CSCI471/971
Modern Cryptography
Public Key Cryptography II

Rupeng Yang
SCIT UOW

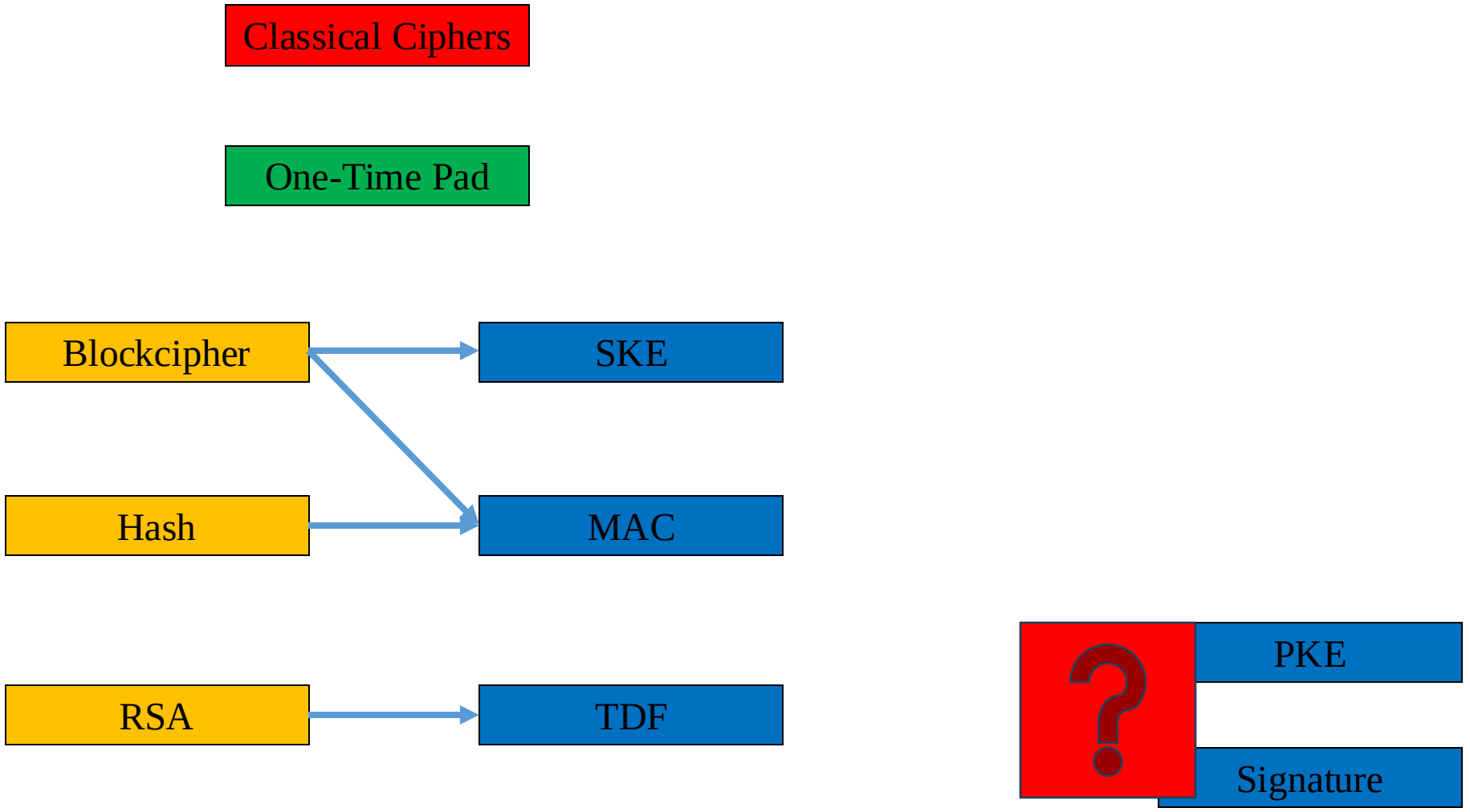
RoadMap

- Week 1-2: Preliminaries

- Week 3-4: Symmetric-Key Cryptography
  - All parties share the same secret key
  - Symmetric-key encryption
  - Message authentication code

