Lec 12: Digital Signatures and Certificates

CSED415: Computer Security

Spring 2024

Seulbae Kim

POSTECH

POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Administrivia

- Lab 03 is out!
  - Due Sunday, April 7
  - Breaking a faulty cryptographic scheme and a game

Cryptography roadmap

Scheme Goal	Symmetric Key	Asymmetric Key
Confidentiality	✅ One Time Pad (OTP) ✅ Block ciphers (DES, AES) ✅ Stream ciphers	✅ ElGamal encryption ✅ RSA encryption
Integrity & Authentication	✅ Message Authentication Code (MAC)	• Digital signature
CIA at the same time	✅ Authenticated encryption	

Tools

✅ Secure key exchange

✅ Hash

Digital Signatures

Missing integrity and authenticity

- Asymmetric enc/decryption, like the symmetric schemes, only provide confidentiality, but not integrity
  - MAC solves integrity problem for symmetric-key settings

→ Can we use asymmetric encryption to provide integrity and authenticity of messages?

Authenticity in real life

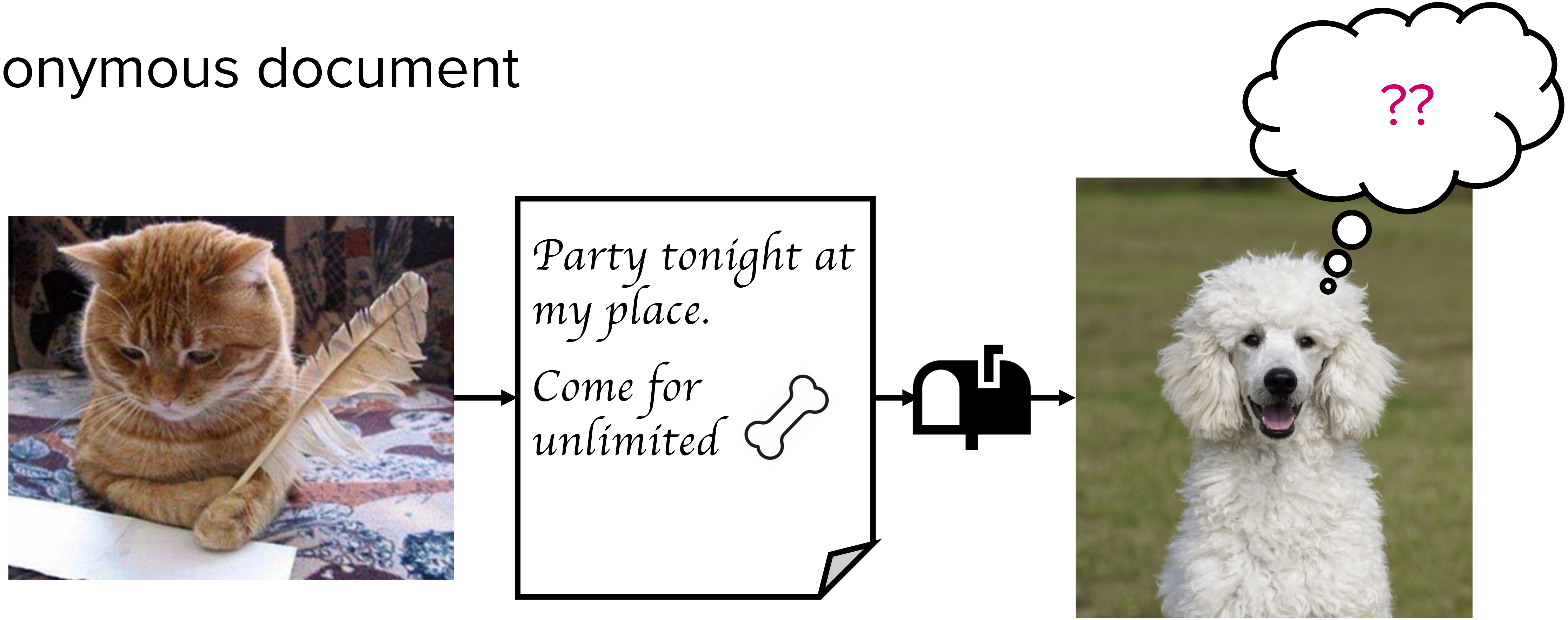
- Anonymous document

Party tonight at my place.

Come for unlimited 🦴

Authenticity in real life

- Anonymous document



Party tonight at my place.

Come for unlimited 🦴

Authenticity in real life

- Signed (stamped) document

Digital signatures

- Key idea:
  - Asymmetric schemes use two keys: private key and public key
  - Only the **owner of the private key can sign** messages using the private key
  - **Everyone else can verify** the signature using the public key

Digital signatures

- Method:
  - Given: A key pair (k_p, k_s)
    - k_s: private key (also known as signing key or secret key)
    - k_p: public key
  - $S(k_s, m)$: Sign m using secret key k_s to generate signature σ
  - $V(k_p, m, \sigma)$: Verify signature σ of message m using public key k_p

