CSE 484 / CSE M 584 (Autumn 2011)

# Asymmetric Cryptography

Daniel Halperin

Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

# Class updates

- (Short) Homework 3
  - Due next Wednesday
  - Individual assignment
- (Short) Lab 3 out after class today
  - Short, fun privacy “scavenger hunt”
  - Groups of 1 to 3

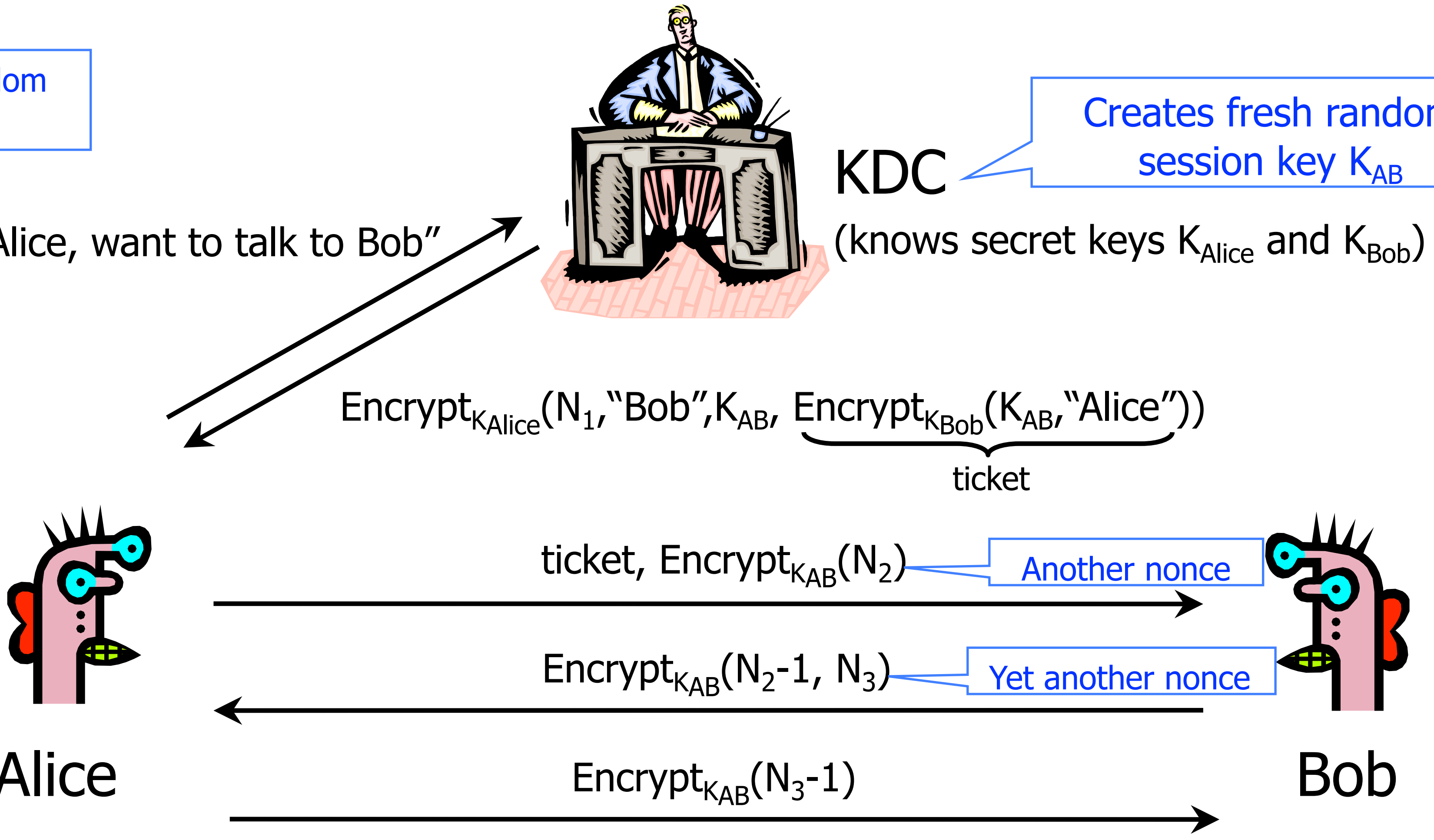
# Homework 2 notes

- (TA request: put name on every page)
- 30 people with public keys: how many key transfers?
- What is the average complexity of breaking a 56-bit key?

# Crypto Protocols

- Last time:
  - Key establishment with 2 parties
- Today:
  - Key establishment with authority

# Private-Key Needham-Schroeder



Fresh, random **nonce**

$N_1$, "I'm Alice, want to talk to Bob"

KDC

(knows secret keys $K_{Alice}$ and $K_{Bob}$)

Creates fresh random session key $K_{AB}$

$\text{Encrypt}_{K_{Alice}}(N_1, \text{"Bob"}, K_{AB}, \underbrace{\text{Encrypt}_{K_{Bob}}(K_{AB}, \text{"Alice"})}_{\text{ticket}})$

ticket, $\text{Encrypt}_{K_{AB}}(N_2)$

Another nonce

$\text{Encrypt}_{K_{AB}}(N_2\text{-}1, N_3)$

Yet another nonce

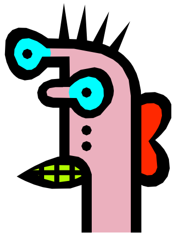
$\text{Encrypt}_{K_{AB}}(N_3\text{-}1)$

Alice

Bob

# Reflection Attack

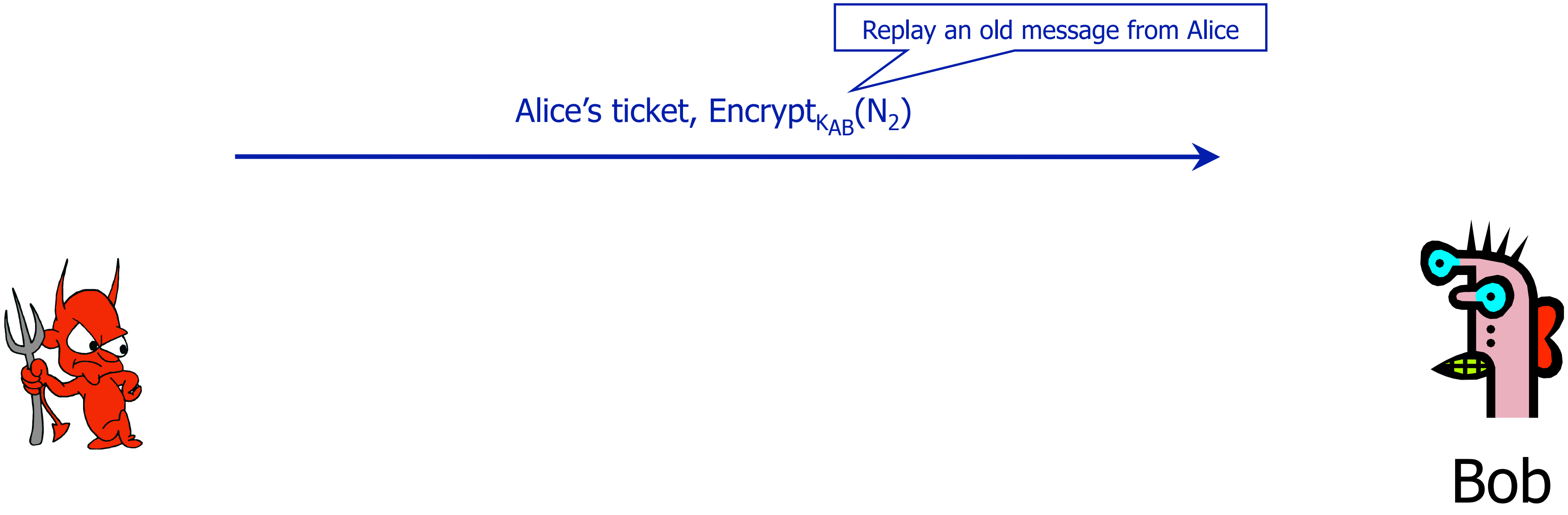
- Suppose symmetric encryption is in ECB/CBC mode…
  - (Easier to see with ECB mode, so assume that)



Bob

# Reflection Attack

- Suppose symmetric encryption is in ECB/CBC mode…
  - (Easier to see with ECB mode, so assume that)

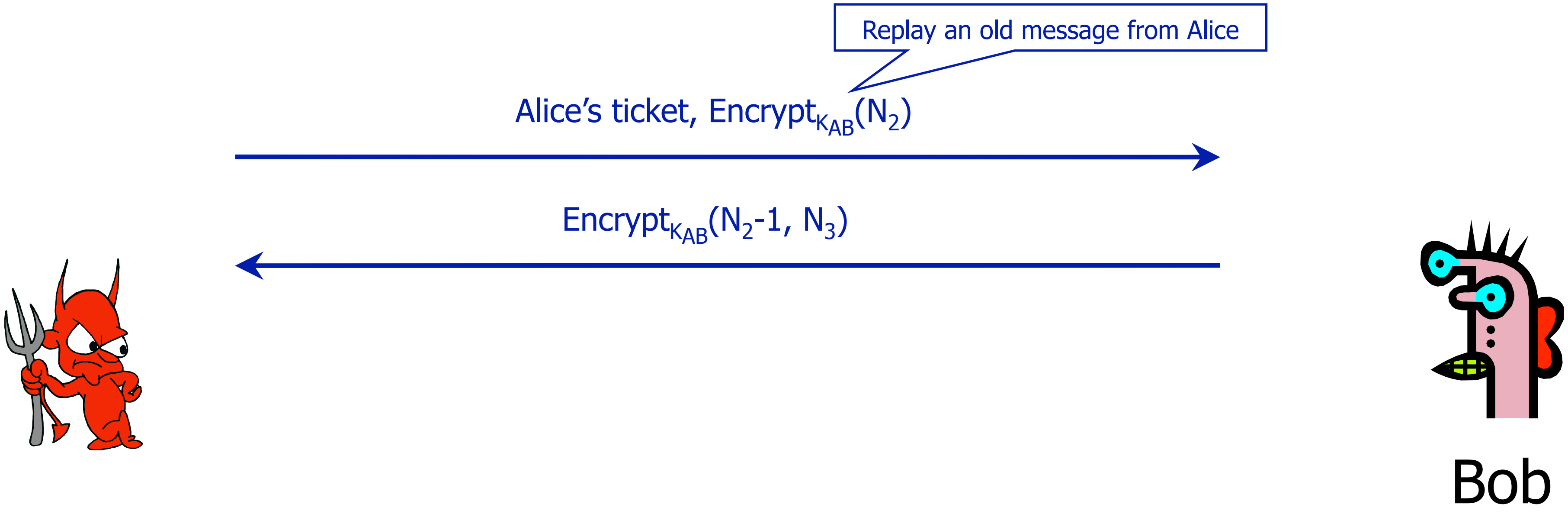


Replay an old message from Alice

Alice’s ticket, $\text{Encrypt}_{KAB}(N_2)$

# Reflection Attack

- Suppose symmetric encryption is in ECB/CBC mode…
  - (Easier to see with ECB mode, so assume that)



Replay an old message from Alice

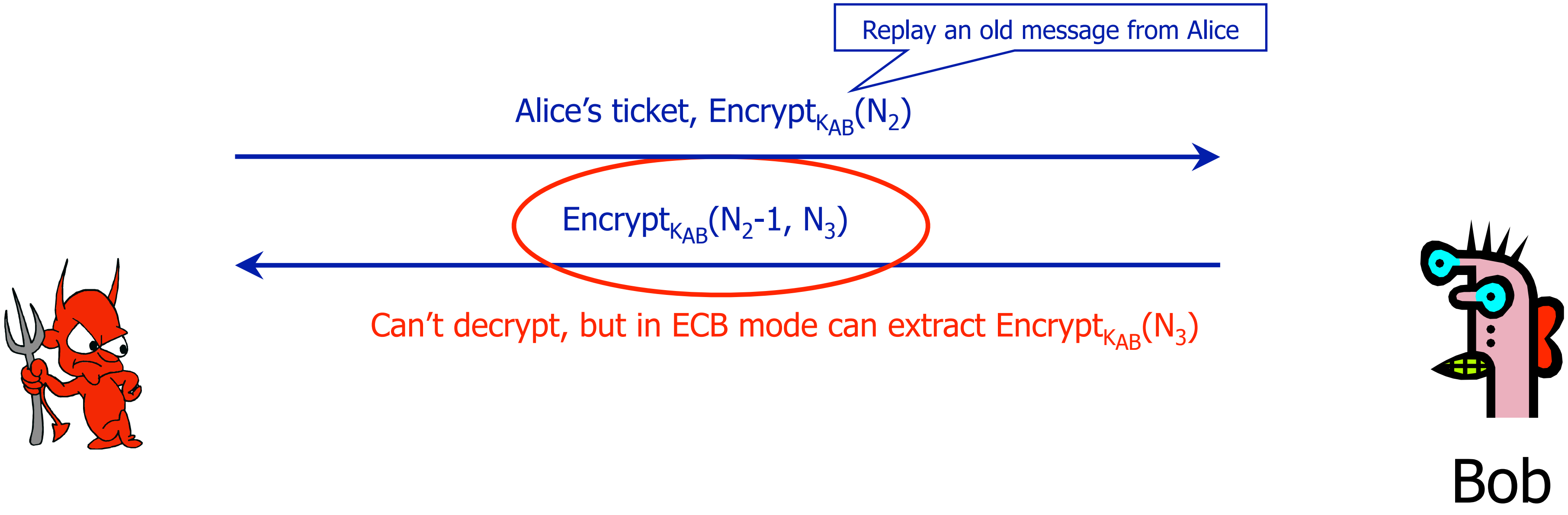
Alice’s ticket, $\text{Encrypt}_{K_{AB}}(N_2)$

