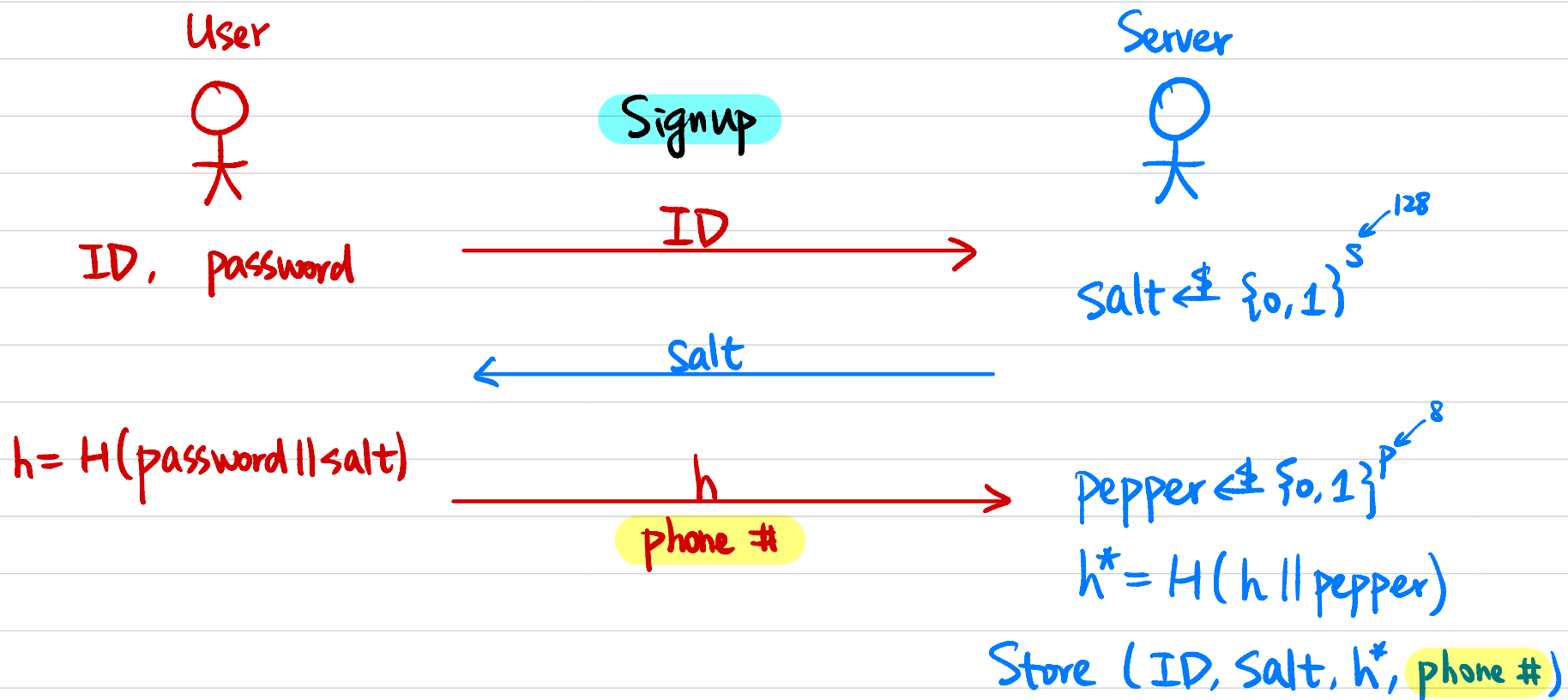


CSCI 1515 Applied Cryptography

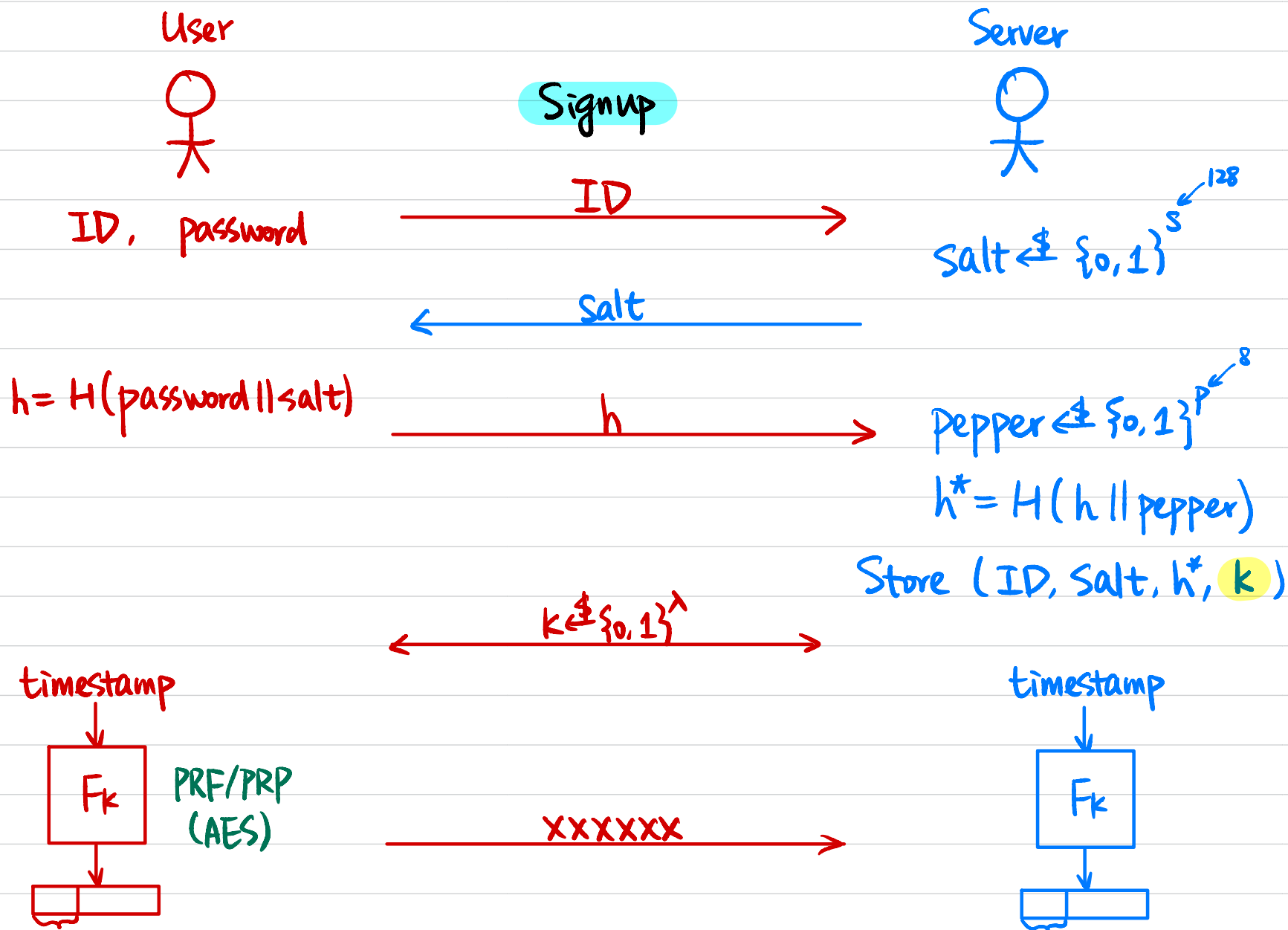
This Lecture:

- Two-Factor Authentication (continued)
- Putting it All Together: Secure Authentication
- Public Key Infrastructure (PKI)
- Case Study: Secure Shell Protocol (SSH)
 - Secure Messaging / Group Chats
 - Single Sign-On (SSO) Authentication

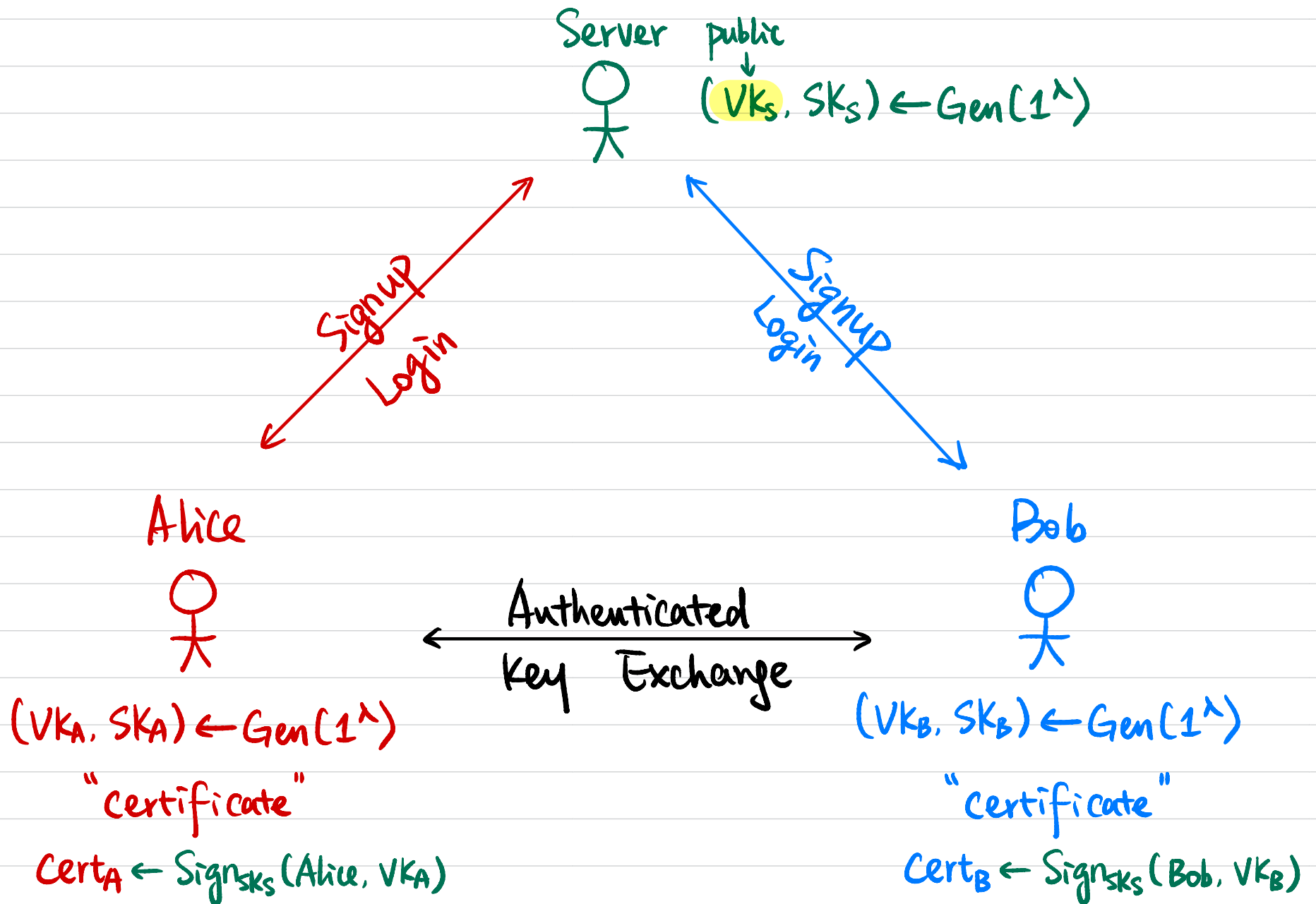
Two-Factor Authentication (2FA) ① SMS



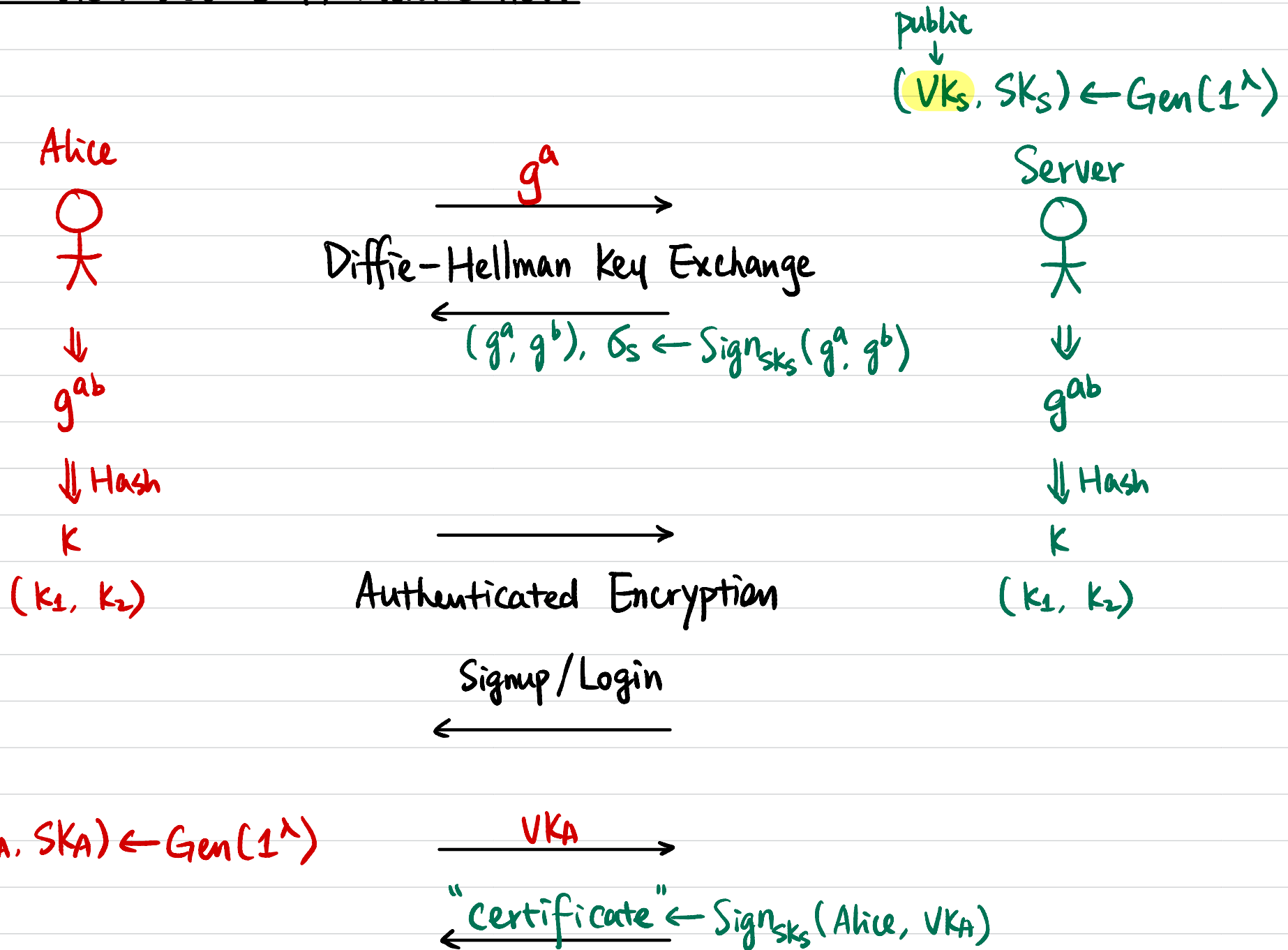
Two-Factor Authentication (2FA) ② app-generated code



