# CSCI471/971
# Modern Cryptography

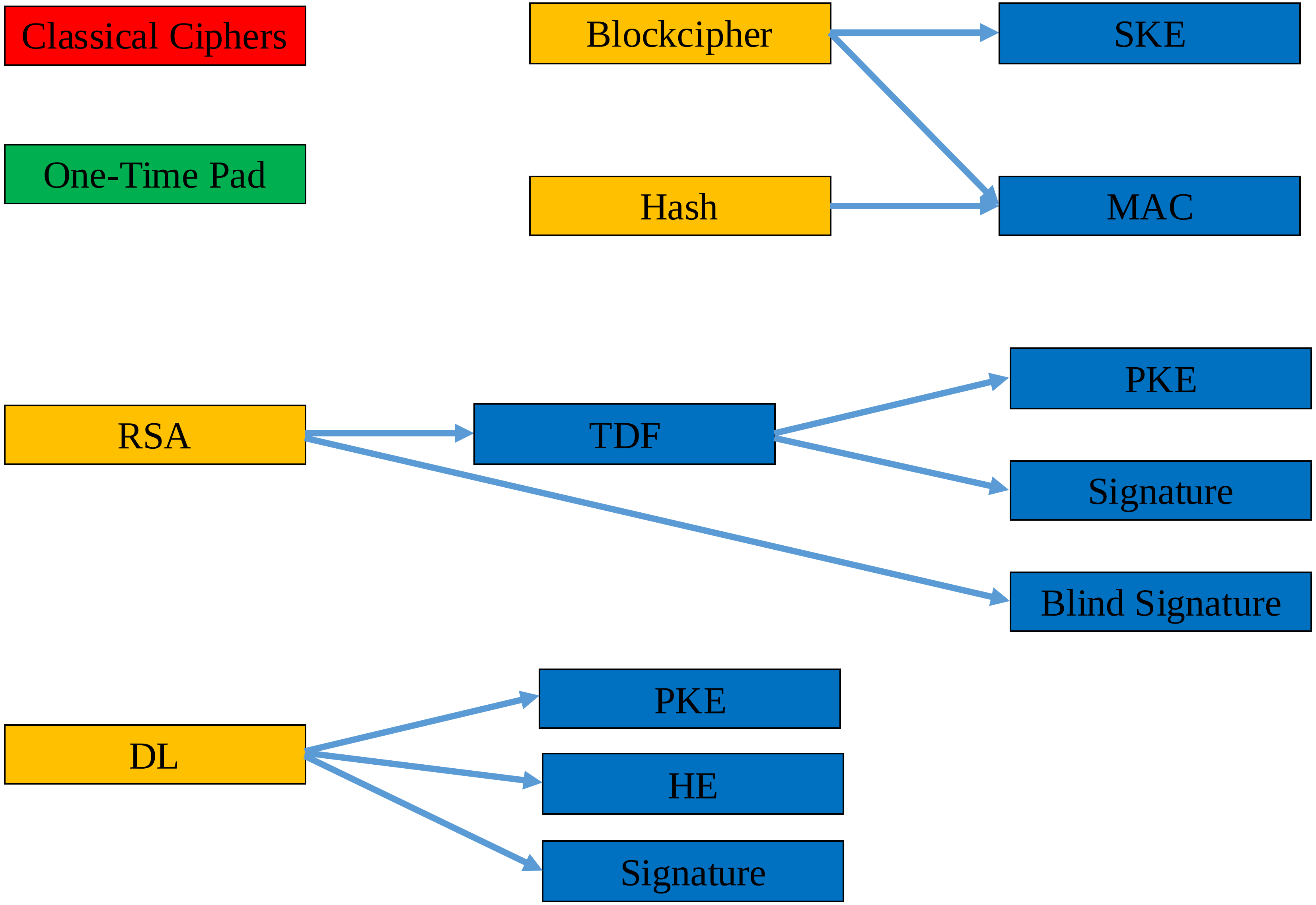
## Key Management

Rupeng Yang
SCIT UOW

# *RoadMap*

- Week 1-2: Preliminaries
- Week 3-4: Symmetric-Key Cryptography
- Week 5-7: Public-Key Encryption and Digital Signature
- Week 8: Key Management

