

# Secured Socket Layer

Assoc. Prof. Trần Minh Triết  
PhD. Trương Toàn Thịnh



[fit@hcmus](mailto:fit@hcmus)

KHOA CÔNG NGHỆ THÔNG TIN  
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

# SSL / TLS in daily life?

Wells Fargo Account Summary - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Favorites Print Home

Address [https://online.wellsfargo.com/mn1\\_aa1\\_on/cgi-bin/session.cgi?sessargs=coAn76axS2xltPX8uoCT8rRBfMMdJldx](https://online.wellsfargo.com/mn1_aa1_on/cgi-bin/session.cgi?sessargs=coAn76axS2xltPX8uoCT8rRBfMMdJldx) Go Links Yahoo maps Mapblast Dictionary

Home | Help Center | Contact Us | Locations | Site Map | Apply | **Sign Off**

**WELLS FARGO**

**Account Summary** Last Log On: January 06, 2004

> Account Summary  
Brokerage  
Bill Pay  
Transfer  
Account Services  
My Message Center

Stay organized with FREE 24/7 access to Online Statements. Sign up today.

Sign up for the Wells Fargo Rewards® program and get 2,500 points. Learn More.

Wells Fargo Accounts OneLook Accounts

**Tip:** Select an account's balance to access the Account History.  
**NEW** [Enroll for Online Statements](#) [My Message Center](#)

**Cash Accounts**

Account	Account Number	Available Balance
Checking <a href="#">Add Bill Pay</a>		
<b>Total</b>		

To end your session, be sure to Sign Off.

Account Summary | Brokerage | Bill Pay | Transfer | My Message Center | Sign Off  
Home | Help Center | Contact Us | Locations | Site Map | Apply

© 1995 - 2003 Wells Fargo. All rights reserved.

Internet

# The evolution of the protocol

- SSL 1.0
  - Internal research in Netscape (~early 1994?)
  - “Lost in the mists of time”
  - Main author: Taher Elgamal
- SSL 2.0
  - Netscape published on 11/1994
  - There are still a few problems
  - Expired on 2011 (RFC 6176)
- SSL 3.0
  - Published in RFC 6101
  - Netscape and Paul Kocher designed (11/1996)
  - Expired on 6/2015 (RFC 7568)

# The evolution of the protocol

- TLS 1.0 và TLS 1.1
  - Internet standards based on SSL 3.0 (01/1999)
  - Incompatible works with SSL 3.0
  - Expired on ~ 3/2020
  - TLS 1.1 updated from TLS 1.0
  - TLS 1.0 published in RFC 2246 and TLS 1.1 in RFC 4346
- TLS 1.2
  - Updated from TLS 1.1
  - Currently in use
  - Published in RFC 5246
- TLS 1.3
  - Updated from TLS 1.2
  - Used by some libraries, browsers and software
  - Published in RFC 8446