Putting it All Together: Secure Authentication



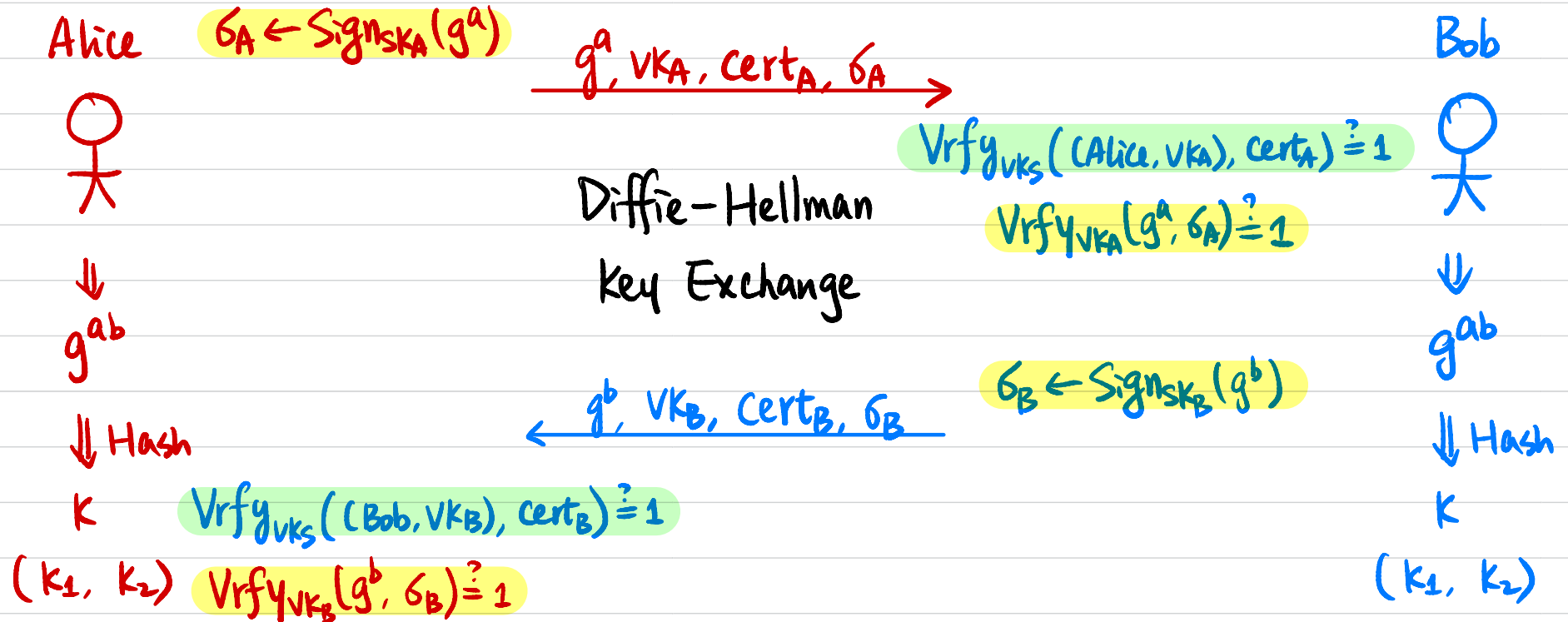
One-Sided Secure Authentication



Two-Sided Authenticated Key Exchange

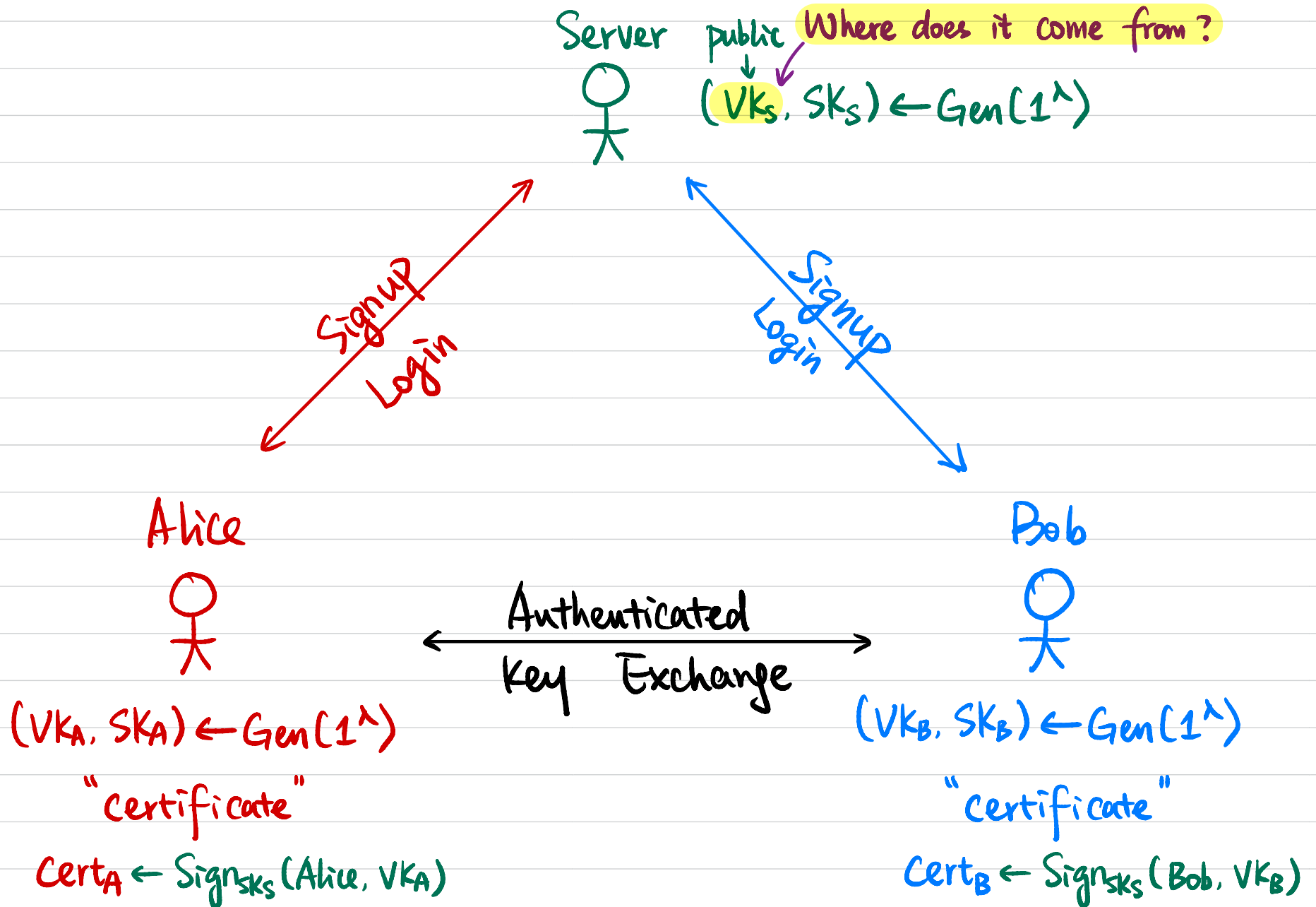
$(VK_A, SK_A); \text{cert}_A \leftarrow \text{Sign}_{SK_A}(\text{Alice}, VK_A)$

$(VK_B, SK_B); \text{cert}_B \leftarrow \text{Sign}_{SK_B}(\text{Bob}, VK_B)$



→
Authenticated Encryption
(Encrypt-then-MAC)
←

Putting it All Together: Secure Authentication



Public Key Infrastructure (PKI)

