IERG 4130 – 2026 Fall

香港中文大學
CUHK

Public-Key Encryption and Digital Signatures

Yajin ZHOU

https://yajin.org

Credits: the slides are from CS161 of UC Berkeley

Summary

- **True randomness requires sampling a physical process**
  - Slow and expensive (low entropy)
- PRNG: An algorithm that uses a little bit of true randomness to generate a lot of random-looking output
  - Security: Computationally indistinguishable from truly random bits

Summary: Diffie-Hellman Key Exchange



- **Algorithm:**
  - Alice chooses a and sends $g^a \bmod p$ to Bob
  - Bob chooses b and sends $g^b \bmod p$ to Alice
  - Their shared secret is $(g^a)^b = (g^b)^a = g^{ab} \bmod p$
- **Issues**
  - *Not* secure against MITM
  - Both parties must be online
  - Does not provide authenticity

Public-Key Cryptography



- **In public-key schemes, each person has two keys**
  - **Public key**: Known to everybody
  - **Private key**: Only known by that person
  - Keys come in pairs: every public key corresponds to one private key
- **Uses number theory**
  - Examples: Modular arithmetic, factoring, discrete logarithm problem
- **Benefit: No longer need to assume that Alice and Bob already share a secret**
- **Drawback: Slower than symmetric-key cryptography**

Public-Key Encryption

- Everybody can encrypt with the public key
- Only the recipient can decrypt with the private key

Public-Key Encryption: Definition



- **Three parts:**
  - KeyGen() → *PK*, *SK*: Generate a public/private keypair, where *PK* is the public key, and *SK* is the private (secret) key
  - Enc(*PK*, *M*) → *C*: Encrypt a plaintext *M* using public key *PK* to produce ciphertext *C*
  - Dec(*SK*, *C*) → *M*: Decrypt a ciphertext *C* using secret key *SK*
- **Properties**
  - **Correctness**: Decrypting a ciphertext should result in the message that was originally encrypted
    - Dec(*SK*, Enc(*PK*, *M*)) = *M* for all *PK*, *SK* ← KeyGen() and *M*
  - **Efficiency**: Encryption/decryption should be fast
  - **Security**: IND-CPA, same as for symmetric-key encryption

Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	• One-time pads • Block ciphers with chaining modes (e.g. AES-CBC) • Stream ciphers	• RSA encryption • ElGamal encryption
Integrity, Authentication	• MACs (e.g. HMAC)	• Digital signatures (e.g. RSA signatures)

- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)

- Key management (certificates)
- Password management

ElGamal Encryption

- Diffie-Hellman key exchange is great: It lets Alice and Bob share a secret over an insecure channel
- Problem: Diffie-Hellman by itself isn't enough to send an encrypted message. The secret $g^{ab} \bmod p$ is random.
- Idea: Let’s modify Diffie-Hellman so it supports encrypting and decrypting messages directly

ElGamal Encryption: Protocol

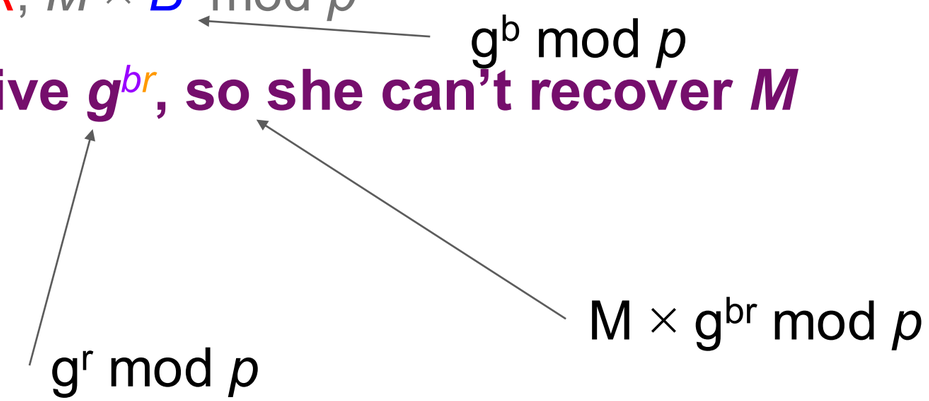


- **KeyGen():**
  - Bob generates private key b and public key $B = g^b \bmod p$
    - Intuition: Bob is completing his half of the Diffie-Hellman exchange
- **Enc(B, M):**
  - Alice generates a random r and computes $R = g^r \bmod p$
    - Intuition: Alice is completing her half of the Diffie-Hellman exchange
  - Alice computes $M \times B^r \bmod p$
    - $M \times (g^b)^r \bmod p$
    - Intuition: Alice derives the shared secret and multiples her message by the secret
  - Alice sends $C_1 = R$, $C_2 = M \times B^r \bmod p$
- **Dec(b, C_1, C_2)**
  - Bob computes $C_2 \times C_1^{-b} = M \times B^r \times R^{-b} = M \times g^{br} \times g^{-br} = M \bmod p$
    - Intuition: Bob derives the (inverse) shared secret and multiples the ciphertext by the inverse shared secret

