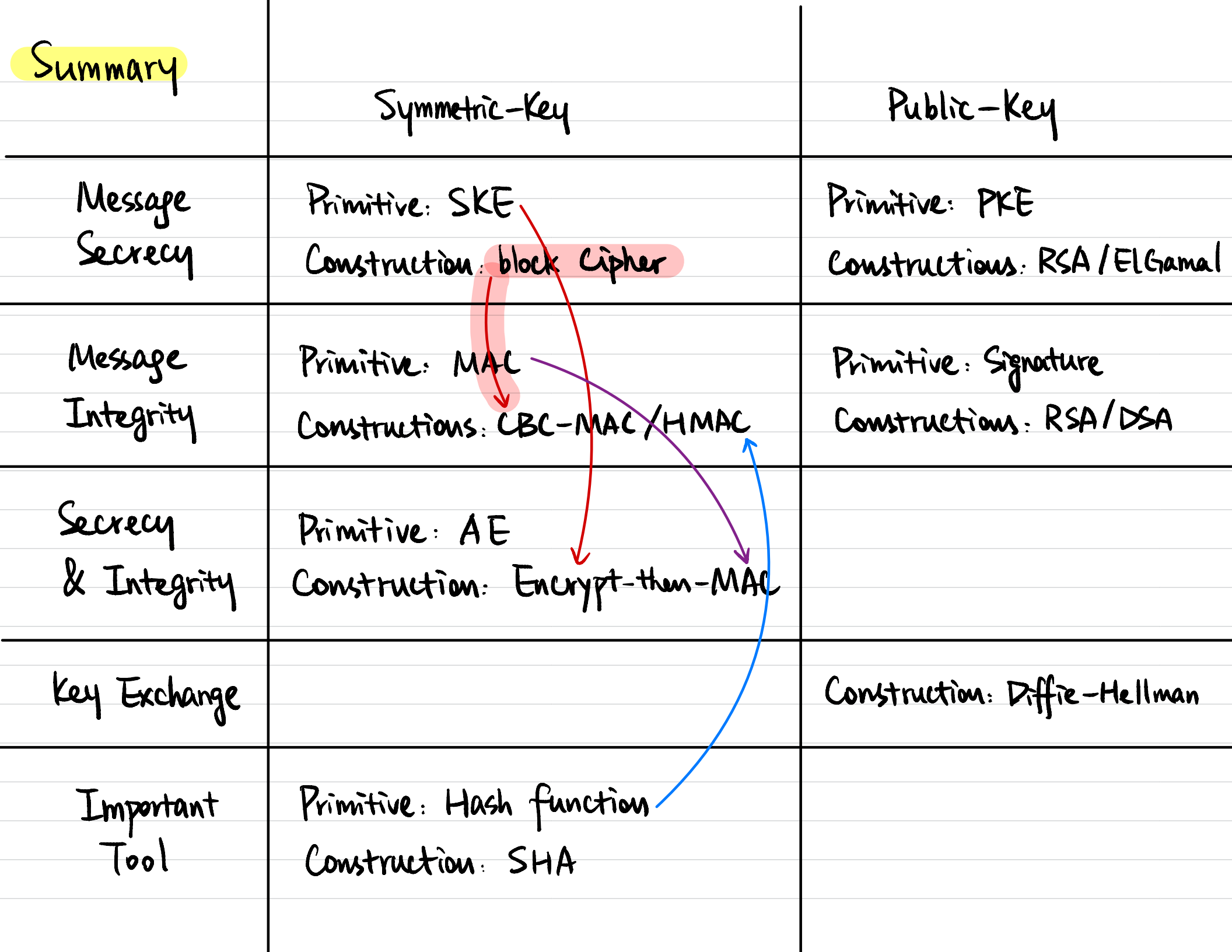
CSCI 1515 Applied Cryptography

This Lecture:

- Block Cipher and Modes of Operation (Continued)
- CBC-MAC
- Case Study: Secure Shell (SSH)
- Certificates and Public Key Infrastructure (PKI)

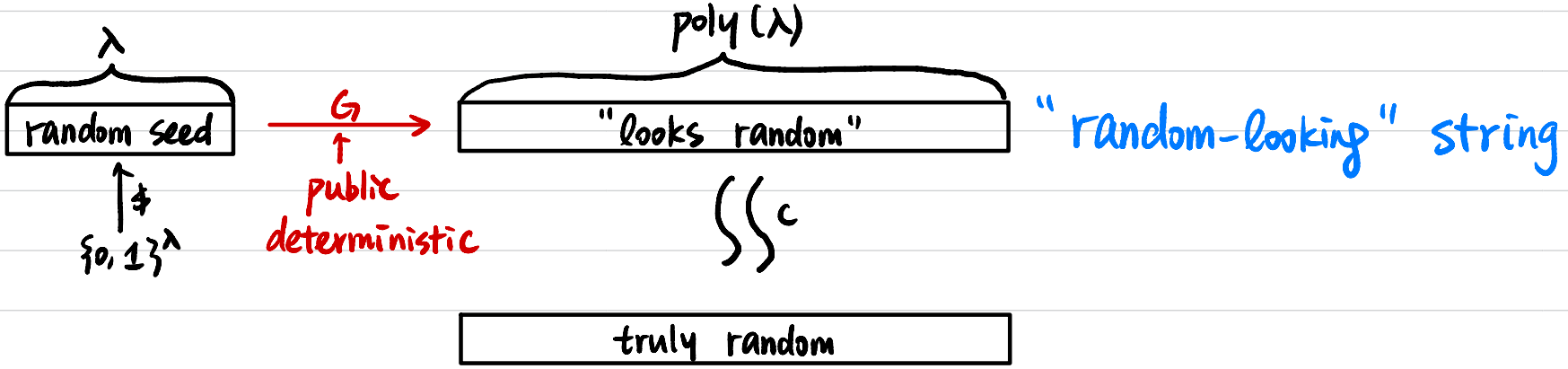
Summary

	Symmetric-Key	Public-Key
Message Secrecy	Primitive: SKE Construction: block cipher	Primitive: PKE Constructions: RSA / ElGamal
Message Integrity	Primitive: MAC Constructions: CBC-MAC / HMAC	Primitive: Signature Constructions: RSA / DSA
Secrecy & Integrity	Primitive: AE Construction: Encrypt-then-MAC	
Key Exchange		Construction: Diffie-Hellman
Important Tool	Primitive: Hash function Construction: SHA	



Pseudorandom Function (PRF)

Pseudorandom Generator (PRG)



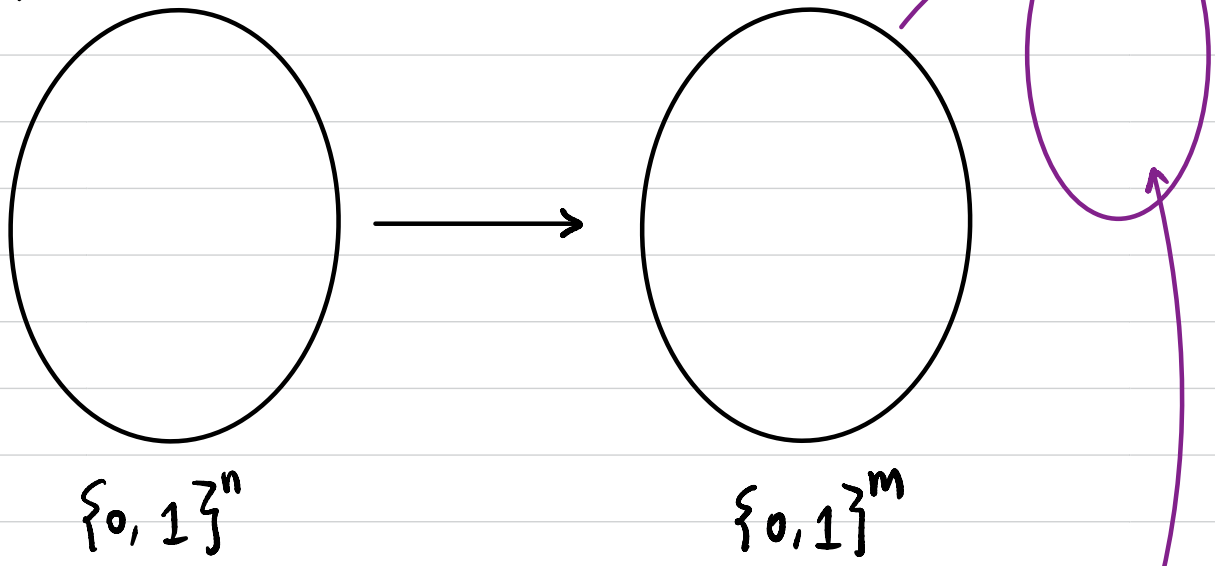
"random-looking" string

Pseudorandom Function (PRF):