## Some limitations in SSL 2.0

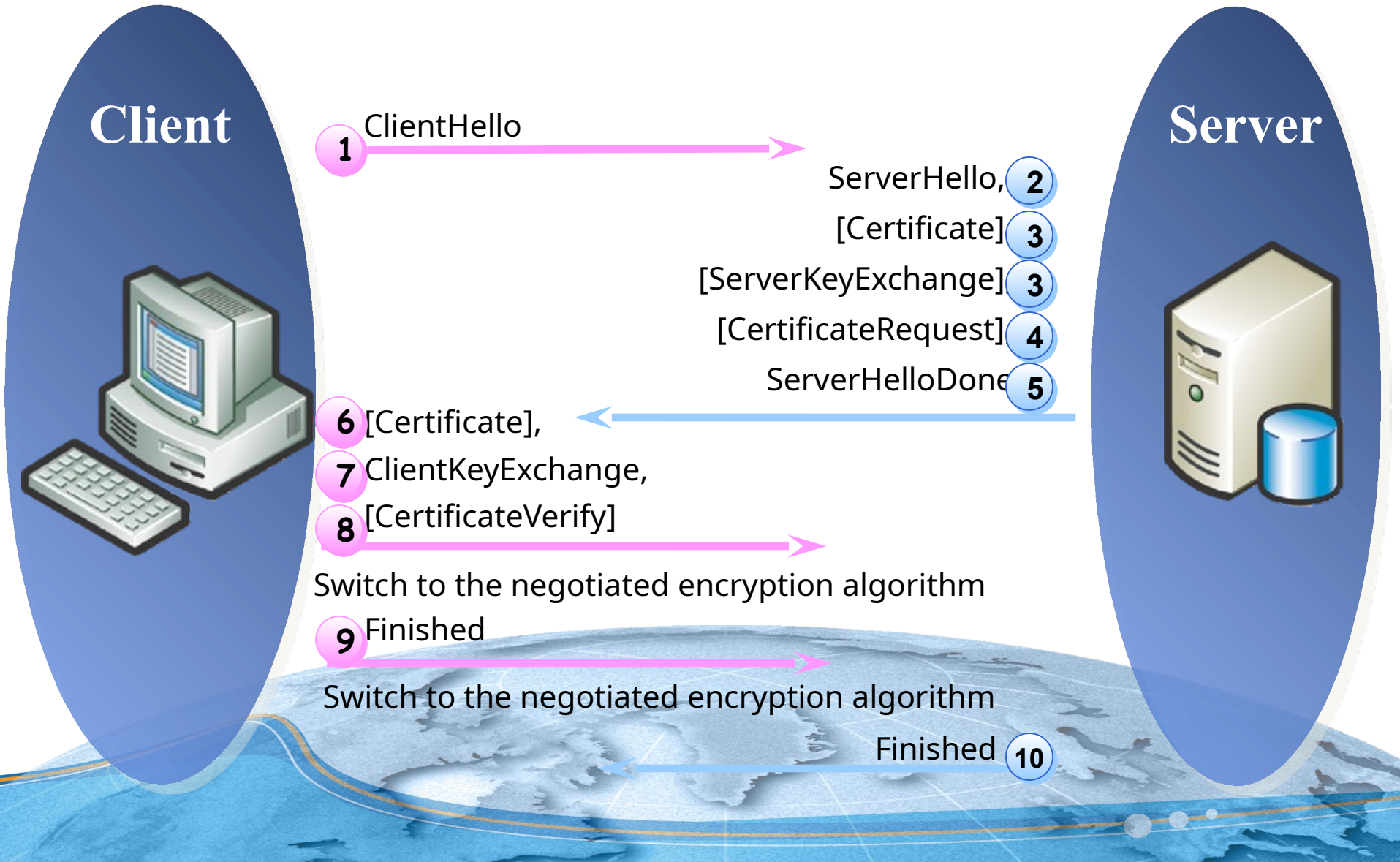
- Key length is too short
  - In weakened modes before being released to the public, SSL 2.0 has narrowed the key length for authentication to 40 bits.
- Generate weak MAC
- Vulnerable to integrity attack
  - SSL 2.0 adds bytes (padding) to the MAC in block cipher modes, but has no control over padding length validation. This makes it possible for an attacker to delete some bytes at the end of the message.
- Ciphersuite rollback attack
  - An attacker can modify the “ClientHello” message to “trick” the server into choosing an old version or a weak algorithm

# Basics of TLS (TLS Handshake Protocol)

- TLS consists of two protocols:
  - **Handshake protocol:** use asymmetric encryption to establish a shared secret key between client and server
    - Include Client and Server
    - Negotiate protocol version and the set of cryptographic algorithms to be used
      - Compatibility & interoperability between different implementations of the general protocol
    - Authenticate client and server (optional)
      - Use an e-certificate to know the partner's public key and confirm each other's identity
    - Use public key to establish public-secret
  - **Record protocol:** Use the secret key established in the handshake protocol to protect communication between client and server



# Structure of Handshake protocol (in general)



# ClientHello

## Client



1 ClientHello



## Server



Client reports (data NOT encrypted):

- Version of in-use protocol
- Supported encryption algorithms



# ClientHello

- 4 bytes timestamp, 28 bytes random value
- session\_id:
  - = 0 for new connection of existing session
  - = 0 for new connection of new session
- client\_version: maximum version value
- cipher\_suites: Ordered list of supported client algorithms
- compression\_methods: an ordered list of compression algorithms that the client supports

```
struct {  
    Timestamp          timestamp;  
    ProtocolVersion    client_version;  
    Random             random;  
    SessionID          session_id;  
    CipherSuite         cipher_suites;  
    CompressionMethod  compression_methods;  
} ClientHello
```

Highest version of the protocol supported by the client

Session id (if the client wants to resume an old session)