Certificate Authority

(CA)



public
 $(VK_s, SK_s) \leftarrow \text{Gen}(1^\lambda)$

Certificate Request

Certificate Request

Google



$(VK_A, SK_A) \leftarrow \text{Gen}(1^\lambda)$

$\text{Cert}_A \leftarrow \text{Sign}_{SK_s}(\text{Google}, VK_A)$

Amazon



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

$\text{Cert}_B \leftarrow \text{Sign}_{SK_s}(\text{Amazon}, VK_B)$

Public Key Infrastructure (PKI)

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

Bob



"bob.com"

Certificate signing request (CSR)

public

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

Certificate Authority
(CA)



$$\leftarrow (bob.com; VK_B), \sigma \leftarrow \text{Sign}_{SK}(bob.com, VK_B)$$

"Certificate"

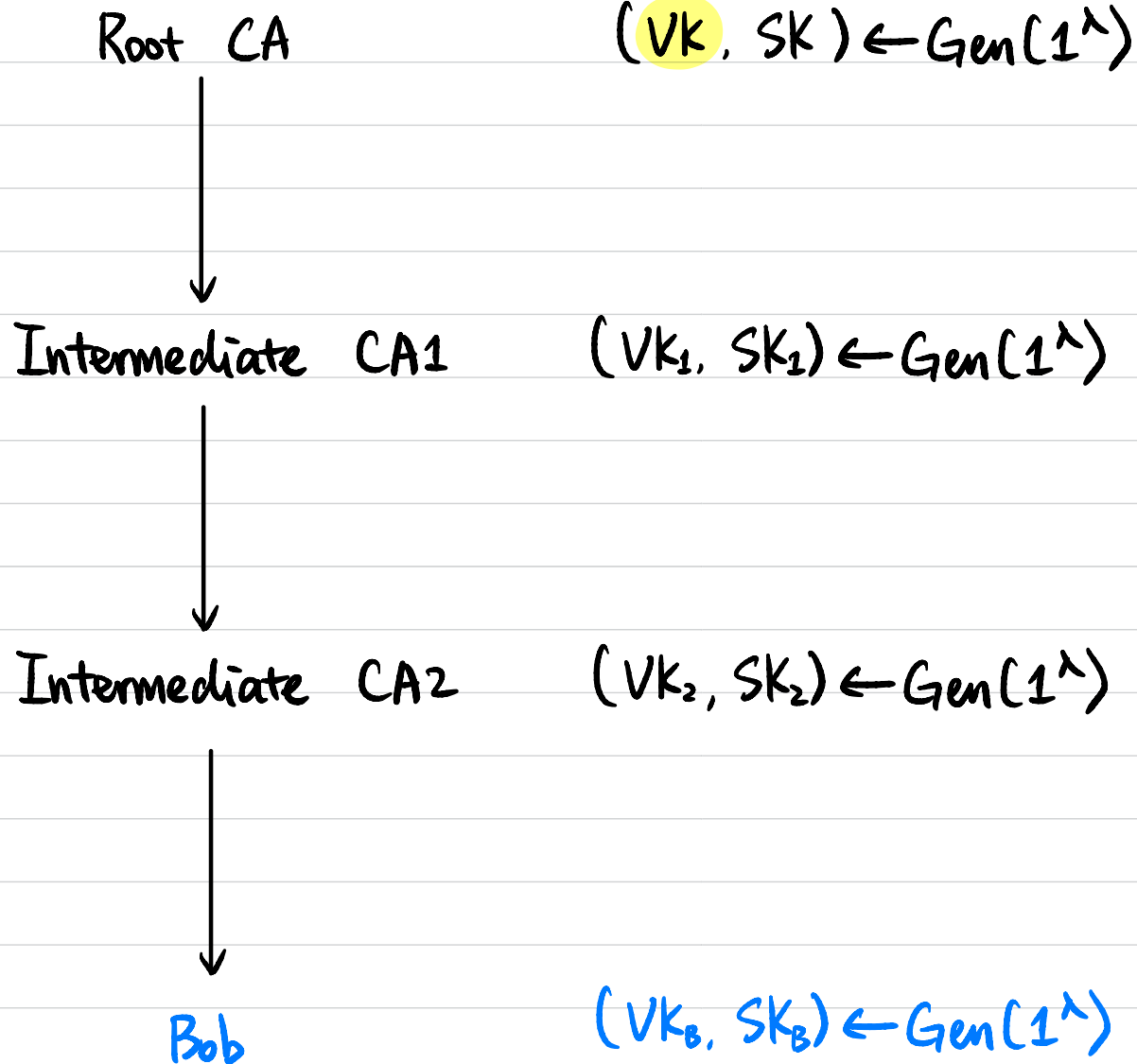
Standard: X.509 certificate

Subject Name	
Country	US
State/Province	CA
Locality	Menlo Park
Organization	Facebook, Inc.
Common Name	*.facebook.com
Issuer Name	
Country	US
Organization	DigiCert Inc
Organizational Unit	www.digicert.com
Common Name	DigiCert SHA2 High Assurance Server CA
Serial Number	0E CB 09 39 B2 B1 01 54 B8 95 70 C7 B2 2B 7A 47
Version	3
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)

Not Valid Before	Wednesday, August 27, 2014 at 5:00:00 PM Pacific Daylight Time
Not Valid After	Friday, December 30, 2016 at 4:00:00 AM Pacific Standard Time
Public Key Info	
Algorithm	Elliptic Curve Public Key (1.2.840.10045.2.1)
Parameters	Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)
Public Key	65 bytes : 04 D8 D1 DD 35 BD E2 59 B6 FB 9B 1F 54 15 8C DB BF 4E 58 BD 47 BE B8 10 FC 22 E9 D2 9E 98 F8 49 2A 25 FB 94 46 E4 42 99 84 50 1C 5F 01 FD 14 25 31 5C 4E D9 64 FD C5 0C B3 46 D2 A1 BC 70 B4 87 8E
Key Size	256 bits
Key Usage	Encrypt, Verify, Derive
Signature	256 bytes : AA 91 AE 52 01 8C 60 F6 02 B6 94 EB AF 6E EB DD 3C C8 E1 6F 17 AB B8 28 80 EC DC 54 82 56 24 C1 16 08 E1 C2 C8 3E 3C 0F 53 18 40 7F DF 41 36 93 95 5F B1 D9 35 43 5E 94 60 F9 D6 A7...