$\text{Encrypt}_{K_{AB}}(N_2\text{-}1, N_3)$

Bob

# Reflection Attack

- Suppose symmetric encryption is in ECB/CBC mode…
  - (Easier to see with ECB mode, so assume that)



Replay an old message from Alice

Alice's ticket, Encrypt$_{KAB}$($N_2$)

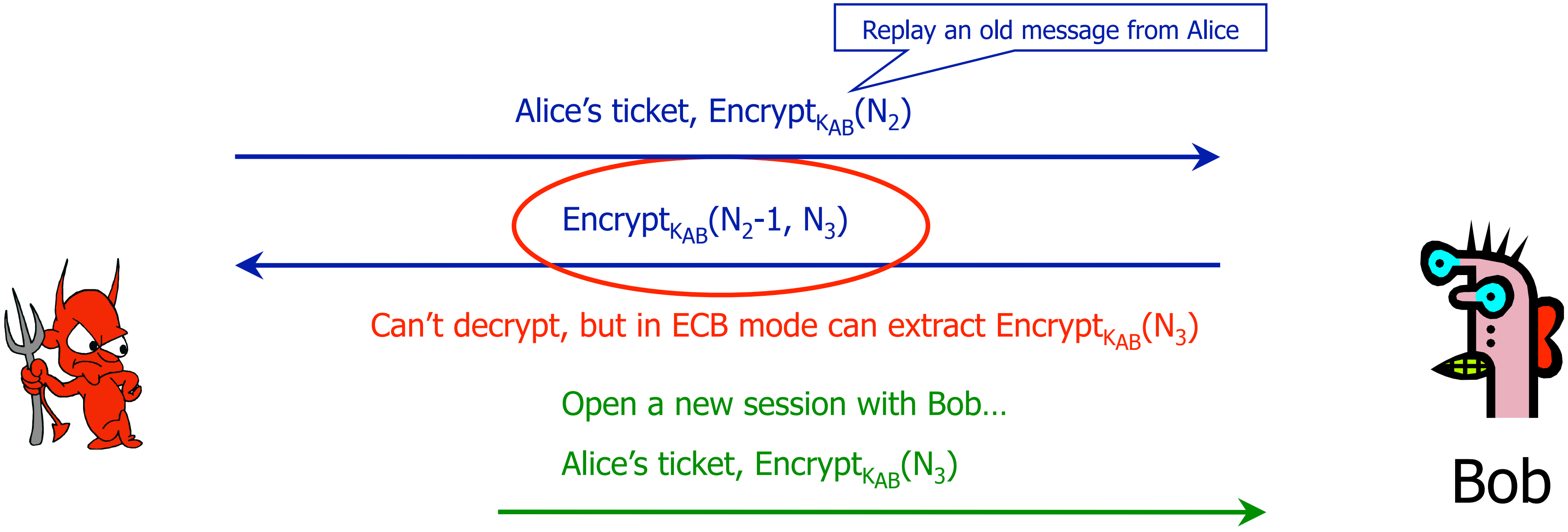
Encrypt$_{KAB}$($N_2$-1, $N_3$)

Can't decrypt, but in ECB mode can extract Encrypt$_{KAB}$($N_3$)

Bob

# Reflection Attack

- Suppose symmetric encryption is in ECB/CBC mode…
  - (Easier to see with ECB mode, so assume that)



Replay an old message from Alice

Alice’s ticket, $\text{Encrypt}_{KAB}(N_2)$

$\text{Encrypt}_{KAB}(N_2\text{-}1, N_3)$

Can’t decrypt, but in ECB mode can extract $\text{Encrypt}_{KAB}(N_3)$

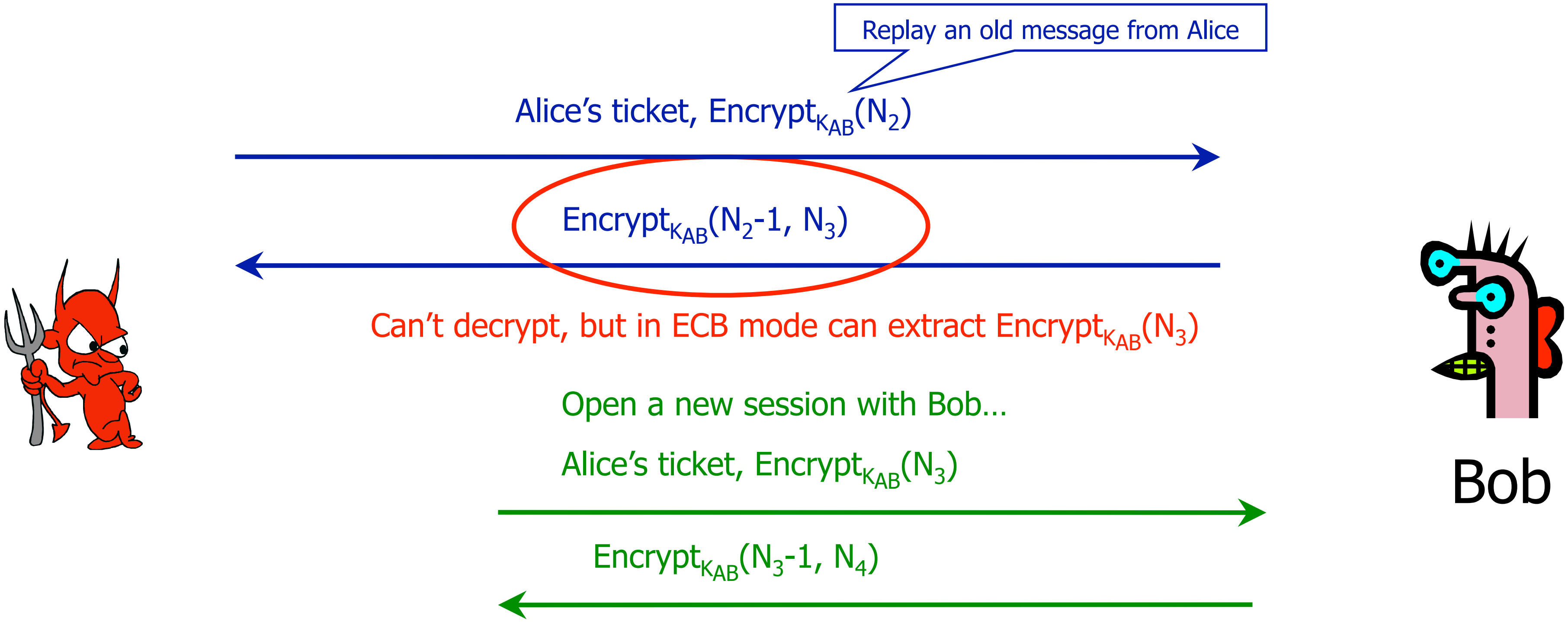
Open a new session with Bob…

Alice’s ticket, $\text{Encrypt}_{KAB}(N_3)$

Bob

# Reflection Attack

- Suppose symmetric encryption is in ECB/CBC mode…
  - (Easier to see with ECB mode, so assume that)



Replay an old message from Alice

Alice’s ticket, $\text{Encrypt}_{KAB}(N_2)$

$\text{Encrypt}_{KAB}(N_2\text{-}1, N_3)$

Can’t decrypt, but in ECB mode can extract $\text{Encrypt}_{KAB}(N_3)$

Open a new session with Bob…

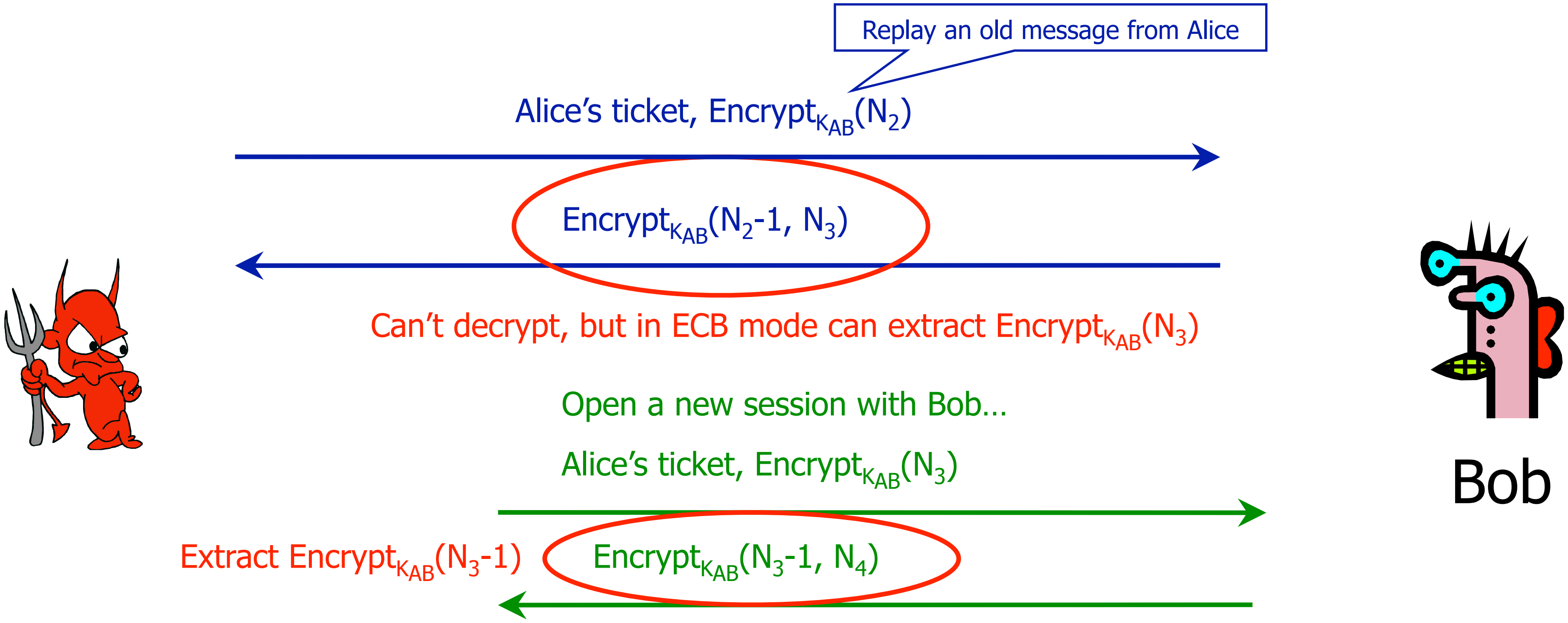
Alice’s ticket, $\text{Encrypt}_{KAB}(N_3)$

$\text{Encrypt}_{KAB}(N_3\text{-}1, N_4)$

Bob

# Reflection Attack

- Suppose symmetric encryption is in ECB/CBC mode…
    - (Easier to see with ECB mode, so assume that)



