Diffie-Hellman Key Exchange and Public-Key Encryption

CS 161 Spring 2024 - Lecture 9

A physical demo

Public-Key Cryptography

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
  - Contrast with symmetric-key cryptography (uses XORs and bit-shifts)
- Messages are numbers
  - Contrast with symmetric-key cryptography (messages are bit strings)
- Benefit: No longer need to assume that Alice and Bob already share a secret
- Drawback: Much slower than symmetric-key cryptography
  - Number theory calculations are much slower than XORs and bit-shifts

Diffie-Hellman Key Exchange

Textbook Chapter 10

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	• One-time pads • Block ciphers with chaining modes (e.g. AES-CBC)	• 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

Discrete Log Problem and Diffie-Hellman Problem

- Assume everyone knows a large prime p (e.g. 2048 bits long) and a generator g
  - Don't worry about what a generator is
- **Discrete logarithm problem** (**discrete log problem**): Choose a long random number a *(e.g., 2048 bits),* given g, p, g^a mod p: it is computationally hard to find a
- **Diffie-Hellman assumption**: Choose long random numbers a and b *(e.g., 2048 bits),* given g, p, g^a mod p, and g^b mod p, no polynomial time attacker can distinguish between a random value R and g^{ab} mod p.
  - Intuition: The best known algorithm is to first calculate a and then compute $(g^b)^a$ mod p, but this requires solving the discrete log problem, which is hard!
  - Note: Multiplying the values doesn't work, since you get g^{a+b} mod $p \neq g^{ab}$ mod p

Discrete Log Problem and Diffie-Hellman Problem

For a random large a, b, R:

g, p, $g^a \bmod p$, $g^b \bmod p$, $g^{ab} \bmod p$

~ ← Indistinguishable from the perspective of a polynomial time attacker

g, p, $g^a \bmod p$, $g^b \bmod p$, R

Diffie-Hellman Key Exchange

Public: g, p

Alice	Eve	Bob
 Alice	Eve 	Bob 
Generate a		Generate b
Calculate $g^a \bmod p$	g^a → ← g^b	Calculate $g^b \bmod p$
Receive $g^b \bmod p$		Receive $g^a \bmod p$
Calculate $(g^b)^a \bmod p$		Calculate $(g^a)^b \bmod p$
$a, g^a, g^b \Rightarrow g^{ab}$	$g^a, g^b \not\Rightarrow g^{ab}$	$b, g^a, g^b \Rightarrow g^{ab}$

Private key → Generate a

Public key → Calculate $g^a \bmod p$

Shared symmetric key is K=g^{ab}

Shared symmetric key is K=g^{ab}

Ephemerality of Diffie-Hellman

- Diffie-Hellman can be used ephemerally (called Diffie-Hellman ephemeral, or DHE)
  - **Ephemeral**: Short-term and temporary, not permanent
  - Alice and Bob discard a, b, and $K = g^{ab} \bmod p$ when they're done
  - Because you need a and b to derive K, you can never derive K again!
  - Sometimes K is called a **session key**, because it's only used for a an ephemeral session
- Benefit of DHE: **Forward secrecy**
  - Eve records everything sent over the insecure channel
  - Alice and Bob use DHE to agree on a key $K = g^{ab} \bmod p$
  - Alice and Bob use K as a symmetric key
  - After they're done, discard a, b, and K
  - Later, Eve steals all of Alice and Bob's secrets
  - Eve can't decrypt any messages she recorded: Nobody saved a, b, or K, and her recording only has $g^a \bmod p$ and $g^b \bmod p$!