ElGamal Encryption: Security



- Recall Diffie-Hellman problem: Given $g^a \bmod p$ and $g^b \bmod p$, hard to recover $g^{ab} \bmod p$
- **ElGamal sends these values over the insecure channel**
  - Bob's public key: B
  - Ciphertext: R, $M \times B^r \bmod p$
- **Eve can't derive g^{br}, so she can't recover M**



ElGamal Encryption: Issues

- **Malleability: The adversary can tamper with the message**
  - The adversary can manipulate $C_1' = C_1$, $C_2' = 2 \times C_2 = 2 \times M \times g^{br}$ to make it look like $2 \times M$ was encrypted
- **Caution: Needs additional padding and other modifications to make it secure**

Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	• One-time pads • Block ciphers with chaining modes (e.g. AES-CBC) • Stream ciphers	• RSA encryption • ElGamal encryption
Integrity, Authentication	• MACs (e.g. HMAC)	• Digital signatures (e.g. RSA signatures)

- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)

- Key management (certificates)
- Password management

RSA Encryption: Definition



- **KeyGen():**
  - Randomly pick two large primes, p and q
    - Efficient algorithm: pick random numbers and then use a test to see if the number is prime
  - Compute $N = pq$
    - N is usually between 2048 bits and 4096 bits long
  - Choose e
    - Requirement: e is relatively prime to $(p - 1)(q - 1)$
    - Requirement: $2 < e < (p - 1)(q - 1)$
  - Compute $d = e^{-1} \bmod (p - 1)(q - 1)$
    - Efficient algorithm for computing multiplicative inverses: Extended Euclid's algorithm (but specifics are out of scope)
  - **Public key**: N and e
  - **Private key:** d

RSA Encryption: Definition

- Enc(e, N, M) = $M^e \bmod N$
- Dec(d, C) = $C^d \bmod N$
- For correctness, we need:

  Dec(d, Enc(e, N, M)) = $(M^e)^d = M \bmod N$

Correctness of RSA Encryption



- Theorems: If $x = y \bmod p$, and $x = y \bmod q$, then $x = y \bmod pq$ (Chinese remainder theorem); $a^{p-1} = 1 \bmod p$ (Fermat's little theorem).
- From key generation, we know $d = e^{-1} \bmod (p - 1)(q - 1)$

  Therefore, $ed = 1 \bmod (p - 1)(q - 1)$

  Therefore, $ed - 1$ is a multiple of $(p - 1)(q - 1)$
- $M^{ed} = M^{ed-1} M \pmod p$

  $= M^{\text{some multiple of } p-1} M \pmod p$ (using definition of $ed - 1$)

  $= M^{(p-1)c} M \pmod p$

  $= M \pmod p$ (using FLT)
- **Similarly $M^{ed} = M \pmod q$**
- **Therefore $M^{ed} = M \pmod N$**
- **Therefore Dec(d, Enc(e, M)) = M**

RSA Encryption: Correctness



1. **Theorem: $M^{ed} \equiv M \bmod N$**
2. **Euler's theorem: $a^{\varphi(N)} \equiv 1 \bmod N$**
   - $\varphi(N)$ is the totient function of N
   - If N is prime, $\varphi(N) = N - 1$ (Fermat's little theorem)
   - If p and q are prime (and different), $\varphi(pq) = (p - 1)(q - 1)$
   - See CS 70 for a more detailed explanation
3. **Notice: $ed \equiv 1 \bmod (p - 1)(q - 1)$ so $ed \equiv 1 \bmod \varphi(N)$**
   - This means that $ed = k\varphi(n) + 1$ for some integer k
4. **(1) can be written as $M^{k\varphi(N) + 1} \equiv M \bmod N$**
5. **$M^{k\varphi(N)}M^1 \equiv M \bmod N$**
6. **$1M^1 \equiv M \bmod N$ by Euler's theorem**
7. **$M \equiv M \bmod N$**

RSA Encryption: Security

- RSA problem: Given N and $C = M^e \bmod N$, it is hard to find M
- Current best solution is to factor N (unknown whether there is an easier way)
- Best algorithms known for factoring or the RSA problem take exponential time

RSA Encryption: Issues



- **Caveat: using RSA encryption in the form we have shown is not secure**
  - It's deterministic, so not IND-CPA secure
  - There are other subtle issues
- **Solution: Use an appropriate (probabilistic) padding scheme**

