf1\xa9\x91\x98w#\xaa\x
02\x14\x17\xae\xe0\x16\xb6\xf7\x14\xe9\x12\xfe\xe9N(!\xc7\xec\x01\x03{\xb1'
signature is a valid signature for data
Traceback (most recent call last):
  File "/Users/orbbyrp/Desktop/CSCI 361 (2024)/codes/sig.py", line 20, in <modul
e>
    public_key.verify(signature, data2,hashes.SHA256())
cryptography.exceptions.InvalidSignature
iMac:codes orbbyrp$
```

The codes are implemented using the pyca/cryptography library (https://cryptography.io/).

- This is a python library depending on OpenSSL

Public-Key Encryption from Cyclic Groups

Public-Key Encryption



Public-Key Encryption

- KeyGen(λ): Taking as input a security parameter λ, the P.P.T. algorithms returns (pk,sk)

- Enc(pk, M): Taking as input a message M and a public key pk, the algorithm returns a ciphertertext denoted by CT.

$$CT \leftarrow Enc(pk, M)$$

- Dec(CT,sk): Taking as input ciphertext CT and the secret key sk, the algorithm returns M or $\perp$.

Public-Key Encryption

- **Correctness**: For all generated (pk,sk) and all CT$\leftarrow$Enc(pk, M), we have

$$\Pr[\mathrm{Dec}(CT, sk)=M]=1$$

Security Model (IND-CPA)

Setup: The challenger chooses a key pair (pk,sk) and pk is given to the adversary.

Challenge: The adversary chooses any two different messages M_0 and M_1. The challenger chooses a random c and computes the challenge ciphertext

$$CT^* = Enc(M\_c, pk),$$

which is given to the adversary.

Guess: The adversary returns the guess c’ and wins if c’=c.

We say that the encryption is secure if every P.P.T adversary can only win the game with **negligible** advantage defined as

$$\Pr[c' = c] - \tfrac{1}{2}$$

Preliminaries on Cyclic Group

- Let (G,g,p) be a cyclic group, where G is the set of group element, g is the generator, and p is the group order:
    - $G=\{ g^0, g^1 , . . . , g^{p-1}\}$
    - $g^p=1$
- The following operations are easy in the group (G,g,p):
    - Given any h_1, h_2 in G, it is easy to compute $h_1 \cdot h_2$
        - For any h in G and for any x,y in [0,p-1], given h^x and h^y, it is easy to compute $h^{x+y} =h^x \cdot h^y$
        - For any h_1, h_2 in G and for any x in[0,p-1], given h_1^x and h_2^x , we can compute $(h_1 \cdot h_2)^x= h_1^x \cdot h_2^x$
    - Given any h in G and any x in [0,p-1], it is easy to compute h^x
- The following operations are hard in the group (G,g,p):
    - Given g^x, it is hard to compute x (The DL problem)
    - Given g^x and g^y, it is hard to compute g^{xy} (The CDH problem)
    - Given g^x and g^y, it is hard to distinguish g^{xy} from a random group element in G (The DDH problem)

The El Gamal Cryptosystem

- **Key generation**:
    - Choose a group G of order q and a generator *g* of G. Then choose a uniform $x \in Z_q$ and compute $h = g^x$.
    - The public key is (G, q, g, h) and the private key is (G, q, g, x). The message space is G.
- **Encryption:**
    - on input a public key pk = (G, q, g, h) and a message $m \in G$, choose a uniform $r \in Z_q$ and output the ciphertext $(g^r, h^r \cdot m)$
- **Decryption:**
    - on input a private key sk = (G, q, g, x). and a ciphertext (c_1, c_2) , output $m = c_2 (c_1^x)^{-1}$.

- **Correctness.**
- The El Gamal is not a trapdoor function.

Security of El Gamal Encryption

- The security of the secret key depends on the difficulty of the discrete logarithm problem directly. Hardness of DLP is the *necessary* condition for the security of El Gamal encryption.
- But it is unknown if DLP is the sufficient condition for the security of El Gamal encryption.

Security of El Gamal Encryption

- If one can solve the CDH problem, then it can decrypt without having to know the secret key. Hardness of CDH is the *necessary* condition for the security of El Gamal encryption.