Diffie-Hellman Key Exchange

Public: g, p

Alice

Bob

Private key

Generate a

Generate b

Calculate $g^a \bmod p$

Public key

Calculate $g^b \bmod p$

Q: What if Mallory can actively change messages (malicious attacker)?

g^a

g^b

Receive $g^b \bmod p$

Receive $g^a \bmod p$

Calculate $(g^b)^a \bmod p$

Calculate $(g^a)^b \bmod p$

$a, g^a, g^b \Rightarrow g^{ab}$

$g^a, g^b \not\Rightarrow g^{ab}$

$b, g^a, g^b \Rightarrow g^{ab}$

Shared symmetric key is K=g^{ab}

Shared symmetric key is K=g^{ab}

Diffie-Hellman: Man-in-the-middle attack

Public: g, p

Alice	Mallory		Bob
			
Generate a	Generate m		Generate b
Calculate $g^a \bmod p$	Calculate $g^m \bmod p$		Calculate $g^b \bmod p$
Receive $g^m \bmod p$	Receive $g^a \bmod p$	Receive $g^b \bmod p$	Receive $g^m \bmod p$
Calculate $(g^m)^a \bmod p$	Calculate $(g^a)^m \bmod p$	Calculate $(g^b)^m \bmod p$	Calculate $(g^m)^b \bmod p$
$a, g^a, g^m \Rightarrow g^{am}$	$m, g^m, g^a \Rightarrow g^{am}$	$m, g^m, g^b \Rightarrow g^{bm}$	$b, g^b, g^m \Rightarrow g^{bm}$

Diffie-Hellman: Issues

- Diffie-Hellman is not secure against a MITM adversary
- DHE is an *active protocol*: Alice and Bob need to be online at the same time to exchange keys
  - What if Bob wants to encrypt something and send it to Alice for her to read later?
- Diffie-Hellman does not provide *authentication*
  - You exchanged keys with someone, but Diffie-Hellman makes no guarantees about who you exchanged keys with; it could be Mallory!

Summary: Diffie-Hellman Key Exchange

- Algorithm:
  - Alice chooses a and sends g^a mod p to Bob
  - Bob chooses b and sends g^b mod p to Alice
  - Their shared secret is $(g^a)^b = (g^b)^a = g^{ab}$ mod p
- Diffie-Hellman provides forwards secrecy: Nothing is saved or can be recorded that can ever recover the key
- Diffie-Hellman can be performed over other mathematical groups, such as elliptic-curve Diffie-Hellman (ECDH)
- Issues
  - *Not* secure against MITM
  - Both parties must be online
  - Does not provide authenticity

Public-Key Encryption

Textbook Chapter 11

Public-Key Encryption

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

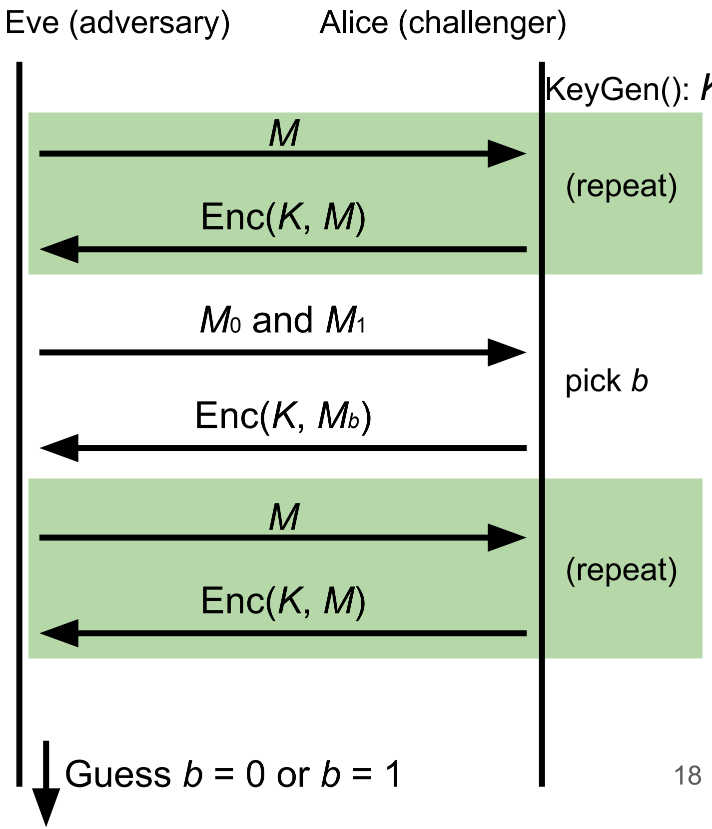
Public-Key Encryption: Definition

- 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**: Similar to IND-CPA, but Alice (the challenger) just gives Eve (the adversary) the public key, and Eve doesn't request encryptions, except for the pair M_0, M_1
    - You don't need to worry about this game (it's called "semantic security")