Difference in key usage

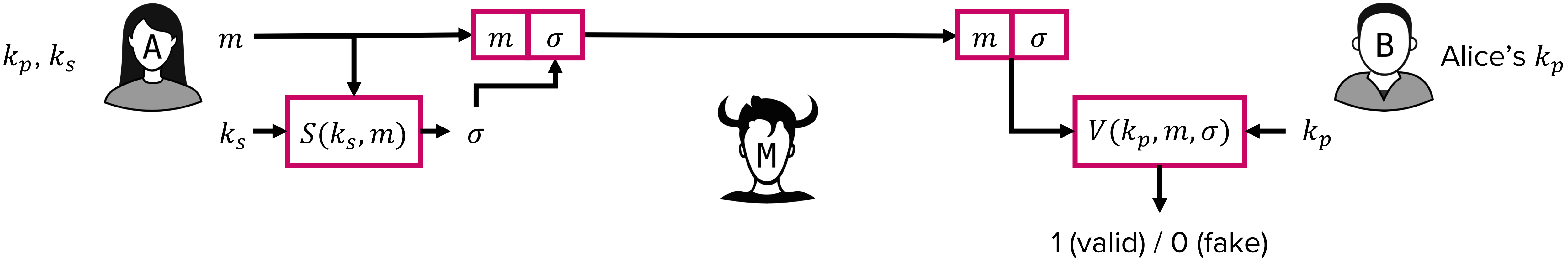
- Note: Digital signatures use key pair in the opposite order of asymmetric encryption schemes
  - Asymmetric encryption:
    - Alice (sender) encrypts using Bob's (receiver's) public key k_p
    - Bob (receiver) decrypts using his (receiver's) secret key k_s
  - Digital signature:
    - Alice (sender) signs using her (sender's) secret key k_s
    - Bob or anyone (receiver) verifies using Alice's (sender's) public key k_p

MAC vs Digital signature

- In a MAC scheme (symmetric):
  - The verifier must share a secret (key k) with the sender
  - Consequently, the verifier could potentially impersonate the sender!
    - Generate MAC tags using the shared key
- In a digital signature scheme (asymmetric):
  - The verifier utilizes the sender's public key
    - Does not require any shared secret
  - Consequently, the verifier cannot impersonate the sender!
    - Only who owns the private key (i.e., the sender) can generate valid signatures

Security of DS

- DS scheme



- Intuition for security (Same as MAC's)
  - **Unforgeability**: No polynomial time adversary should be able to produce forgery (i.e., m and sig σ, where m was never queried to S) with non-negligible probability, even after seeing multiple legitimate (m, σ) pairs

Security of DS

- Let's utilize the Vanilla RSA encryption for building S and V
  - Recall RSA:
    - Select two large primes p and q. $N = pq$
    - Compute $\varphi(N) = (p-1)(q-1)$
    - Select k_p, which is coprime to $\varphi(N)$ // $k_p = e$ (notation in Lec 10)
    - Compute $k_s = k_p^{-1} \, mod \, \varphi(N)$ // $k_s = d$ (notation in Lec 10)
    - Ciphertext $c \leftarrow E(k_p, N, m) = m^{k_p} \, mod \, N$
    - Decrypted $m \leftarrow D(k_s, N, c) = c^{k_s} \, mod \, N$

→ Key property (Euler's theorem): $m^{k_p k_s} \, mod \, N = m$

The order of k_p and k_s does not matter!

Security of DS

- Let’s utilize the Vanilla RSA encryption for building S and V
  - Message m, secret key k_s, public key (k_p, N)
  - Sign $S(N, k_s, m)$: $\sigma \leftarrow m^{k_s} \ mod \ N$
  - Send m and σ
  - Verify $V(N, k_p, \sigma)$:
    - $m' \leftarrow \sigma^{k_p} \ mod \ N$ // message retrieved by decrypting σ
    - If $m = m'$ then return 1, else return 0

→ Can an attacker forge a valid pair (m, σ)?

Yes! Any attacker can forge $m = 1$ and $\sigma = 1$.

Verification: $m' \leftarrow \sigma^{k_p} \ mod \ N = 1^{k_p} \ mod \ N = 1$.