Client-supported cryptographic algorithms (e.g., RSA or Diffie-Hellman)

# ServerHello

## Client



1  $C, \text{Version}_c, \text{suite}_c, N_c$

ServerHello 2

Server responds (data NOT encrypted):

- Highest version supported by server and client
- The strongest set of cryptographic algorithms supported by the client

## Server



- The structure consists of
  - 32 bytes random value
  - session\_id: new value or old value reused
  - version:  $\min\{\text{version}_{\text{client supports}}, \text{version}_{\text{maximum server supports}}\}$
  - cipher\_suite list: list of selected algorithms (select only one algorithm in each category)
  - compression list: Select only from client-supported compression algorithms
  - Key exchange method
    - RSA: need to certify the recipient's public key
    - Fixed DH: Both parties must have a public-key certificate
    - Ephemeral DH: Both sides need the key to sign and certify the public-key
    - Anonymous DH: don't validate DH key, man-in-the-middle attack possible
    - Fortezza: rarely used

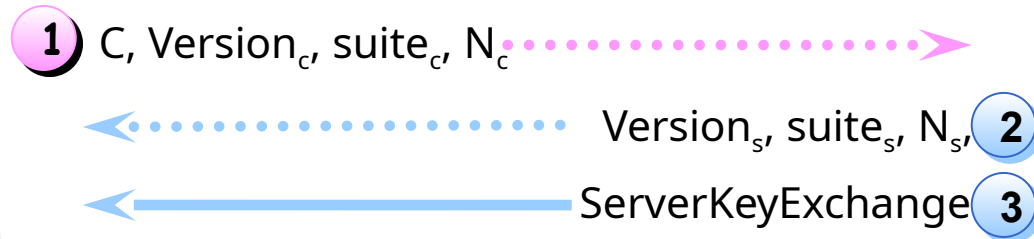
- CipherAlgorithm
  - RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza
- MACAlgorithm
  - MD5 or SHA-1
- CipherType
  - stream or block
- IsExportable: true or false
- HashSize
  - 0, 16 or 20 bytes
- Key Material: used to generate a write-key
- IV Size: size of IV in CBC

# ServerKeyExchange

## Client



## Server



Server sends its certificate of public-key (RSA or Diffie-Hellman, depending on the selected set of algorithms)



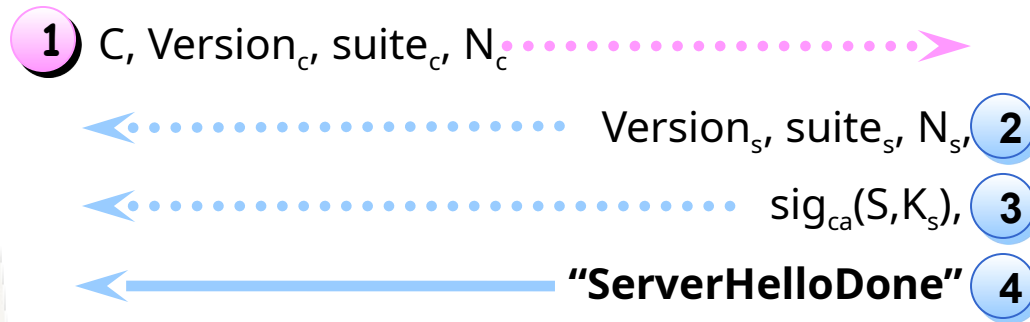
- ☐ No need for RSA and Fixed DH
- ☐ Needed with Anonymous DH, Ephemeral DH
- ☐ Use with RSA if the server only has a key for signing. The server then sends a temporary public key (RSA) to the client
- ☐ Message ServerKeyExchange:
  - ☐ Signed by server
  - ☐ Signature on hash value of:
    - ☐ ClientHello.random, ServerHello.random
    - ☐ Parameter of Server Key Exchange

# ClientKeyExchange

Client



Server



# ClientKeyExchange

## Client



## Server



- 1  $C, \text{Version}_c, \text{suite}_c, N_c$
- 2  $\text{Version}_s, \text{suite}_s, N_s$
- 3  $\text{sig}_{ca}(S, K_s)$
- 4 "ServerHelloDone"
- 5 **ClientKeyExchange**

Client generates a secret-key (encrypted with the server's public-key) and sends it to the server