# Roadmap


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

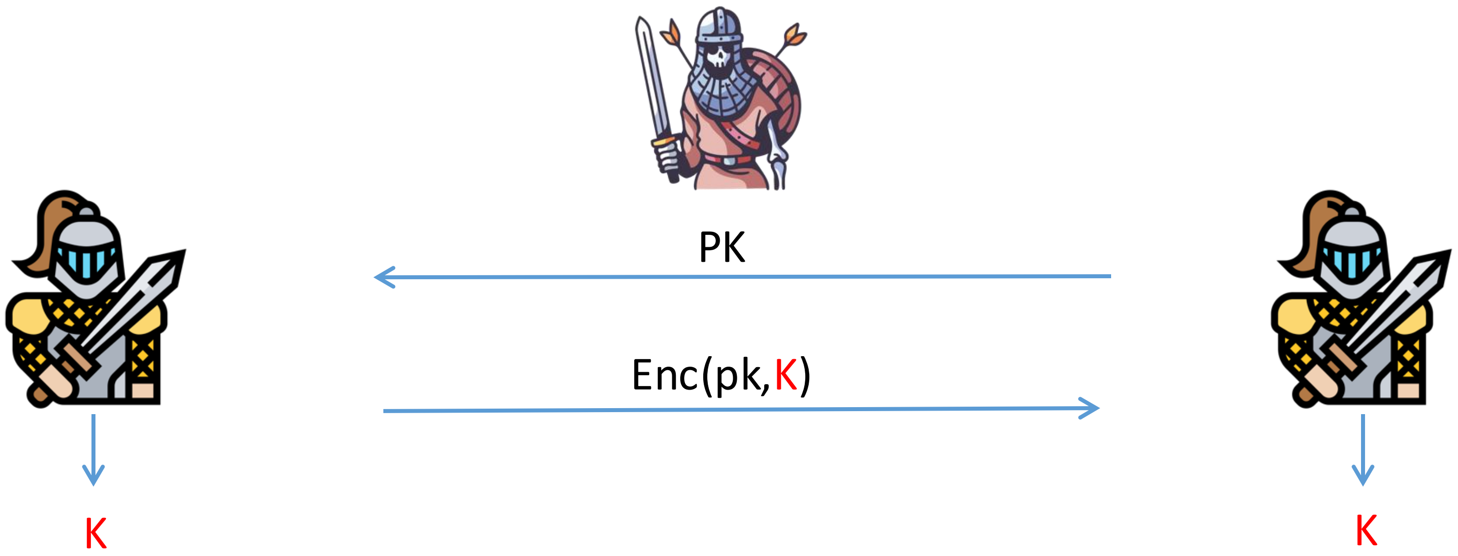
# Motivation

- When discussing symmetric-key cryptosystems, we always assume that secret keys are shared securely between the sender and the receiver. But how?
  - We can use a secure channel.
  - But what if there is no secure channel between the sender and the receiver?

# Key Exchange

# How to share a key securely?

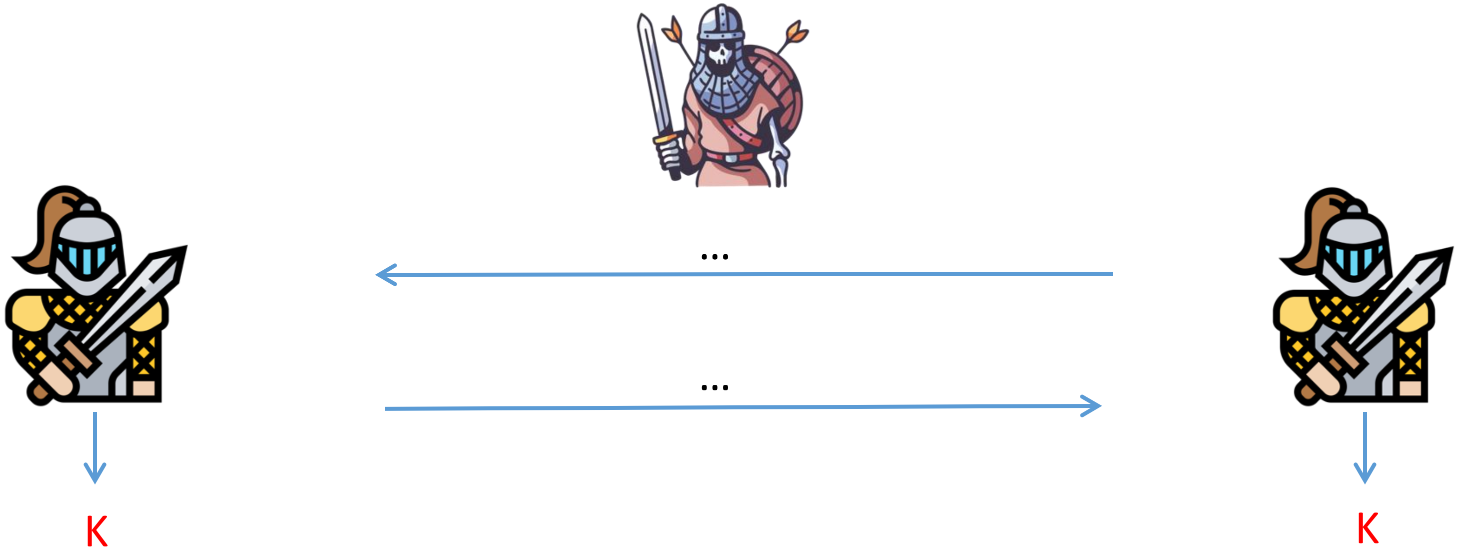
- We can use a PKE scheme. If the public key of the receiver is not known to the sender in advance, then we need complete the above task in the following two steps:




- We do not use the PKE/Signature scheme to protect the communication directly because it is much more expensive to run a PKE/Signature than running a SKE/MAC.

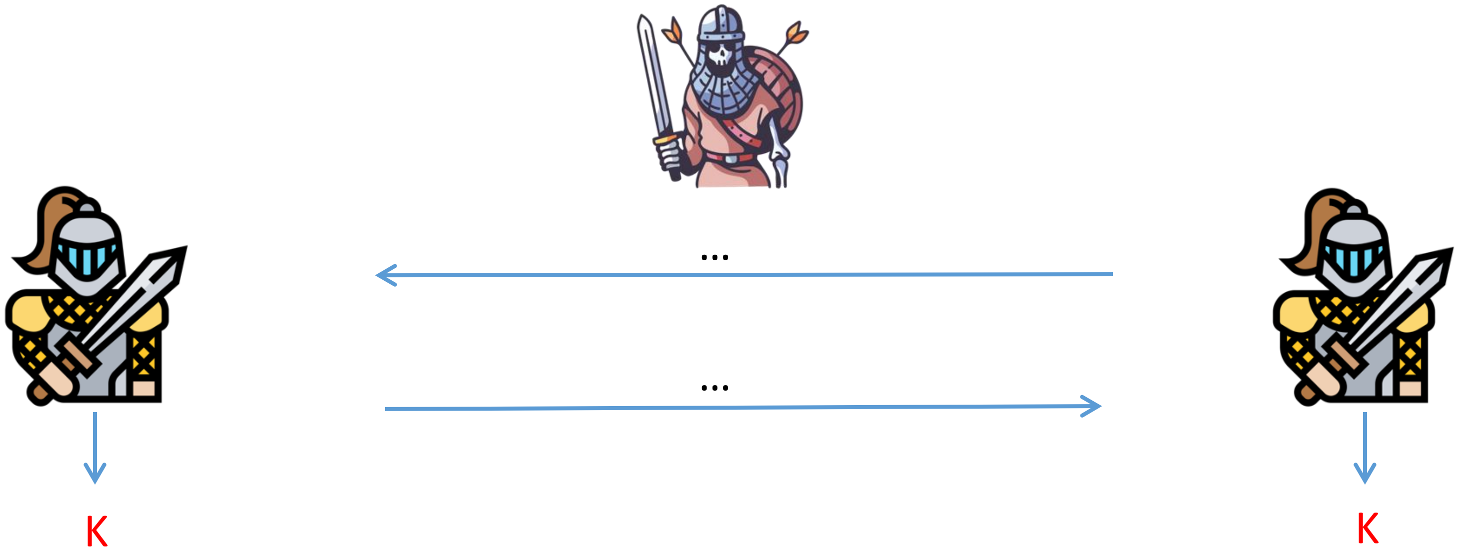
# How to transform a message securely?