Figure 13.4: An example X.509 certificate

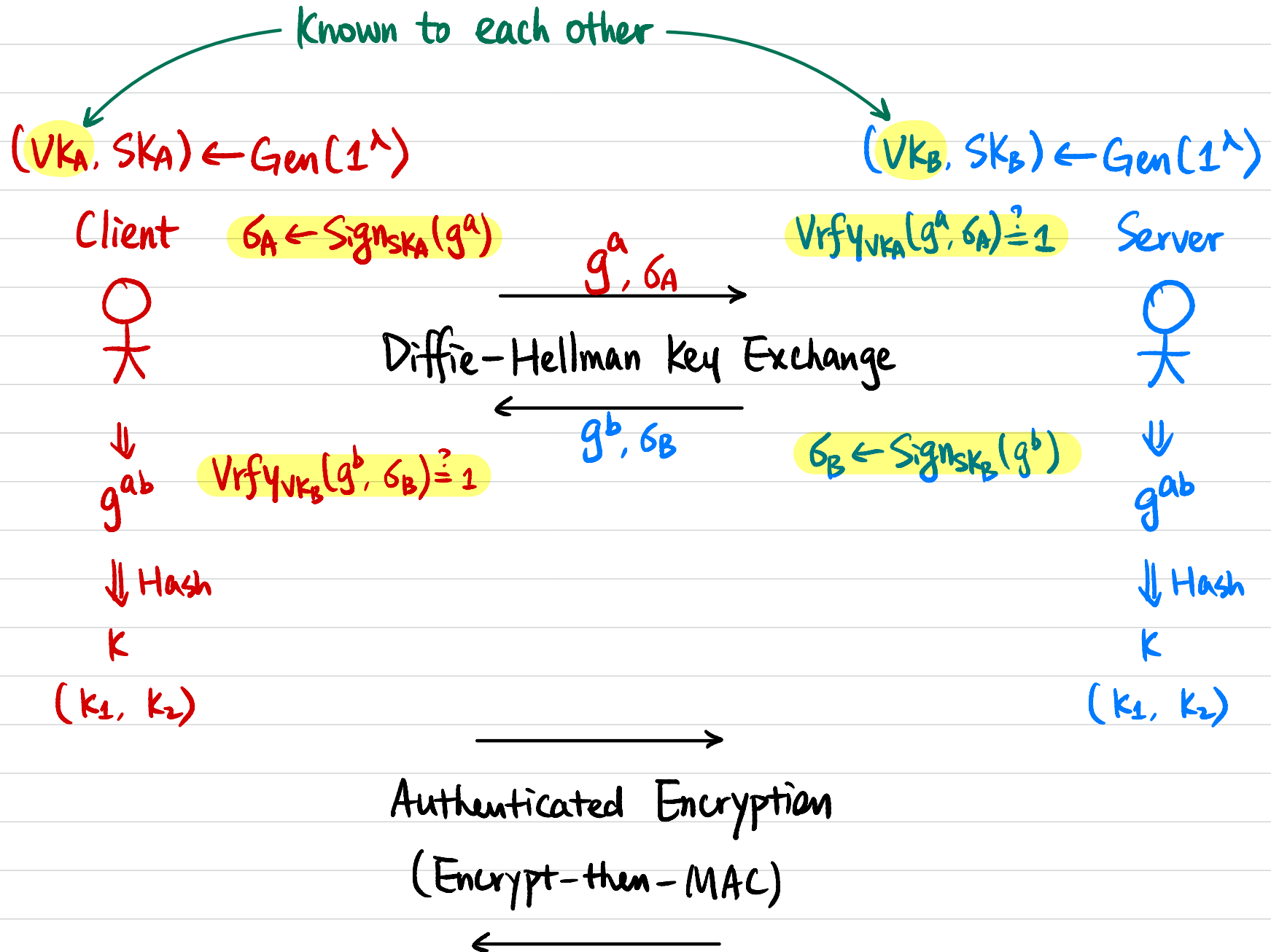
Certificate Chain



Certificate Revocation

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

Case Study: Secure Shell Protocol (SSH)



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 enter])
```

- 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]
```

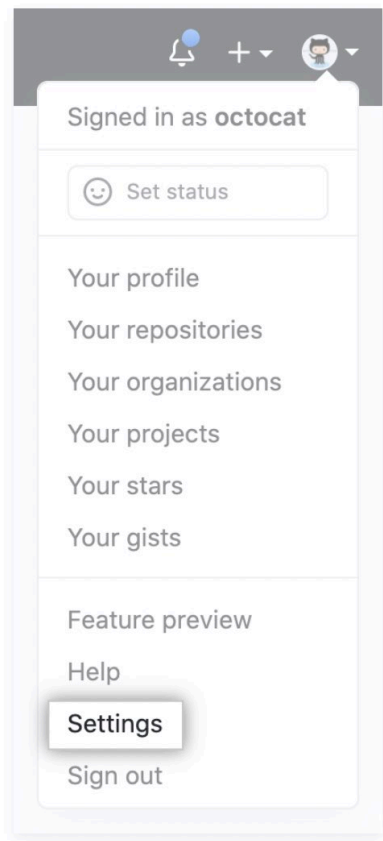
1 Copy the SSH public key to your clipboard.