Recall IND-CPA for symmetric key encryption

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice
3. Alice randomly chooses either M_0 or M_1 to encrypt and sends the encryption back
   - Alice does not tell Eve which one was encrypted!
4. Eve may again choose plaintexts to send to Alice and receives their ciphertexts
5. Eventually, Eve outputs a guess as to whether Alice encrypted M_0 or M_1

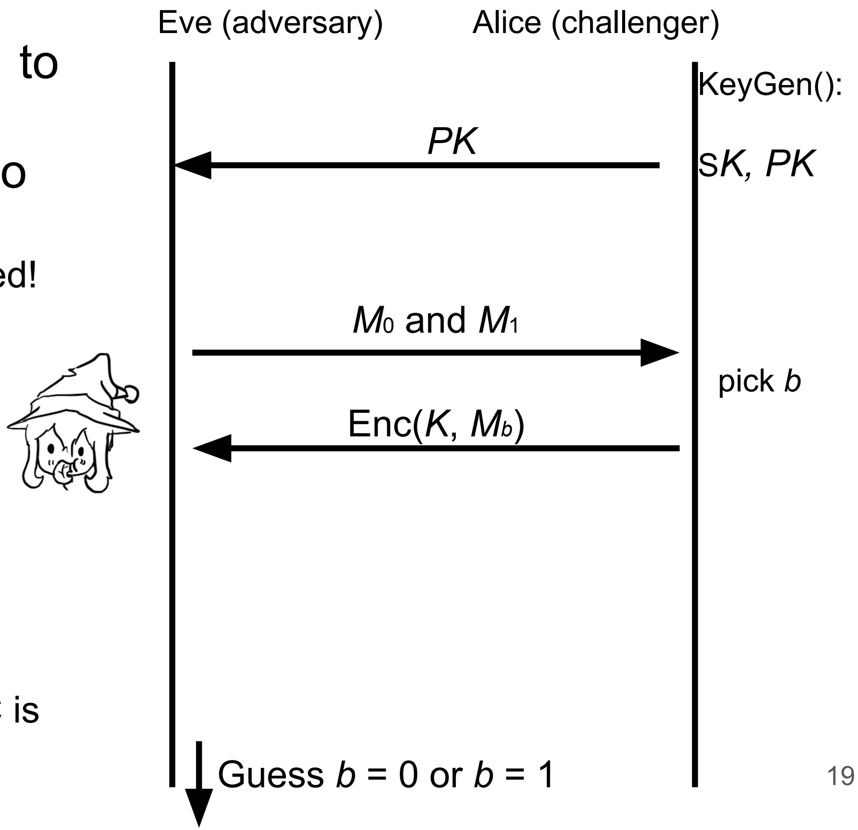
- An encryption scheme is IND-CPA secure if for all polynomial time attackers Eve:
  - Eve can win with probability ≤ 1/2 + Ɛ, where Ɛ is *negligible.*



Semantic security (IND-CPA for public-key encryption)

1. Eve issues a pair of plaintexts M_0 and M_1 to Alice
2. Alice randomly chooses either M_0 or M_1 to encrypt and sends the encryption back
    - Alice does not tell Eve which one was encrypted!
3. Eventually, Eve outputs a guess as to whether Alice encrypted M_0 or M_1

- An encryption scheme is semantically secure if for all polynomial time attackers Eve:
    - Eve can win with probability ≤ 1/2 + Ɛ, where Ɛ is *negligible.*



ElGamal Encryption

Textbook Chapter 11.4

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	• One-time pads • Block ciphers with chaining modes (e.g. AES-CBC)	• 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

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

ElGamal Encryption: Issues

- Is ElGamal encryption semantically secure?
  - No. The adversary can send $M_0 = 0$, $M_1 \neq 0$
  - Additional padding and other modifications are needed to make it semantically secure
- Malleability: The adversary can tamper with the message, so no integrity
  - 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