- We may not hope to use a full PKE scheme, because
  - There is no PKE yet.
    - A solution to share the same secret key via an insecure channel was due to DH in 1976.
    - The first PKE scheme was proposed by RSA in 1978.
  - Other solutions are more efficient than PKE.
- A protocol that establish a shared secret key via an insecure channel is denoted as a key exchange protocol.



# Key Exchange Protocol

- A key exchange protocol involves two parties that communicate via a public channel.
- Both parties do not take any input.
- The goal of the protocol is to establish a shared secret that is hidden to anyone observes the public channel.



# Key Exchange Protocol

- A key exchange protocol involves two parties that communicate via a public channel.
- Both parties do not take any input.
- The goal of the protocol is to establish a shared secret that is hidden to anyone observes the public channel.

- A key exchange protocol should satisfy
  - Correctness: if both parties run honestly, then they will get the same secret key.
  - Security: an adversary that observes the communication between honest parties cannot learn any information about the shared secret key.
    - This is usually defined by requiring that the adversary cannot distinguish the shared secret key from a random string.
    - The security definition only works for a passive adversary and it is not secure against an active attacker, e.g., the man-in-the-middle attack.
    - We need an authenticated key exchange protocol to defend against active attacks.

# 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)

# Diffie-Hellman Key Exchange

- We assume that all parties agree on a common group G of order q and a common generator g of G.

Sample $X_A$ in [0,q-1]

$Y_A = g^{X_A}$ →

Sample $X_B$ in [0,q-1]

← $Y_B = g^{X_B}$

$K = (Y_B)^{X_A} = g^{X_A X_B}$

$K = (Y_A)^{X_B} = g^{X_A X_B}$

# Diffie-Hellman Key Exchange

- We assume that all parties agree on a common group G of order q and a common generator g of G.
- Step One:
  - A chooses a random number $X_A$ in [0,q-1] and computes $Y_A=g^{X_A}$.
  - B chooses a random number $X_B$ in [0,q-1] and computes $Y_B=g^{X_B}$.
  - Then A sends $Y_A$ to B and B sends $Y_B$ to A.
- Step two:
  - A computes $K_A=(Y_B)^{X_A}$.
  - B computes $K_B=(Y_A)^{X_B}$.
- Both $K_A$ and $K_B$ are equal to $g^{X_A X_B}$.

# Security of Diffie-Hellman protocol

- If the DDH problem is hard in the group G.
  - The protocol can prevent the adversary from distinguishing the shared secret key from a random string, i.e., the adversary cannot learn any information about the shared secret key.

# The parameters of schemes based on DLP

- 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

# Man-in-the-Middle Attack for Diffie-Hellman Key Exchange Protocol

- In the protoocl:
  - Step One:
    - A chooses a random number $X_A$ in [0,q-1] and computes $Y_A=g^{XA}$.
    - B chooses a random number $X_B$ in [0,q-1] and computes $Y_B=g^{XB}$.
    - ~~Then A sends $Y_A$ to B and B sends $Y_B$ to A.~~
    - The adversary cuts the communication between A and B, and
      - It samples $X_E$ in [0,q-1] and computes $Y_E=g^{XE}$
      - Then it sends $Y_E$ to both A and B.
  - Step two:
    - A computes $K'_A=(Y_E)^{XA}= g^{XAXE}$
    - B computes $K'_B=(Y_E)^{XB} = g^{XBXE}$
    - The adversary computes $K'_A=(Y_A)^{XE} = g^{XAXE}$ and $K'_B=(Y_B)^{XE} = g^{XBXE}$.
    - That is, both A and B are sharing secret key with the attacker.
- The problem can be solved if we add authentications in the protocol.

# Motivation

- When discussing symmetric-key cryptosystems, we always assume that secret keys are shared securely between the sender and the receiver. But how?
  - We can use a secure channel.
  - We can use a (authenticated) key exchange protocol if no secure channel is available?
- When discussing public-key cryptosystems, we always assume that the correct public keys are distributed. But how?
  - We can use a secure channel.
  - But what if there is no secure channel?