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
```

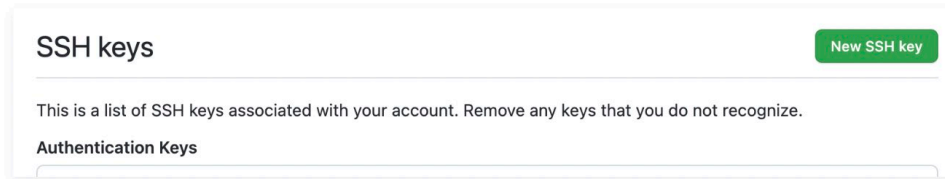
Tip: If `pbcopy` isn't working, you can locate the hidden `.ssh` folder, open the file in your favorite text editor, and copy it to your clipboard.

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



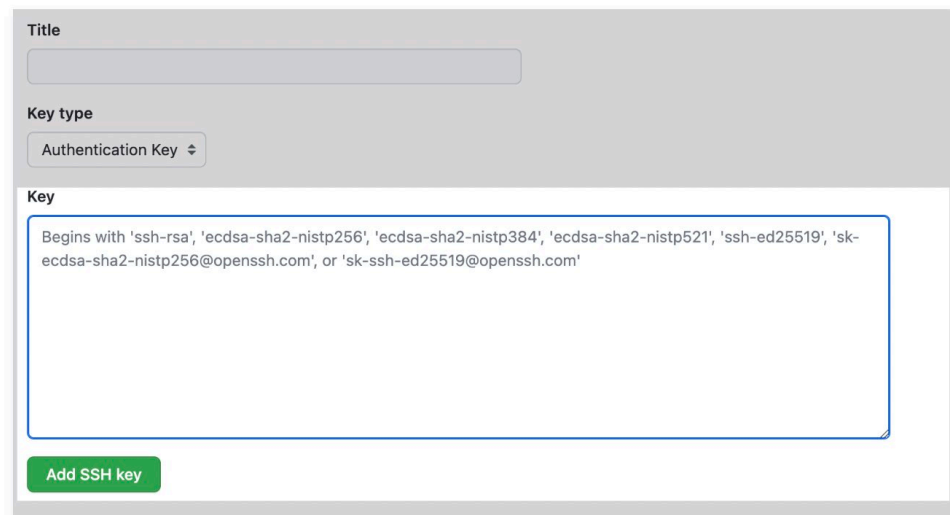
3 In the "Access" section of the sidebar, click  **SSH and GPG keys**.

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



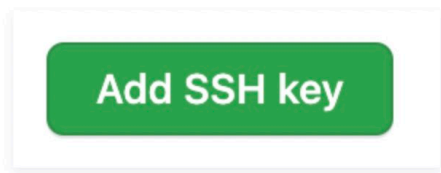
The screenshot shows the 'SSH keys' section of a GitHub account. At the top right is a green button labeled 'New SSH key'. Below the header, a message states: 'This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.' Underneath, there is a section titled 'Authentication Keys' with a scrollable list area below it.

- 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.



The screenshot shows the 'Add SSH key' form. It has three main sections: 'Title' with a text input field; 'Key type' with a dropdown menu currently set to 'Authentication Key'; and 'Key' with a large text area. The 'Key' field contains placeholder text: 'Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com''. At the bottom left of the form is a green button labeled 'Add SSH key'.

- 8 Click **Add SSH key**.



A close-up of the green 'Add SSH key' button, which has white text and rounded corners.

- 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:nThbg6kXUpJWGl7E1IG0CspRomTxdCARLviKw6E5SY8.
> Are you sure you want to continue connecting (yes/no)?
```