- If one can recover the message m from a ciphertext $(c_1, c_2) = (g^r, h^r \cdot m)$ given only the public key (G, q, g, h) , then it can compute $h^r = g^{rx}$ given $h = g^x$ and g^r. Hardness of CDH is the *sufficient* condition for the one-way security of El Gamal encryption.

Security of El Gamal Encryption

- If one can solve the DDH problem, then it can distinguish encryption of 1 and encryption of a random group element. Hardness of DDH is the *necessary* condition for the IND-CPA security of El Gamal encryption.
- If we further assume hardness of the decisional Diffie-Hellman problem, the ciphertext would be indistinguishable from random group elements, i.e., El Gamal is IND-CPA secure.

Security of ElGamal encryption in different groups

- The group Z_p^* = {1, 2, …, p-1} where p is a large prime number
  - The DLP and the CDH problem are assumed to be hard in Z_p^* for large enough p.
    - The El Gamal encryption in Z_p^* has one-way CPA security.
  - The DDH problem is easy in Z_p^*.
    - The El Gamal encryption in Z_p^* does not have IND-CPA security.
- The prime order subgroup of Z_p^*
  - The DLP, the CDH problem, and the DDH problem are assumed to be hard in this subgroup if the group order is large enough.
    - The El Gamal encryption in this subgroup has IND-CPA security.

The parameters of schemes based on DLP

- If we use the group Z_p^*. The modulus p should have the same size as that of the RSA modulus N for the same security level
  - 80-bit security: p is a 1024-bit prime number
  - 112-bit security: p is a 2048-bit prime number
  - 128-bit security: p is a 3072-bit prime number
- If we use the subgroup G of Z_p^* that has prime order q
  - 80-bit security: p is a 1024-bit prime number and q is a 160-bit prime number
  - 112-bit security: p is a 2048-bit prime number and q is a 224-bit prime number
  - 128-bit security: p is a 3072-bit prime number and q is a 256-bit prime number

Another Look at ElGamal and (Fully) Homomorphic Encryption

The El Gamal Cryptosystem

- **Key generation**:
  - Choose a group G of order q and a generator *g* of G. Then choose a uniform $x \in Z_q$ and compute $h = g^x$.
  - The public key is (G, q, g, h) and the private key is (G, q, g, x). The message space is G.
- **Encryption:**
  - on input a public key pk = (G, q, g, h) and a message $m \in G$, choose a uniform $r \in Z_q$ and output the ciphertext $(g^r, h^r \cdot m)$
- **Decryption:**
  - on input a private key sk = (G, q, g, x). and a ciphertext (c_1 , c_2) , output $m= c_2 (c_1^x)^{-1}$.

- Now, let us try to ”multiple” two ciphertexts…

The El Gamal Cryptosystem

- **Key generation**:
  - Choose a group G of order q and a generator *g* of G. Then choose a uniform $x \in Z_q$ and compute $h = g^x$. x
  - The public key is (G, q, g, h) and the private key is (G, q, g, x). The message space is G.
- **Encryption:**
  - on input a public key pk = (G, q, g, h) and a message $m \in G$, choose a uniform $r \in Z_q$ and output the ciphertext $(g^r, h^r \cdot m)$
- **Decryption:**
  - on input a private key sk = (G, q, g, x). and a ciphertext (c_1 , c_2) , output $m= c_2 \, (c_1^x)^{-1}$.