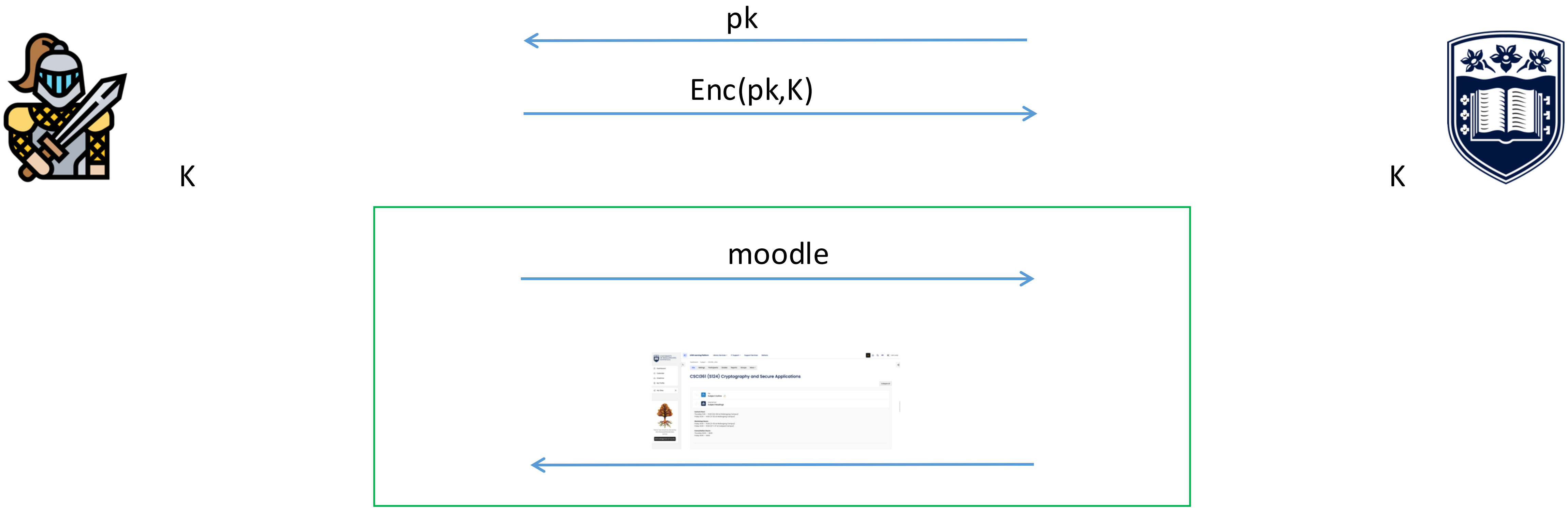
# Key Management and PKI

# What happens when you visit a website (securely)

This is not secure!!!! (because the public key could be replaced by the adversary.)

Can we guarantee integrity by using a signature?

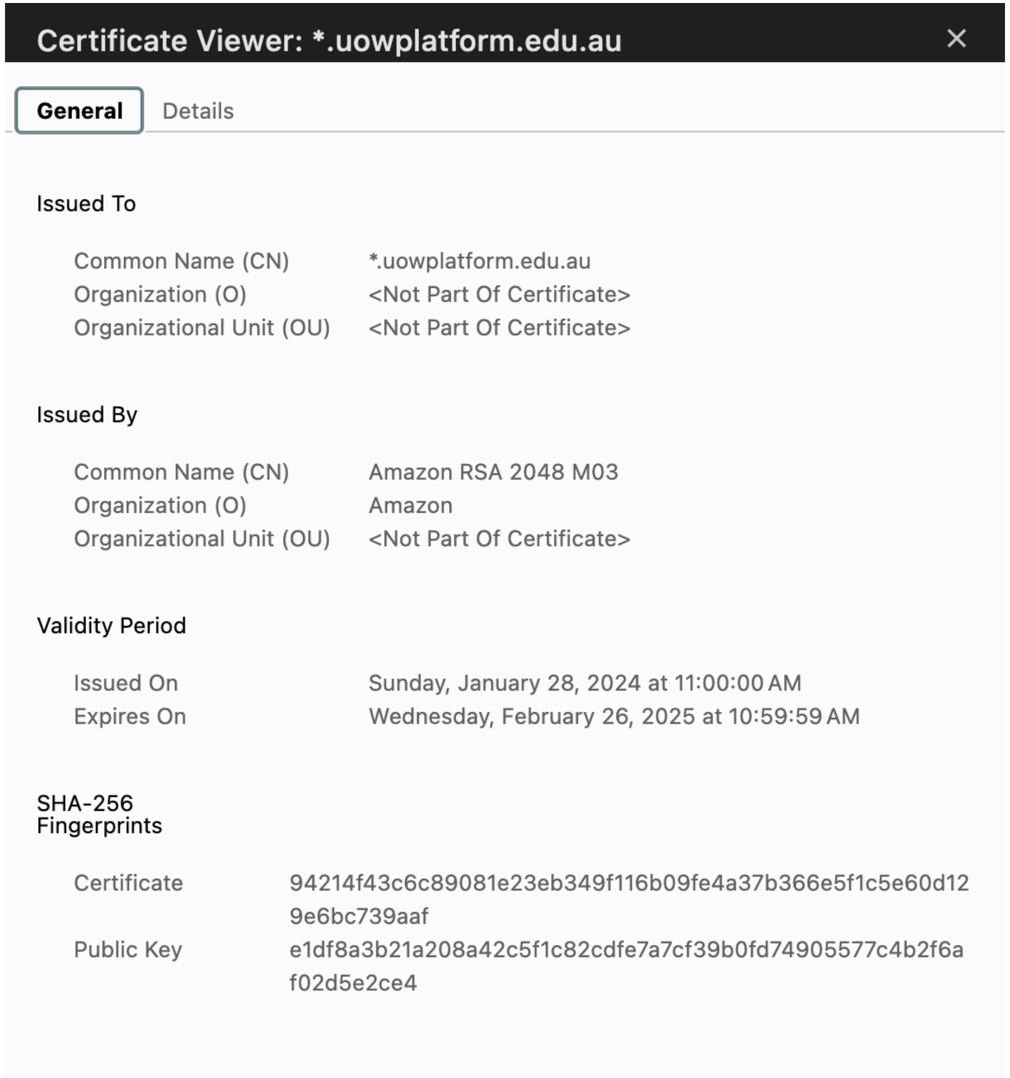
Again, how to get the public key of siganture!