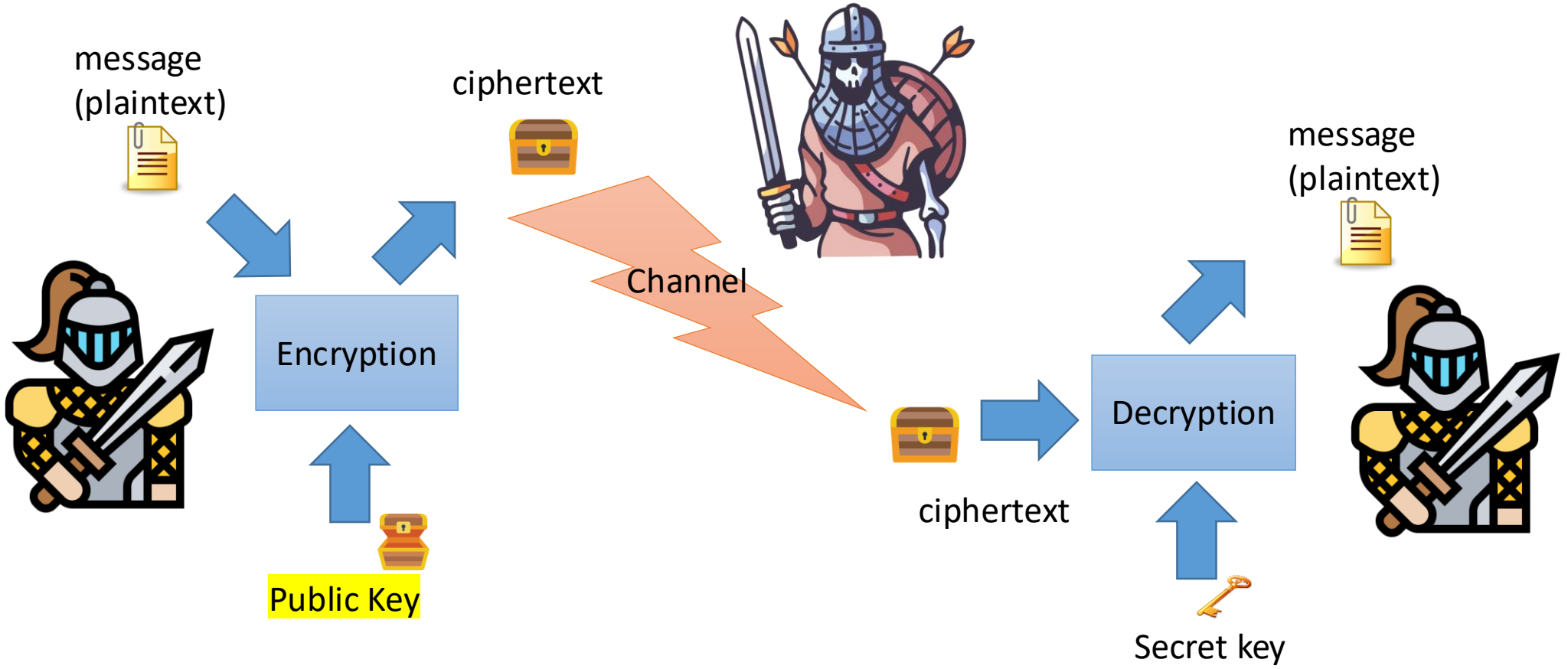
- Week 5: Attempts to Construct Public-Key Cryptography
- Week 6: Secure Public-Key Encryption and Digital Signature

Roadmap


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

Definition of Public-Key Encryption

Public-Key Encryption



Public-Key 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. Any party can encrypt a sensitive message using the public key and Bob can decrypt the ciphertext using the secret key.

The data that Alice sent to Bob is unknown to the adversary.
Bob doens't even need to know who is Alice (no need to share a secret key!)

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←Enc(pk, M), we have

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

Security Model of Public-Key Encryption

- What is the adversary's capabilities?

- *Ciphertext only*: Attacker only knows some ciphertexts.
- *Known plaintext*: Attacker knows some plaintext-ciphertext pairs.
- *Chosen plaintext*: Attacker is allowed to choose some plaintexts, and receives the corresponding ciphertexts.
- *Chosen ciphertext*: Attacker is allowed to choose some plaintexts, and receives the corresponding ciphertexts; attacker is also allowed to choose some ciphertexts and receives the corresponding plaintexts.

Security Model of Public-Key Encryption

- What is the adversary's capabilities?

- ~~*Ciphertext only*: Attacker only knows some ciphertexts.~~
- ~~*Known plaintext*: Attacker knows some plaintext-ciphertext pairs.~~
- *Chosen plaintext*: Attacker is allowed to choose some plaintexts, and receives the corresponding ciphertexts.
- *Chosen ciphertext*: Attacker is allowed to choose some plaintexts, and receives the corresponding ciphertexts; attacker is also allowed to choose some ciphertexts and receives the corresponding plaintexts.

- In PKE, the adversary can encrypt by itself, thus it does not make sense to consider the ciphertext-only attack and the known-plaintext attack, and there is no need to query ciphertext in CPA. (Note that the adversary should know pk)

Security Model of Public-Key Encryption

- What is the adversary's capabilities?

- ~~*Ciphertext only*: Attacker only knows some ciphertexts.~~
- ~~*Known plaintext*: Attacker knows some plaintext-ciphertext pairs.~~
- *Chosen plaintext*: Attacker is allowed to choose some plaintexts, and receives the corresponding ciphertexts.
- *Chosen ciphertext*: Attacker is allowed to choose some plaintexts, and receives the corresponding ciphertexts; attacker is also allowed to choose some ciphertexts and receives the corresponding plaintexts.

- In PKE, the adversary can encrypt by itself, thus it does not make sense to consider the ciphertext-only attack and the known-plaintext attack, and there is no need to query ciphertext in CPA. (Note that the adversary should know pk)

Security Model of Public-Key Encryption

What is the adversary's security goal?

- Onewayness: Attacker cannot recover the plaintext.
- Semantic Security: Attacker cannot learn any information about the plaintext.
- Indistinguishability : Given a ciphertext CT* and two messages M_0 and M_1 where CT*=Enc(M_c, K), the adversary is going to compute c from {0,1}.

Public-Key Encryption: IND-CPA Security

Algorithms KeyGen, Enc, Des are public.

1. Run KeyGen to get (pk,sk)

2. Send pk to the Attacker

3.1 Attacker sends m_0, m_1 to challenger

3.2 Randomly choose a bit b, run Enc(pk, m_b) to get c*

3.3 Send c* to the Attacker

We assume that ($|m_0|=|m_1|$)

4. Attacker returns a guess bit b'

Attacker wins if b = b'!