"random-looking" function

Pseudorandom Function (PRF)

$k \xleftarrow{\$} \{0,1\}^\lambda$ $\quad F_k :$



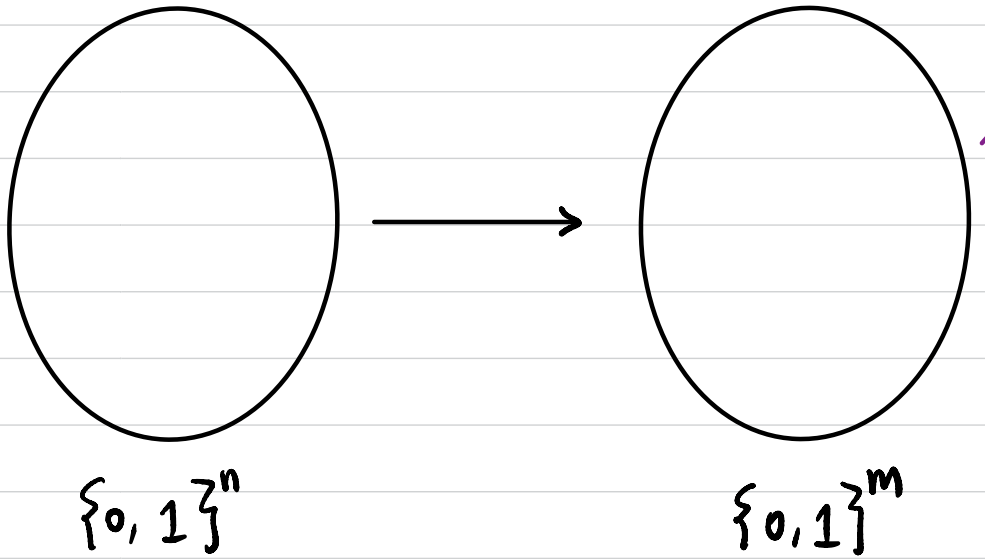
How many possible F_k's ?

2^λ

$\S\S$ C (not knowing k)

$f \xleftarrow{\$} \{ F \mid F: \{0,1\}^n \to \{0,1\}^m \}$

$f :$

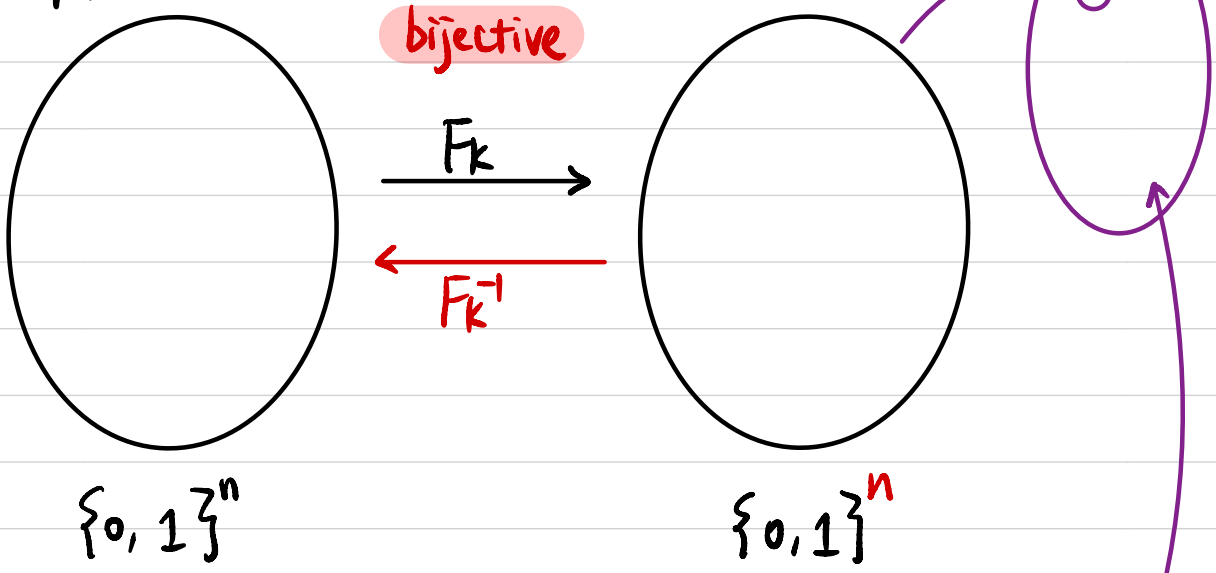


How many possible f's ?

$(2^m)^{2^n}$

Pseudorandom Permutation (PRP)

$k \xleftarrow{\$} \{0,1\}^\lambda$ $\quad F_k:$



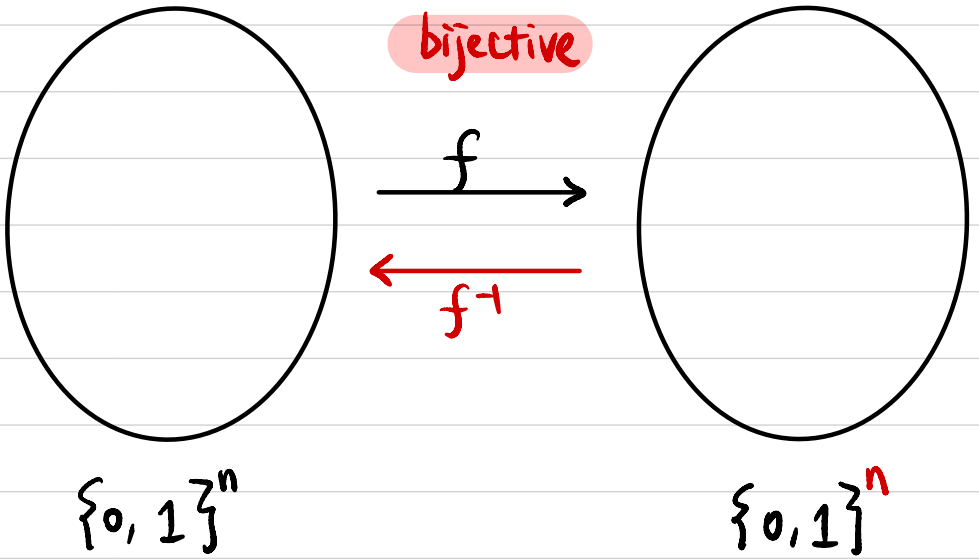
How many possible F_k's ?

2^λ

$\mathcal{C}$ (not knowing k)

$f \xleftarrow{\$} \{ F \mid F: \{0,1\}^n \to \{0,1\}^n,$ F is bijective $\}$

$f:$



How many possible f's ?