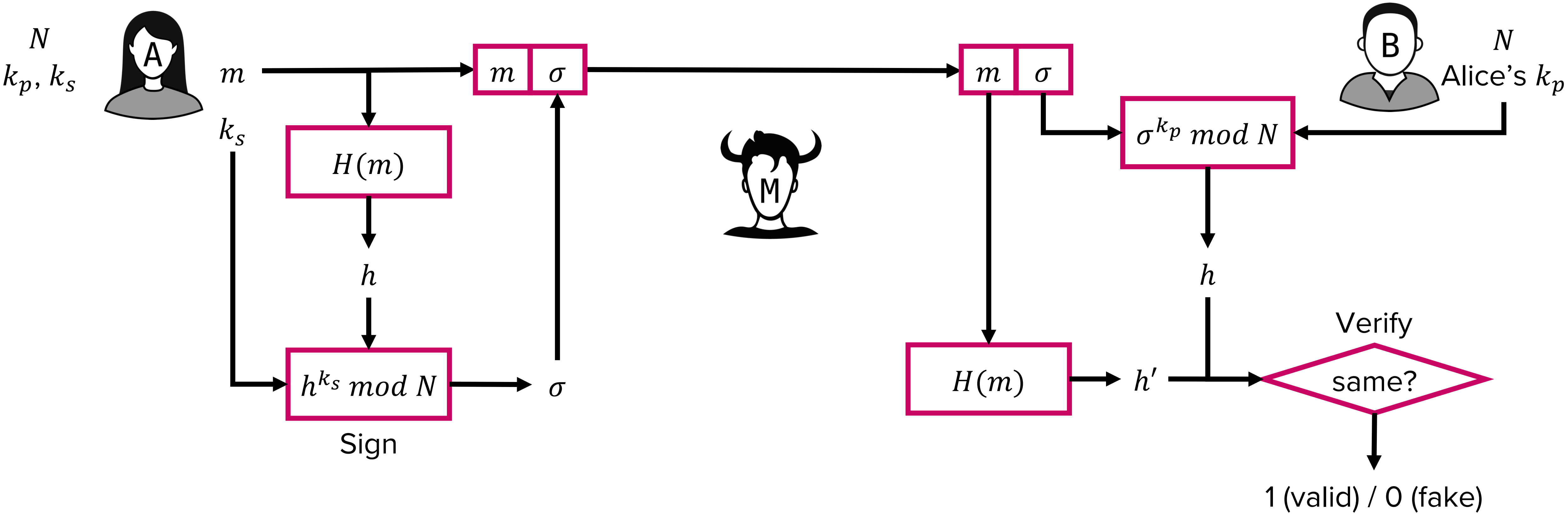
$m = m'$ holds. Return true

Secure DS: Hash-then-sign

- Countermeasure: Hash the message first
  - Message m, secret key k_s, public key (k_p, N)
  - $h \leftarrow H(m)$
  - Sign $S(N, k_s, h)$: $\sigma \leftarrow h^{k_s}\ mod\ N$
  - Send m and σ
  - Verify $V(N, k_p, \sigma)$:
    - $h \leftarrow H(m)$ // compute the hash of the received message m
    - $h' \leftarrow \sigma^{k_p}\ mod\ N$ // hash retrieved by decrypting σ
    - if $h = h'$ then return 1, else return 0

→ The previous forgery using $(m = 1, \sigma = 1)$ no longer works

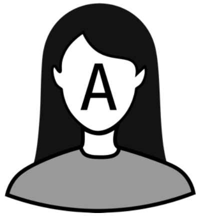
Summary: Digital signature using hash and RSA



We can now provide integrity using an asymmetric scheme!

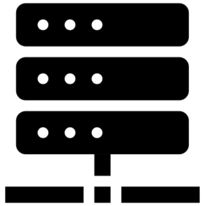
Digital signature in practice

- SSH (secure shell) – passwordless authentication


Alice

Initial setup for account "Alice"

1. Alice logs in using password
2. Register Alice's public key in `/home/Alice/.ssh/authorized_keys`
3. Disable password login in ssh configuration


Server

Passwordless login

1. Alice wants to log in
2. Alice signs her identity using her secret key and sends it to the server
3. Using the stored public key of Alice, the server verifies Alice's identity
4. Alice logs in without using password

Only Alice can securely log in as long as her secret key is not leaked

Rethinking “authentication” problem

- Pizza prank
  - Mallory creates an e-mail order: and signs the order with his secret key

```
Dear Pizza Store,
Please deliver me four pepperoni pizzas.
Thank you,
- Bob
```

  - Mallory sends the order to Pizza Store
  - Pizza Store asks Mallory, “Hey Bob, send us your public key”
  - Mallory sends his public key
  - Pizza Store verifies the signature and delivers four pepperoni pizzas to Bob
  - Bob is vegan

Are public keys enough for strong authentication?

Cryptography roadmap

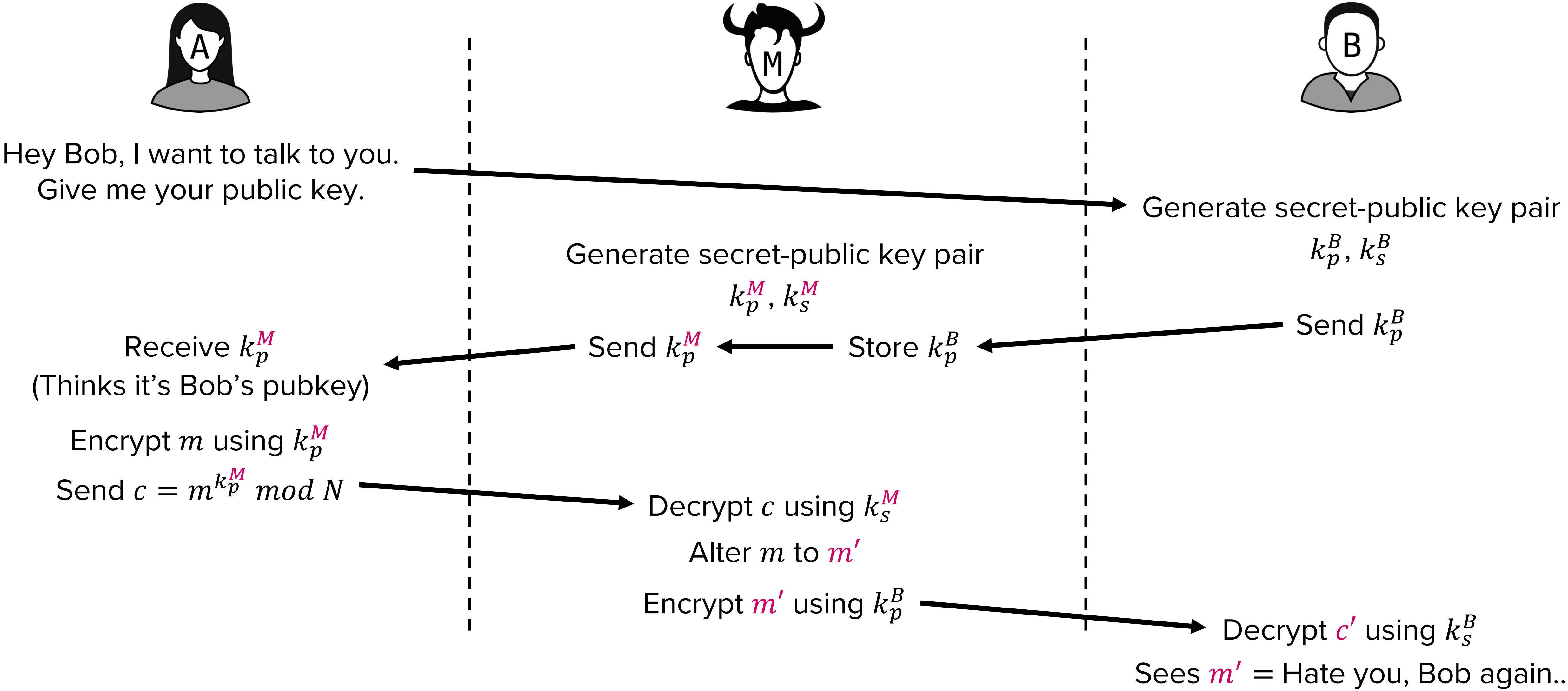
Scheme Goal	Symmetric Key	Asymmetric Key
Confidentiality	✅ One Time Pad (OTP) ✅ Block ciphers (DES, AES) ✅ Stream ciphers	✅ ElGamal encryption ✅ RSA encryption
Integrity & Authentication	✅ Message Authentication Code (MAC)	✅ Digital signature
CIA at the same time	✅ Authenticated encryption	