# *Certificate (Idea)*

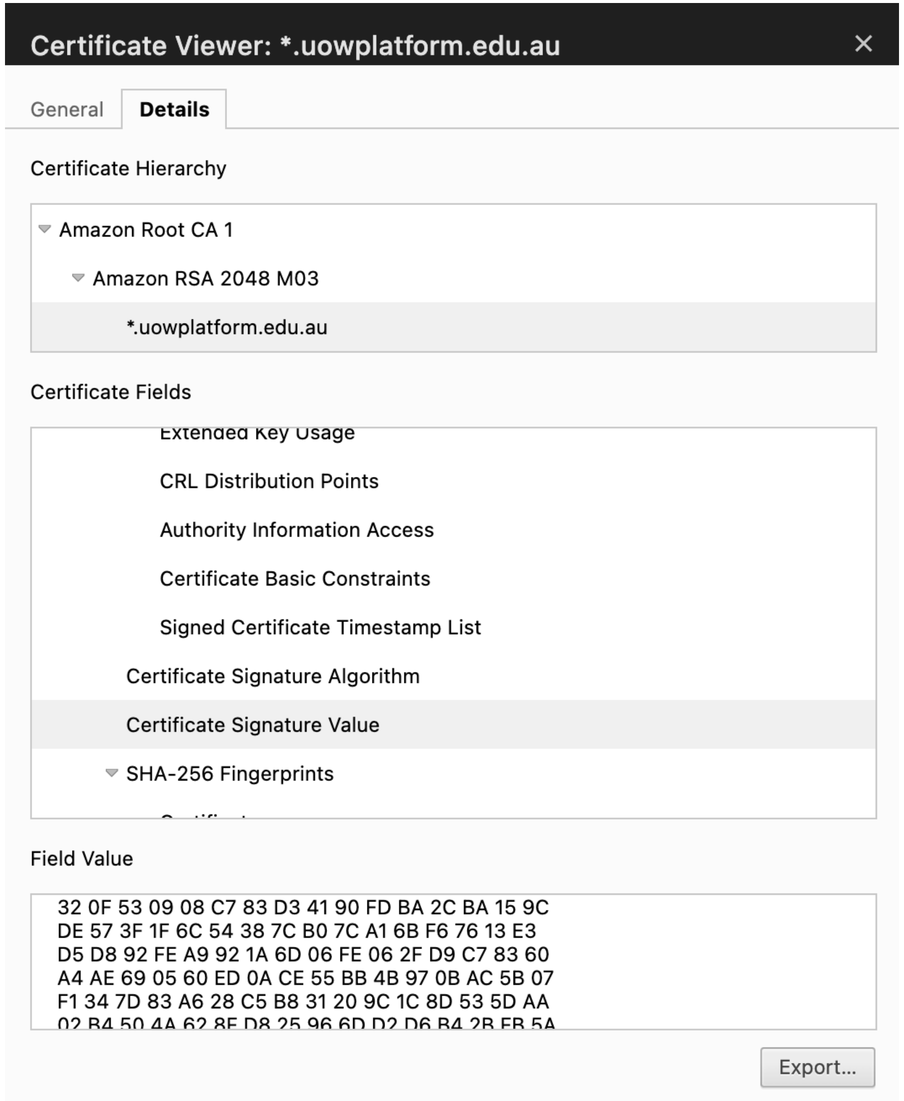
- We can solve the problem by asking a trusted third party to sign the public key of Alice.
- It is impossible to directly assume that an entity knows the public key of another entity, but it is reasonable to assume that we ALL know the public key of a trusted third party, such as Google.
- The trusted party is called a certificate authority (CA).

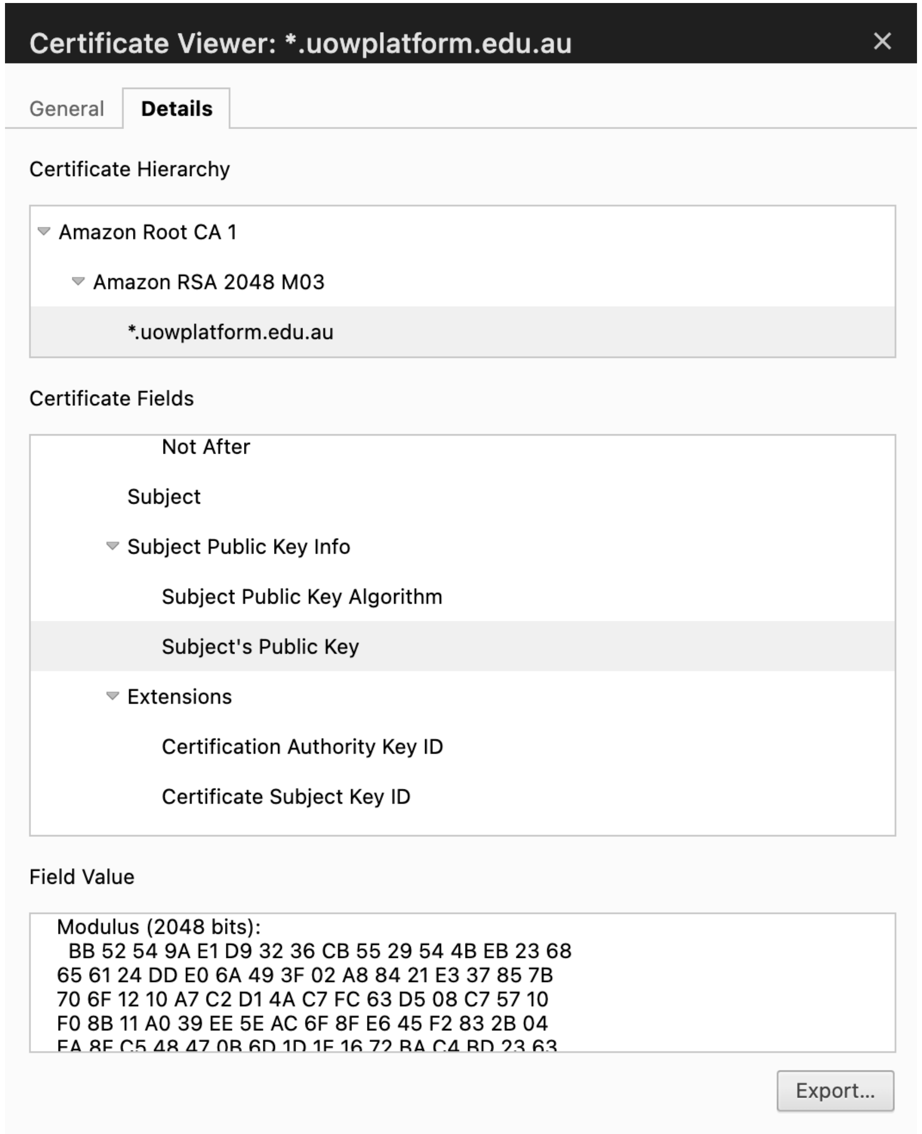
# Certificate (Idea)