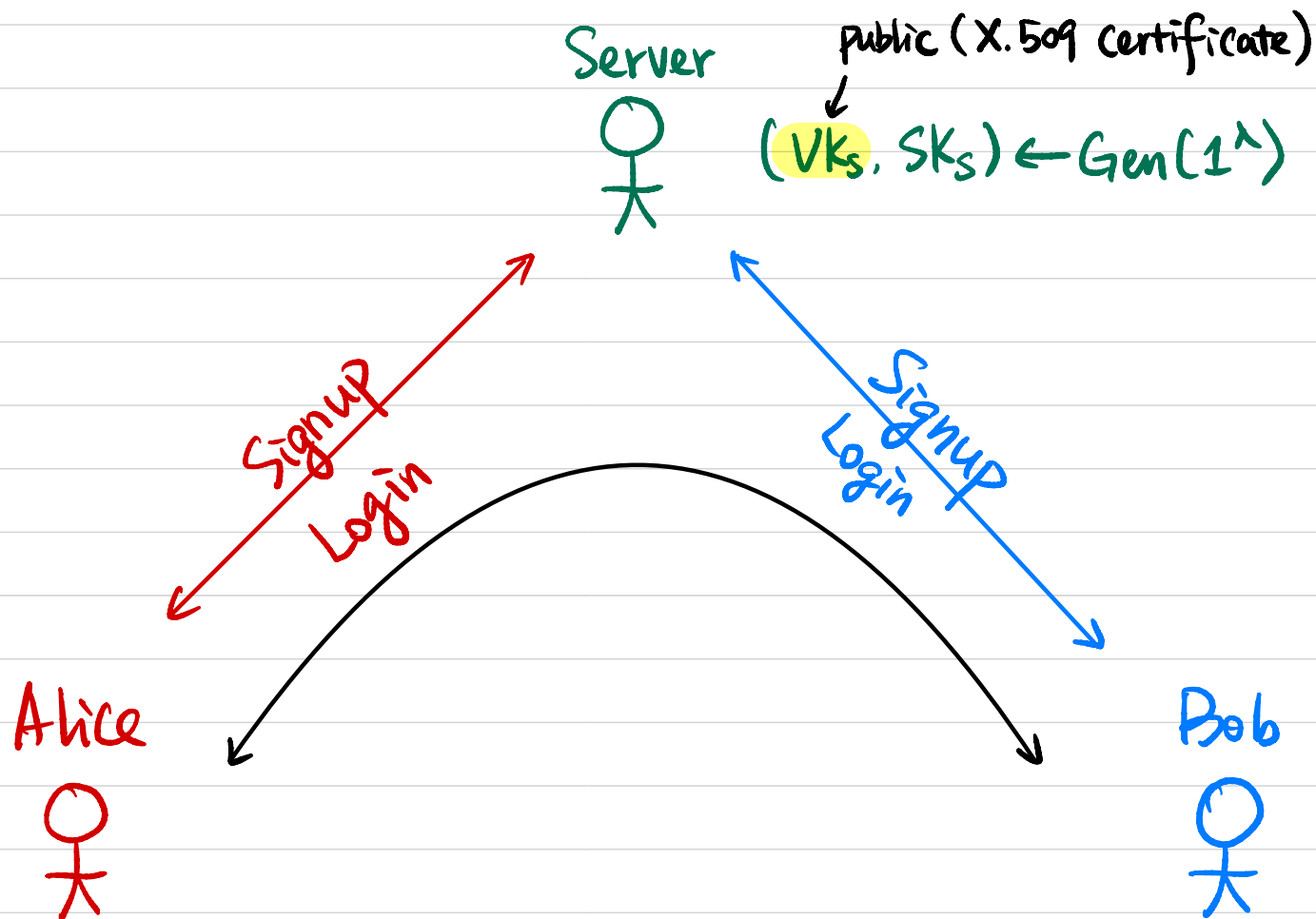
- 3 Verify that the fingerprint in the message you see matches [GitHub's public key fingerprint](#). If it does, then type `yes` :

```
> 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 \(publickey\)](#)".

Case Study: Secure Messaging



How would you design it?

Group Chat ?

Server
public (X.509 certificate)
 $(VK_s, SK_s) \leftarrow \text{Gen}(1^\lambda)$

Signup
Login

Alice

Signup
Login

Charlie

Signup
Login

Bob

- m revealed to server?
- group structure revealed to server?
- all same key / pairwise keys?

How would you design it ?

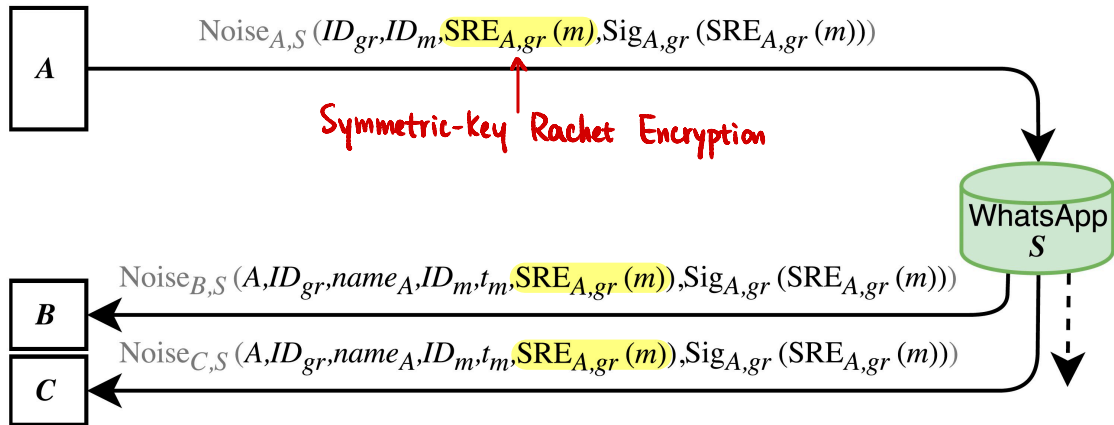