Homomorphic Evaluation:

$(c_1 , c_2) = (g^r, h^r \cdot m)$

$(c'_1 , c'_2) = (g^{r'}, h^{r'} \cdot m')$

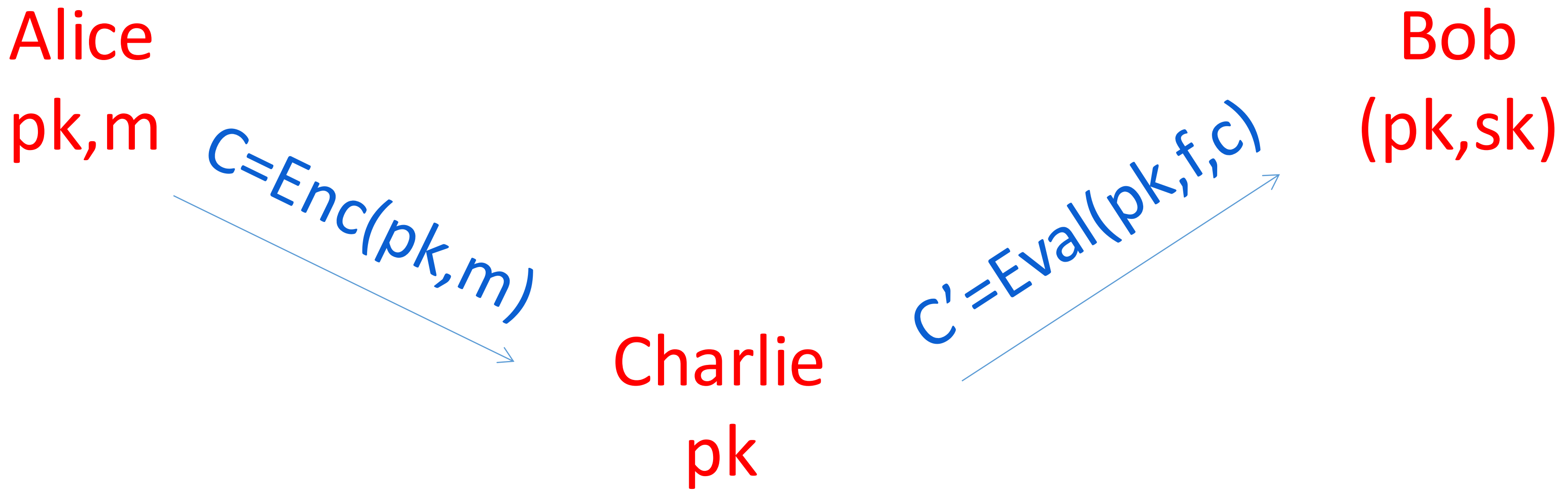
$(c_1 \cdot c'_1, c_2 \cdot c'_2) = (g^{r+r'}, h^{r+r'} \cdot m \cdot m')$

Homomorphic Encryption

- An encryption scheme that supports homomorphic operations over ciphertexts are denoted as homomorphic encryption.
- Why homomorphic encryption is useful?
  - Assume that you would like to compute the product of some numbers, which are distributed among different users.
  - The numbers are encrypted before being sent to you to make them secure.
  - In addition, you do not want to decrypt all ciphertexts and compute the result; instead, you would like to borrow the computation power of some untrusted cloud.
  - Then the ciphertexts will be sent to the cloud and the cloud will compute the product of the ciphertexts.
  - Finally, you decrypt the ciphertext returned by the cloud to recover the product of the numbers.
- It would be more useful if we can perform any computation over the encrypted data.
  - That is denoted as fully homomorphic encryption.

Fully Homomorphic Encryption

Bob generates a key pair (pk,sk). pk is public key and published to all others, while sk is secret and only known by Bob.



The data that Alice sent to Bob is unknown to the adversary, even Charlie.

Fully Homomorphic Encryption

- KeyGen(λ): Taking as input a security parameter λ, the P.P.T. algorithms returns (pk,sk)

- Enc(pk, M): Taking as input a message M and a public key pk, the algorithm returns a ciphertertext denoted by CT.

  CT←Enc(pk, M)

- Dec(CT,sk): Taking as input ciphertext CT and the secret key sk, the algorithm returns M or $\perp$.