An Encryption Scheme is Secure if ***NO*** efficient attacker can win with a probability of ½+1/poly(λ).

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

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

Security Model (IND-CCA)

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

Phase 1: The adversary can choose any CT for decryption queries.

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.

Phase 2: The adversary can choose any CT different from CT* for decryption queries.

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

$$\mathbf{Pr[c'=c]-½}$$

Security Model (IND-CCA1)

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

Phase 1: The adversary can choose any CT for decryption query.

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.

~~Phase 2: The adversary can choose any CT different from CT* for decryption query.~~

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

$$\mathbf{Pr[c'=c]-½}$$

PKE with IND-CPA Security from One-Way Trapdoor Function

One-Way Trapdoor Function

(Definition) A function f:{0,1}*→{0,1}* is a one-way trapdoor function if

- Easy to Compute: There exists a P.P.T algorithm that can compute f(x) for any x.
- Hard to Invert: For every P.P.T. adversary, given f(x), where x is sampled uniformly at random, we have

$$Pr[f(\ A(f(x))\ ) = f(x)] \leq negl$$

- Easy to Invert with Trapdoor: There exists a trapdoor td and a P.P.T algorithm that given td and f(x), it is easy to compute x.

- We can view the encryption algorithms of the textbook RSA encryption as the function and view their decryption algorithms as the inverse of the function.
- The function is defined by the public key and the trapdoor is the secret key.

Pseudorandomness from One-Way (Trapdoor) Functions

Let f be a one-way function. Functions g and gl are constructed as follows: set $g(x,r) \stackrel{\text{def}}{=} (f(x),r)$, for $|x| = |r|$, and define

$$\text{gl}(x,r) \stackrel{\text{def}}{=} \bigoplus_{i=1}^{n} x_i \cdot r_i,$$

we have

$$Pr[A(f(x),r\,) = gl(x,r)] \leq \frac{1}{2} + negl$$

where x and r are sampled uniformly at random, i.e., gl(x,r) is pseudorandom given f(x) and r.

Public-Key Encryption from one-way trapdoor functions

- KeyGen(λ): Taking as input a security parameter λ, the key generation algorithm generates (f, td) and sets pk=f, sk=td

- Enc(pk, M): Taking as input a message M of one bit and a public key pk, the encryption algorithm samples a uniform x and a uniform r, then it returns a ciphertertext denoted by CT:

$$CT \leftarrow (f(x), r, gl(x,r) \oplus M)$$

- Dec(CT,sk): Taking as input a ciphertext CT=(C1,C2,C3) and the secret key td, the decryption algorithm uses td to invert C1 and gets x, then it computes the message as $gl(x,r) \oplus C3$.

Public-Key Encryption from one-way trapdoor functions

- KeyGen(λ): Taking as input a security parameter λ, the key generation algorithm generates (f, td) and sets pk=f, sk=td

- Enc(pk, M): Taking as input a message M of one bit and a public key pk, the encryption algorithm samples a uniform x and a uniform r, then it returns a ciphertertext denoted by CT:

$$CT \leftarrow (f(x), r, gl(x, r) \oplus M)$$

- Dec(CT,sk): Taking as input a ciphertext CT=(C1,C2,C3) and the secret key td, the decryption algorithm uses td to invert C1 and gets x, then it computes the message as $gl(x, r) \oplus C3$.

- The scheme is IND-CPA secure if f is a secure one-way trapdoor function.

Preliminaries on RSA

- It is easy to generate large random prime numbers
- It is easy to compute y s.t. $xy = 1 \, mod \, N$ given x and N that are relatively prime.
- Given any number N = pq, we define ϕ(N)= (p-1)(q-1), then for any a that is relatively prime with N, we have

$$a^{\phi(N)} = 1 \, mod \, N$$

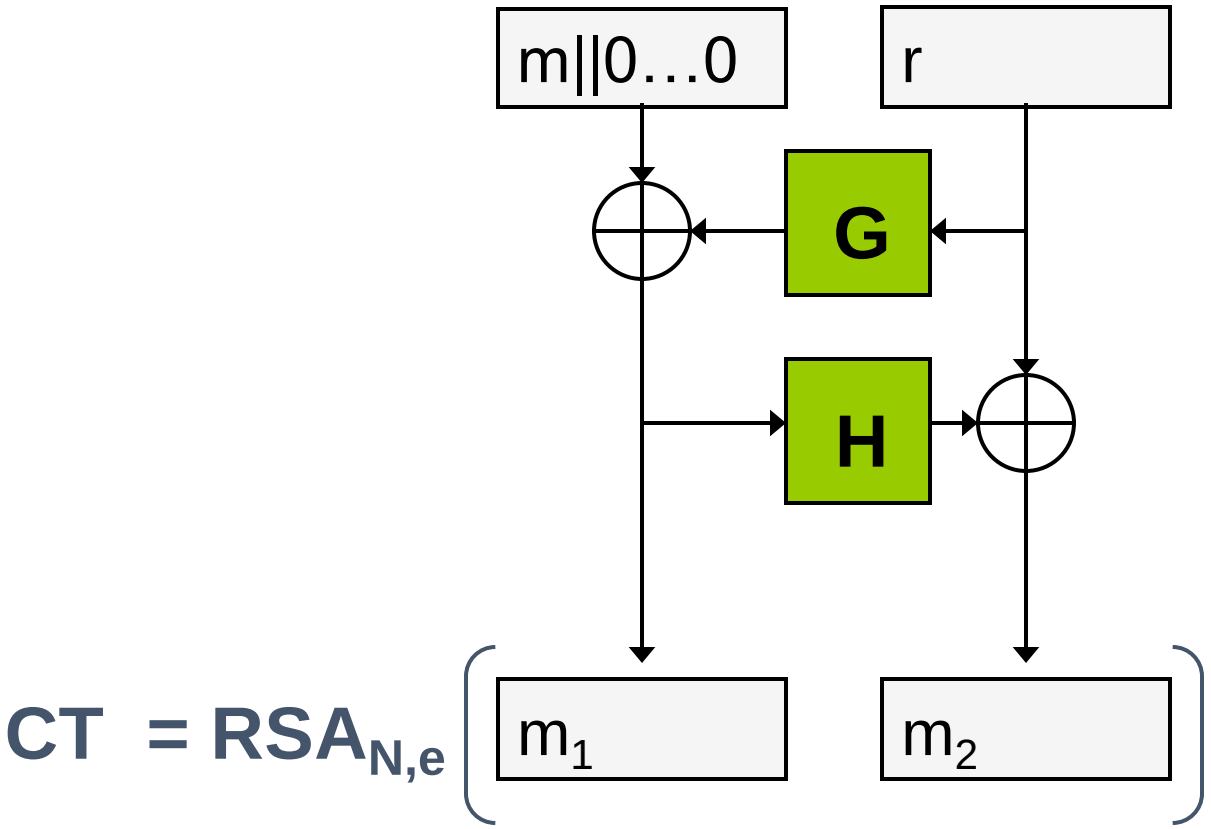
- It is easy to compute x^e given x and e.
- It is hard to factorize N=pq if p and q are large enough primes.