- The public key of Amazon is pk*.   (Certificate Authority)
  (We trust this public key. Suppose that everyone trusts this)
- When we browse “www.uow.edu.au”, the web server sends its public key pk to us. But we don’t know pk belongs to UOW or not.
- The CA uses sk* to genera a digital signature S_UOW on
  M=”pk belongs to UOW”
- With pk and S_UOW, we know that pk bleongs to UOW

# Certificate (Idea)



# Get A Certificate

- The system can be sketched as follows:
  - Alice securely sends $PK_A$ to the certificate authority.
  - Alice receives a certificate $C_A$ that binds $PK_A$ to Alice. The main component in $C_A$ is a signature on $PK_A$ and some necessary auxiliary information, which is signed by the certificate authority.
  - This certificate can be verifiable by everyone who has the public key of the certificate authority.
  - A certificate has the following form:

    $M = [PK_A$, Alice's ID, validity period, …].

    $S_A = Sign_{SK_T}(M)$

    $C_A=(M,S_A,\ldots)$

# Use A Certificate

- When Bob wants to send an encrypted message to Alice:
  - He obtains Alice's certificate.
  - Verifies the signature in the certificate using the public key of the certificate authority.
  - Verifies the identity of the owner of the certificate
  - Verifies the certificate has not been expired, etc.
  - Extracts $PK_A$ and uses it to encrypt the message.

- Question: how to ensure Alice has the correct public key of the certificate authority?

# Trusted Root Certificates

- Question: how to ensure Bob has the correct public key of the the certificate authority?
  - If it is a root certificates authority, its public key will be hardwired in the code of web browser/operation system.
  - If it is not a root certificates, then its public key also includes a certificate authenticating its public key. The certificate is issued by another certificates authority.

# Trusted Root Certificates in MACOS

**AAA Certificate Services**
Root certificate authority
Expires: Monday, January 1, 2029 at 10:59:59 Australian Eastern Daylight Time
This certificate is valid