Replay an old message from Alice

Alice's ticket, $\text{Encrypt}_{KAB}(N_2)$

$\text{Encrypt}_{KAB}(N_2\text{-}1, N_3)$

Can't decrypt, but in ECB mode can extract $\text{Encrypt}_{KAB}(N_3)$

Open a new session with Bob…

Alice's ticket, $\text{Encrypt}_{KAB}(N_3)$

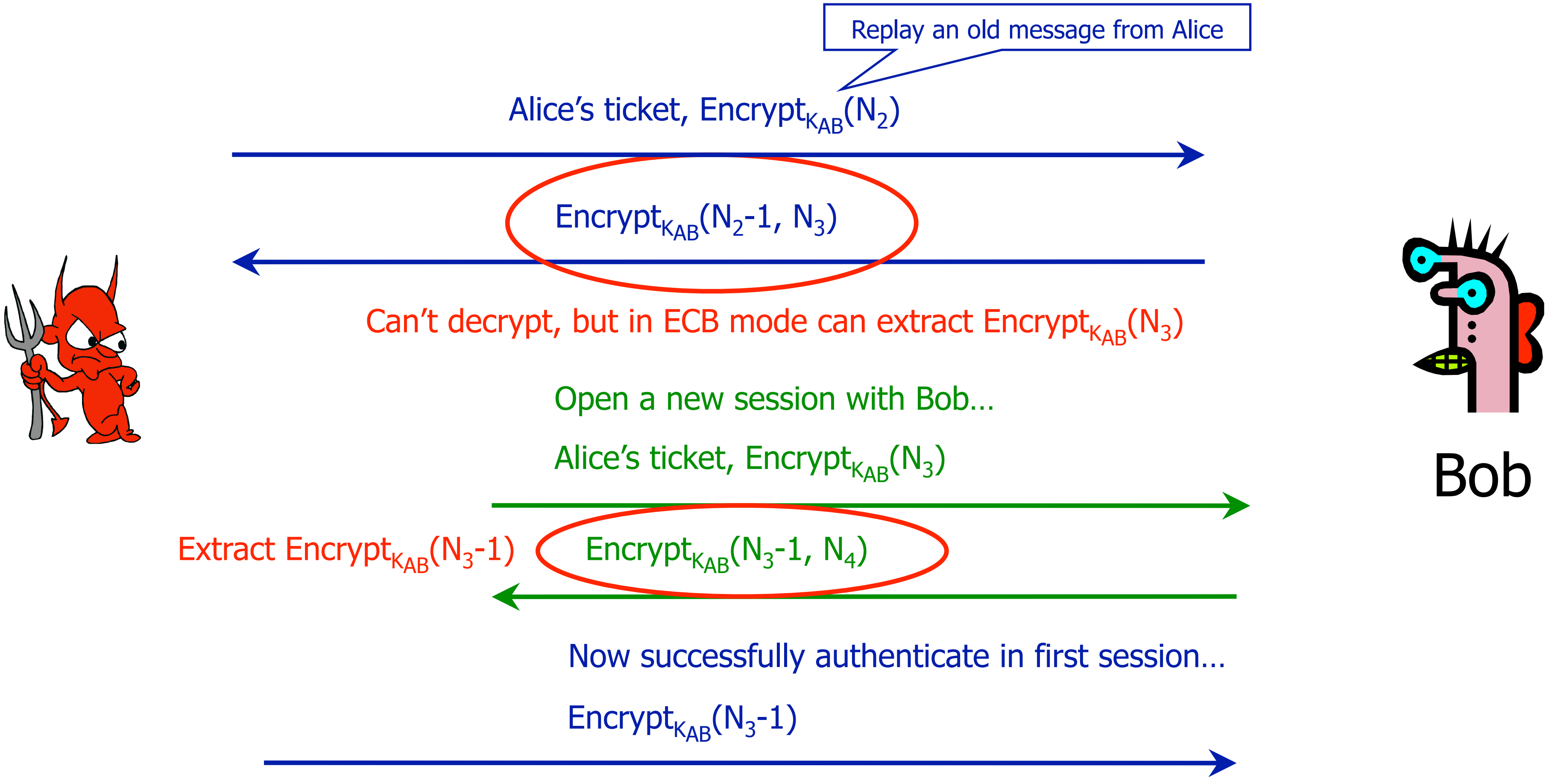
Extract $\text{Encrypt}_{KAB}(N_3\text{-}1)$

$\text{Encrypt}_{KAB}(N_3\text{-}1, N_4)$

Bob

# Reflection Attack

- Suppose symmetric encryption is in ECB/CBC mode…
  - (Easier to see with ECB mode, so assume that)



Replay an old message from Alice

Alice's ticket, Encrypt$_{KAB}$(N$_2$)

Encrypt$_{KAB}$(N$_2$-1, N$_3$)

Can't decrypt, but in ECB mode can extract Encrypt$_{KAB}$(N$_3$)

Open a new session with Bob…

Alice's ticket, Encrypt$_{KAB}$(N$_3$)

Extract Encrypt$_{KAB}$(N$_3$-1)

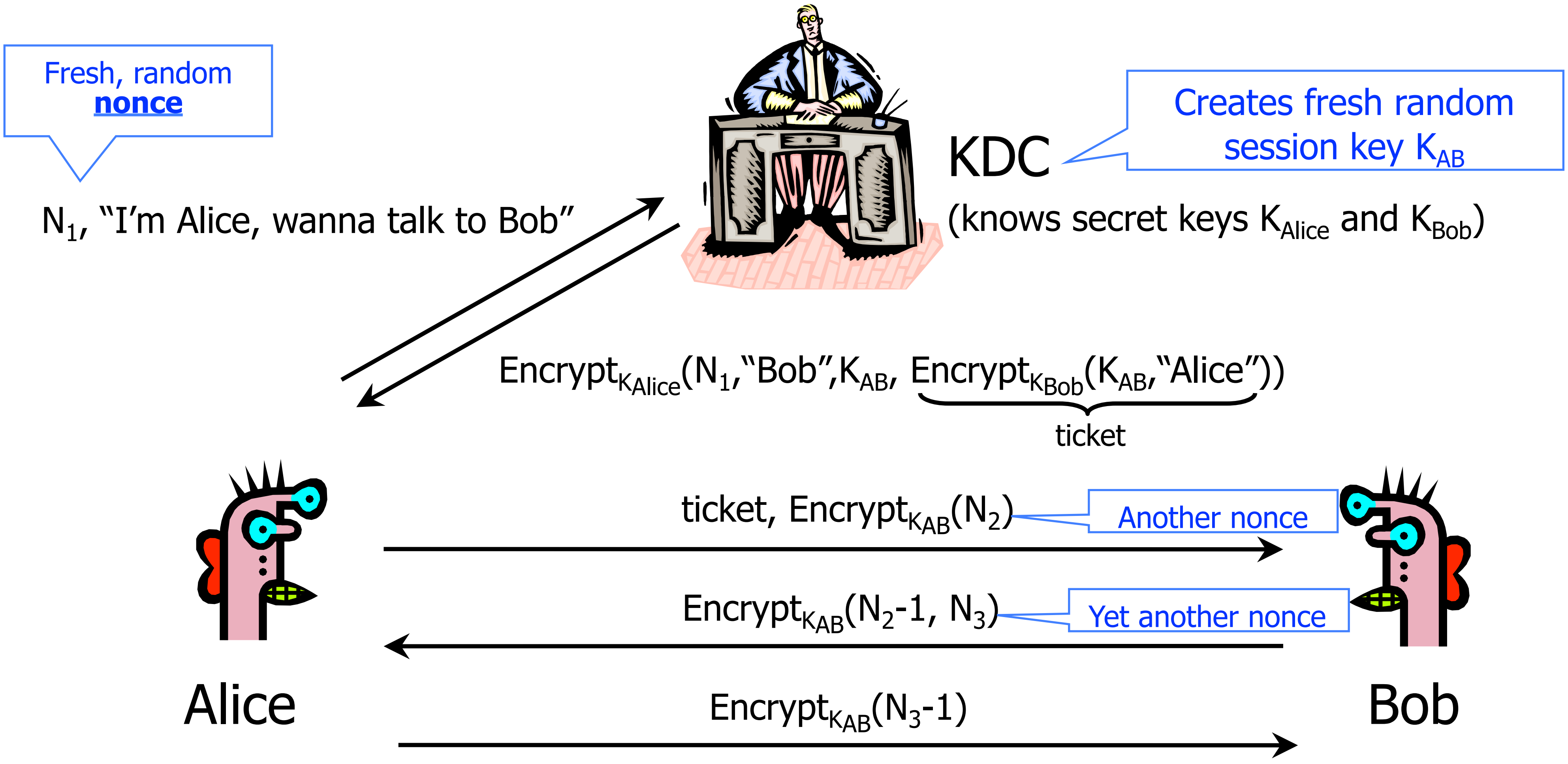
Encrypt$_{KAB}$(N$_3$-1, N$_4$)

Now successfully authenticate in first session…

Encrypt$_{KAB}$(N$_3$-1)

Bob

# Private-Key Needham-Schroeder



Fresh, random **nonce**

$N_1$, "I'm Alice, wanna talk to Bob"

KDC

(knows secret keys $K_{Alice}$ and $K_{Bob}$)

Creates fresh random session key $K_{AB}$

$\text{Encrypt}_{K_{Alice}}(N_1, \text{"Bob"}, K_{AB}, \underbrace{\text{Encrypt}_{K_{Bob}}(K_{AB}, \text{"Alice"})}_{\text{ticket}})$

ticket, $\text{Encrypt}_{K_{AB}}(N_2)$

Another nonce

$\text{Encrypt}_{K_{AB}}(N_2\text{-}1, N_3)$

Yet another nonce

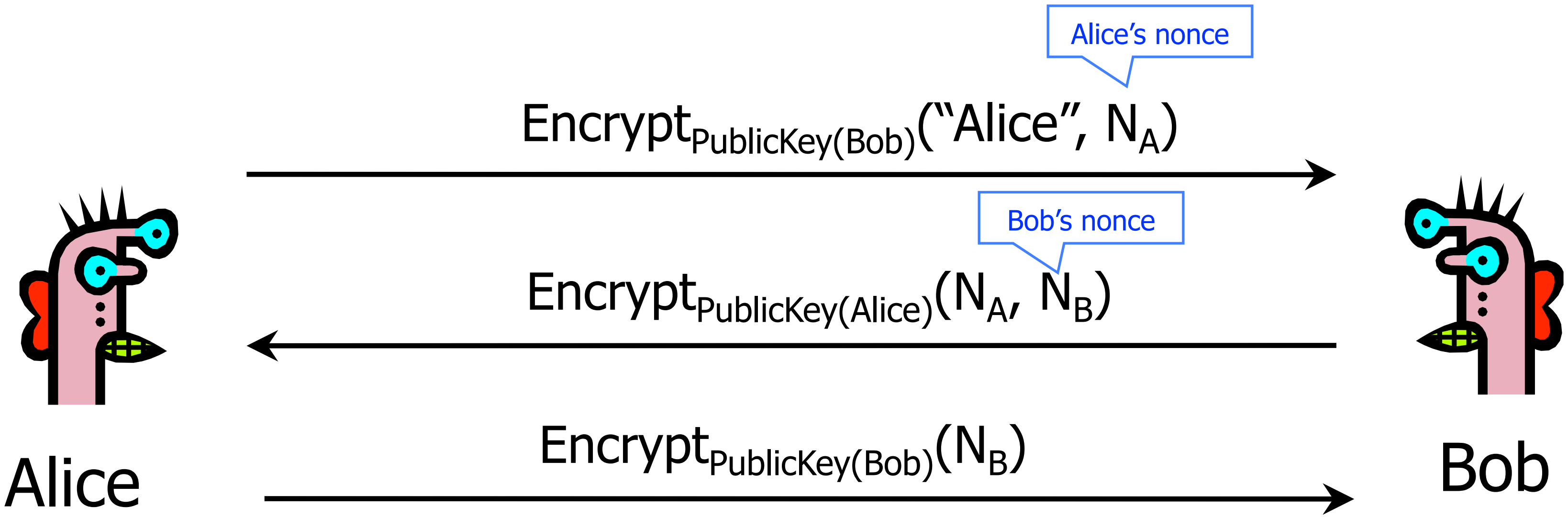
$\text{Encrypt}_{K_{AB}}(N_3\text{-}1)$

Alice

Bob

- Another issue: If learn $K_{AB}$ after session completes, then can re-use. (Solution: timestamps, nonces.)