Really?

Tools
- ✅ Secure key exchange
- ✅ Hash

Certification Authorities

Problem: Distributing public keys



Hey Bob, I want to talk to you.
Give me your public key.

Generate secret-public key pair
k_p^B, k_s^B

Generate secret-public key pair
k_p^M, k_s^M

Send k_p^B

Store k_p^B

Send k_p^M

Receive k_p^M
(Thinks it's Bob's pubkey)

Encrypt m using k_p^M

Send $c = m^{k_p^M} \ mod \ N$

Decrypt c using k_s^M

Alter m to m'

Encrypt m' using k_p^B

Decrypt c' using k_s^B

Sees m' = Hate you, Bob again..

Problem: Distributing public keys

Hey Bob, I want to talk to you.

Man-in-the-Middle (MitM) attack becomes possible by merely replacing the public key

Alter m to m'

Encrypt m' using k_p^B

Decrypt c' using k_s^B

Sees m' = Hate you, Bob again..

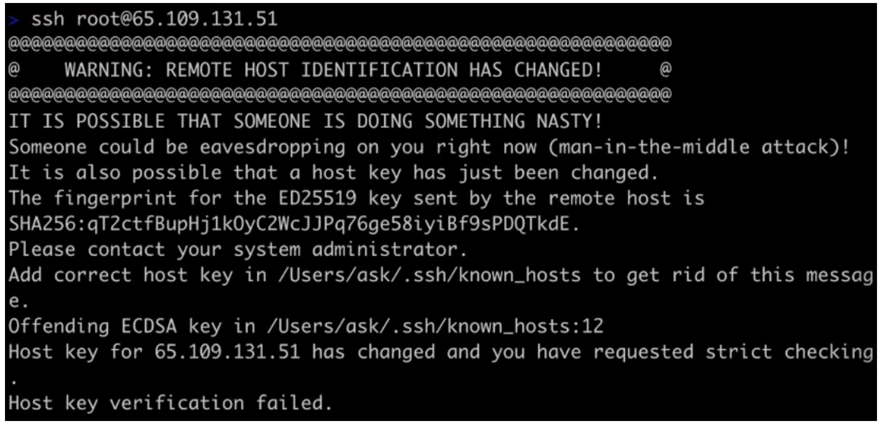
Problem: Distributing public keys

- Countermeasure idea
  - Sign Bob’s public key to prevent tampering?
- Dilemma:
  - For verification, we require his **public key**
  - Yet, the purpose was to verify Bob’s **public key** in the first place
  - Creates a circular problem!
    - Alice cannot fully trust any public key

We need a “root of trust”!

Establishing root of trust: Trust-on-first-use (TOFU)

- Trust the public key that is used for the initial communication and warn the user if the key changes in the future
  - Rationale: Attacks are not frequent, so assume that the initial communication was not attacked
  - Used by SSH (Secure Shell)
    - Connect to a new server from my machine
    - Server's identification is saved on my machine (in `~/.ssh/known_hosts`)
    - If the server sends a different identification, we can suspect an MitM attack



```
> ssh root@65.109.131.51
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ED25519 key sent by the remote host is
SHA256:qT2ctfBupHj1kOyC2WcJJPq76ge58iyiBf9sPDQTkdE.
Please contact your system administrator.
Add correct host key in /Users/ask/.ssh/known_hosts to get rid of this messag
e.
Offending ECDSA key in /Users/ask/.ssh/known_hosts:12
Host key for 65.109.131.51 has changed and you have requested strict checking
.
Host key verification failed.
```