RSA as a Trapdoor One-Way Function

- Generating the function and the trapdoor
  1. Choose two large primes p and q. Compute n = pq and $m=\phi(n)= (p-1)(q-1)$.
  2. Choose a random e, such that $1 \leq e \leq m - 1$ and gcd(e,m)=1.
  3. Finds d such that ed=1 mod m.
  4. The **function** is described by **(e, n)**.

  The **trapdoor** is **(d, n)**.
- Computing the function given an input X: $Y = X^e \bmod n$.
- Inverting the function on Y: $X = Y^d \bmod n$.

Optimal Asymmetric Encryption Padding (RSA-OAEP)



- G, H: cryptographic hash functions
- r: a random string
- m_1= m \xor G(r)
- m_2=H(m_1) \xor r

Decryption:

1. Invert E_{pk} on the ciphertext.
2. Invert the Feistel Network on the result of the first step.
3. Check the results of the first two steps and returns the message or an error symbol.

The RSA-OAEP (and Rabin-OAEP) can provide IND-CPA/IND-CCA security.

Implementing PKE

```
import os
from cryptography.hazmat.primitives.asymmetric import
rsa,padding
from cryptography.hazmat.primitives import hashes

# Key Generation
private_key = rsa.generate_private_key(
public_exponent=65537,
key_size=2048)
public_key=private_key.public_key()

# Encryption
message = b"encrypted data"
ciphertext = public_key.encrypt(message,padding.OAEP(
mgf=padding.MGF1(algorithm=hashes.SHA256()),
algorithm=hashes.SHA256(),
label=None))
print(ciphertext)

# Decryption
dmessage= private_key.decrypt(ciphertext,padding.OAEP(
mgf=padding.MGF1(algorithm=hashes.SHA256()),
algorithm=hashes.SHA256(),
label=None))
print(dmessage)
```