OAEP

- **Optimal asymmetric encryption padding** (**OAEP**): A method of adding padding that introduces randomness and makes RSA secure
  - Different from “padding” used for symmetric encryption, used to add randomness instead of dummy bytes
- **RSA-OAEP is IND-CPA secure**

Hybrid Encryption



- **Issues with public-key encryption**
  - We can only encrypt small/short messages, because of the modulo operator
  - It is slow
  - Result: Public-key encryption is bad for large messages
- **Hybrid encryption**: Randomly generate a key *K*, encrypt message under key *K* using symmetric encryption, and encrypt *K* using public-key encryption
  - Now we can encrypt large amounts of data quickly using symmetric encryption, and we still have the benefits of public-key encryption
- **Almost all cryptographic systems use hybrid encryption**

Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	• One-time pads • Block ciphers with chaining modes (e.g. AES-CBC) • Stream ciphers	• RSA encryption • ElGamal encryption
Integrity, Authentication	• MACs (e.g. HMAC)	• Digital signatures (e.g. RSA signatures)

- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)

- Key management (certificates)
- Password management

Digital Signatures

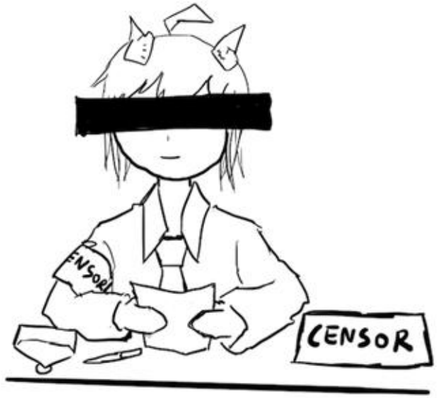


- **Public-key cryptography is good because we don’t need to share a secret key**
- Digital signatures are the public-key way of providing integrity/authenticity to data
- We need to assume that Alice and Bob can communicate public keys without Mallory interfering
  - We will see how to fix this limitation later

Public-key Signatures

- Only the owner of the private key can sign messages with the private key
- Everybody can verify the signature with the public key







Digital Signatures: Definition



- **Three parts:**
  - KeyGen() → *PK*, *SK*: Generate a public/private keypair, where *PK* is the verify (public) key, and *SK* is the signing (secret) key
  - Sign(*SK*, *M*) → *sig*: Sign the message *M* using the signing key *SK* to produce the signature *sig*
  - Verify(*PK*, *M*, *sig*) → {0, 1}: Verify the signature *sig* on message *M* using the verify key *PK* and output 1 if valid and 0 if invalid
- **Properties**
  - **Correctness**: Verification should be successful for a signature generated over any message
    - Verify(*PK*, *M*, Sign(*SK*, *M*)) = 1 for all *PK*, *SK* ← KeyGen() and *M*
  - **Efficiency**: Signing/verifying should be fast
  - **Security**: EU-CPA, same as for MACs

Digital Signatures in Practice



- **If you want to sign message *M*:**
  - First hash *M*
  - Then sign H(*M*)
- **The hash allows signing arbitrarily long messages**
- **Digital signatures provide integrity *and authenticity* for *M***
  - The digital signature acts as proof that the private key holder signed H(*M*), so you know that *M* is authentically endorsed by the private key holder

RSA Signatures



- **Recall RSA encryption: $M^{ed} \equiv M \bmod N$**
  - There is nothing special about using *e* first or using *d* first!
  - If we “encrypt” using *d*, then anyone can “decrypt” using *e*
    - Given *x* and $x^d \bmod N$, can’t recover *d* because of discrete-log problem, so *d* is safe

RSA Signatures: Definition



- **KeyGen():**
  - Same as RSA encryption:
    - **Public key**: *N* and *e*
    - **Private key:** *d*
- **Sign(*d*, *M*) = $H(M)^d$ mod *N***
- **Verify(*e*, *N*, *M*, *sig*): Verify that $H(M) \equiv sig^e$ mod *N***

Summary: Public-Key Cryptography



- **Public-key cryptography: Two keys; one undoes the other**
- **Public-key encryption: One key encrypts, the other decrypts**
  - Security properties similar to symmetric encryption
  - ElGamal: Based on Diffie-Hellman
    - The public key is g^b, and C_1 is g^r.
  - RSA: Produce a pair e and d such that $M^{ed} = M \bmod N$
    - Needs padding etc. to be IND-CPA secure
- Hybrid encryption: Encrypt a symmetric key, and use the symmetric key to encrypt the message
- **Digital signatures: Integrity and authenticity for asymmetric schemes**
  - RSA: Same as RSA encryption, but “encrypt” the hash with the *private* key