# Public-Key Needham-Schroeder



Alice's nonce

$\text{Encrypt}_{\text{PublicKey(Bob)}}(\text{“Alice”}, N_A)$

Bob's nonce

$\text{Encrypt}_{\text{PublicKey(Alice)}}(N_A, N_B)$

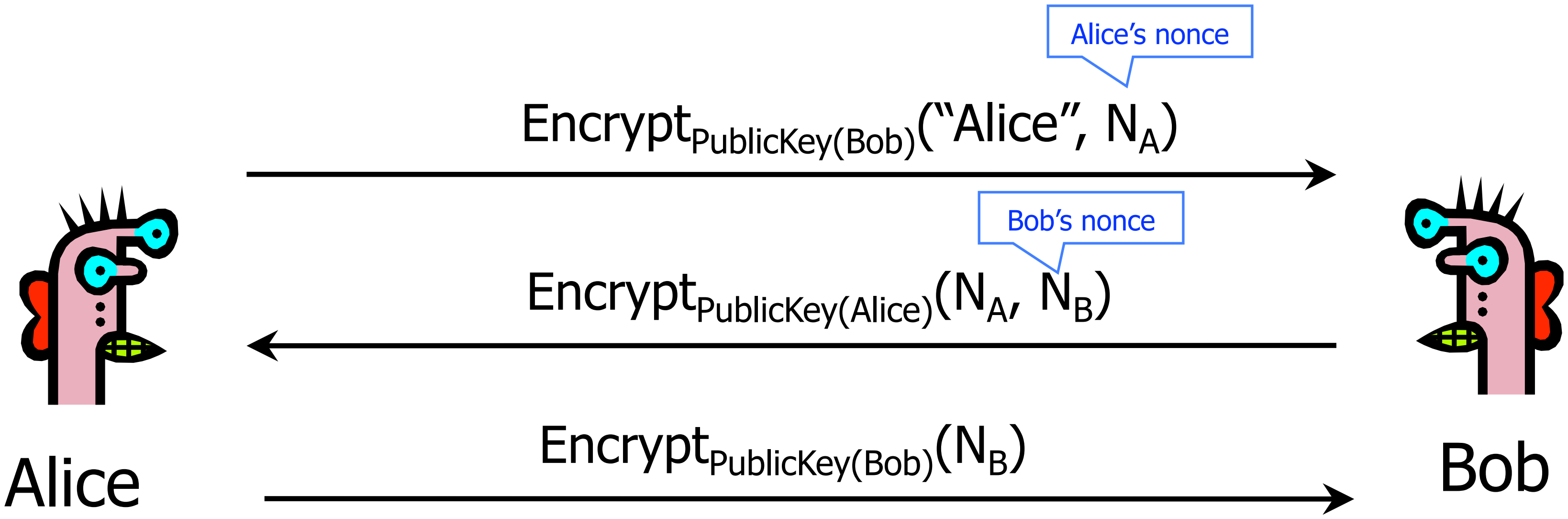
$\text{Encrypt}_{\text{PublicKey(Bob)}}(N_B)$

Alice

Bob

Create new key from $N_A$ and $N_B$, e.g., $N_A \oplus N_B$

# Public-Key Needham-Schroeder



Alice's nonce

Encrypt$_{\text{PublicKey(Bob)}}$("Alice", $N_A$)

Bob's nonce

Encrypt$_{\text{PublicKey(Alice)}}$($N_A$, $N_B$)

Encrypt$_{\text{PublicKey(Bob)}}$($N_B$)

Alice

Bob

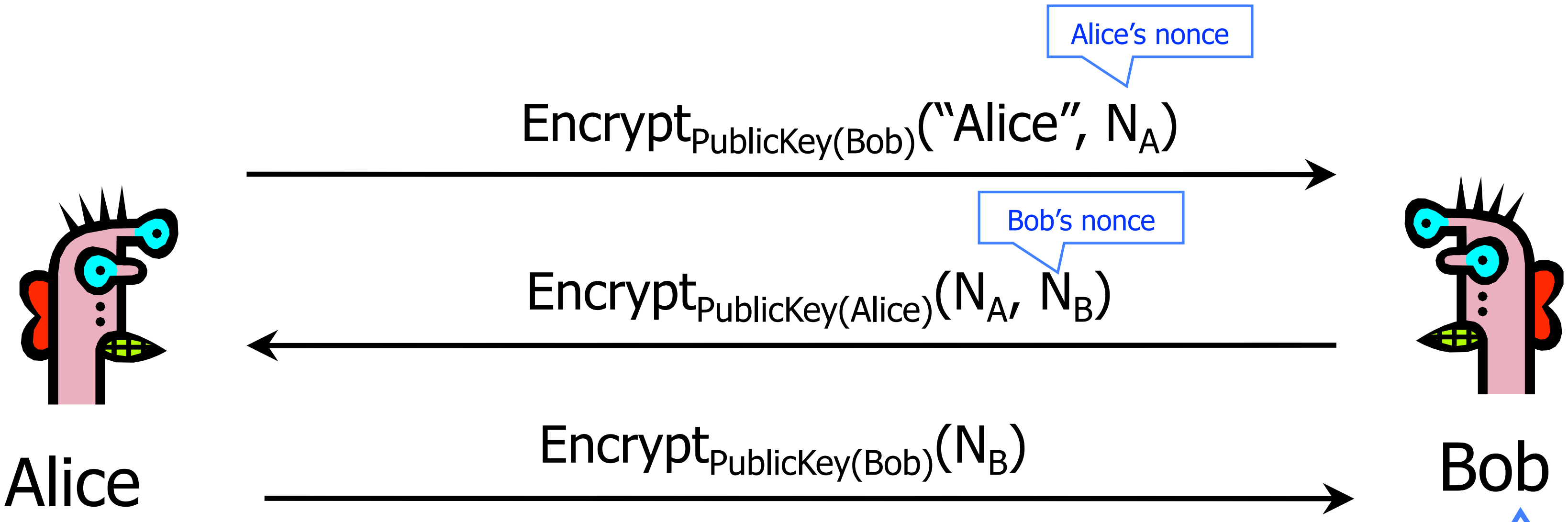
Create new key from $N_A$ and $N_B$, e.g., $N_A \oplus N_B$

Alice's reasoning:

- The only person who could know $N_A$ is the person who decrypted 1st message
- Only Bob can decrypt message encrypted with Bob's public key
- Therefore, Bob is on the other end of the line

**Bob is authenticated!**

# Public-Key Needham-Schroeder



Alice's nonce

Encrypt$_{\text{PublicKey(Bob)}}$("Alice", $N_A$)

Bob's nonce

Encrypt$_{\text{PublicKey(Alice)}}$($N_A$, $N_B$)

Encrypt$_{\text{PublicKey(Bob)}}$($N_B$)

Create new key from $N_A$ and $N_B$, e.g., $N_A \oplus N_B$

Alice's reasoning:

- The only person who could know $N_A$ is the person who decrypted 1st message
- Only Bob can decrypt message encrypted with Bob's public key
- Therefore, Bob is on the other end of the line

**Bob is authenticated!**

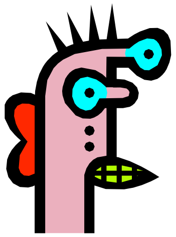
Bob's reasoning:

- The only way to learn $N_B$ is to decrypt 2nd message
- Only Alice can decrypt 2nd message
- Therefore, Alice is on the other end

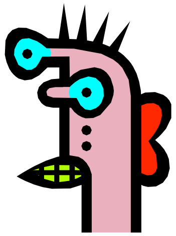
**Alice is authenticated!**

# Attack on Needham-Schroeder

[published by Gavin Lowe]



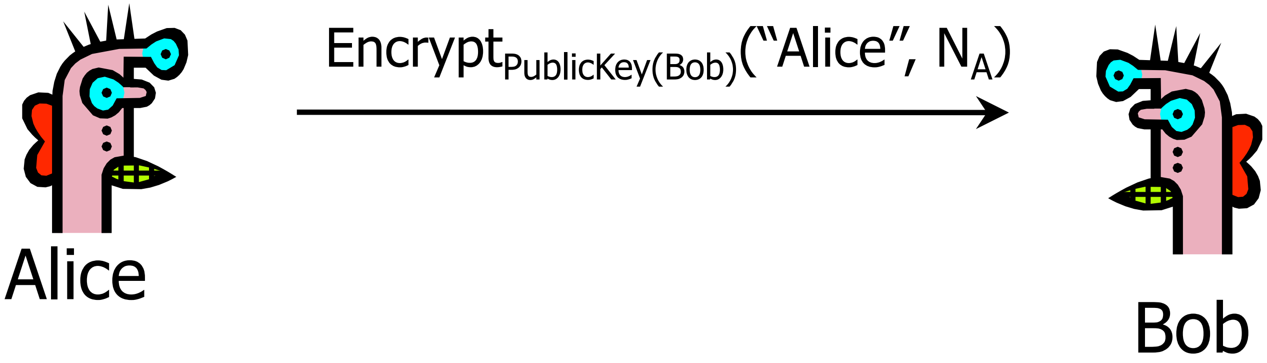
Alice



Bob

# Attack on Needham-Schroeder

[published by Gavin Lowe]



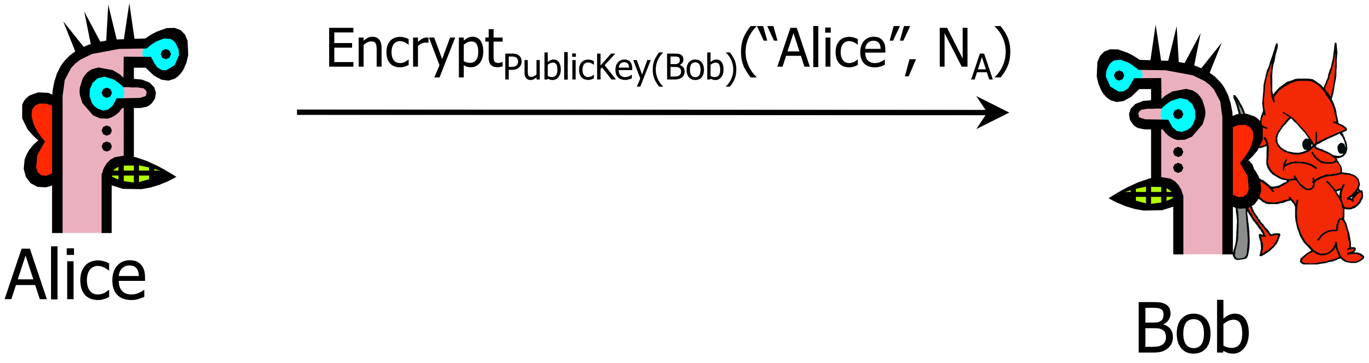
Alice

$\text{Encrypt}_{\text{PublicKey(Bob)}}(\text{“Alice”}, N_A)$

Bob

# Attack on Needham-Schroeder

[published by Gavin Lowe]



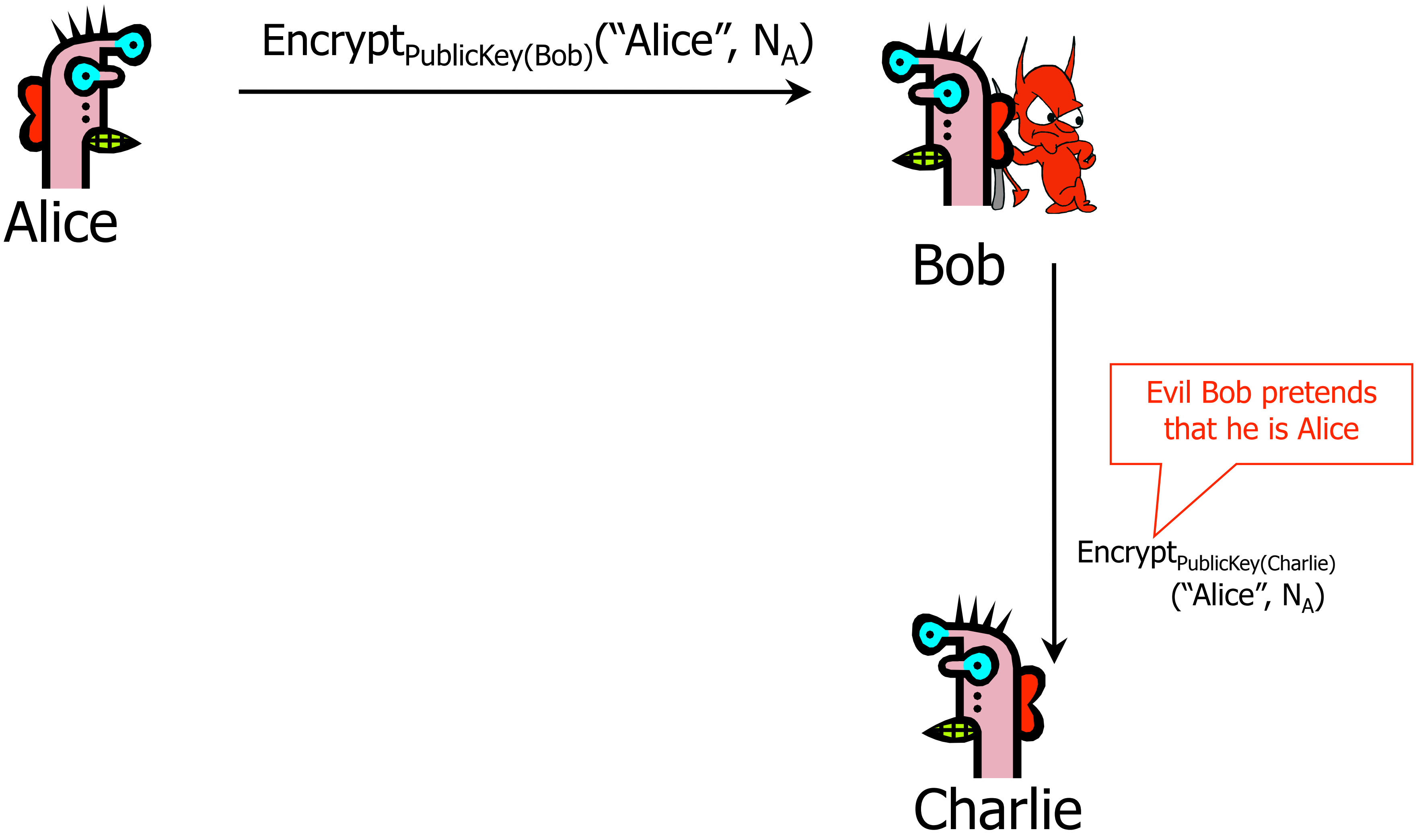
$\text{Encrypt}_{\text{PublicKey(Bob)}}(\text{“Alice”}, N_A)$

Alice

Bob

# Attack on Needham-Schroeder

[published by Gavin Lowe]