RSA Encryption

Textbook Chapter 11.3

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	• One-time pads • Block ciphers with chaining modes (e.g. AES-CBC)	• 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
    - Done by picking random numbers and then using a test to see if the number is (probably) 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 $d = e^{-1} \bmod (p - 1)(q - 1)$
    - Algorithm: Extended Euclid's algorithm (CS 70)
  - **Public key**: N and e
  - **Private key:** d

RSA Encryption: Definition

- Enc(e, N, M):
  - Output $M^e \bmod N$
- Dec(d, C):
  - Output $C^d = (M^e)^d \bmod N$

RSA Encryption: Correctness

1. Theorem: $M^{ed} \equiv M \bmod N$
2. Euler's theorem: for all positive coprime-with-N a, $a^{\varphi(N)} \equiv 1 \bmod N$
   a. $\varphi(N)$ is the totient function of N
   b. If N is prime, $\varphi(N) = N - 1$ (Fermat's little theorem)
   c. For a semi-prime pq, where p and q are prime, $\varphi(pq) = (p - 1)(q - 1)$
   d. This is out-of-scope CS 70 knowledge

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

(1) can be written as $M^{k\varphi(N) + 1} \equiv M \bmod N$

$M^{k\varphi(N)}M^1 \equiv M \bmod N$

$1M^1 \equiv M \bmod N$ by Euler's theorem

$M \equiv M \bmod N$

RSA Encryption: Security

- **RSA problem**: Given large $N = pq$ and $C = M^e \bmod N$, it is hard to find M
  - No harder than the factoring problem (if you can factor N, you can recover d)
- Current best solution is to factor N, but unknown whether there is an easier way
  - Factoring problem is assumed to be hard (if you don't have a massive quantum computer, that is)

RSA Encryption: Issues

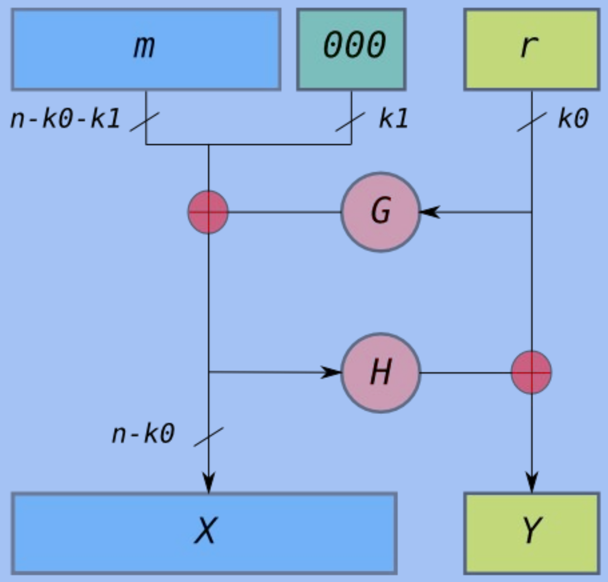
- Is RSA encryption semantically secure?
  - No. It's deterministic. No randomness was used at any point!
- Sending the same message encrypted with different public keys also leaks information
  - $m^{e_a} \bmod N_a$, $m^{e_b} \bmod n_b$, $m^{e_c} \bmod N_c$
  - Small m and e leaks information
    - e is usually small (~16 bits) and often constant (3, 17, 65537)
- Side channel: A poor implementation leaks information
  - The time it takes to decrypt a message depends on the message and the private key
  - This attack has been successfully used to break RSA encryption in OpenSSL
- Result: We need a probabilistic padding scheme

OAEP

- **Optimal asymmetric encryption padding** (**OAEP**): A variation of RSA that introduces randomness
  - Different from “padding” used for symmetric encryption, used to add randomness instead of dummy bytes
- Idea: RSA can only encrypt “random-looking” numbers, so encrypt the message with a random key
- RSA encryption is proved semantically secure assuming a stronger version of the RSA problem and using OAEP padding