Problem: Assumption is too strong

Establishing root of trust: Certification Authority

- Certification Authority (CA) binds a public key to a specific entity (E)
  - Serves as a trusted third party (TTP)
- Procedure
  - Bob registers his public key with CA, providing a “proof of identity”
  - CA creates an identity binding of Bob and his public key
  - The binding, digitally signed by CA’s private key, is the certificate

$$\boxed{id^B \leftarrow B \bowtie k_p^B} \longrightarrow \boxed{S(k_s^{CA}, id^B)} \longrightarrow \boxed{cert^B}$$

Certification Authority (CA)

- Now when Alice wants Bob’s public key
  - Alice gets Bob’s certificate ($cert^B$) from the CA
  - Alice applies the CA’s public key to verify Bob’s identity

$cert^B \rightarrow V(k_p^{CA}, cert^B) \rightarrow id^B \leftarrow B \bowtie k_p^B$

  - If Alice trusts the CA (root of trust), Alice can trust that Bob’s public key is truly Bob’s

Building a practical CA

- Naïve idea: Make a central, trusted directory (TD) from which you can fetch anyone's public key
  - The TD has a public/secret key pair: k_p^{TD} and k_s^{TD}
  - The directory publishes k_p^{TD} to everyone
  - When someone requests Bob's public key, the directory sends a certificate for Bob's identity
    - $cert^B$, which is $B \bowtie k_p^B$ signed using k_s^{TD}
  - If you trust the TD, you trust every public key

Building a practical CA

- Naïve idea: Make a central, trusted directory (TD) from which you can fetch anyone's public key
- Problems
  - **Scalability**: One directory will not have enough computing power to serve all entities in the entire world
  - **Single point of failure**:
    - If the TD fails, every service depending on TD becomes unavailable
    - If the TD is compromised, you cannot trust anyone
    - If the TD is compromised, it is extremely difficult to recover

Building a practical CA

- Practical idea #1: Hierarchical trust model
  - The roots of trust may **delegate** the identity bindings and signing power to other authorities
    - Alice's public key is $\boldsymbol{k_p^A}$ and I trust her to sign for POSTECH
    - Bob's public key is $\boldsymbol{k_p^B}$ and I trust him to sign for the CSE department
    - Charlie's public key is $\boldsymbol{k_p^C}$. (I don't let him sign for anyone else)
  - Hierarchy
    - Root CA
    - Alice and Bob are intermediate CAs

Solves the scalability problem

Building a practical CA

- Practical idea #2: Multiple trust anchors
  - There are more than 200 root CAs in the world
  - Most operating systems provide a built-in list of trusted root CAs
    - 161 root CAs and 10 blocked CAs in MacOS 14
  - Most web browsers, too

Solves the single-point-of-failure problem

Building a practical CA

- New problem: Revocation
    - What if a CA messes up and issues a bad certificate?
        - e.g., CA: “Bob’s public key is k_p^M”
    - Everyone will trust the wrong public key
    - If Mallory signs messages, people will think Bob did

We need to be able to revoke bad certificates!

Building a practical CA – Revocation

- Approach #1: Each certificate has an expiration date
  - When the certificate expires, request a new certificate from a CA
  - Bad certificates will eventually become invalid once they expire
- Strength: No bad certificate remain forever
- Weakness: Everybody must renew frequently (overhead)
  - Frequent renewal: More security, less usability
  - Infrequent renewal: Less security, more usability

Building a practical CA – Revocation

- Approach #2: Periodically release a list of invalidated certificates
  - Users must periodically download a Certification Revocation List
- Strength: Real-time revocation (immediately add to the list)
- Weakness:
  - Size of list grows linearly to the number of revoked certificates
  - Cannot know which certificates are revoked before downloading CRL

Current certificate standard: X.509

- Certificate contains
  - Issuer's name
  - Entity's name, address, domain name, …
  - Entity's public key
  - Digital signature of the certificate (signed with the issuer's secret key)

- Core components
  - Certificates and CAs
  - Certificate revocation list

Summary

- Certificate: A signed attestation of identity
- Trusted directory: Once server holds all keys
- Certificate authorities: Provide delegated trust from a pool of multiple root CAs
    - Root CA can sign certificates for intermediate CAs
    - Certificates can be revoked (timed expiry or revocation list)

Cryptography roadmap

Scheme Goal	Symmetric Key	Asymmetric Key
Confidentiality	✅ One Time Pad (OTP) ✅ Block ciphers (DES, AES) ✅ Stream ciphers	✅ ElGamal encryption ✅ RSA encryption
Integrity & Authentication	✅ Message Authentication Code (MAC)	✅ Digital signature + ✅ CA
CIA at the same time	✅ Authenticated encryption	???