Alice

$\text{Encrypt}_{\text{PublicKey(Bob)}}(\text{“Alice”}, N_A)$

Bob

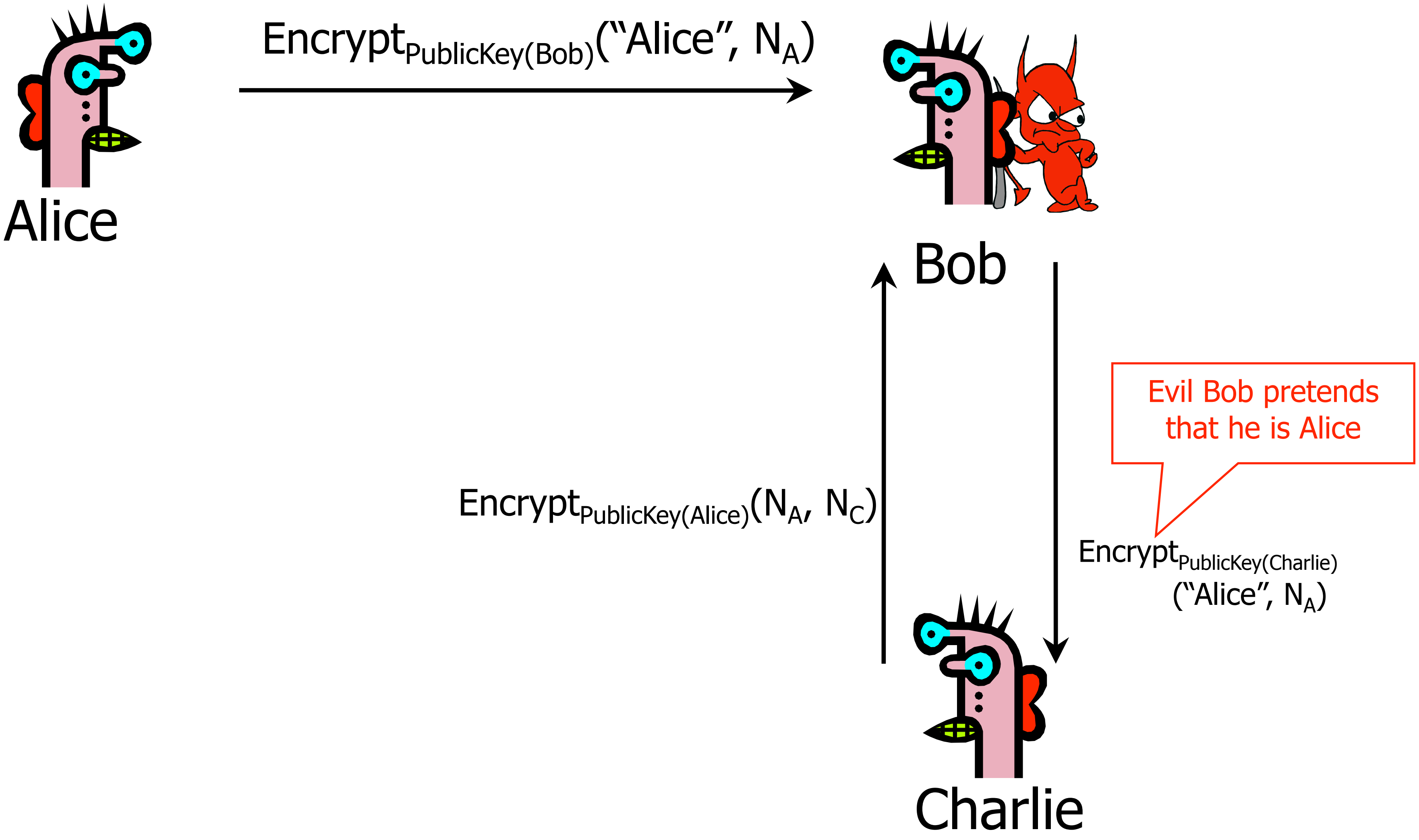
Evil Bob pretends that he is Alice

$\text{Encrypt}_{\text{PublicKey(Charlie)}}(\text{“Alice”}, N_A)$

Charlie

# Attack on Needham-Schroeder

[published by Gavin Lowe]



$\text{Encrypt}_{\text{PublicKey(Bob)}}(\text{“Alice”}, N_A)$

Alice

Bob

$\text{Encrypt}_{\text{PublicKey(Alice)}}(N_A, N_C)$

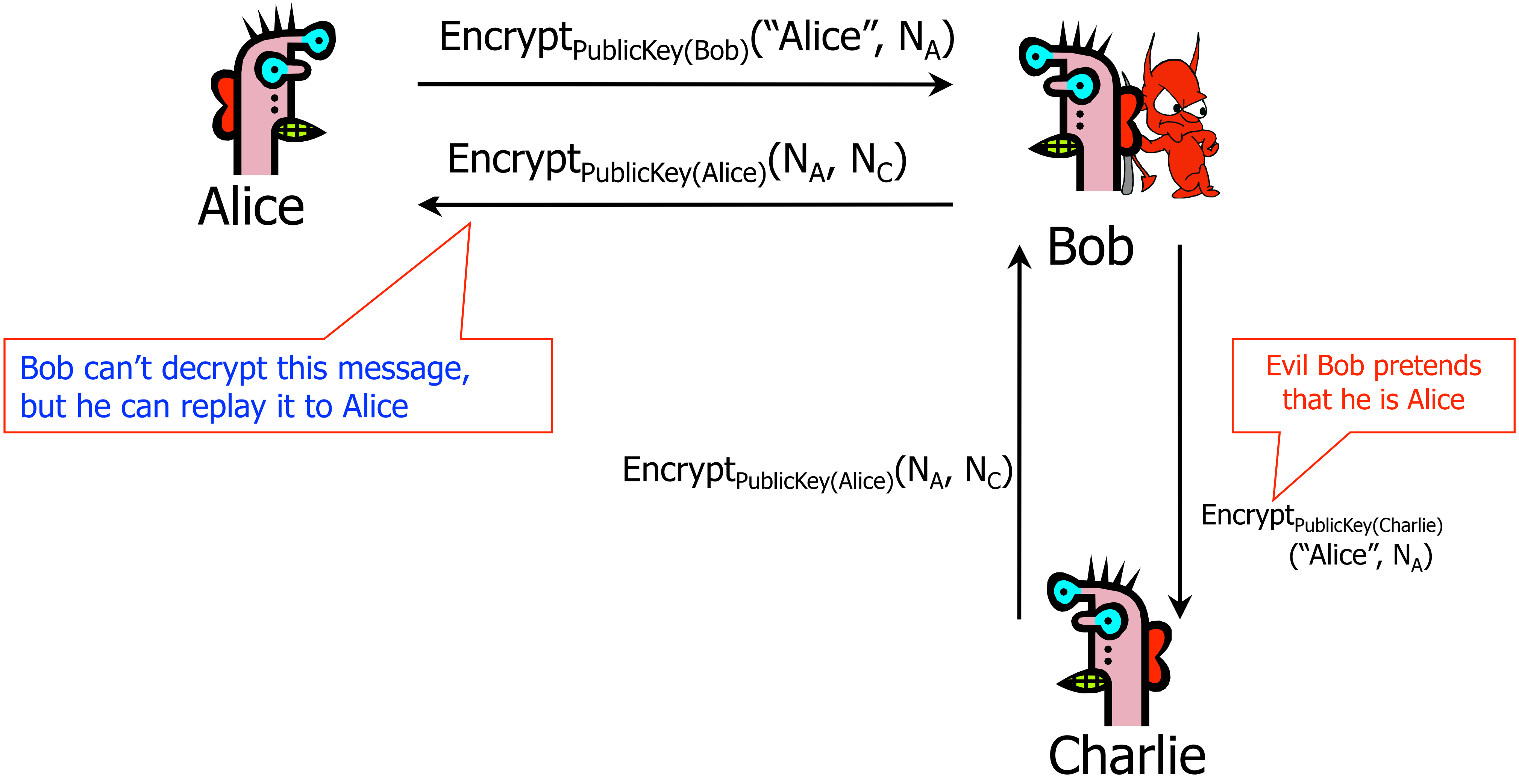
Evil Bob pretends that he is Alice

$\text{Encrypt}_{\text{PublicKey(Charlie)}}(\text{“Alice”}, N_A)$

Charlie

# Attack on Needham-Schroeder

[published by Gavin Lowe]



Alice

$\text{Encrypt}_{\text{PublicKey(Bob)}}(\text{“Alice”}, N_A)$

Bob

$\text{Encrypt}_{\text{PublicKey(Alice)}}(N_A, N_C)$

Bob can’t decrypt this message, but he can replay it to Alice

Evil Bob pretends that he is Alice

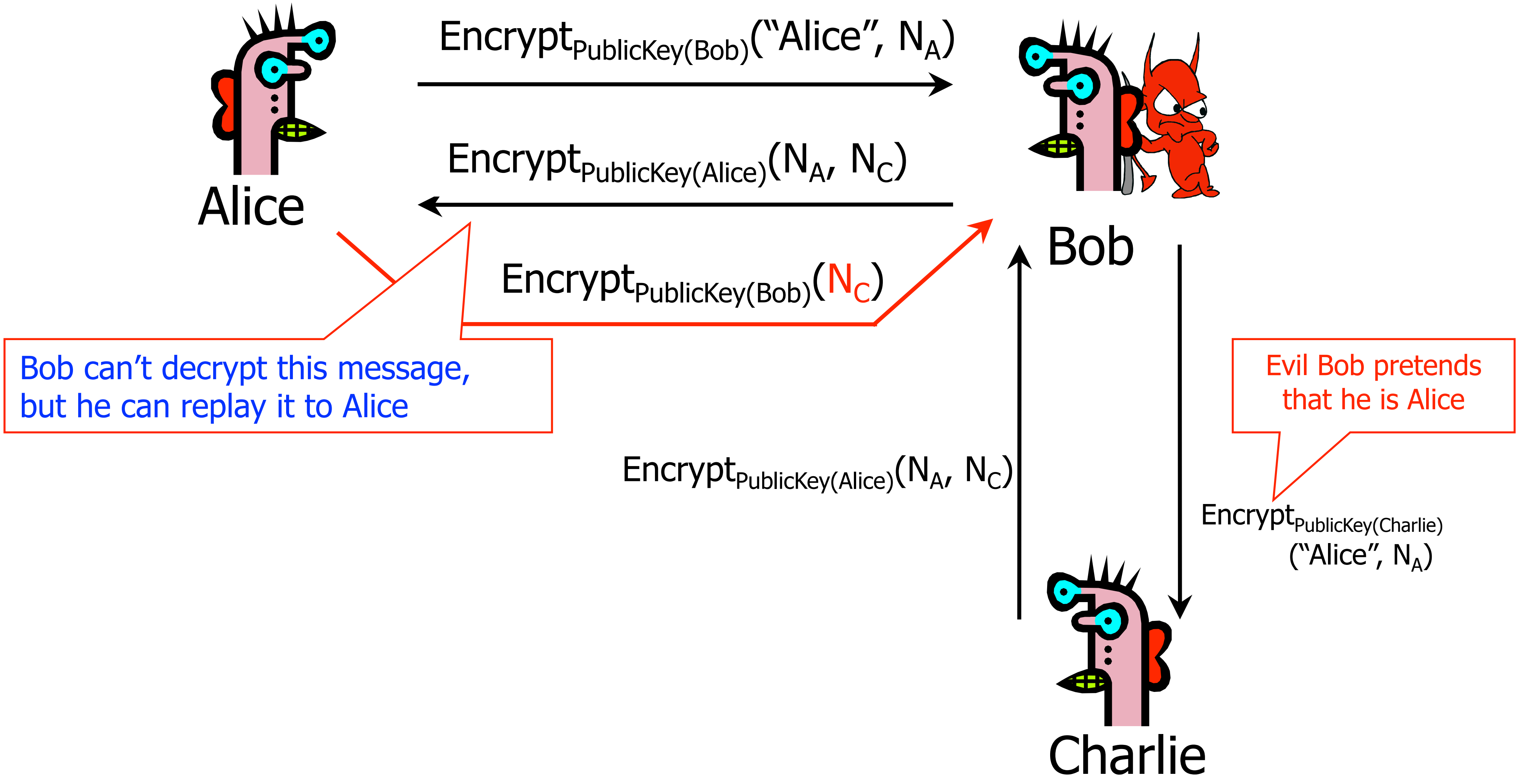
$\text{Encrypt}_{\text{PublicKey(Charlie)}}(\text{“Alice”}, N_A)$

$\text{Encrypt}_{\text{PublicKey(Alice)}}(N_A, N_C)$

Charlie

# Attack on Needham-Schroeder

[published by Gavin Lowe]



Alice

Encrypt$_{\text{PublicKey(Bob)}}$(“Alice”, $N_A$)

Encrypt$_{\text{PublicKey(Alice)}}$($N_A$, $N_C$)

Encrypt$_{\text{PublicKey(Bob)}}$($N_C$)