$2^n!$

Block Cipher

$$F: \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$$

λ: key length — AES: $n = 128$

n: block length — $\lambda = 128/192/256$

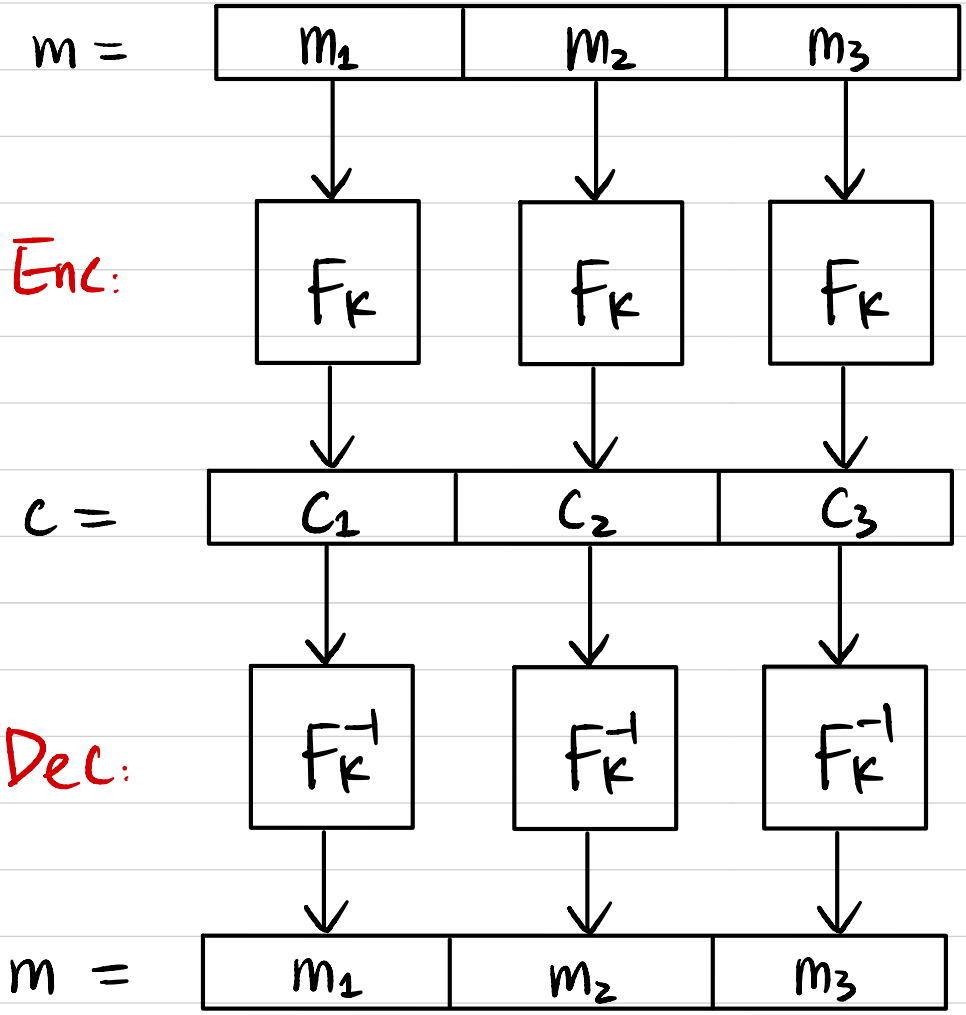
It is assumed to be a pseudorandom permutation (PRP).

Construct an SKE scheme from F for arbitrary-length messages.

- $k \xleftarrow{\$} \{0,1\}^\lambda$
- $Enc_k(m)$ — $|m| = \alpha \cdot \lambda$ (If not, pad it to a multiple of λ)
- $Dec_k(c)$

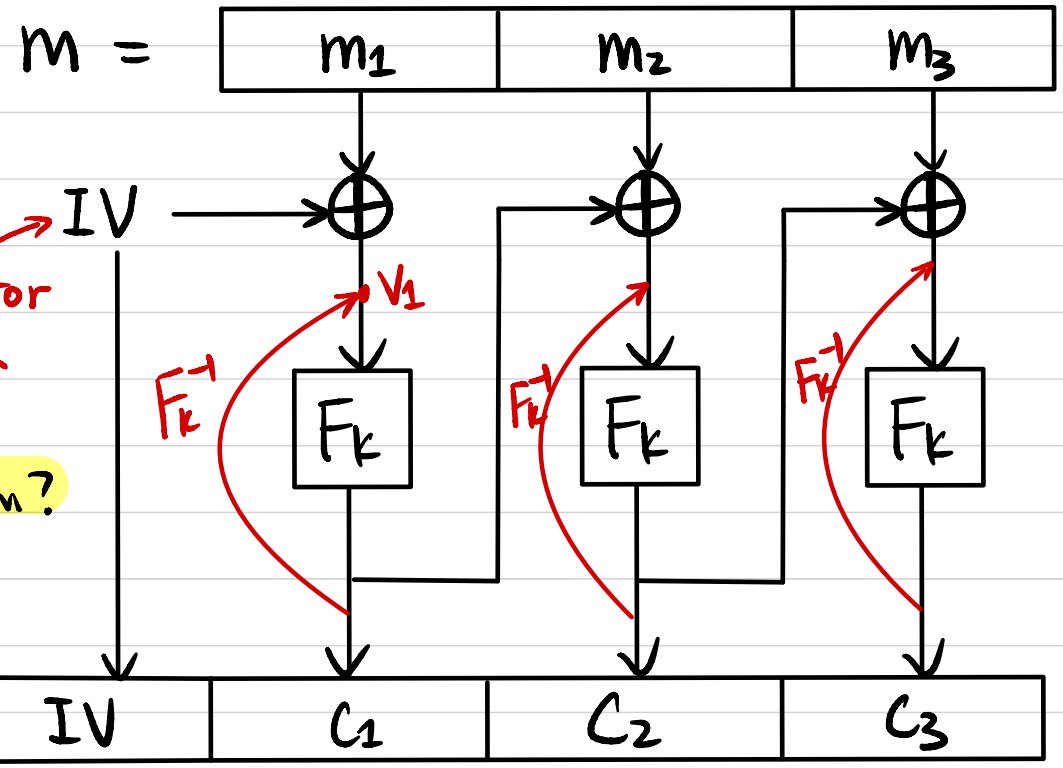
Goal: CPA (Chosen Plaintext Attack) Security

Electronic Code Book (ECB) Mode



CPA Secure? No! Deterministic!

Cipher Block Chaining (CBC) Mode



Initialization Vector

$IV \xleftarrow{\$} \{0,1\}^{\lambda}$

What if not random?

$m_1 \oplus IV \to V_1$

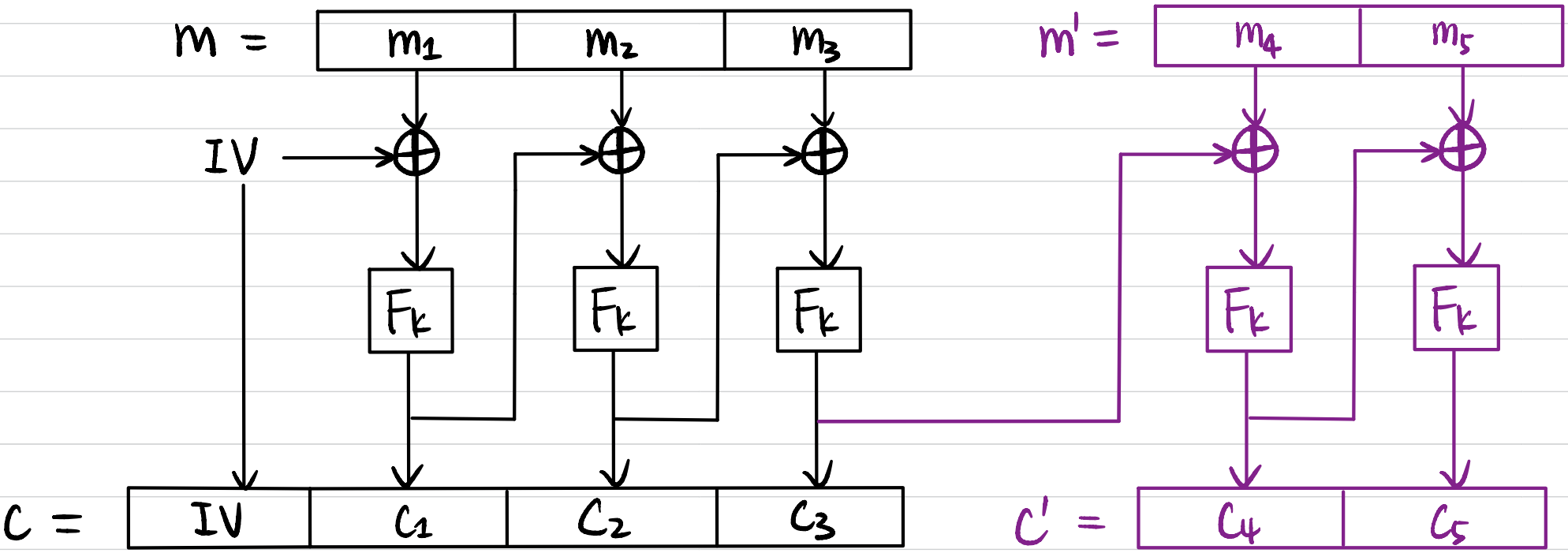
$IV \oplus V_1 \to m_1$

How to decrypt?

CPA secure? YES!