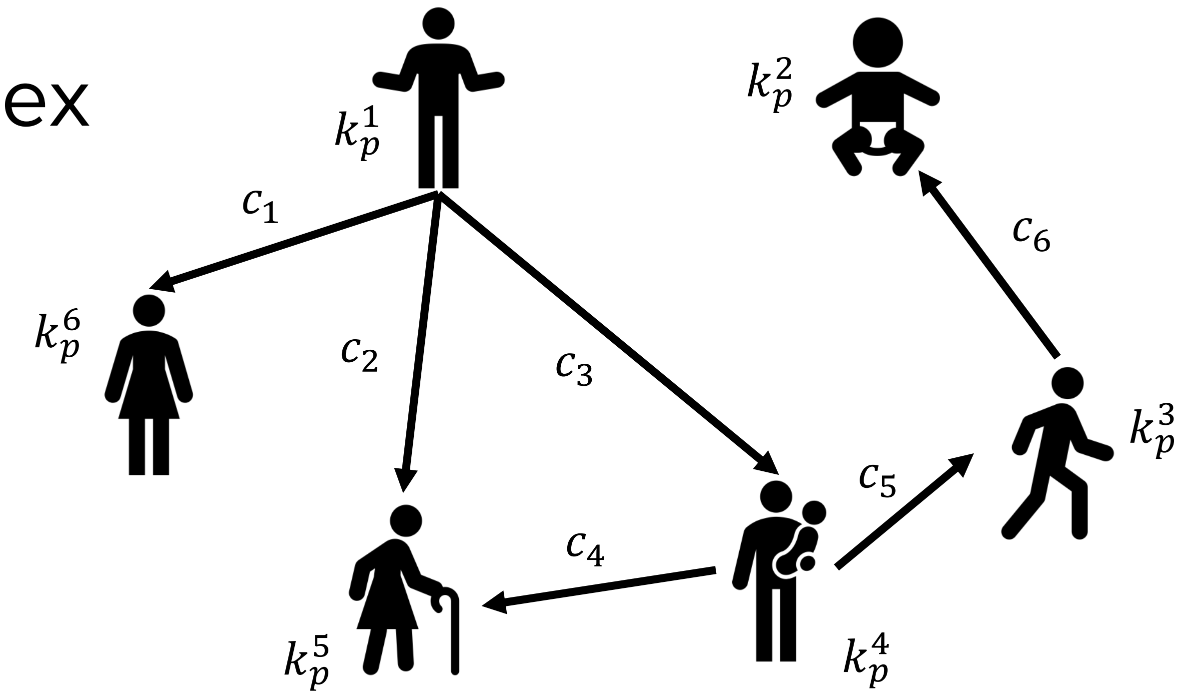
Tools
- ✅ Secure key exchange
- ✅ Hash

Multi-user Setting and Signcryption

Multi-user setting complications

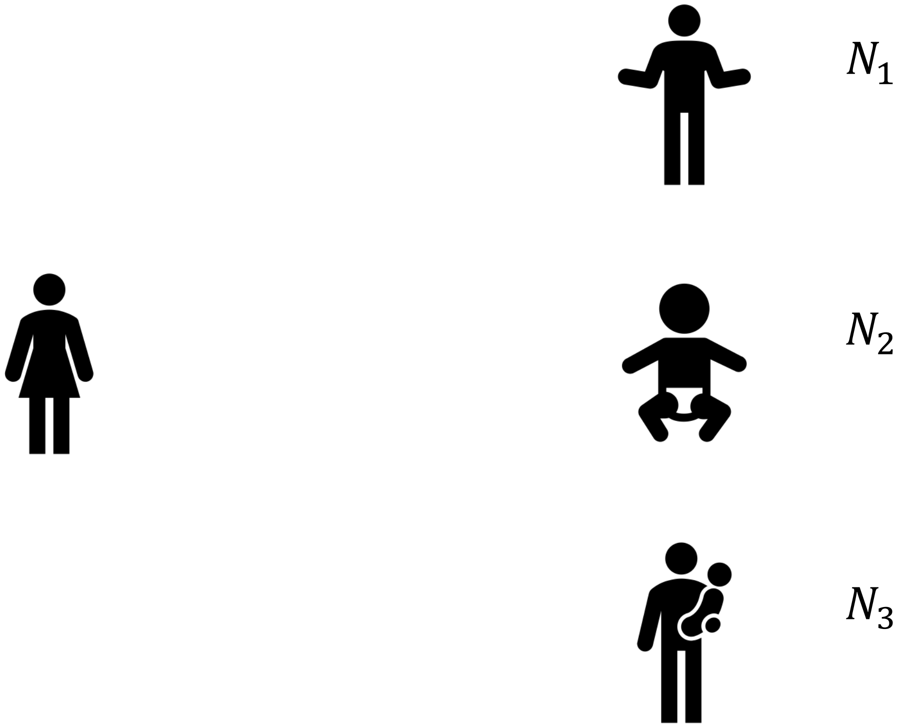
- Security of asymmetric schemes considered a single user
  - "Can sender have confidentiality?"
  - "Can receiver verify a signature?"