Figure 5. Schematic depiction of traffic, generated for a message m from sender A to receivers B, C in group gr with $\mathcal{G}_{gr} = \{A, B, C\}$ in WhatsApp.

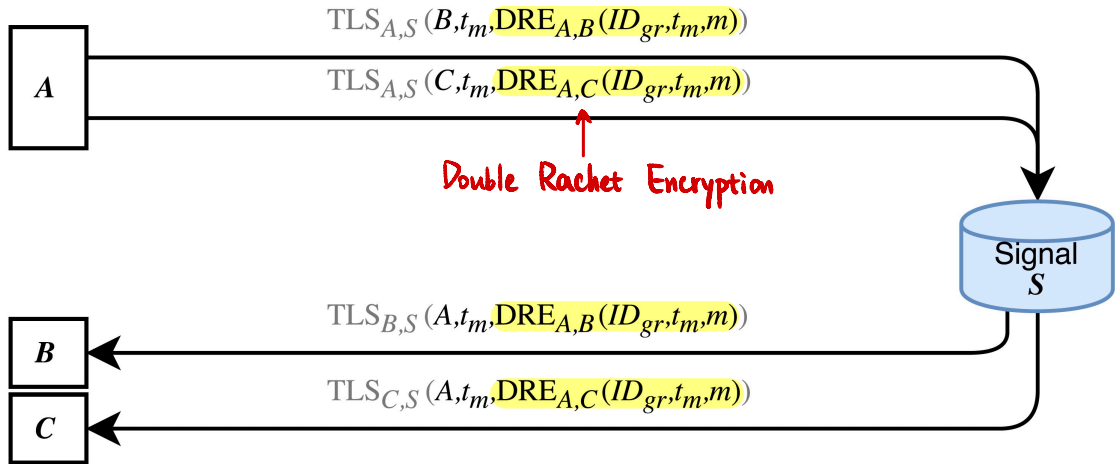


Figure 3. Schematic depiction of Signal's traffic, generated for a message m from sender A to receivers B and C in group gr with $\mathcal{G}_{gr} = \{A, B, C\}$. Transport layer protection is not in the analysis scope (gray).

Case Study: Single Sign-On (SSO) Authentication

User



← Password-Based Authentication →

Request "token" →

← "token"
(Signature / MAC)

→ "token"

Authentication
Server



k

MAC

Service Provider

k

sk

Signature

Service Provider



vk

- OAuth / OpenID: Sign-in with Google / Apple / Brown / ...
- Kerberos: enterprises