# ClientKeyExchange

## □ Structure

```
struct {  
    select (KeyExchangeAlgorithm) {  
        case rsa: EncryptedPreMasterSecret;  
        case diffie_hellman: ClientDiffieHellmanPublic;  
    } exchange_keys  
} ClientKeyExchange
```

**RSA or Diffie-Hellman key  
exchange algorithm**

```
struct {  
    ProtocolVersion client_version;  
    opaque random[46];  
} PreMasterSecret
```

**Client-side protocol  
version**

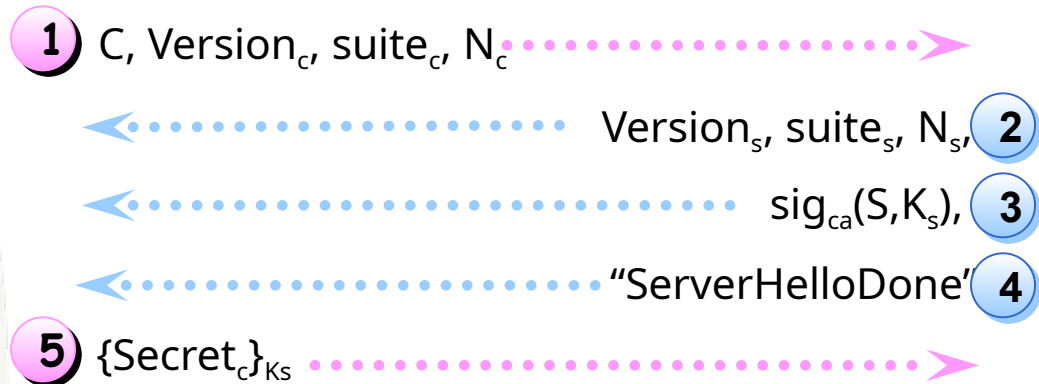


# “Core” SSL

Client



Server



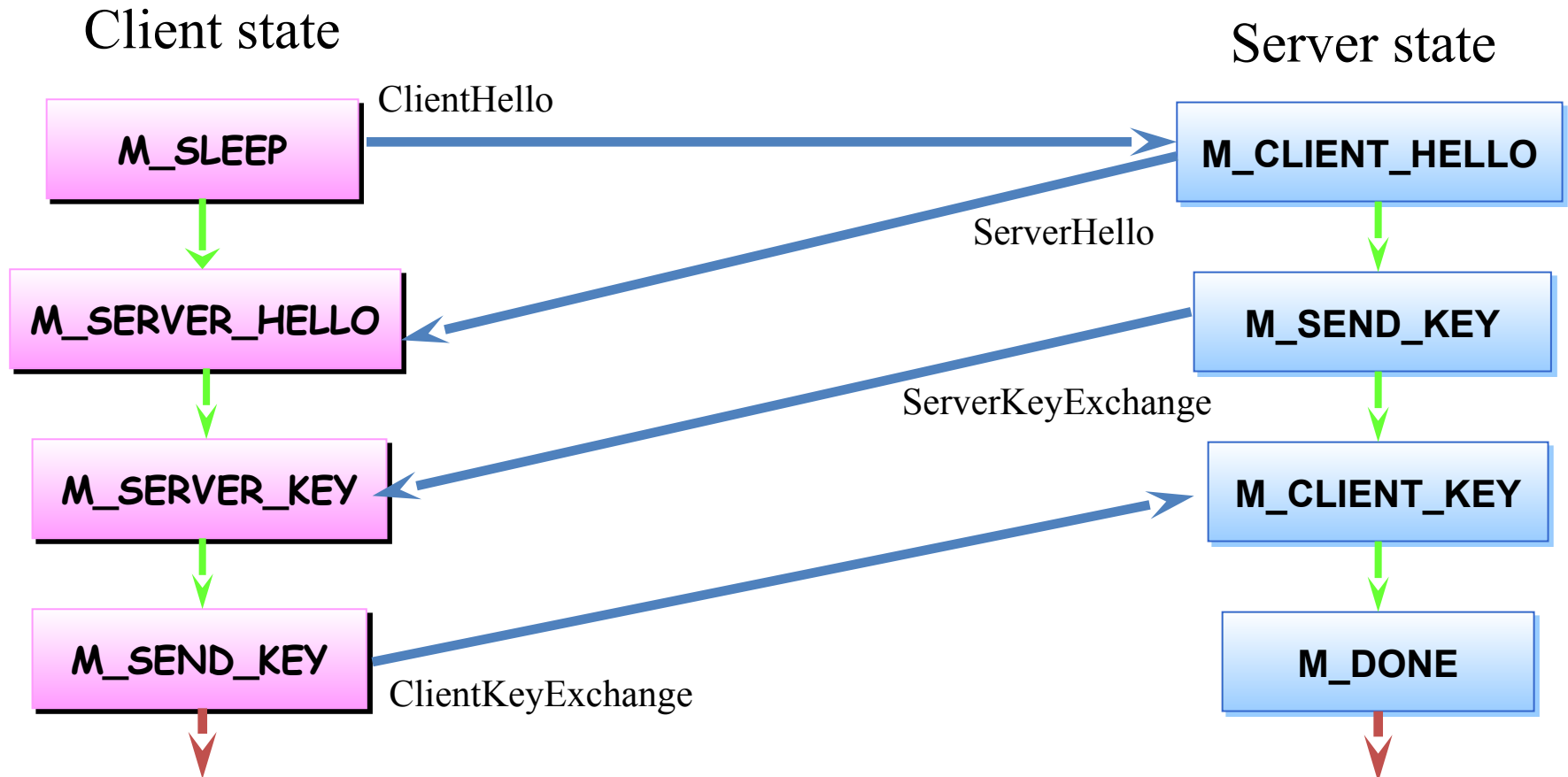
If the protocol is correct, from now on, C and S share a common secret key