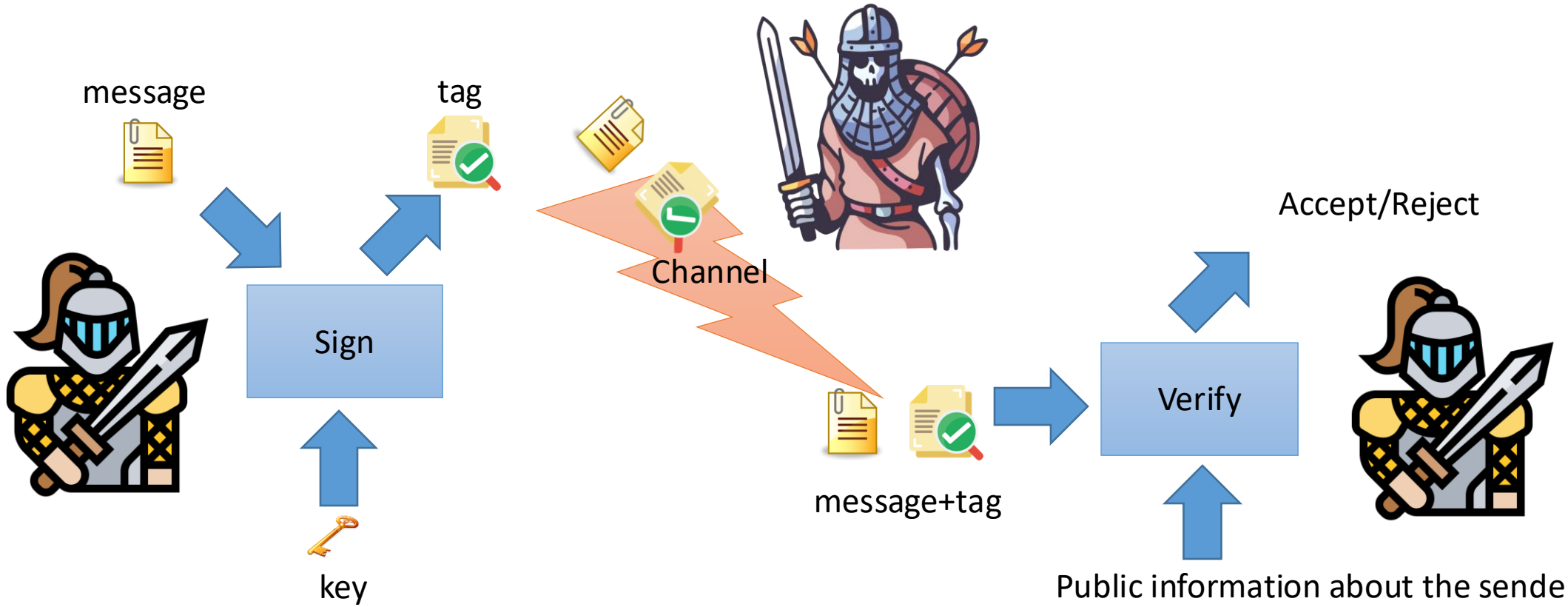
```
iMac:codes orbbyrp$ python3.11 pke.py
b'{y\x8fI\x0b\xe0\xfd\x0cl\x1b^-\x17P\xb0\xddK\x87\xf7o\xef\tEJ\x98M\x0e\xdb\xf9
\x1ch\xda\xeb\x84\xeb\x0c\x83\xe2\xac\xdc}\xfa!\xb8a\n\x9eW7\xdd\xebe\x1a\xc5\xb
1j\xb04\xd4y1\xf2K\x19\x14I\xc9\xae\xf0RP\xe4\xf0?.\x0ed;\x10\x80^bR\xad\\\xd8R\
xd5afe\xae\xe9\x84^\xbc-\xed\xda\xe2\xf8\x7f\xf8\xba\x1a\xb5^\xd9\xc3\xa8\x07\xf
1\xd2)+\x99f\xa1\xd0z\x8c6\x96\xfc?e\x11L]\xc2\xbfe\x96,\xc3\xba\xe1%C\xf2\xbd=\
xd6\xf0\xe6x\x13S\xb3G\t7"\r(\x94\xf6\xa7[\x97X\xea\xfd\xff\xd4T-\xb7\xf60\xcc\x
ae|sQA(\x96sq\xa9\xd4b|\x1e\xdfp\xba\xa5\x85-\xf2\x86d\x95\xa4@\xaa\xa6\xfeM\xbe
\xbd\xc4\x02]\xdd\xd3&\x07\xac\xa3\x9dP(`\x8c\x8a%#\xeb`\xe4\xce\x85\xec\xd1\n\x
868\x91D%o<E\x00"\x99t\xc8~\x02\xefr\r\xde\x7f\xd9\xed\x98\x13\x86\x035S'
b'encrypted data'
```

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

- This is a python library depending on OpenSSL

Definition of Digital Signature

Digital Signatures



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.

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

- 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

$$\Pr[\text{Verify}(S,M,pk)=1]=1$$

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

Security Model for Signature

- Our objective is to prevent the adversary from modifying the message without being detected.
- The adversary can modify both the message and the tag, since it controls the communication channel.
- Thus, we require that the adversary cannot generate a valid message tag pair if it modifies the message
- **Security Goal**
  - **Unforgeable**: It is hard to generate a valid tag for a *new* message.

Security Model for Signature

- Adversary's capability
  - Known tag attack: The adversary knows some messages and the associated tags for them.
  - Chosen Message Attack: Attacker is allowed to choose some messages, and receives the corresponding tags.

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).

Secure Signature from One-Way Trapdoor Function

One-Way Trapdoor Function

(Definition) A function f:{0,1}*→{0,1}* is a one-way trapdoor function if

- Easy to Compute: There exists a P.P.T algorithm that can compute f(x) for any x.
- Hard to Invert: For every P.P.T. adversary, given f(x), where x is sampled uniformly at random, we have

$$Pr[f(\ A(f(x))\ ) = f(x)] \leq negl$$

- Easy to Invert with Trapdoor: There exists a trapdoor td and a P.P.T algorithm that given td and f(x), it is easy to compute x.

- We can view the encryption algorithms of the textbook RSA encryption as the function and view their decryption algorithms as the inverse of the function.
- The function is defined by the public key and the trapdoor is the secret key.

Digital Signatures from One-Way Trapdoor Functions

- KeyGen(λ): Taking as input a security parameter λ, the P.P.T. algorithms generates (f, td) and specifies a secure hash function h: {0,1}* $\rightarrow Z_N^*$. Then it sets pk=(f,h), sk=td.

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

  $S \leftarrow f^{-1}(h(M))$: This is to invert h(M)

- Verify(S,M,pk): Taking as input signed message (S,M) and the public key pk, the P.P.T. algorithm computes and accepts the signature if f(S)=h(M).

- Correctness.

Security of The Signature

- To generate a valid signature s for a message m, you need to ensure both H(m) and f(s) are the same value x.
  - H is a secure hash function, so, it is hard to compute $H^{-1}(x)$
  - f is also one way, it is also hard to compute $f^{-1}(x)$.
  - So, you need to reach the same x from both m and s, which is also hard.
- If the adversary can further make some signing oracle queries, we also need the hash function to be collision resistant.
- To support formal proof, H should be modelled as an idealized random oracle.