| Name | Kind | Date Modified | Expires | Keychain |
|---|---|---|---|---|
| AAA Certificate Services | certificate | -- | Jan 1, 2029 at 10:59:59 | System Roots |
| AC RAIZ FNMT-RCM | certificate | -- | Jan 1, 2030 at 11:00:00 | System Roots |
| ACCVRAIZ1 | certificate | -- | Dec 31, 2030 at 20:37:37 | System Roots |
| Actalis Authentication Root CA | certificate | -- | Sep 22, 2030 at 21:22:02 | System Roots |
| AffirmTrust Commercial | certificate | -- | Jan 1, 2031 at 01:06:06 | System Roots |
| AffirmTrust Networking | certificate | -- | Jan 1, 2031 at 01:08:24 | System Roots |
| AffirmTrust Premium | certificate | -- | Jan 1, 2041 at 01:10:36 | System Roots |
| AffirmTrust Premium ECC | certificate | -- | Jan 1, 2041 at 01:20:24 | System Roots |
| Amazon Root CA 1 | certificate | -- | Jan 17, 2038 at 11:00:00 | System Roots |
| Amazon Root CA 2 | certificate | -- | May 26, 2040 at 10:00:00 | System Roots |
| Amazon Root CA 3 | certificate | -- | May 26, 2040 at 10:00:00 | System Roots |
| Amazon Root CA 4 | certificate | -- | May 26, 2040 at 10:00:00 | System Roots |
| ANF Global Root CA | certificate | -- | Jun 6, 2033 at 03:45:38 | System Roots |
| Apple Root CA | certificate | -- | Feb 10, 2035 at 08:40:36 | System Roots |
| Apple Root CA - G2 | certificate | -- | May 1, 2039 at 04:10:09 | System Roots |
| Apple Root CA - G3 | certificate | -- | May 1, 2039 at 04:19:06 | System Roots |
| Apple Root Certificate Authority | certificate | -- | Feb 10, 2025 at 11:18:14 | System Roots |
| Atos TrustedRoot 2011 | certificate | -- | Jan 1, 2031 at 10:59:59 | System Roots |
| Autoridad de Certificacion Firmaprofesional CIF A62634068 | certificate | -- | Dec 31, 2030 at 19:38:15 | System Roots |
| Baltimore CyberTrust Root | certificate | -- | May 13, 2025 at 09:59:00 | System Roots |
| Buypass Class 2 Root CA | certificate | -- | Oct 26, 2040 at 19:38:03 | System Roots |
| Buypass Class 3 Root CA | certificate | -- | Oct 26, 2040 at 19:28:58 | System Roots |
| CA Disig Root R2 | certificate | -- | Jul 19, 2042 at 19:15:30 | System Roots |
| Certainly Root E1 | certificate | -- | Apr 1, 2046 at 10:00:00 | System Roots |
| Certainly Root R1 | certificate | -- | Apr 1, 2046 at 10:00:00 | System Roots |
| Certigna | certificate | -- | Jun 30, 2027 at 01:13:05 | System Roots |
| certSIGN ROOT CA | certificate | -- | Jul 5, 2031 at 03:20:04 | System Roots |
| certSIGN ROOT CA G2 | certificate | -- | Feb 6, 2042 at 20:27:35 | System Roots |
| Certum CA | certificate | -- | Jun 11, 2027 at 20:46:39 | System Roots |
| Certum EC-384 CA | certificate | -- | Mar 26, 2043 at 18:24:54 | System Roots |
| Certum Trusted Network CA | certificate | -- | Dec 31, 2029 at 23:07:37 | System Roots |
| Certum Trusted Network CA 2 | certificate | -- | Oct 6, 2046 at 18:39:56 | System Roots |
| Certum Trusted Root CA | certificate | -- | Mar 16, 2043 at 23:10:13 | System Roots |
| CFCA EV ROOT | certificate | -- | Dec 31, 2029 at 14:07:01 | System Roots |
| Chambers of Commerce Root | certificate | -- | Oct 1, 2037 at 02:13:44 | System Roots |
| Chambers of Commerce Root - 2008 | certificate | -- | Jul 31, 2038 at 22:29:50 | System Roots |
| Cisco Root CA 2048 | certificate | -- | May 15, 2029 at 06:25:42 | System Roots |

**Amazon Root CA 4**
Root certificate authority
Expires: Saturday, May 26, 2040 at 10:00:00 Australian Eastern Standard Time
This certificate is valid

> Trust

**Details**

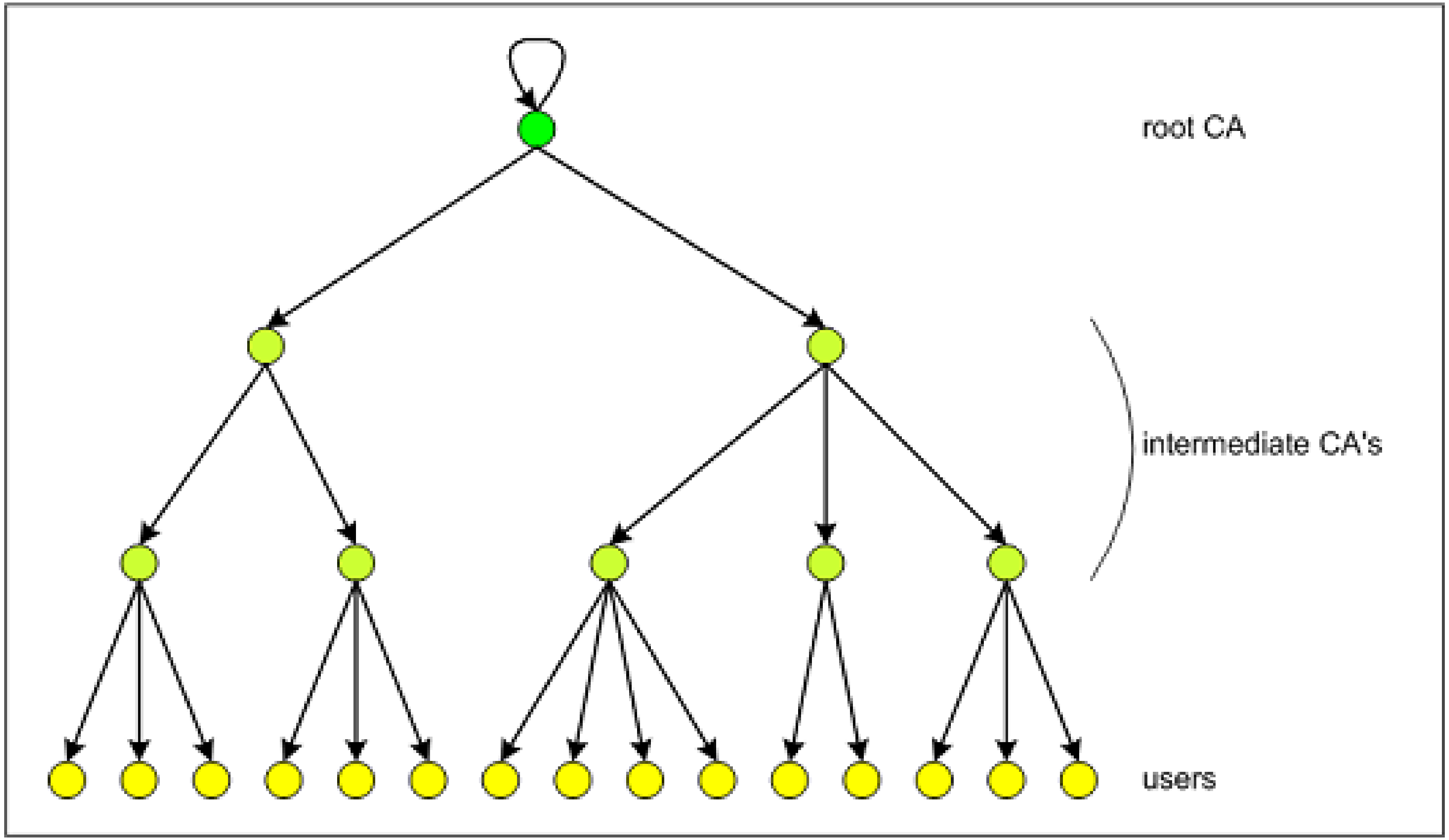
**Subject Name**
**Country or Region** US
**Organization** Amazon
**Common Name** Amazon Root CA 4