Bob

Bob can’t decrypt this message, but he can replay it to Alice

Encrypt$_{\text{PublicKey(Alice)}}$($N_A$, $N_C$)

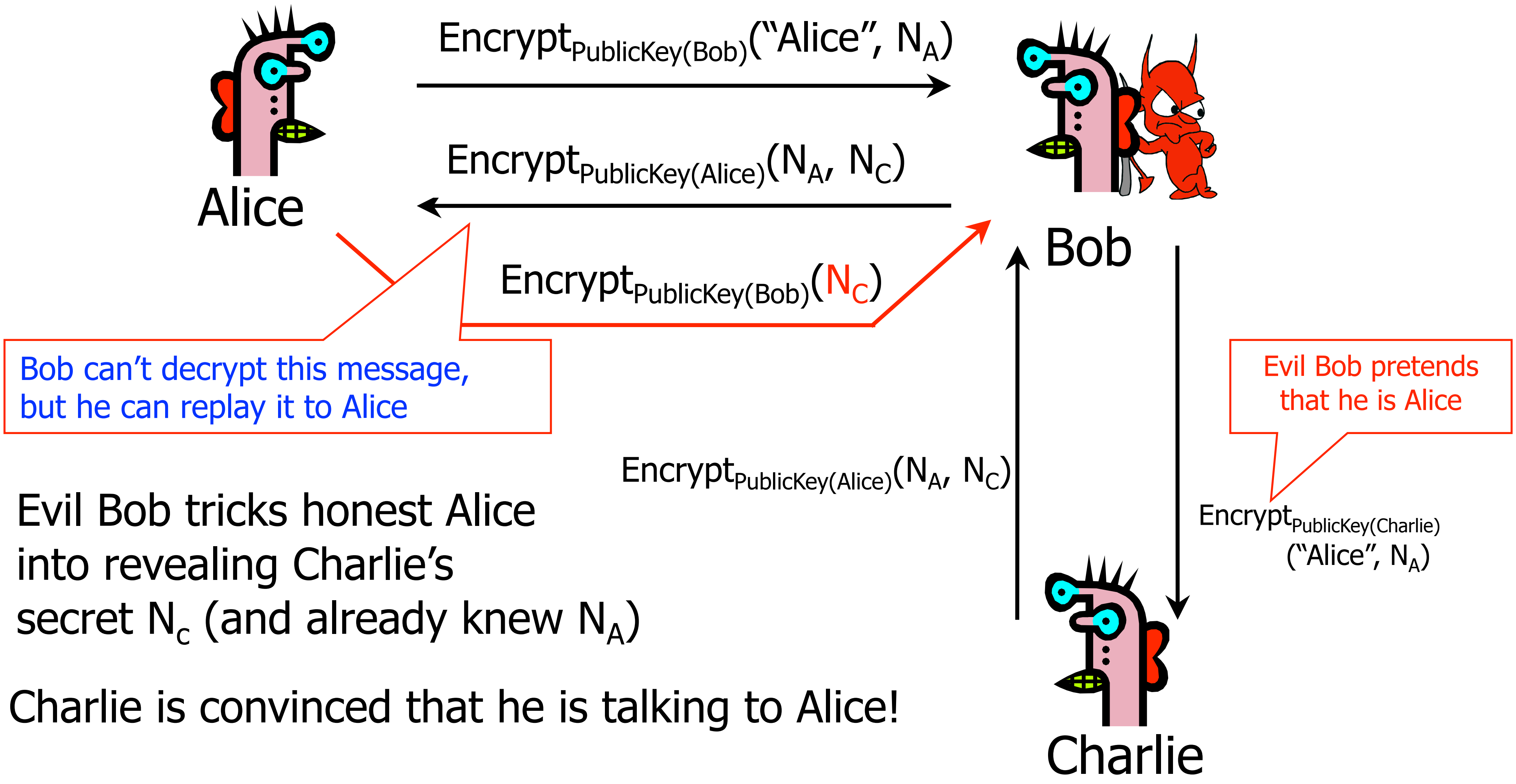
Evil Bob pretends that he is Alice

Encrypt$_{\text{PublicKey(Charlie)}}$ (“Alice”, $N_A$)

Charlie

# Attack on Needham-Schroeder

[published by Gavin Lowe]



Encrypt$_{\text{PublicKey(Bob)}}$("Alice", $N_A$)

Encrypt$_{\text{PublicKey(Alice)}}$($N_A$, $N_C$)

Encrypt$_{\text{PublicKey(Bob)}}$($N_C$)

Bob can't decrypt this message, but he can replay it to Alice

Evil Bob pretends that he is Alice

Encrypt$_{\text{PublicKey(Charlie)}}$("Alice", $N_A$)

Encrypt$_{\text{PublicKey(Alice)}}$($N_A$, $N_C$)

Evil Bob tricks honest Alice into revealing Charlie's secret $N_c$ (and already knew $N_A$)

Charlie is convinced that he is talking to Alice!

# Lessons of Needham-Schroeder

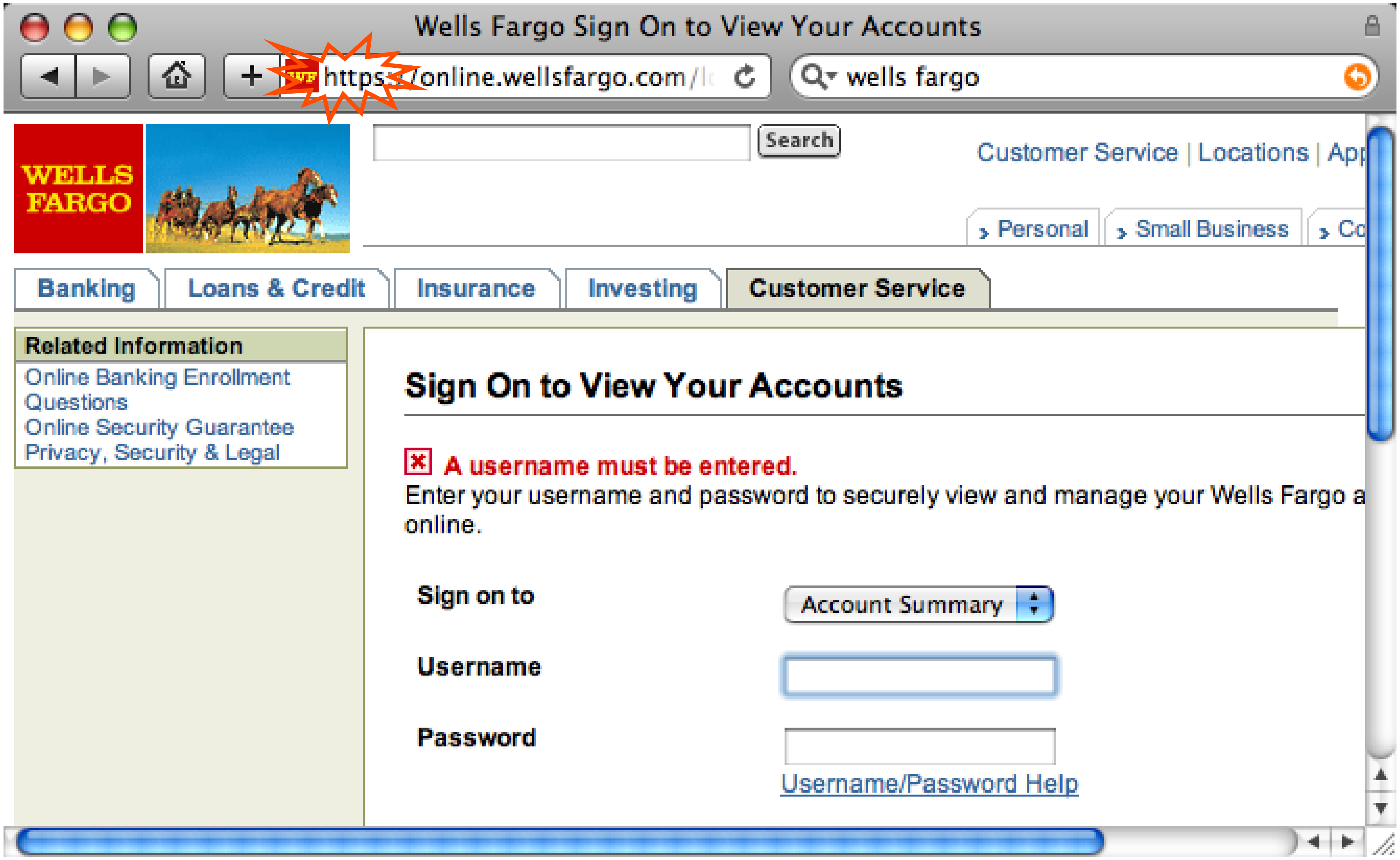
- This is yet another example of design challenges
  - Alice is correct that Bob must have decrypted $\text{Encrypt}_{\text{PublicKey(Bob)}}$(“Alice”, $N_A$), but this does <u>not</u> mean that $\text{Encrypt}_{\text{PublicKey(Alice)}}(N_A, N_B)$ came from Bob
- It is important to realize limitations of protocols
  - The attack requires that Alice willingly talk to attacker
    - Attacker uses a legitimate conversation with Alice to impersonate Alice to Charlie

# SSL

# What is SSL / TLS?

- Transport Layer Security (TLS) protocol, version 1.2
  - De facto standard for Internet security
  - “The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications”
  - In practice, used to protect information transmitted between browsers and Web servers (and mail readers and ...)
- Based on Secure Sockets Layers (SSL) protocol, version 3.0
  - Same protocol design, different algorithms
- Deployed in nearly every Web browser

# SSL / TLS in the Real World



Wells Fargo Sign On to View Your Accounts
https://online.wellsfargo.com/l
wells fargo
WELLS FARGO
Search
Customer Service | Locations | App
Personal
Small Business
Co
Banking
Loans & Credit
Insurance
Investing
Customer Service
Related Information
Online Banking Enrollment
Questions
Online Security Guarantee
Privacy, Security & Legal

Sign On to View Your Accounts

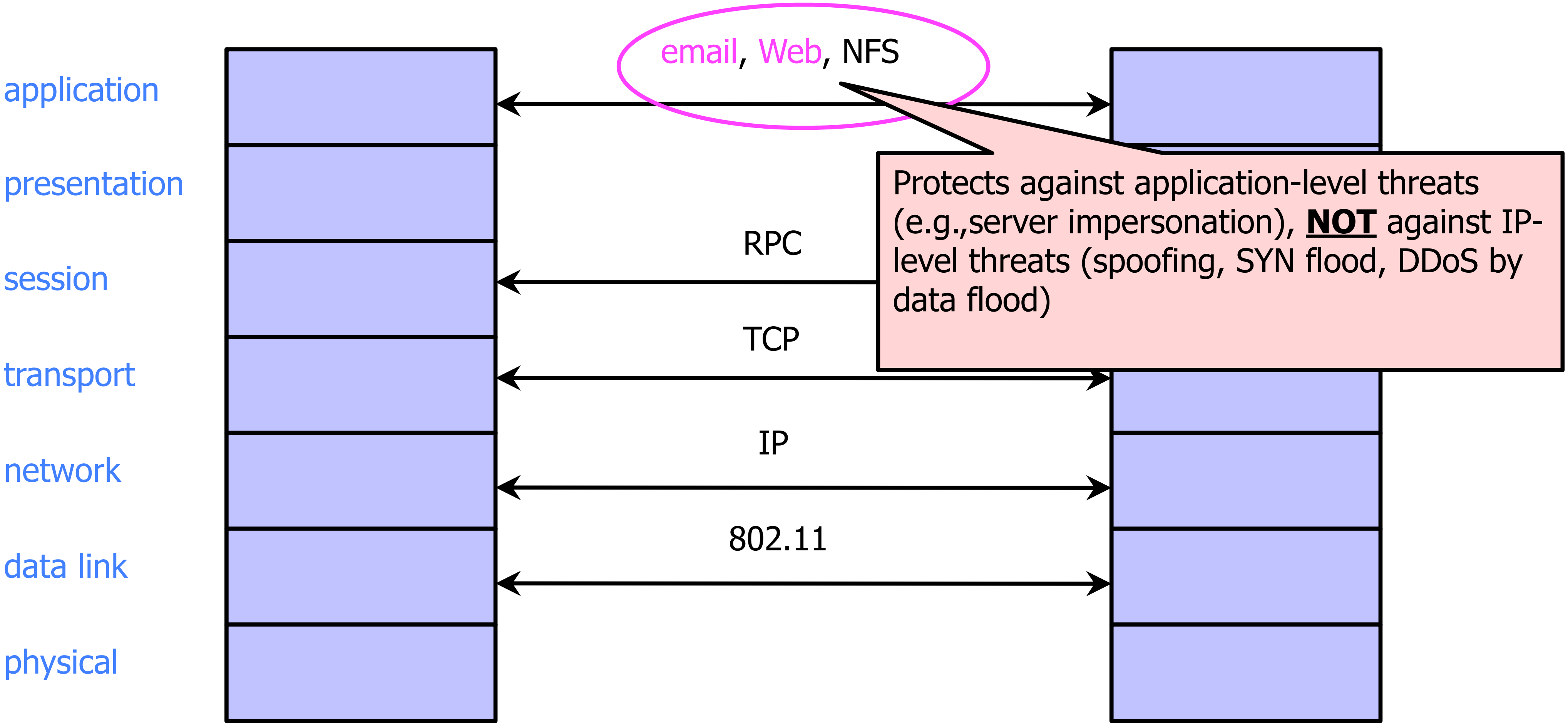
☒ A username must be entered.