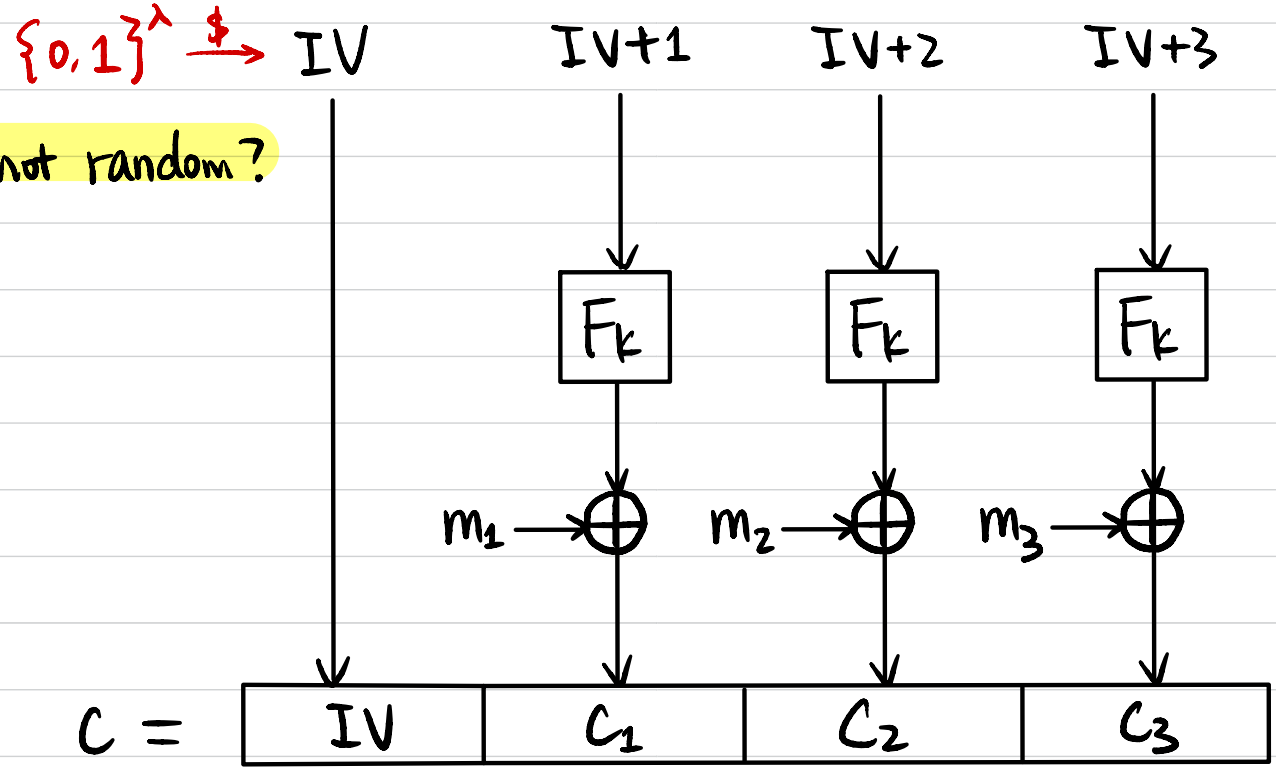
Can we parallelize the computation? NO for Enc, YES for Dec

Chained Cipher Block Chaining (CBC) Mode



CPA secure?

Counter (CTR) Mode



What if not random?

How to decrypt?

CPA secure?

"Stateful" CTR Mode?

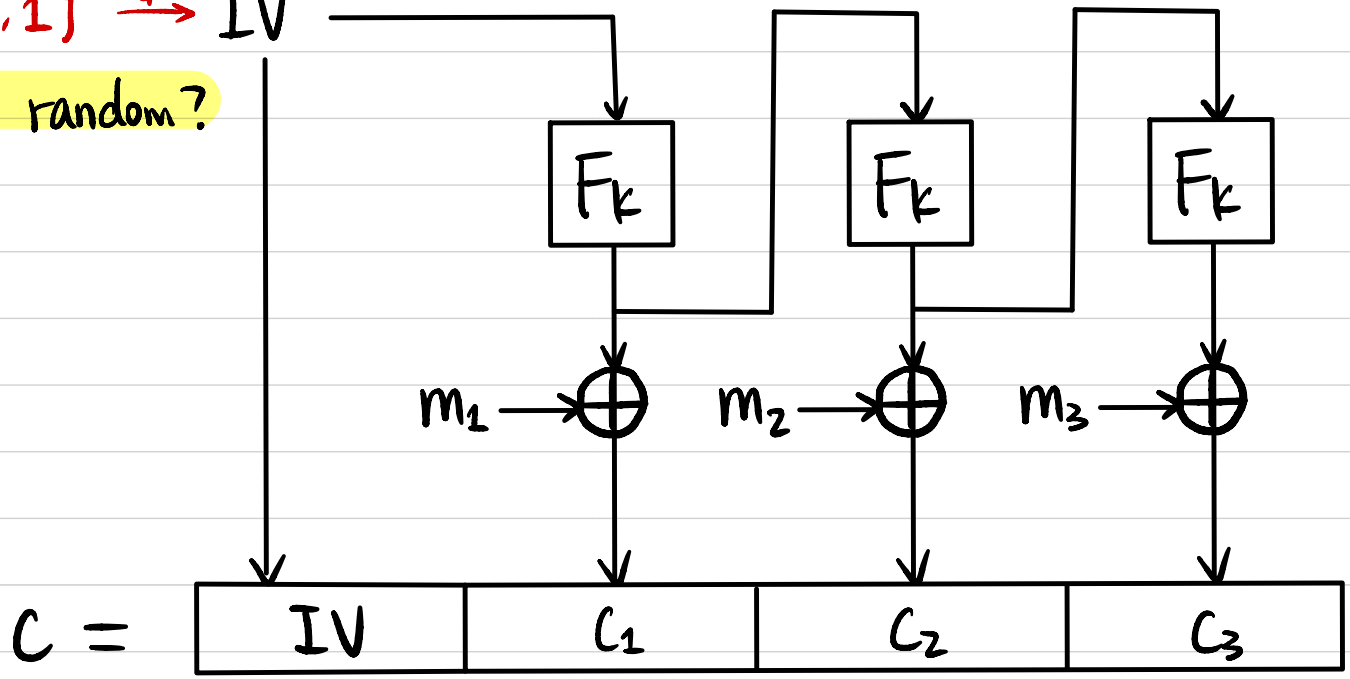
Can we parallelize the computation?

PRG from PRF

Output Feedback (OFB) Mode

$\{0,1\}^\lambda \xrightarrow{\$}$ IV

What if not random?



How to decrypt?

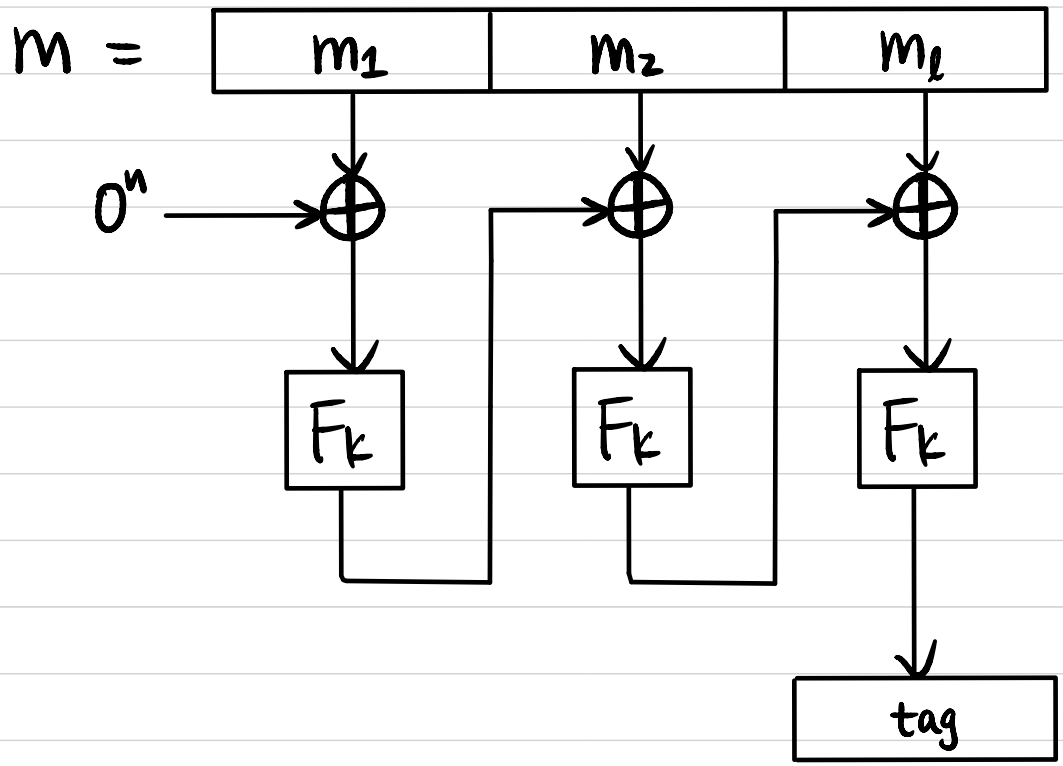
CPA secure?