Preliminaries on RSA

- It is easy to generate large random prime numbers
- It is easy to compute y s.t. $xy = 1 \, mod \, N$ given x and N that are relatively prime.
- Given any number N = pq, we define ϕ(N)= (p-1)(q-1), then for any a that is relatively prime with N, we have

$$a^{\phi(N)} = 1 \, mod \, N$$

- It is easy to compute x^e given x and e.
- It is hard to factorize N=pq if p and q are large enough primes.

RSA Full-Domain Hash (RSA-FDH) Signature

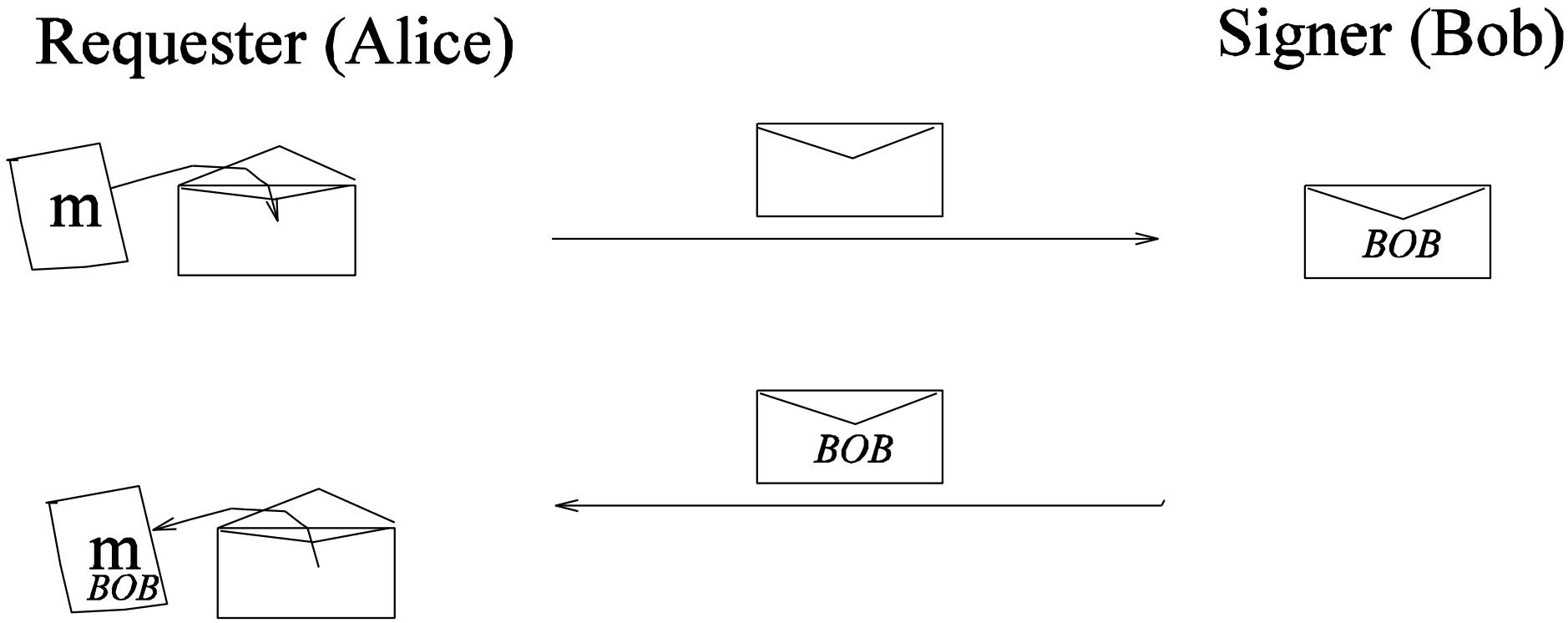
- Key Generation:
  - Generate primes P and Q, compute N = PQ
  - Generate d and e such that de = 1 mod (P-1)(Q-1)
  - Public Key (N, e)
  - Private Key d
  - Specify a secure hash function H: {0,1}^* $\rightarrow Z_N^*$
- Sign:
  - Given message m, compute s = $H(m)^d$ mod N
- Verify:
  - Given message m, signature s, check if H(m) = s^e mod N
- Correctness

Blind Signature and RSA Blind Signature

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 some applications like electronic voting or electronic cash, we need to ensure that each user only has one electronic ballot or a fixed number of electronic coins:
  - To request a ballot, a voter randomly generates a serial number.
  - The voter needs the organization's signature on the ballot (i.e., the serial number) before being able to use it.
  - To ensure privacy of the vote, that is to ensure that the organization cannot link a vote with a particular voter, we need the organization to sign on the serial number without knowing it.
  - In addition, the signature scheme needs to guarantee that the voter cannot show two valid signatures if it only asks for one signature, and this guarantees that the voter can vote for at most once.

Blind Signature



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 σ, where anyone can verify the correctness of (m, σ).
- The requester cannot get/forge signatures on other values.

Blind Signatures

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

- Sign<$Signer_{sk}$,$Requester_{M}$>: This is a protocol run between a signer with the secret key sk and a requester with a message M. After the protocol, the requester learns a signature on M.
  - Usually, it takes 3 steps:
  - First, the requester sends a masked message to the signer
  - Then, the signer signs on the masked message and returns the “masked signature”
  - Finally, the requester recovers the real signature from the "masked signature”

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

Blind Signatures

- **<u>Correctness</u>**: For all generated (pk,sk) and all $S \leftarrow \text{Sign}\langle \text{Signer}_{sk}, \text{Requester}_{M} \rangle$, we have

$$\Pr[\text{Verify}(S,M,pk)=1]=1$$

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

Next, let us try to define the security…
We need to define unforgeability (since it is a signature) and privacy of message (since it is a blind signature).