Switch to using key  
deduced from secret<sub>c</sub>

Switch to using key  
deduced from secret<sub>c</sub>



# States of Client and Server

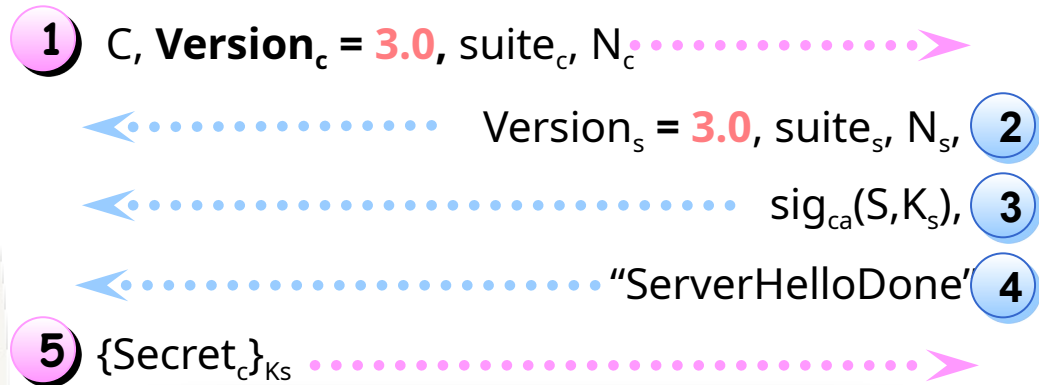


# Main part of SSL 3.0

Client



Server



If the protocol is correct, from now on,  
C and S share the common secret key

Switch to using key  
deduced from secret<sub>c</sub>

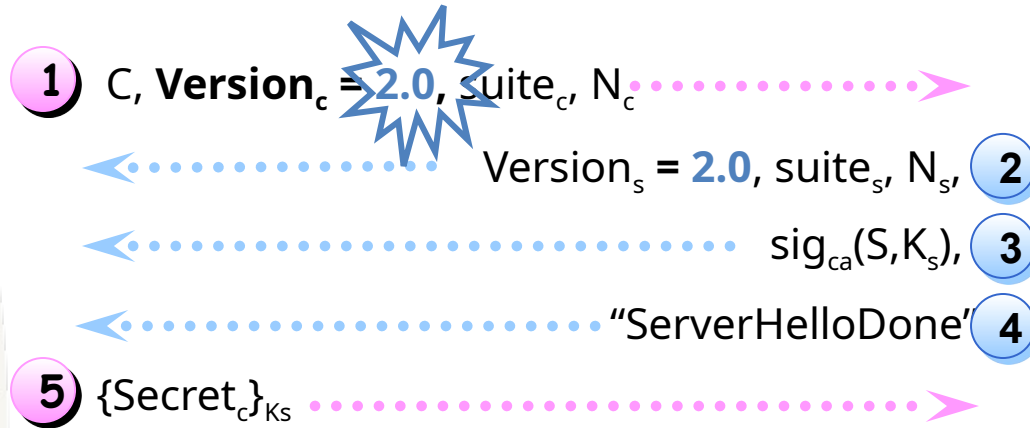
Switch to using key  
deduced from secret<sub>c</sub>