Enter your username and password to securely view and manage your Wells Fargo a online.

| | |
|---|---|
| Sign on to | Account Summary |
| Username | |
| Password | |

Username/Password Help

# Application-Level Protection



Protects against application-level threats (e.g.,server impersonation), **NOT** against IP-level threats (spoofing, SYN flood, DDoS by data flood)

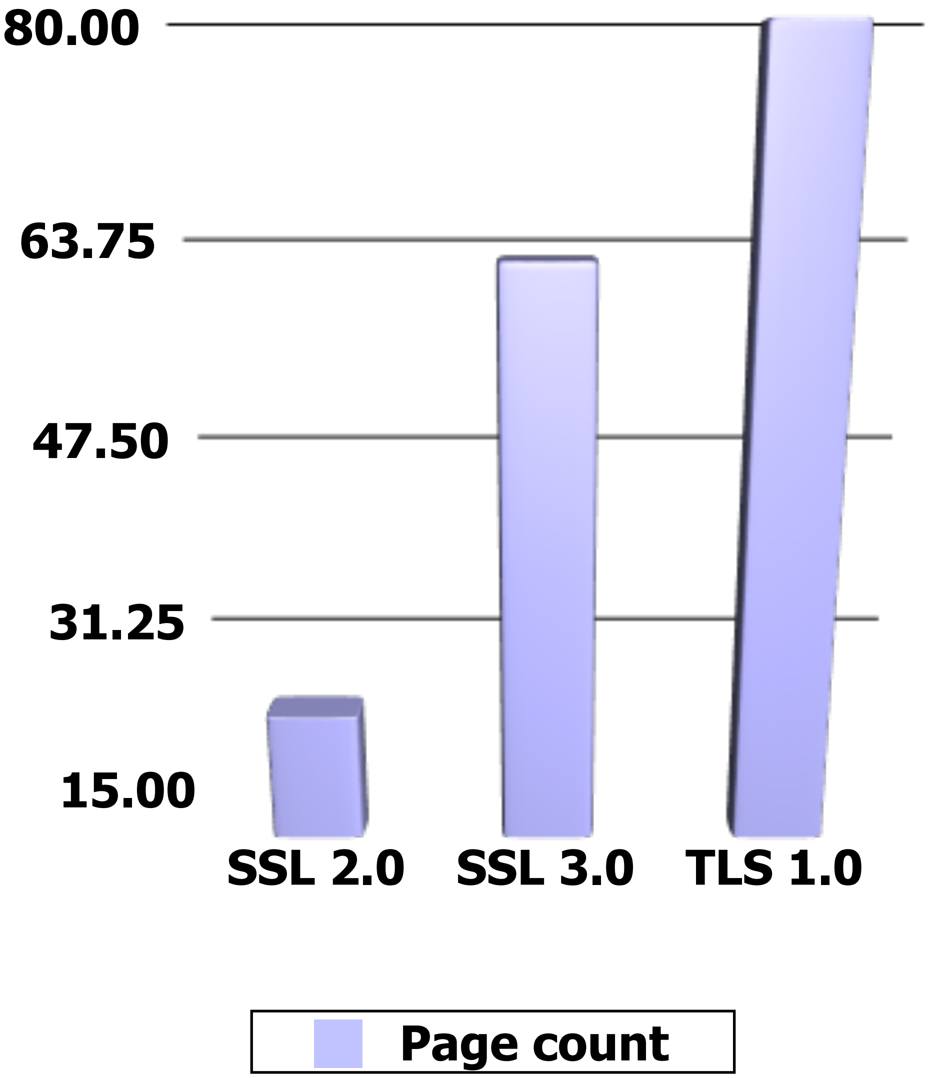
# History of the Protocol

- SSL 1.0
  - Internal Netscape design, early 1994?
  - Lost in the mists of time
- SSL 2.0
  - Published by Netscape, November 1994
  - Several weaknesses
- SSL 3.0
  - Designed by Netscape and Paul Kocher, November 1996
- TLS 1.0
  - Internet standard based on SSL 3.0, January 1999
  - Not interoperable with SSL 3.0
    - TLS uses HMAC instead of earlier MAC; can run on any port
- TLS 1.2
  - Remove dependencies to MD5 and SHA1

# “Request for Comments”

- Network protocols are usually disseminated in the form of an RFC
- TLS version 1.0 is described in RFC 5246
- Intended to be a self-contained definition of the protocol
    - Describes the protocol in sufficient detail for readers who will be implementing it and those who will be doing protocol analysis
    - Mixture of informal prose and pseudo-code

# Evolution of the SSL/TLS RFC



**104 pages for TLS 1.2**

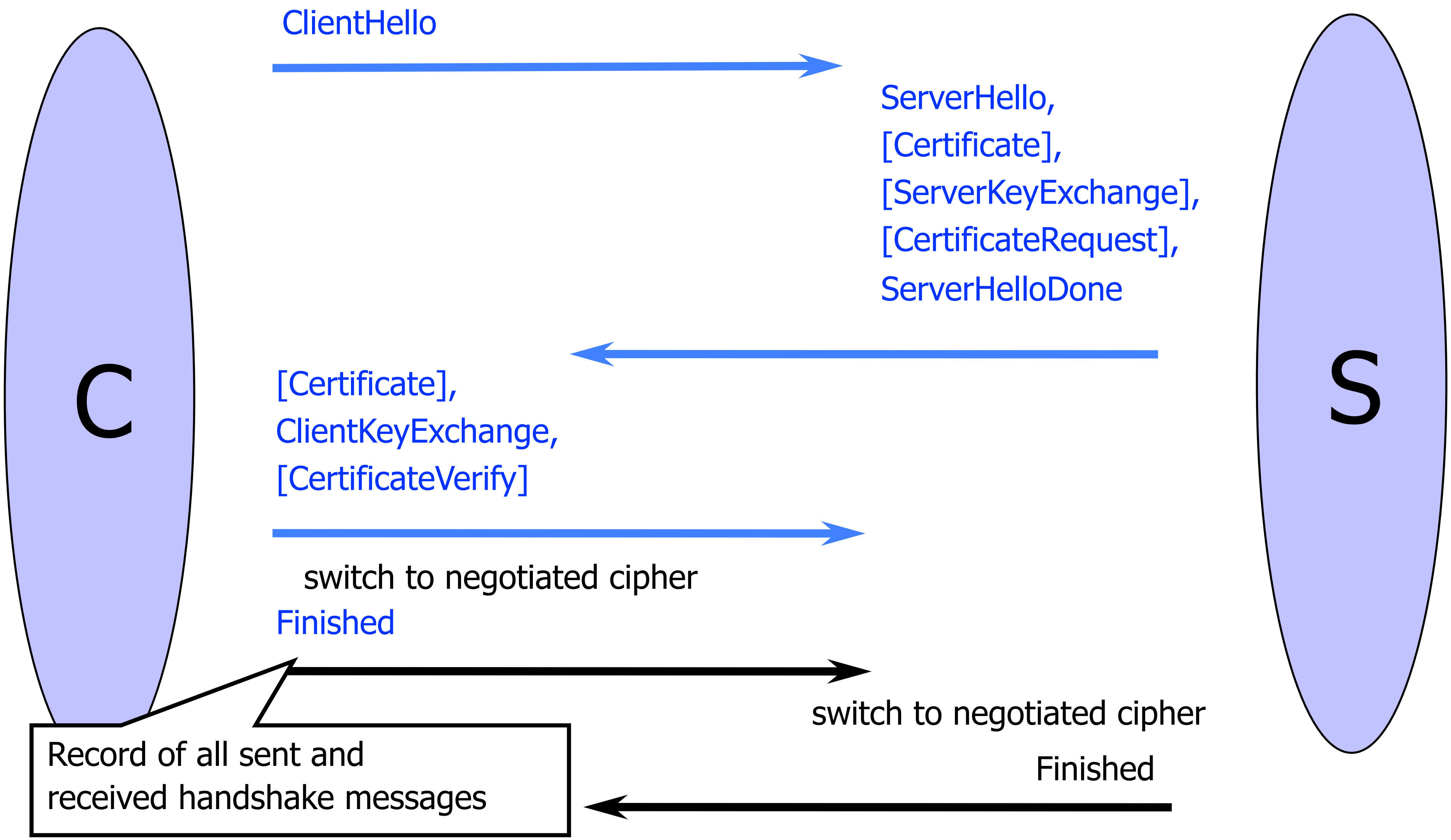
# TLS Basics

- TLS consists of two protocols
  - Familiar pattern for key exchange protocols
- Handshake protocol
  - Use public-key cryptography to establish a shared secret key between the client and the server
- Record protocol
  - Use the secret key established in the handshake protocol to protect communication between the client and the server
- We will focus on the handshake protocol

# TLS Handshake Protocol

- Two parties: client and server
- Negotiate version of the protocol and the set of cryptographic algorithms to be used
  - Interoperability between different implementations of the protocol
- Authenticate client and server (optional)
  - Use digital certificates to learn each other’s public keys and verify each other’s identity
- Use public keys to establish a shared secret

# Handshake Protocol Structure



C

S

ClientHello

ServerHello,
[Certificate],
[ServerKeyExchange],
[CertificateRequest],
ServerHelloDone

[Certificate],
ClientKeyExchange,
[CertificateVerify]

switch to negotiated cipher

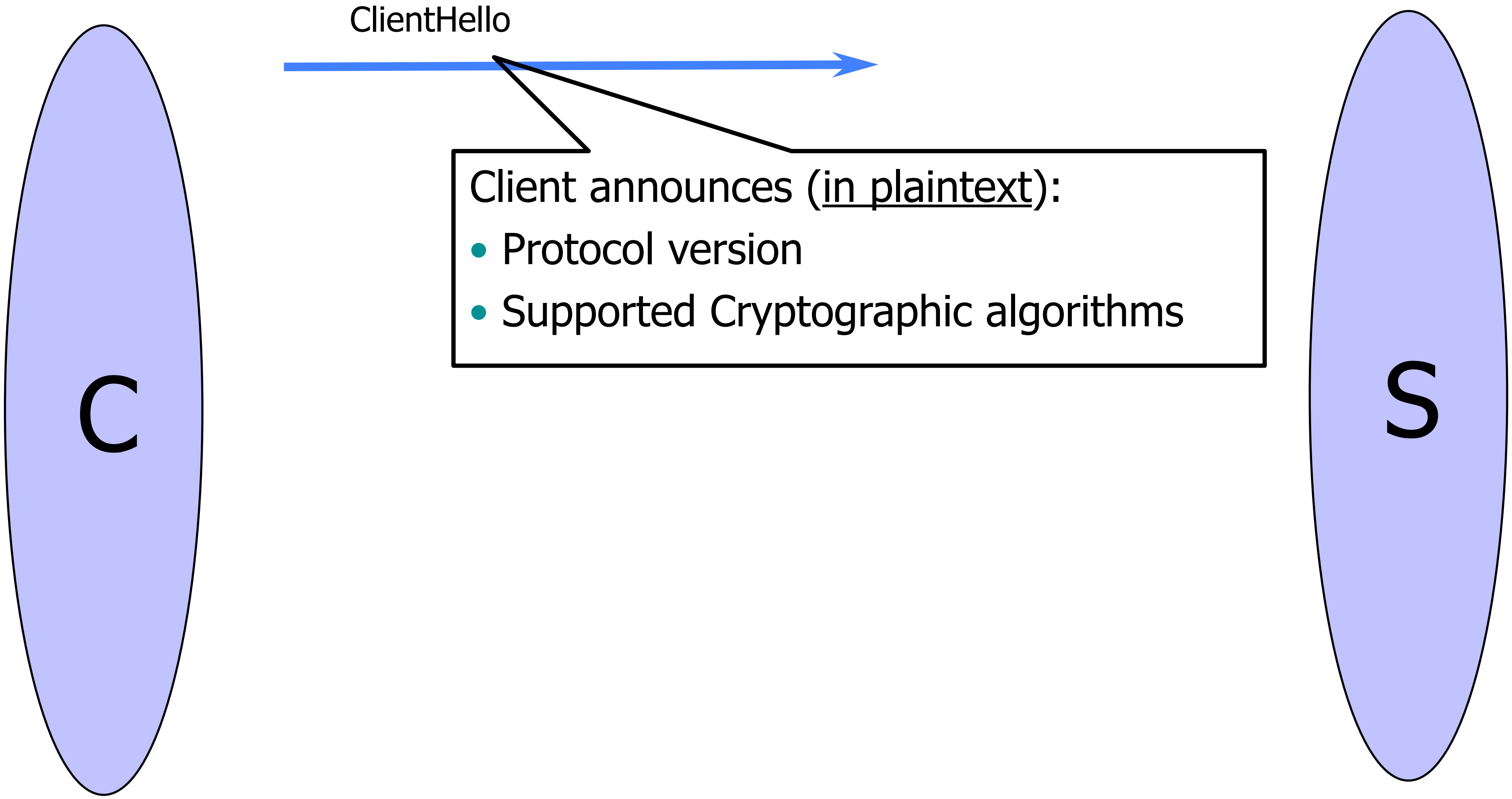
Finished

switch to negotiated cipher

Finished

Record of all sent and
received handshake messages

# ClientHello



Client announces (in plaintext):