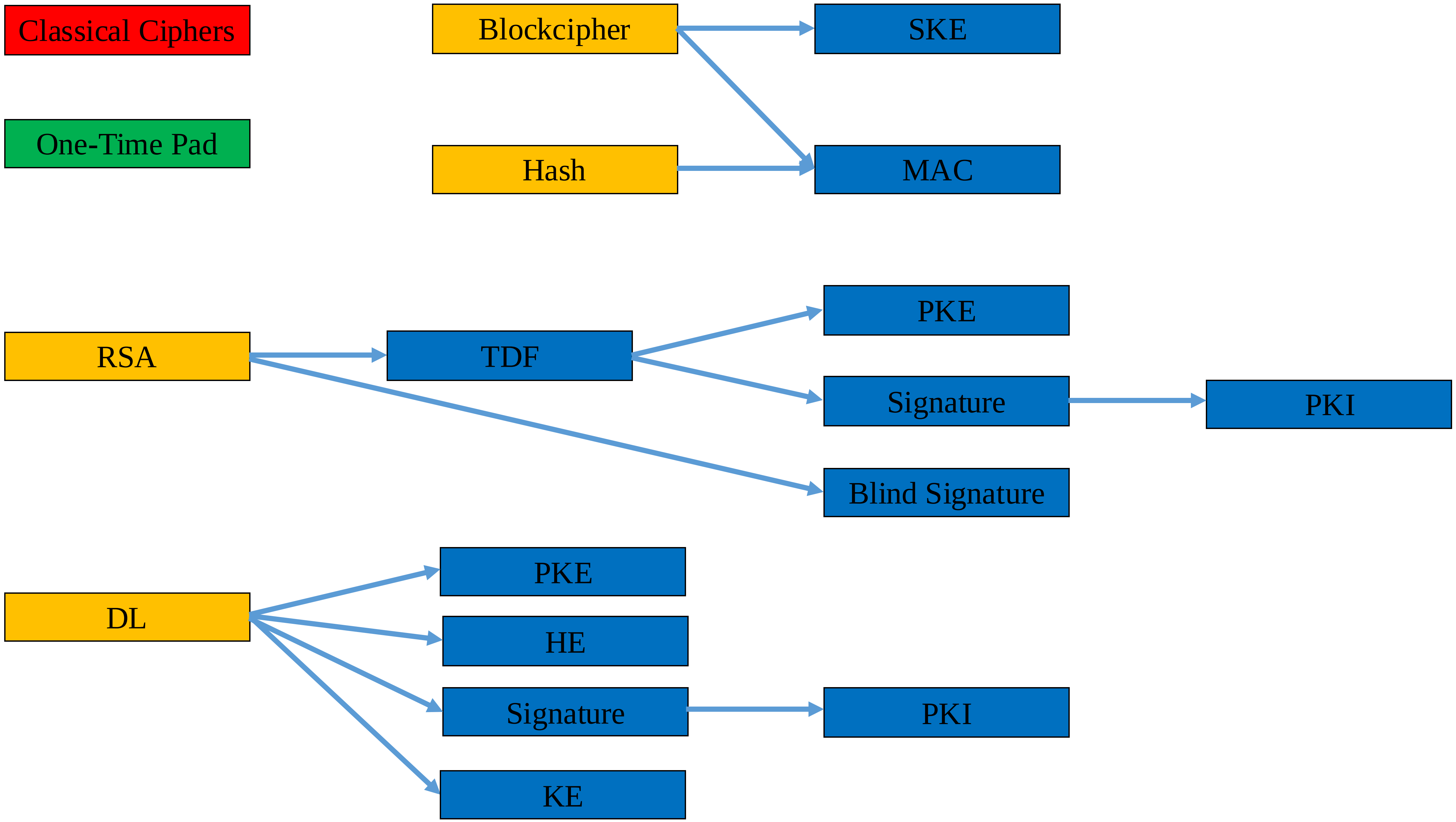
**Issuer Name**
**Country or Region** US
**Organization** Amazon
**Common Name** Amazon Root CA 4

**Serial Number** 06 6C 9F D7 C1 BB 10 4C 29 43 E5 71 7B 7B 2C C8 1A C1 0E
**Version** 3
**Signature Algorithm** ECDSA Signature with SHA-384 ( 1.2.840.10045.4.3.3 )
**Parameters** None

**Not Valid Before** Tuesday, May 26, 2015 at 10:00:00 Australian Eastern Standard Time
**Not Valid After** Saturday, May 26, 2040 at 10:00:00 Australian Eastern Standard Time

**Public Key Info**
**Algorithm** Elliptic Curve Public Key ( 1.2.840.10045.2.1 )
**Parameters** Elliptic Curve secp384r1 ( 1.3.132.0.34 )
**Public Key** 97 bytes : 04 D2 AB 8A 37 4F A3 53 ...
**Key Size** 384 bits


root CA
intermediate CA's
users


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

# Summary

- Key Exchange
  - Motivation and Application Scenario
  - Definition
  - Construction
  - Man-in-the-Middle Attack
- PKI
  - Motivation
  - How to certificate a public key