# Mis-choose old version!

Client



Server



The server was "tricked" and decided to establish communication between S and C using the old version

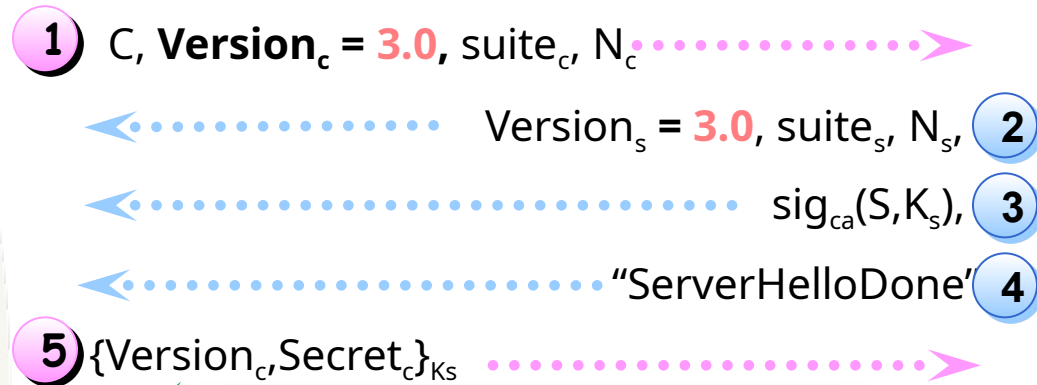


# Adjust SSL

Client



Server



If the protocol is correct, from now on,  
C and S share the common secret key

Prevent attack  
with old version

Added version check  
matches the information in ClientHello

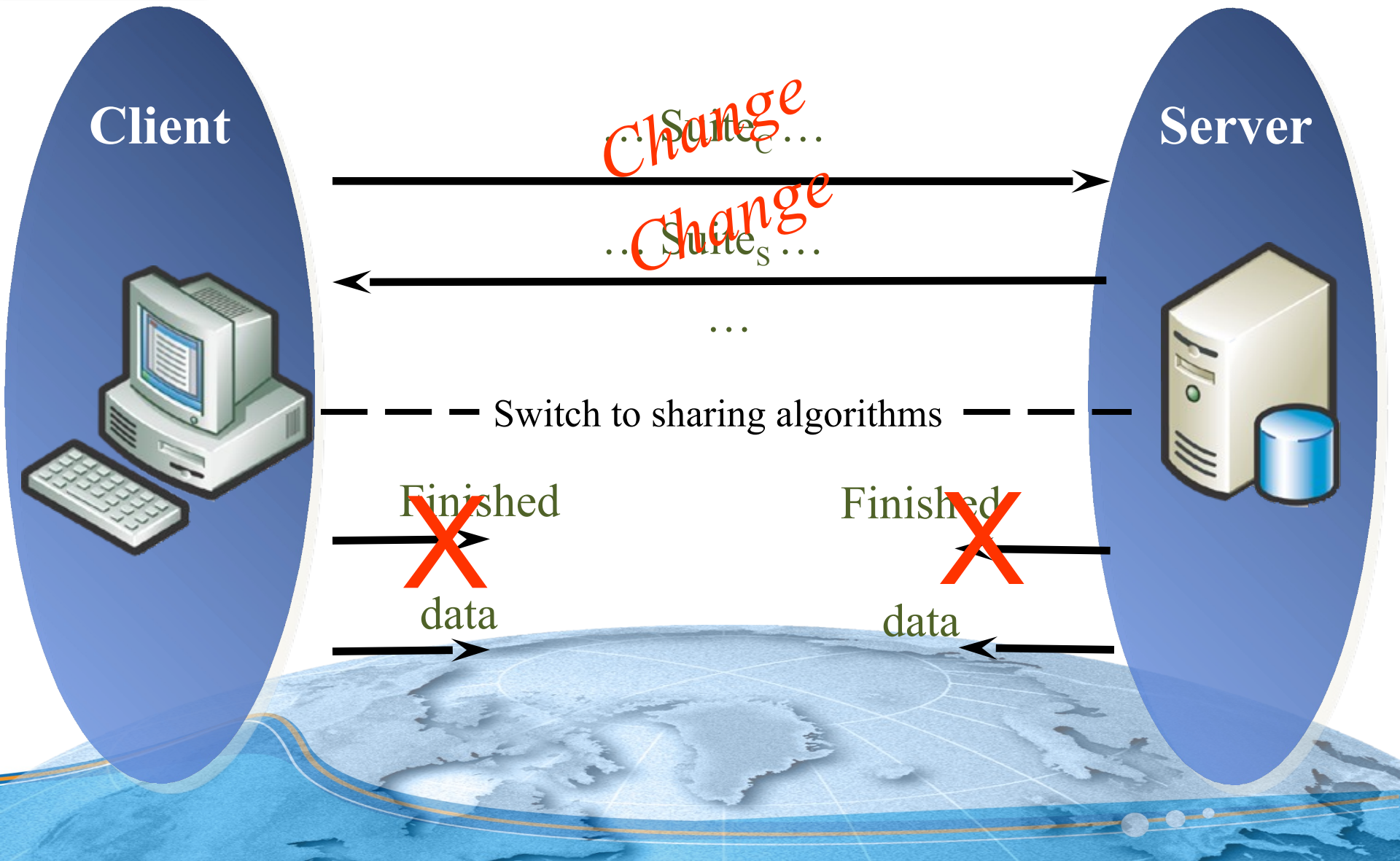
Switch to using key  
deduced from secret<sub>c</sub>