"Stateful" OFB Mode?

Can we parallelize the computation?

PRG from PRF

CBC-MAC

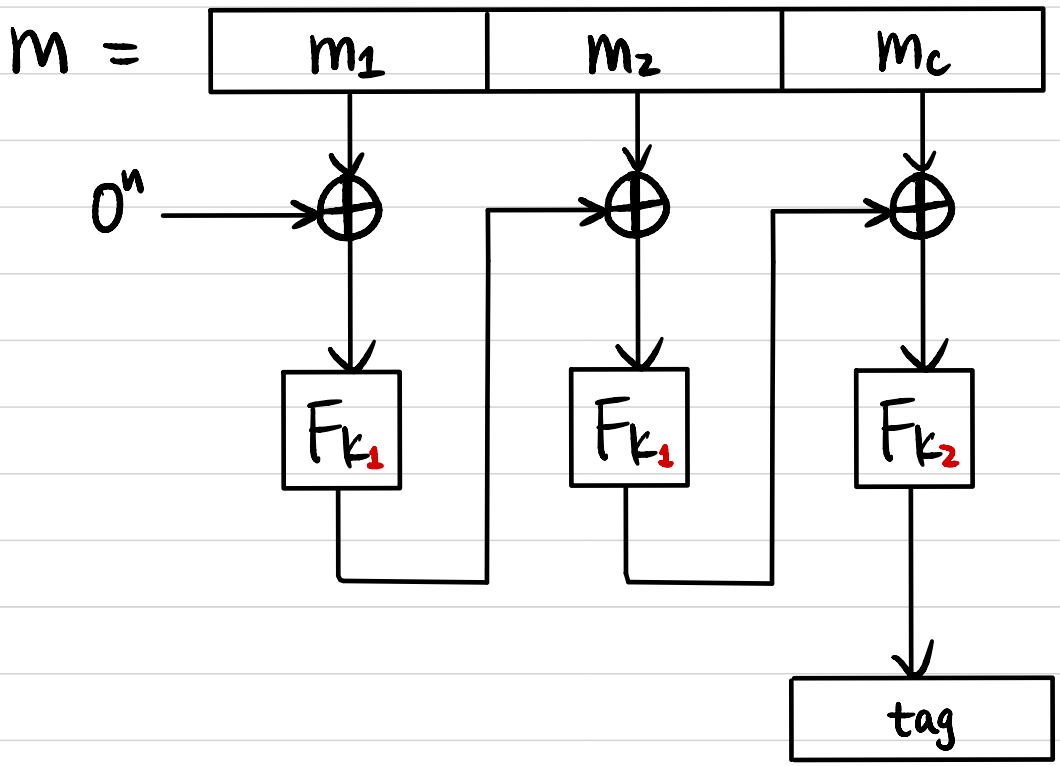


How to verify?

CMA (Chosen Message Attack) Secure?

- Fixed-length messages of length $l \cdot n$
- Arbitrary-length messages

Encrypt-last-block CBC-MAC (ECBC-MAC)



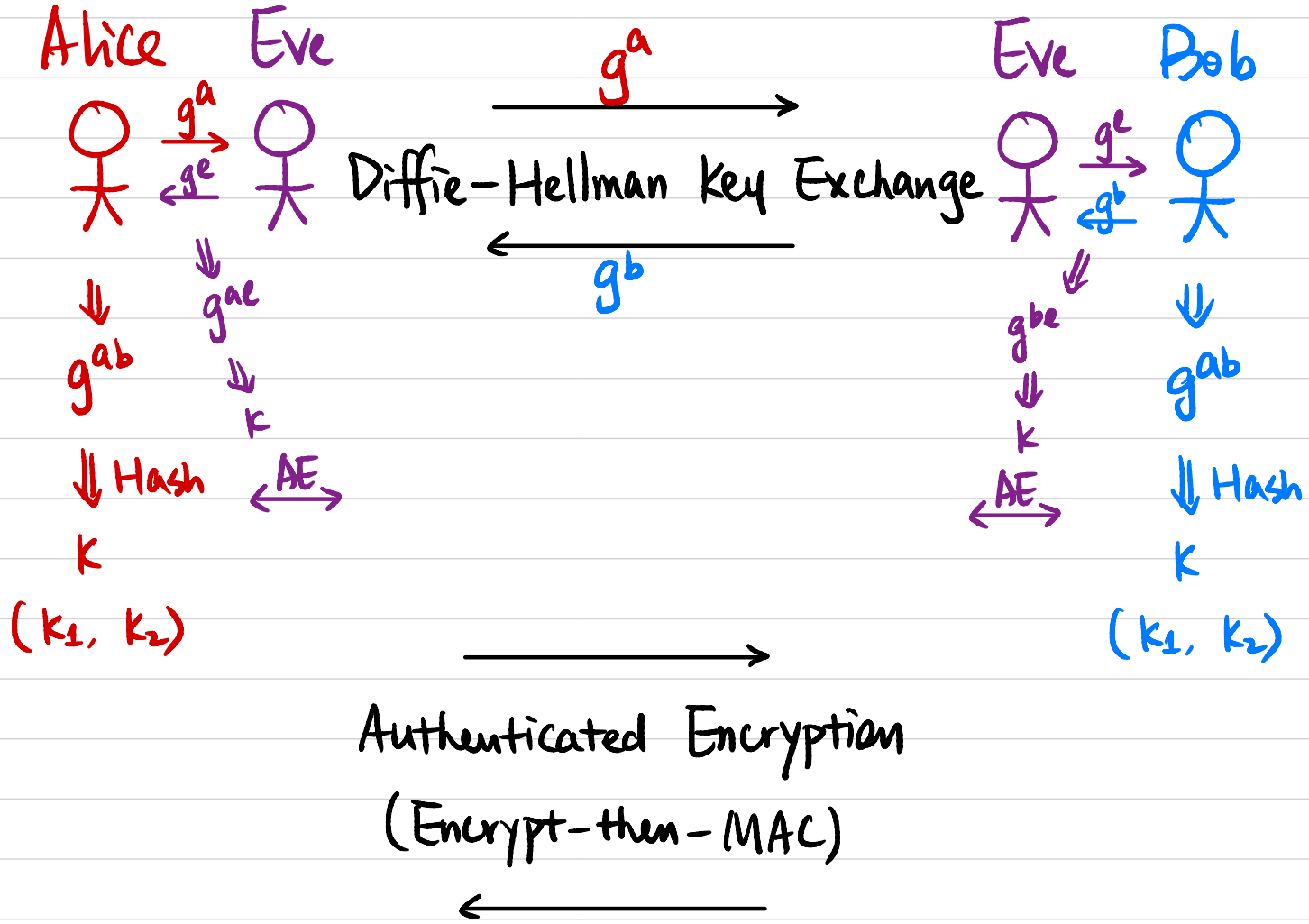
CMA (Chosen Message Attack) Secure?