Unforgeability of Signature

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).

However, in blind signature, anyone can get the signature without revealing the message…

- Can we force the adversary to reveal the message to the challenger?
  - Yes we can define like this, but in the applications, the conditions may not be satisfied….
- The problem is solved by requiring the adversary to give q+1 different sigantures if it has made q queries.
  - The definition is good enough for main applications of blind signature, such as electronic cash or electronic voting.

Unforgeability of Blind Signature

Algorithms KeyGen, Sign, Verify are public.

1. Run KeyGen to get (pk,sk)

2. Send pk to the Attacker

3 Attacker and challenger runs the sign protocol

Repeat for q times

Attacker wins if Verify(pk,m_i*,t_i*)=1 for all i from 1 to q+1, and all m_i* are different.

4. Attacker returns ((m_1*,t_1*), … (m_{q+1}*,t_{q+1}*))

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

Privacy of Blind signature

- Who should be the adversary for privacy?
  - The signer!
  - Our goal is to prevent the singer from knowing any information about the message it has been signed.
- How to define the privacy? Can we define the message indistinguishability following IND-CPA security of PKE?
  - That means, the signer (adversary) selects two messages; the challenger runs the signing protocol with the signer (adversary) using one of them.
  - But this security definition cannot prevent the signer from “marking” you when running the protocol with you, i.e., it may produce a signature that is different from the usual ones and then it can detect you when you use your signature. The attack does not conflict with the security definition.
- How to define the privacy?
  - We allow the signer (adversary) to choose many messages; then the challenger runs the signing protocol with the signer (adversary) using these messages, but in a shuffled order.
  - The adversary’s goal is to match the signing sessions with the messages.
  - We are not going to show the formal definition.

RSA Blind Signatures

- The key generation and the verification algorithm is unchanged.
  - KeyGen:
    - Generate primes P and Q, compute N = PQ
    - Generate d and e such that de = 1 mod (P-1)(Q-1)
    - Public Key (N, e)
    - Private Key (N,d)
  - Verify:
    - Input a message M and a signature S
    - Check $S^e = H(M) \bmod N$

RSA Blind Signatures

- We transform the signing algorithm into a 3-step protocol
  - Message masking (by requester):
    - Choose a random value r
    - Compute m = H(M)
    - Compute B = r^em mod N
    - Send B to the signer
  - Signing (by signer):
    - Signer computes C = B^d mod N
    - Signer sends C back to the requester
  - Signature Extracting (by requester):
    - The requester computes S = C/r mod N
    - S is a valid signature for M

RSA Blind Signatures

- Correctness
  - $C = B^d \bmod N = (r^e m)^d \bmod N = r^{de} m^d \bmod N = r m^d \bmod N$
  - $C/r = m^d \bmod N = H(M)^d \bmod N$
- Unforgeability
  - Unforgeability comes from unforgeability of RSA signature
    - It is possible to generate signature on messages that are not queried!
    - But the attacker can create at most n valid message/signature pairs if he has asked the signer to sign for n times.
- Privacy
  - Unconditional privacy

Implementing Signature

```
import os
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import rsa,padding

# Key Generation
private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
public_key=private_key.public_key()


# Sign
data = b"signed data"
signature = private_key.sign(data,padding.PSS(
      mgf=padding.MGF1(hashes.SHA256()),
      salt_length=padding.PSS.MAX_LENGTH),
   hashes.SHA256())
print(signature)
```