Switch to using key  
deduced from secret<sub>c</sub>

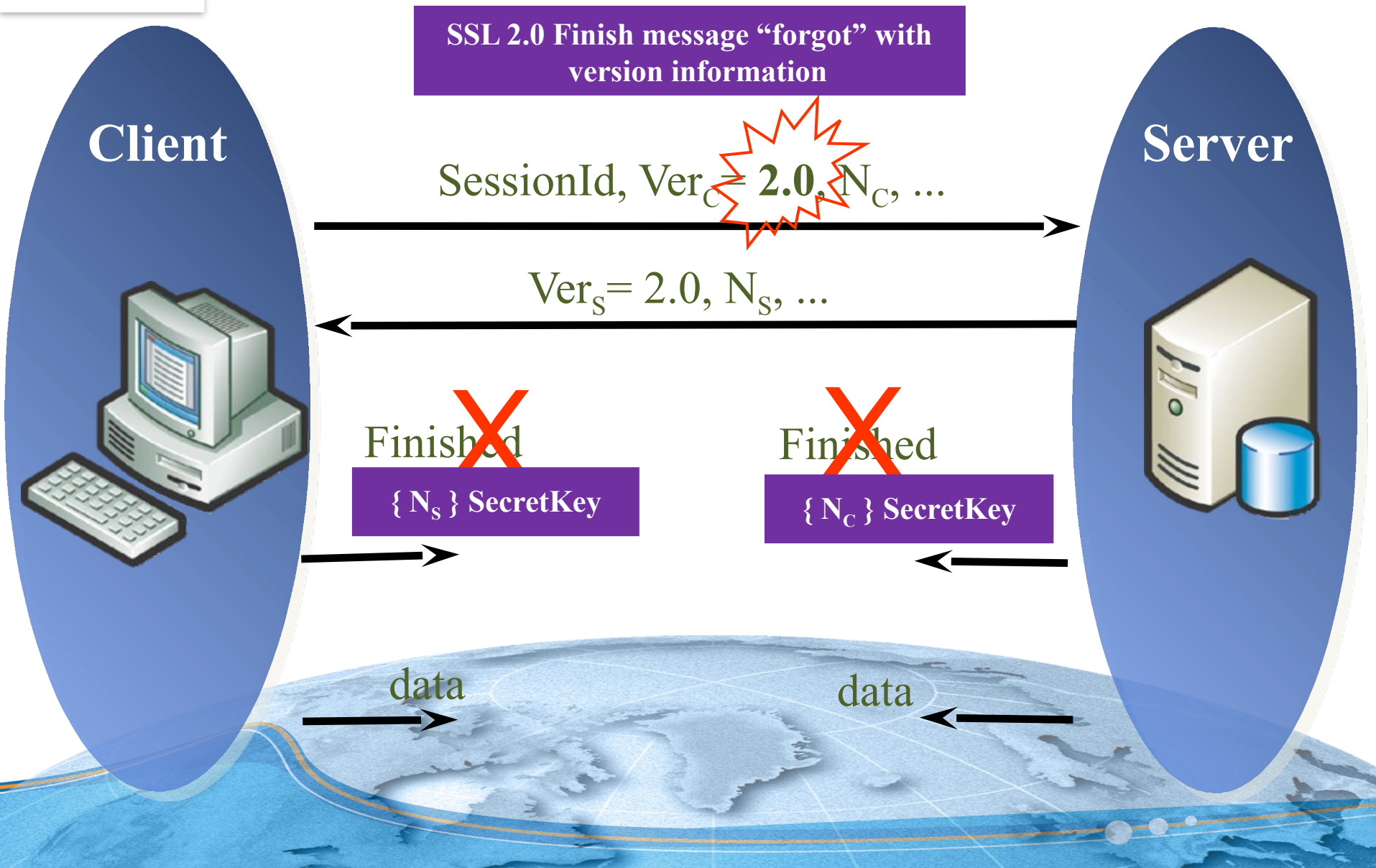
- A = basic protocol
- C = A + certificate of public key
  - Authentication for client and server
- E = C + confirmation message (Finished)
  - Prevent attacks from “tricking” the server to choose the old version or weak algorithms
- F = E + nonce
  - Prevent replay attack