- Eval(pk,f,CT): Taking as input ciphertext CT, public key pk, and a function f, the algorithm returns another ciphertext CT’

Fully Homomorphic Encryption

- **Correctness**: For all generated (pk,sk), all CT←Enc(pk, M), and all function f, we have

  Pr[Dec(Eval(pk,f,CT), sk)=f(M)]=1

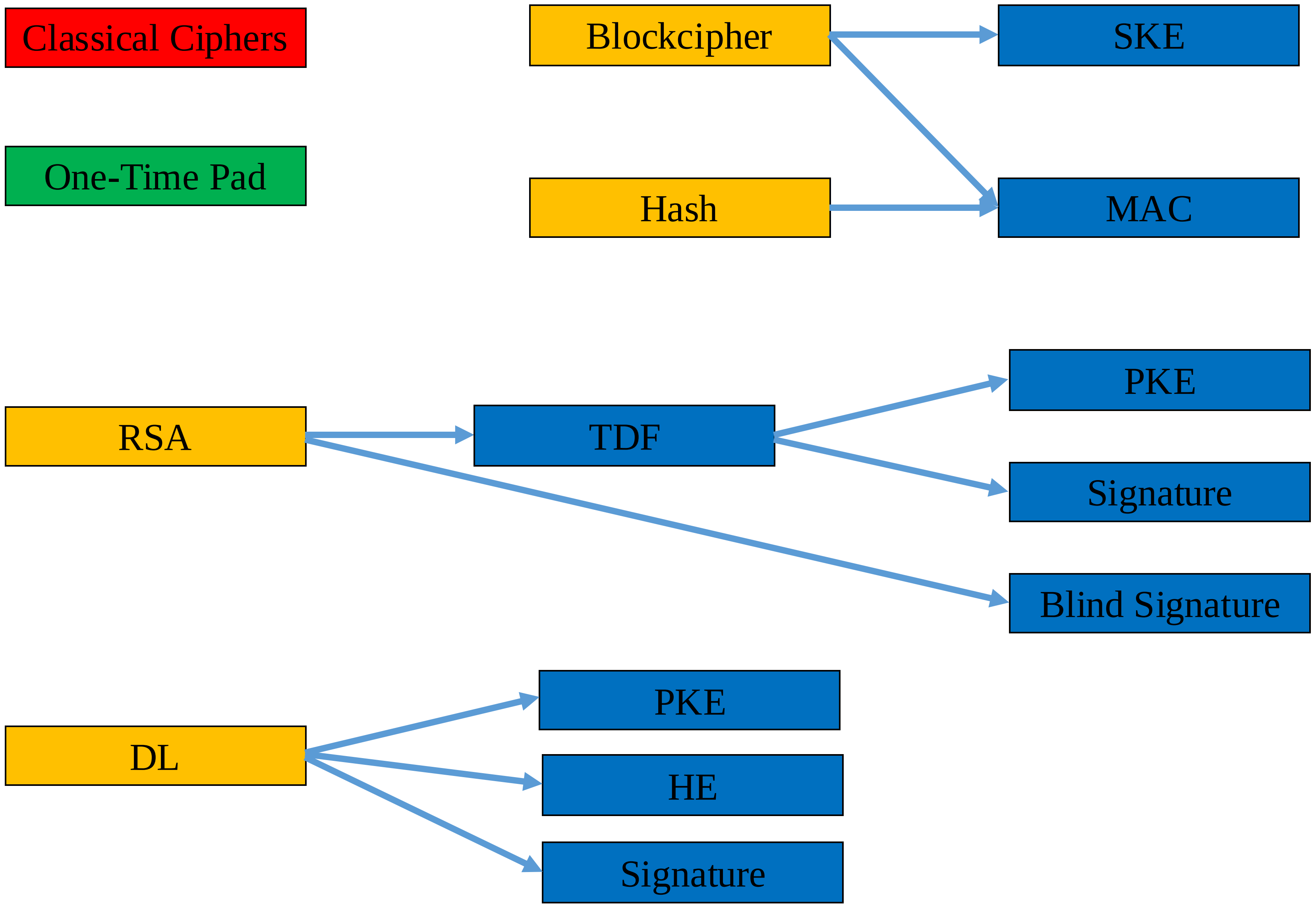
- **Security**: Standard IND-CPA security.

- **Construction:** Existing FHE schemes are constructed from lattice and the concrete constructions are beyond the scope of this subject.

Implementing PKE II

- El Gamal encryption is not naturally supported in main stream cryptography libraries like OpenSSL.
- You can implement it by using existing interfaces such as DH key exchange or DSA signature.
  - You are not recommended to do this unless you have a strong reason.

Roadmap


Classical Ciphers
One-Time Pad
Blockcipher
Hash
SKE
MAC
RSA
TDF
PKE
Signature
Blind Signature
DL
PKE
HE
Signature

Summary

- Cyclic Group
- Schnorr Signature
  - Construction
  - Correctness
  - Security
    - Insecurity of weak implementation
- ElGamal Encryption
  - Construction
  - Correctness
  - Security
- Homomorphic Encryption
  - Notion and syntax
  - Partially homomorphic encryption from ElGamal