- Arbitrary-length messages of length c·n
- Arbitrary-length messages

Summary

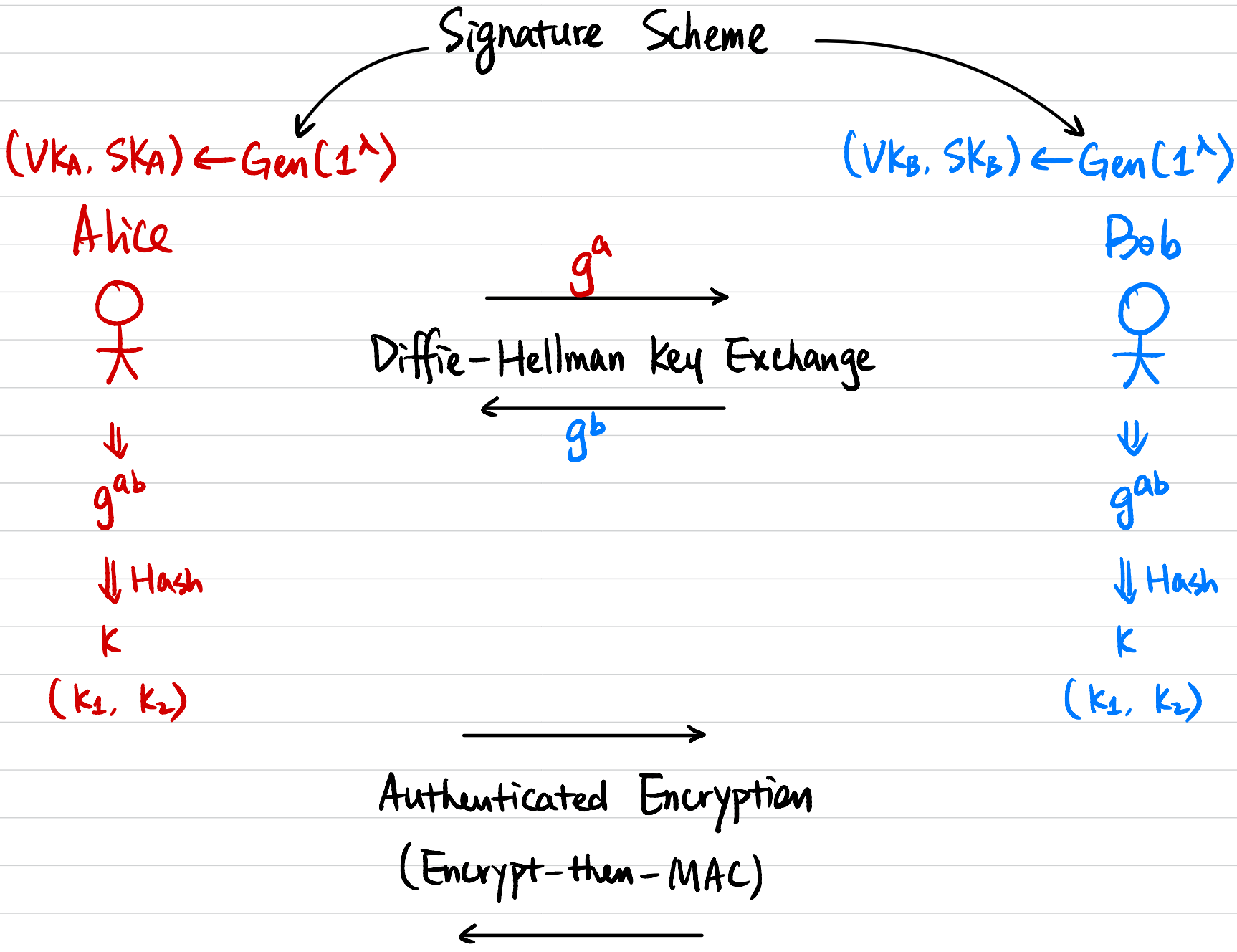
	Symmetric-Key	Public-Key
Message Secrecy	Primitive: SKE Construction: block Cipher	Primitive: PKE Constructions: RSA / ElGamal
Message Integrity	Primitive: MAC Constructions: CBC-MAC / HMAC	Primitive: Signature Constructions: RSA / DSA
Secrecy & Integrity	Primitive: AE Construction: Encrypt-then-MAC	
Key Exchange		Construction: Diffie-Hellman
Important Tool	Primitive: Hash function Construction: SHA	

Putting it Together: Secure Communication

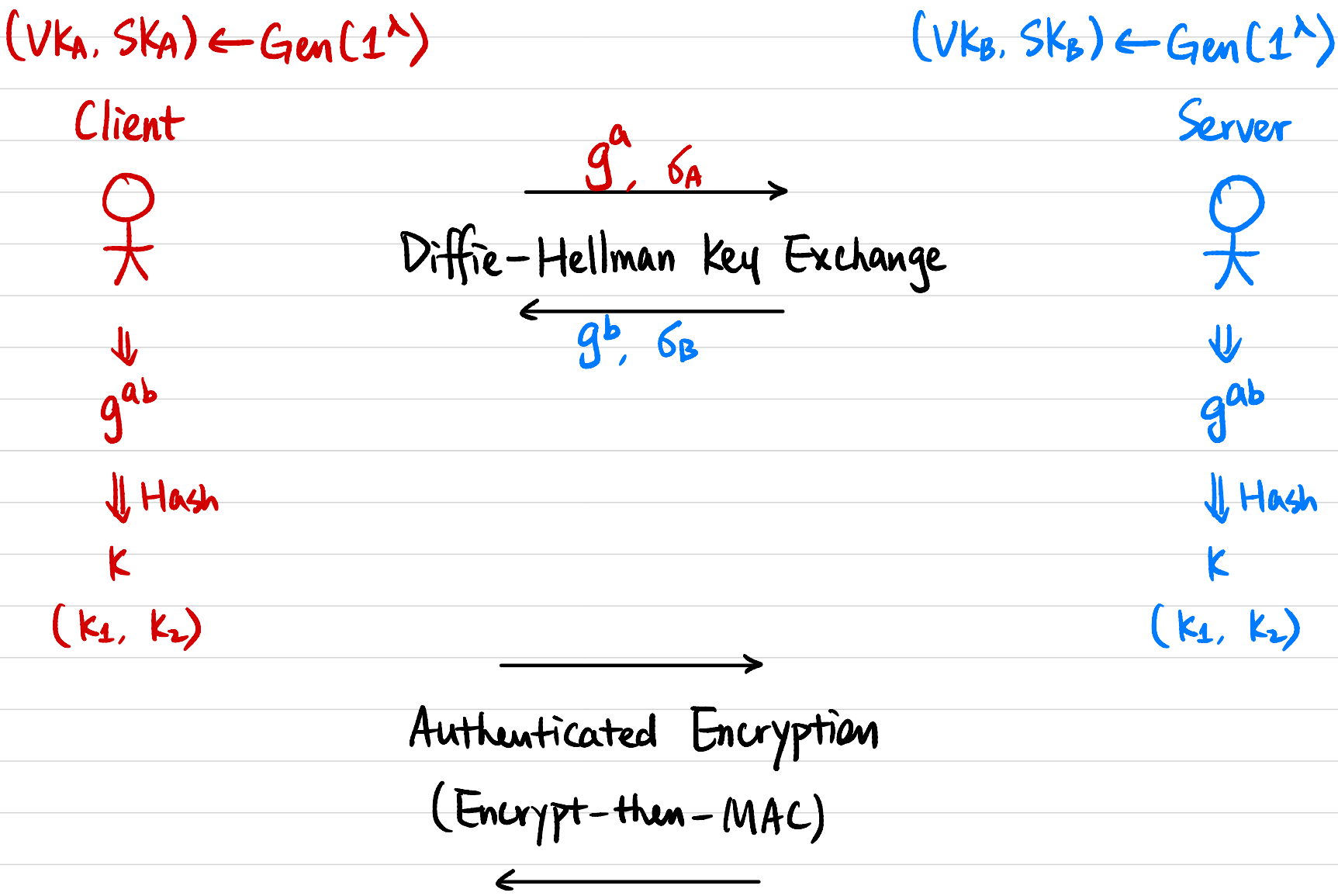


Any security issue?

"Man-in-the-Middle" Attack


Signature Scheme
$(VK_A, SK_A) \leftarrow Gen(1^\lambda)$
$(VK_B, SK_B) \leftarrow Gen(1^\lambda)$
Alice
Bob
g^a
Diffie-Hellman Key Exchange
g^b
g^{ab}
Hash
K
(k_1, k_2)
g^{ab}
Hash
K
(k_1, k_2)
Authenticated Encryption
(Encrypt-then-MAC)

Secure Shell Protocol (SSH)



OSX

Note: Most of these instructions will work on Linux as well.

Keypair Authentication Setup

- In a terminal, run the command `ssh-keygen -t rsa`
- Accept the default location for the key files.
- Enter a strong passphrase for your key. You will be prompted for it twice. **Do not** make this passphrase the same as your Brown password.
- Copy your public key to your desktop with the command `cp .ssh/id_rsa.pub Desktop`
- Upload your public key to your keys page. Wait a few minutes for the gateway to recognize your key.

Generating a new SSH key

You can generate a new SSH key on your local machine. After you generate the key, you can add the key to your account on GitHub.com to enable authentication for Git operations over SSH.

> **Note:** GitHub improved security by dropping older, insecure key types on March 15, 2022.
>
> As of that date, DSA keys (`ssh-dss`) are no longer supported. You cannot add new DSA keys to your personal account on GitHub.com.
>
> RSA keys (`ssh-rsa`) with a `valid_after` before November 2, 2021 may continue to use any signature algorithm. RSA keys generated after that date must use a SHA-2 signature algorithm. Some older clients may need to be upgraded in order to use SHA-2 signatures.

1. Open Terminal.
2. Paste the text below, substituting in your GitHub email address.

   ```
   $ ssh-keygen -t ed25519 -C "your_email@example.com"
   ```

   > **Note:** If you are using a legacy system that doesn't support the Ed25519 algorithm, use:
   >
   > ```
   > $ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
   > ```

   This creates a new SSH key, using the provided email as a label.