- Real world is much more complex



Multi-user setting complications

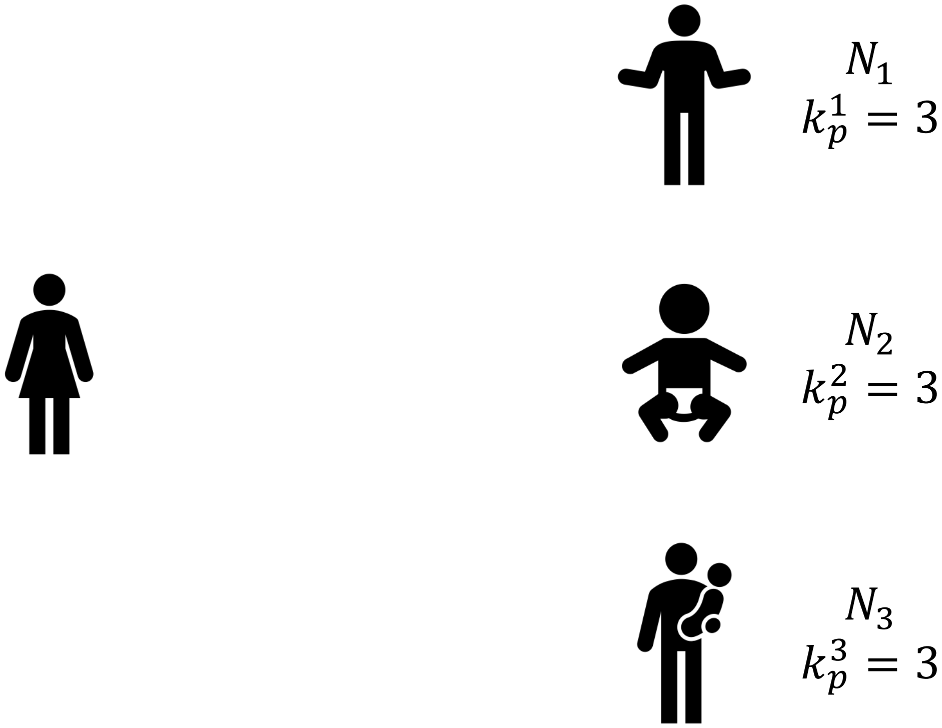
- Hastad-type attack on RSA



Three people select different large numbers N_1, N_2, N_3 for RSA key generation

Multi-user setting complications

- Hastad-type attack on RSA



N_1
$k_p^1 = 3$

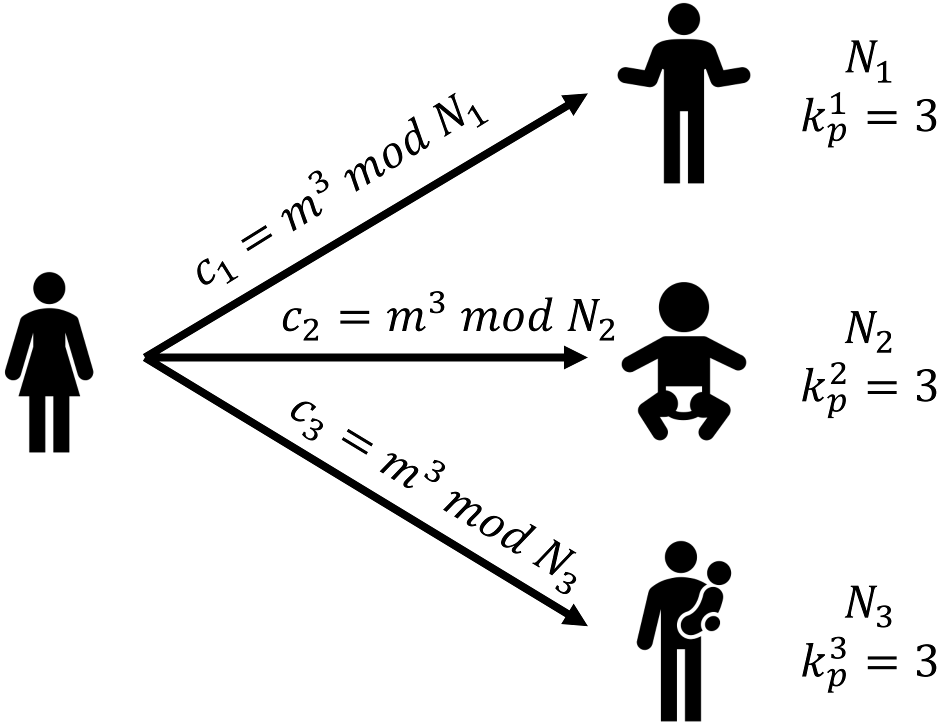
N_2
$k_p^2 = 3$

N_3
$k_p^3 = 3$

Three people happen to select the same public key k_p^i relatively to $\varphi(N_i)$, e.g., $k_p^i = 3$