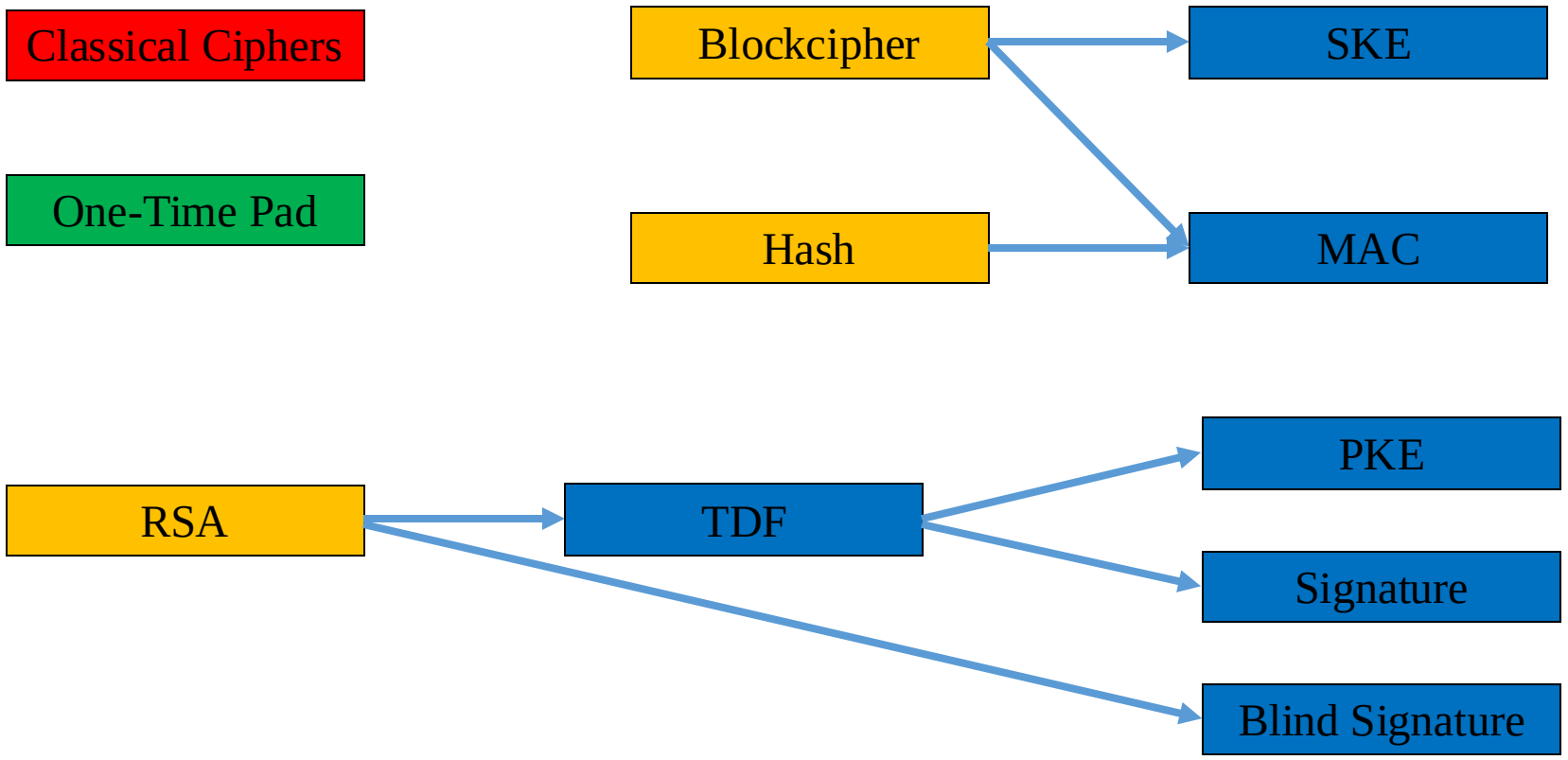
```
data2 = b"signed data2"
public_key.verify(signature, data, padding.PSS(
      mgf=padding.MGF1(hashes.SHA256()),
      salt_length=padding.PSS.MAX_LENGTH),
hashes.SHA256())
print("signature is a valid signature for data")

public_key.verify(signature, data2, padding.PSS(
      mgf=padding.MGF1(hashes.SHA256()),
      salt_length=padding.PSS.MAX_LENGTH),
   hashes.SHA256())
```

```
iMac:codes orbbyrp$ python3.11 sig2.py
b"=\xc1\xd1\x0e\xb5\xbf\xc5\x06\xaf:\x7f\xaeE\xb1 \x9c\xb8\xc8Yg\xb5\x16\xe6\x99
dU5\x1d>\x9dQ\x80\xd8\xca\xb85\xcb^}O\x01;Rt\xa6\xd1\xd4=!\xf0\xd1\xcc\xb9\x12\x
1e\xff7W\xfe>m\xb8\xb6\x1e,\xa3\x90\x9a\xa0\xaf\xd2\x8d\x07\x84\xed\x1b\x97\xe6\
xd9\x9c\x0f@\xb2\x8b^'\xe4P{\x06\x94:b\xac\xcbN\x88\x9ff\x0f\x99y\xc9\xf6\xc9\x0
7\x9b\x02\x0f\x04\xdb\xef\x94\xbd\xb5\xc0\x82\xbc@\xbbu}\xab\xa6\x94\\dpTx\x8c\r
\x14\xc8\t\xc5\x87)\xe6l.\xc5\x89\xc9g\xff\xa6\x15\x9aE\xf8`\x1c=S\t]\x93\xe7J\x
ad\xd5\x87M9Y\xa9\x05L\x91tl\x1e\x0e\x1e\x04\x03\x91\xd7\x9d\x10\xdd?_b\xfb\x0c\
xb7\x9b\xcc\xef\xb5\xack\xe8\x94\xb00<\xe50h\xf9sq\x81\xd2\x86'\xc3\xe1\x0c\x1a;
\xff\xa2\xc8r\xbb\xc0a\xca\x91\xeb2@\x8a{\x03N6\xdf\xcbo\x1a\x1es\xfe{\tF\xcf\xb
8\xaa`\xe5\x14\xbd\x80\xea|\x8c\xe8\xc2e3"
signature is a valid signature for data
Traceback (most recent call last):
  File "/Users/orbbyrp/Desktop/CSCI 361 (2024)/codes/sig2.py", line 26, in <modu
le>
    public_key.verify(signature, data2, padding.PSS(
cryptography.exceptions.InvalidSignature
```

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

- This is a python library depending on OpenSSL

Roadmap


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

Summary

- Secure PKE
  - The Definition
    - Application Scenario
    - Syntax
    - Correctness
    - Security
      - CPA is the minimal requirement
      - IND-CPA security
      - IND-CCA Security
      - IND-CCA1 Security
  - The Construction
    - PKE from one-way trapdoor function
    - RSA as a one-way trapdoor function
    - RSA-OAEP
- Digital Signature
  - The Definition
    - Application Scenario
    - Syntax
    - Correctness
    - Security
  - The Construction
    - Construction from one-way trapdoor function
      - Security*
    - RSA-FDH
  - Blind Signature
    - Definition
      - Application Scenarios
      - Syntax and Correctness
      - Security*
    - RSA Blind Signature