   ```
   > Generating public/private ALGORITHM key pair.
   ```

   When you're prompted to "Enter a file in which to save the key", you can press **Enter** to accept the default file location. Please note that if you created SSH keys previously, ssh-keygen may ask you to rewrite another key, in which case we recommend creating a custom-named SSH key. To do so, type the default file location and replace id_ssh_keyname with your custom key name.

   ```
   > Enter a file in which to save the key (/Users/YOU/.ssh/id_ALGORITHM: [Press ent
   ```

3. At the prompt, type a secure passphrase. For more information, see "Working with SSH key passphrases."

   ```
   > Enter passphrase (empty for no passphrase): [Type a passphrase]
   > Enter same passphrase again: [Type passphrase again]
   ```

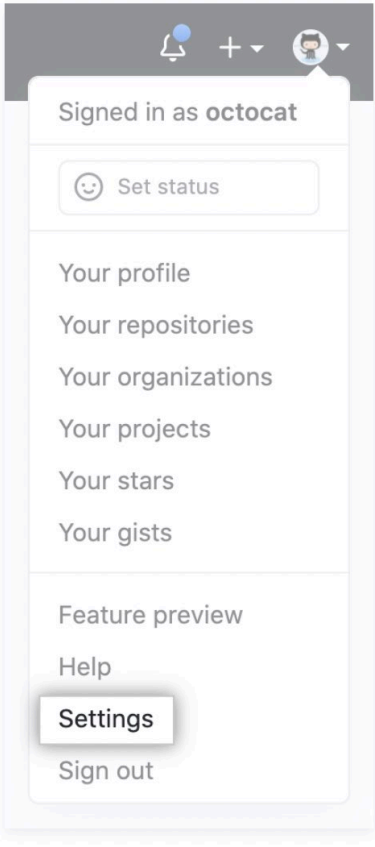
1. Copy the SSH public key to your clipboard.

   If your SSH public key file has a different name than the example code, modify the filename to match your current setup. When copying your key, don't add any newlines or whitespace.

   ```
   $ pbcopy < ~/.ssh/id_ed25519.pub
     # Copies the contents of the id_ed25519.pub file to your clipboard
   ```

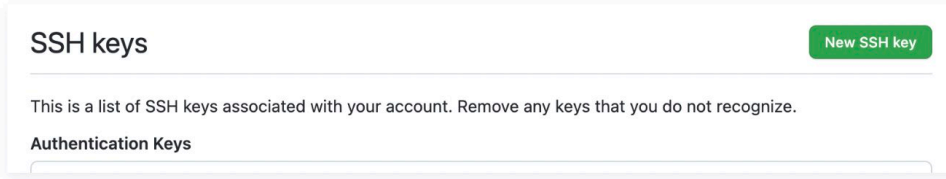
   Tip: If `pbcopy` isn't working, you can locate the hidden `.ssh` folder, open the file in your favorite text editor, and copy it to your clipboard.

2. In the upper-right corner of any page, click your profile photo, then click **Settings**.



3. In the "Access" section of the sidebar, click **SSH and GPG keys**.

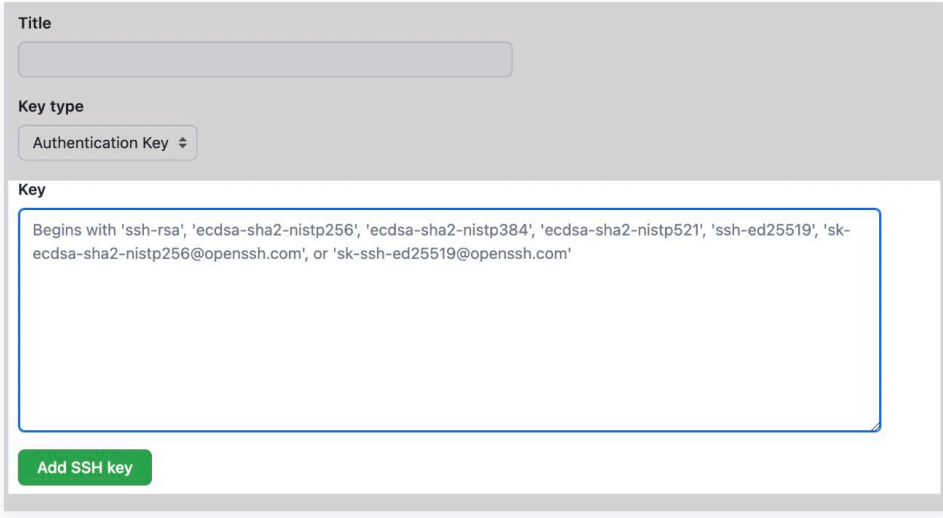
4. Click **New SSH key** or **Add SSH key**.



5. In the "Title" field, add a descriptive label for the new key. For example, if you're using a personal laptop, you might call this key "Personal laptop".

6. Select the type of key, either authentication or signing. For more information about commit signing, see "About commit signature verification."

7. Paste your public key into the "Key" field.



8. Click **Add SSH key**.

9. If prompted, confirm access to your account on GitHub. For more information, see "Sudo mode."

Testing your SSH connection

After you've set up your SSH key and added it to your account on GitHub.com, you can test your connection.

Mac Windows Linux

Before testing your SSH connection, you should have:

- Checked for existing SSH keys
- Generated a new SSH key
- Added a new SSH key to your GitHub account

When you test your connection, you'll need to authenticate this action using your password, which is the SSH key passphrase you created earlier. For more information on working with SSH key passphrases, see "Working with SSH key passphrases".

1. Open Terminal.
2. Enter the following:

```
$ ssh -T git@github.com
# Attempts to ssh to GitHub
```

You may see a warning like this:

```
> The authenticity of host 'github.com (IP ADDRESS)' can't be established.
> RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
> Are you sure you want to continue connecting (yes/no)?
```

3. Verify that the fingerprint in the message you see matches GitHub's public key fingerprint. If it does, then type `yes`:

```
> Hi USERNAME! You've successfully authenticated, but GitHub does not
> provide shell access.
```

> **Note:** The remote command should exit with code 1.

4. Verify that the resulting message contains your username. If you receive a "permission denied" message, see "Error: Permission denied (publickey)".

