# ECE568 笔记汇总

- Cryptography - Block Ciphers
- Cryptography - Ciphers
- Cryptography - Hashes, MACs, and Digital-Signitures
- Cryptography - Public-Key Cryptography
- Cryptography - Stream Ciphers

## Table of Contents

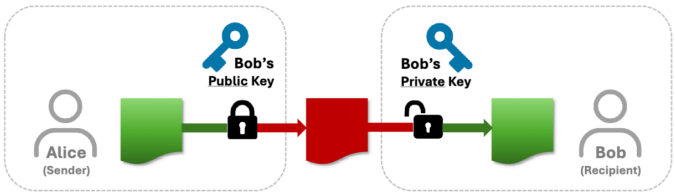


## Public Key Crypto Sys

- pair of keys
  - every user has **public/private key pair**
  - private & public key reveal nothing about each other
  - users distribute pub key and keep priv key in secure place
  - msg encrypted w one key can **ONLY** be decrypted w other key

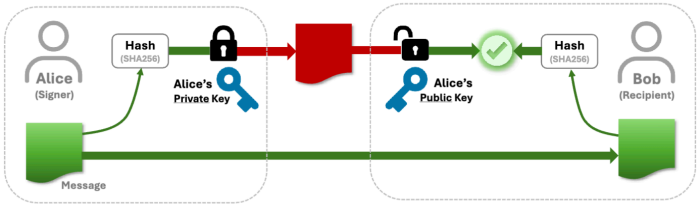
### Public Key Encryption

- sender encrypts msg with recipient's **public key**
  - pub key encryption computationally expensive




### Public Key Signing

- **authentication** and **non-repudiation** (receiver doesn't falsely claim msg came from sender)
  - To sign the msg, a hash of the message, msg is encrypted with sender's **private key**
    - recipient decrypt with sender's **public key**
    - **ONLY** sender could have encrypted the text




## RSA

# Public Key Infrastructure (PKI)

- Public key crypto prevents *Man In The Middle* attacks
  - Alice wants to share key with Bob, encrypt key with Bob's public key prevents eavesdropping
  - But eavesdropper M can lie to A that M's pub key is B's pub key
- **Public-Key Infrastructure** is a system where a **trusted third party** vouches for the identity of a key (key belongs to a principal)
  - B creates pub key and goes to T
  - T sees both Bob and his pub key, creates a certificate saying pub key belongs to B (with other authentication methods key is from Bob) and **signs** it with T's priv key
  - B sends A his own pub key along with the certificate from T
  - A uses T's pub key and certificate to verify B's pub key
- M cannot prevent its pub key is B's pub key since it won't have T's certificate

## X509

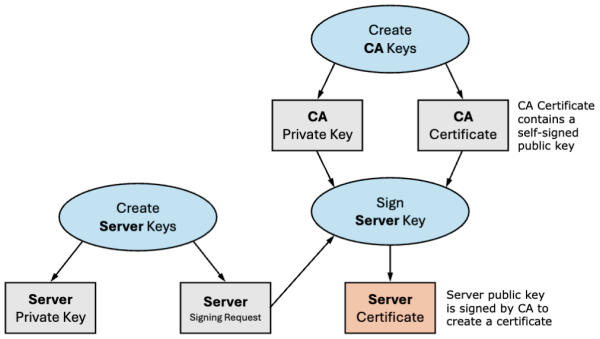
- used in SSL and most digital signatures
- PKI allows using a chain of certificates issued by a hierarchy of CAs
- Not **trusted central server**
  - trust level
  - availability, integrity

### Certificate Authorities

- **CA** (AWS, Azure...)
- when a browser connects to a secure website, the website sends the browser a certificate that can be verified if browser/OS trust the CA that issued the certificate (chain)

## Structure of X.509 Certificate

- X.509 certificate is a standardized identification form with numerous fields
  - **subject** info about bearer (Common Name CN - name of host being authenticated)
  - subject's **public key**
  - expiry (**not-after**) and validity (**not-before**) dates
  - **issuer**: info about CA
  - **certificate signature** digital signature of the first part of the certificate, signed by issuer's priv key




**Create ourselves a CA (Certificate Authority) certificate**

```
$ openssl req -new -x509 -extensions v3_ca -keyout cakey.pem
  -out cacert.pem -days 3650
```

**Generate a new RSA private key for our web server**

```
$ openssl genrsa -des3 -out server.key 1024
```

**Generate a CSR (Certificate Signing Request) for our server's key**

```
$ openssl req -new -key server.key -out server.csr
```

**Sign the CSR with our CA key**

```
$ openssl x509 -req -days 365 -in server.csr -CA cacert.pem
  -CAkey cakey.pem -CAserial serial.txt -Cacreateserial
  -out server.crt
```

- can **revoke** certificates