# Anomaly detection with Finished



## Example: Version Rollback Attack

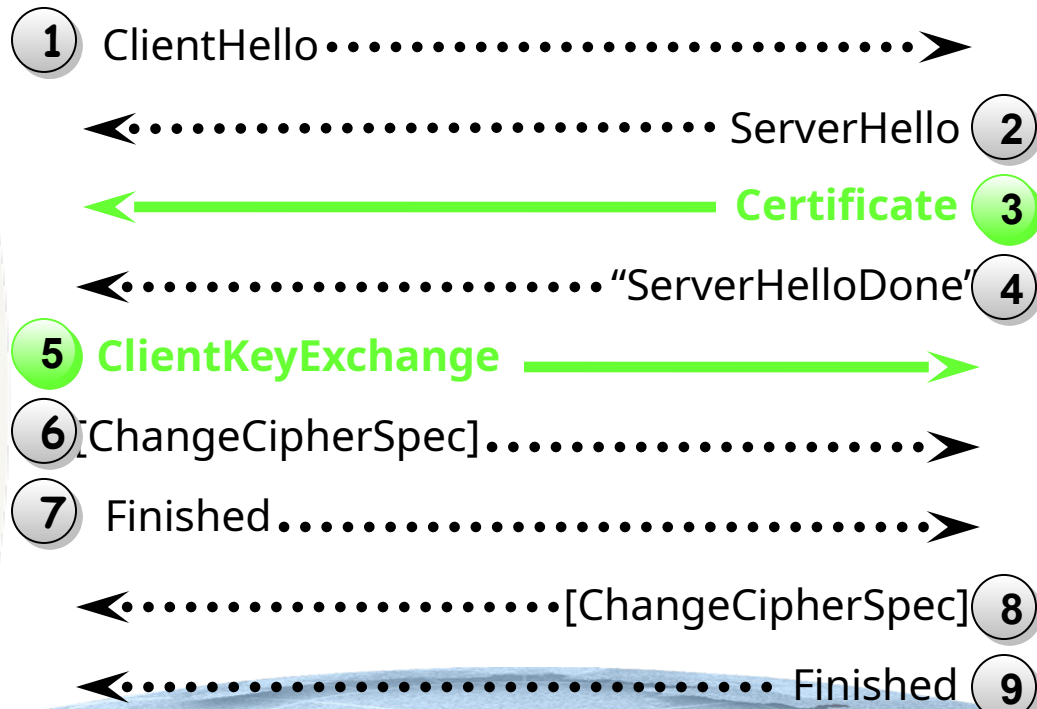


# Server Authentication

Need to use certificate X.509 v3 in  
case of using RSA, Fixed DH,  
Ephemeral DH

Client

Server





# Server & Client Authentication

Client



Server