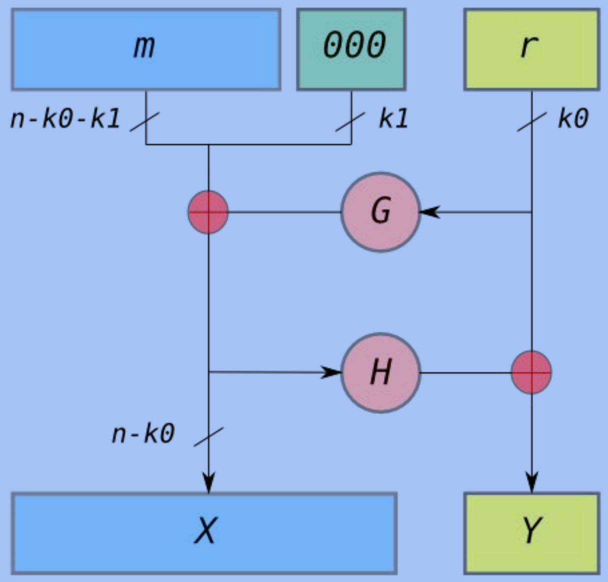
OAEP: Padding

1. k_0 and k_1 constants defined in the standard, and G and H are hash functions
   - M can only be n - k_0 - k_1 bits long
   - G produces a (n - k_0)-bit hash, and H produces a k_0-bit hash
2. Pad M with k_0 0's
   - Idea: We should see 0's here when unpadding, or else someone tampered with the message
3. Generate a random, k_1-bit string r
4. Compute $X = M \,||\, 00...0 \oplus G(r)$
5. Compute $Y = r \oplus H(X)$
6. Result: $X \,||\, Y$



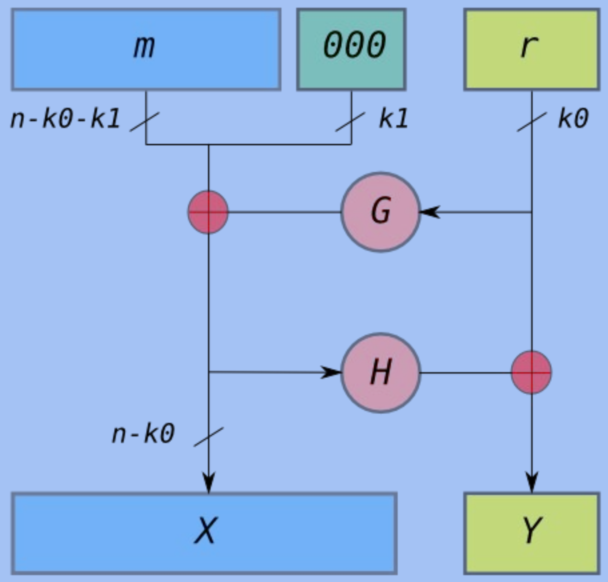
OAEP: Unpadding

1. Compute $r = Y \oplus H(X)$
2. Compute $M \,||\, 00...0 = X \oplus G(r)$
3. Verify that $M \,||\, 00...0$ actually ends in k_1 0's
    - Error if not



OAEP

- Even though *G* and *H* are irreversible, we can recover their inputs using XOR and work backwards
- This structure is called a **Feistel network**
    - Can be used for encryption algorithms if *G* and *H* depend on a key
        - Example: DES (out of scope)
- **Takeaway**: To fix the problems with RSA (it's only secure encrypting random numbers and isn't semantically secure), use RSA with OAEP, abbreviated as RSA-OAEP



Hybrid Encryption

- Issues with public-key encryption
  - Notice: We can only encrypt small messages because of the modulo operator
  - Notice: There is a lot of math, and computers are slow at math
  - Result: Asymmetric doesn’t work for large messages
- **Hybrid encryption**: Encrypt data under a randomly generated key K using symmetric encryption, and encrypt K using asymmetric encryption
  - Benefit: Now we can encrypt large amounts of data quickly using symmetric encryption, and we still have the security of asymmetric encryption
- Almost all cryptographic systems encrypting user data use hybrid encryption