Multi-user setting complications

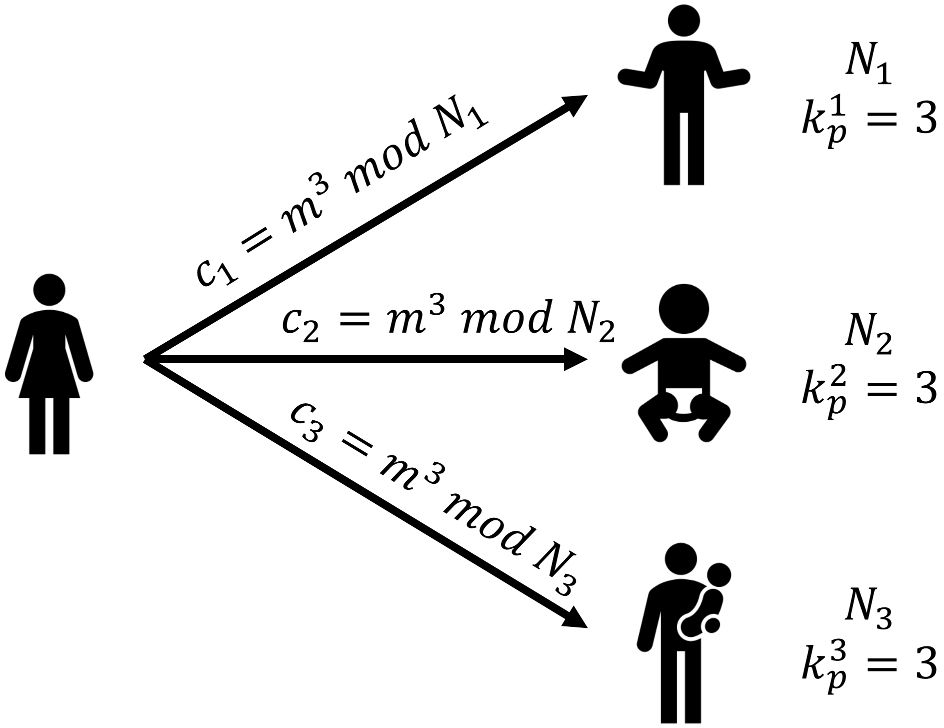
- Hastad-type attack on RSA



The sender wants to send m and RSA-encrypts it using N_i, k_p^i for each recipient

Multi-user setting complications

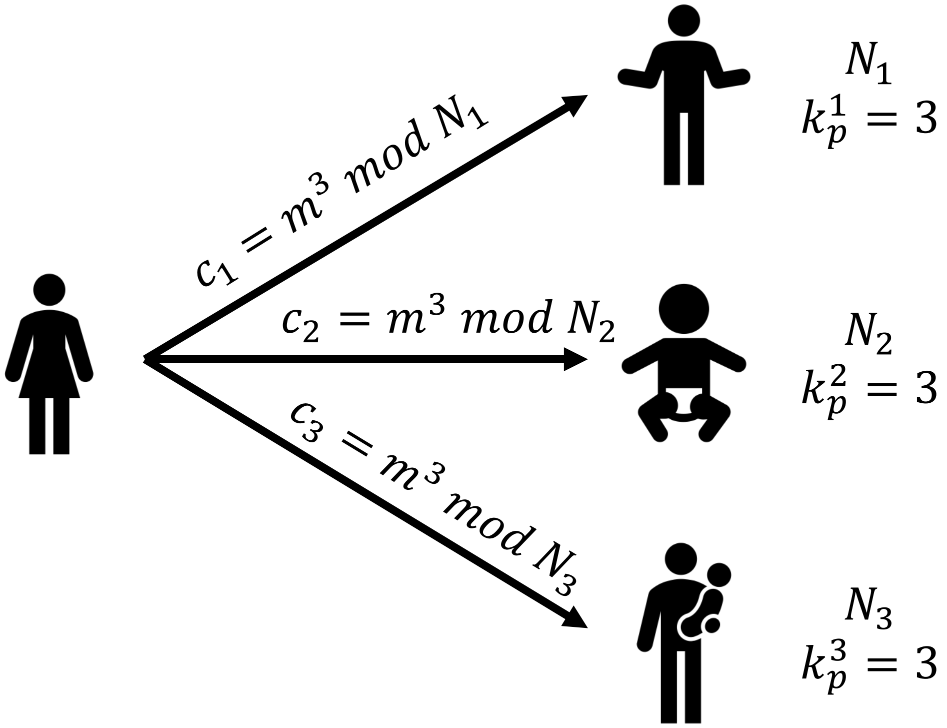
- Hastad-type attack on RSA



Only the three recipients, individually, should be able to decrypt m from c_i using their k_s^i

Multi-user setting complications

- Hastad-type attack on RSA



$c_1 = m^3 \ mod \ N_1$

$c_2 = m^3 \ mod \ N_2$

$c_3 = m^3 \ mod \ N_3$

N_1
$k_p^1 = 3$

N_2
$k_p^2 = 3$

N_3
$k_p^3 = 3$

If N_1, N_2, N_3 are relatively prime,
then by Chinese Remainder Theorem,

- $c_1 = m^3 \ mod \ N_1$
- $c_2 = m^3 \ mod \ N_2$
- $c_3 = m^3 \ mod \ N_3$

can be combined to find:

$$c = m^3 \ mod \ N_1 N_2 N_3$$

Since $m^3 < \mathrm{N_1 N_2 N_3}$, we get $m = \sqrt[3]{c}$

m can be completely recovered using public keys

Signcryption

- Signcryption is a public key-based primitive that assures confidentiality, integrity, and authenticity at the same time
  - Not by separately utilizing encryption and digital signatures
  - Goal is to combine encryption and signing into a single operation
- e.g., sign-then-encrypt?
  - Signing involves an encryption (using a secret key)
  - Encrypting involves another encryption (using a public key)

  → Redundancy (== inefficiency)

Signcryption

- Signcryption presents significant challenges:
  - Strong security should be provided:
    - Indistinguishability under chosen plaintext/ciphertext attacks
    - Unforgeability
  - Multi-user setting poses more challenges
    - e.g., Hastad-type attack

- As of now, no provably-secure algorithm has been developed

Cryptography roadmap

Scheme Goal	Symmetric Key	Asymmetric Key
Confidentiality	✅ One Time Pad (OTP) ✅ Block ciphers (DES, AES) ✅ Stream ciphers	✅ ElGamal encryption ✅ RSA encryption
Integrity & Authentication	✅ Message Authentication Code (MAC)	✅ Digital signature + ✅ CA
CIA at the same time	✅ Authenticated encryption	N/A

Tools

- ✅ Secure key exchange
- ✅ Hash

Coming up next

- What do we do in the real world?
  - Applications (e.g., Internet Security Protocols)
  - Incidents of crypto-based attacks

Questions?