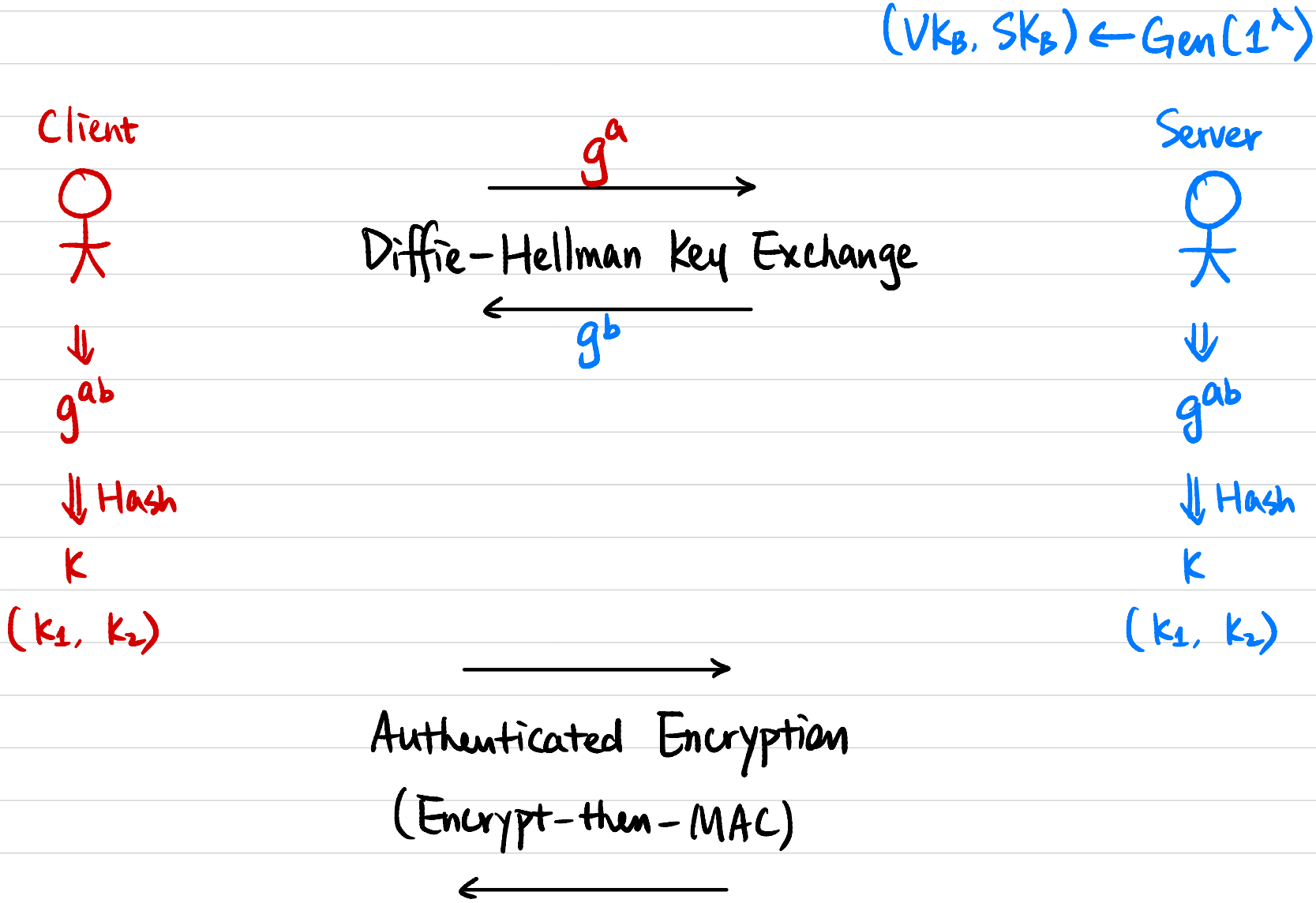
SSH regular use

- In a Terminal, run the command

  ```
  ssh username@ssh.cs.brown.edu
  ```

  (where *username* is your CS login).
- You will receive a prompt that asks if you you're sure you want to connect (yes/no). Type yes and hit enter.
- Enter the passphrase you chose during setup.
- If this is your first time logging in, or you recently changed your University password, you will additionally be asked for your University password and Duo 2-factor auth. Subsequent logins will skip this step.
- You are now connected to a CS department computer.
- If you want to connect to a specific computer, for examble cslab6a, change the command to

  ```
  ssh -t username@ssh.cs.brown.edu host=cslab6a
  ```


$(VK_B, SK_B) \leftarrow Gen(1^\lambda)$
Client
Server
g^a
Diffie-Hellman Key Exchange
g^b
g^{ab}
Hash
K
(k_1, k_2)
g^{ab}
Hash
K
(k_1, k_2)
Authenticated Encryption
(Encrypt-then-MAC)

Public Key Infrastructure (PKI)

$(VK, SK) \leftarrow Gen(1^\lambda)$

Certificate Authority (CA)

$(VK_B, SK_B) \leftarrow Gen(1^\lambda)$

Bob

Cerificate signing request (CSR) →

"bob.com"

← $(bob.com, VK_B), \sigma$

"Certificate"

Standard: X.509 certificate

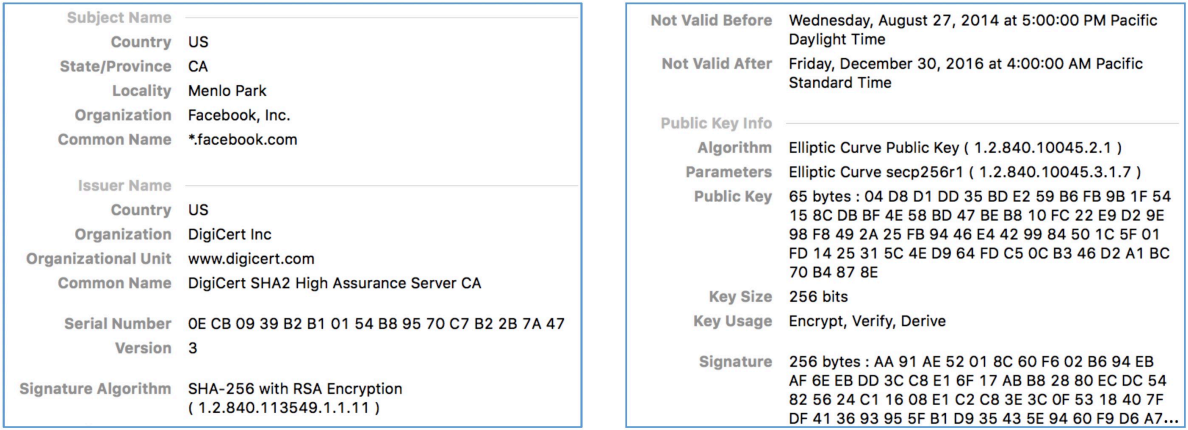
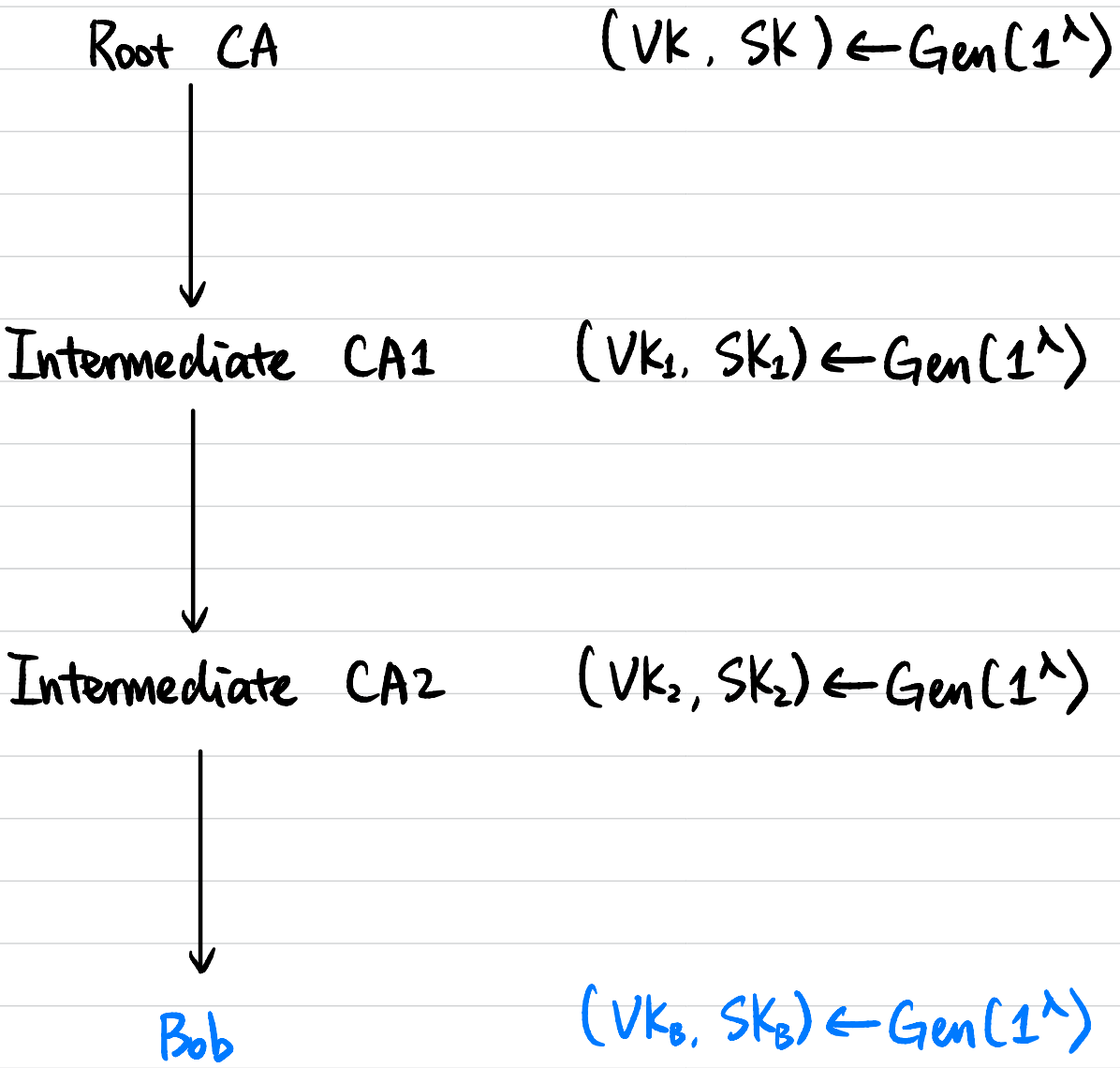


Figure 13.4: An example X.509 certificate

Certificate Chain



Certificate Revocation

- Short-lived certificates
- Certificate revocation lists (CRLs)