- Protocol version
- Supported Cryptographic algorithms

# ClientHello (RFC)

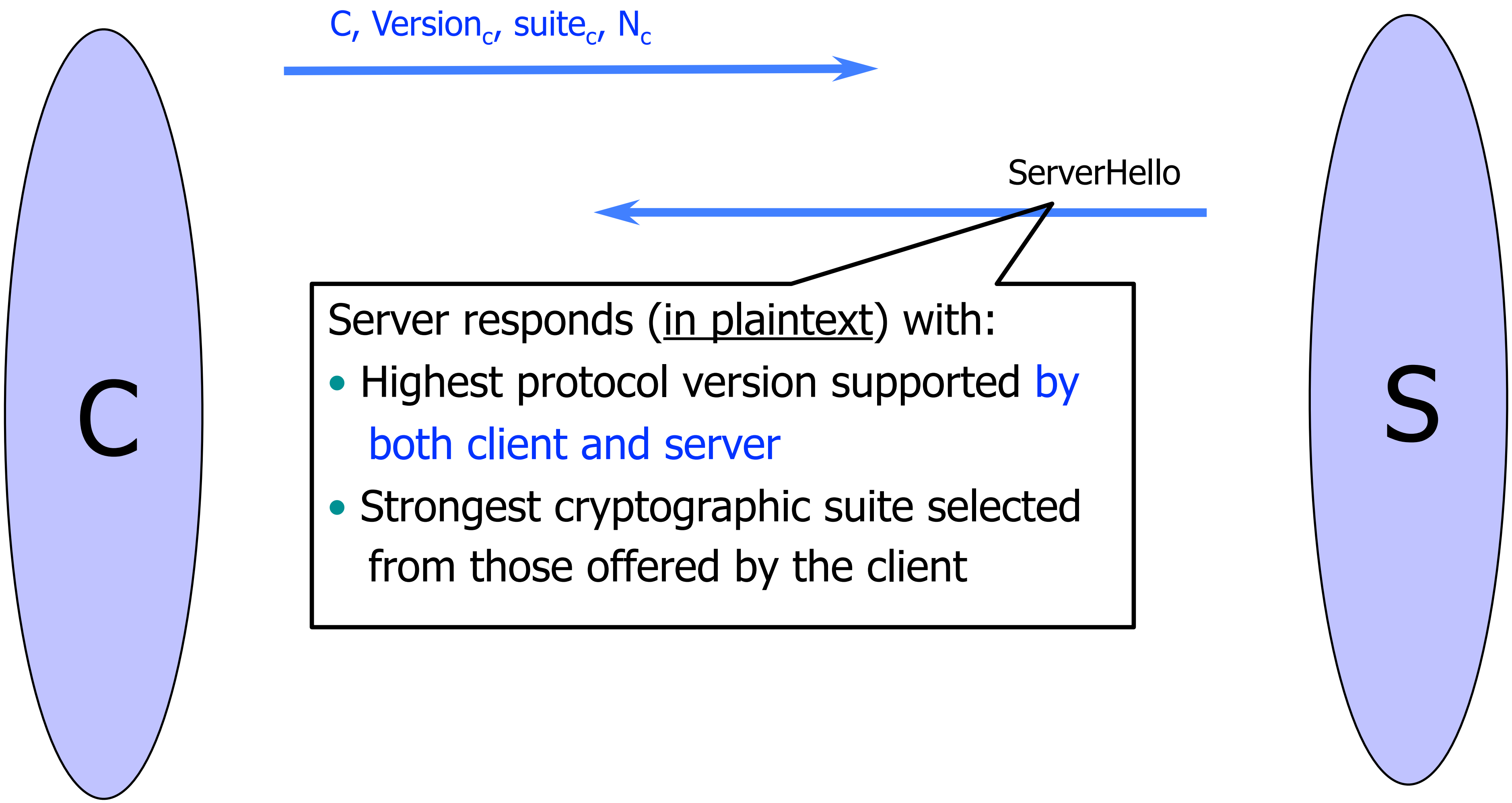
```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites;
    CompressionMethod compression_methods;
} ClientHello
```

Highest version of the protocol supported by the client

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

Set of cryptographic algorithms supported by the client (e.g., RSA or Diffie-Hellman)

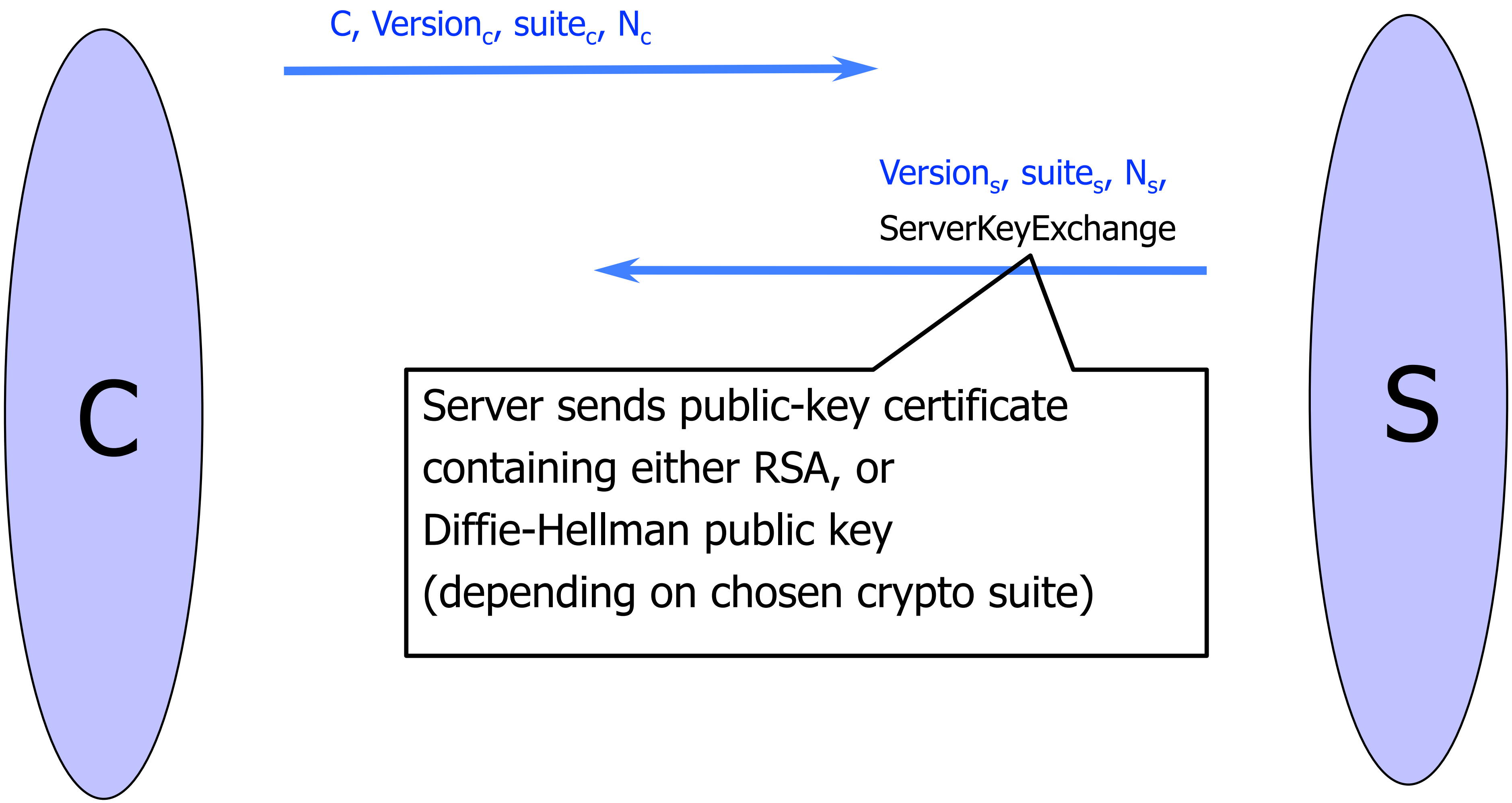
# ServerHello



Server responds (in plaintext) with:

- Highest protocol version supported by both client and server
- Strongest cryptographic suite selected from those offered by the client

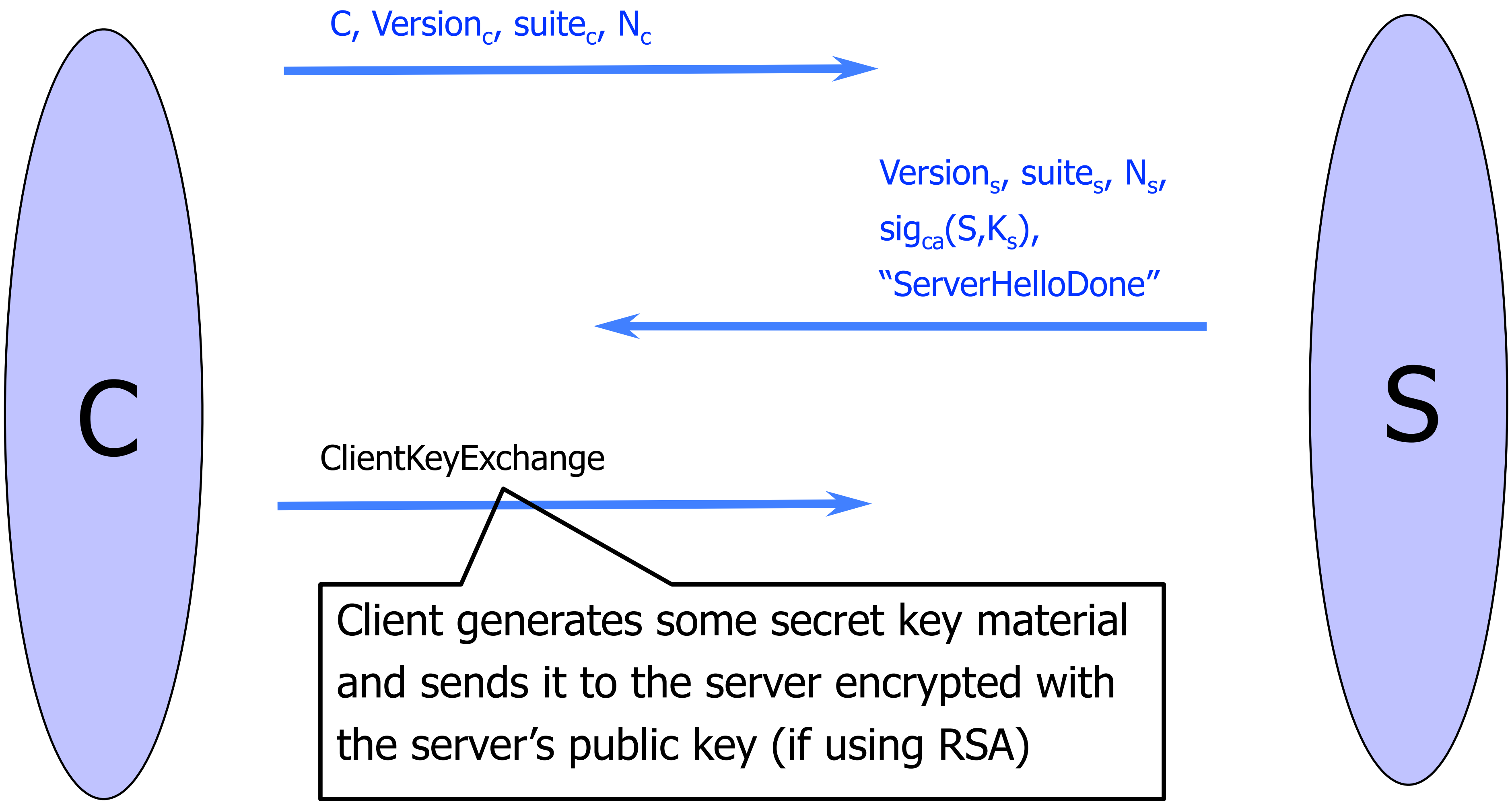
# ServerKeyExchange



C
$C$, Version$_c$, suite$_c$, $N_c$
Version$_s$, suite$_s$, $N_s$,
ServerKeyExchange
S

Server sends public-key certificate containing either RSA, or Diffie-Hellman public key (depending on chosen crypto suite)

# ClientKeyExchange



C

S

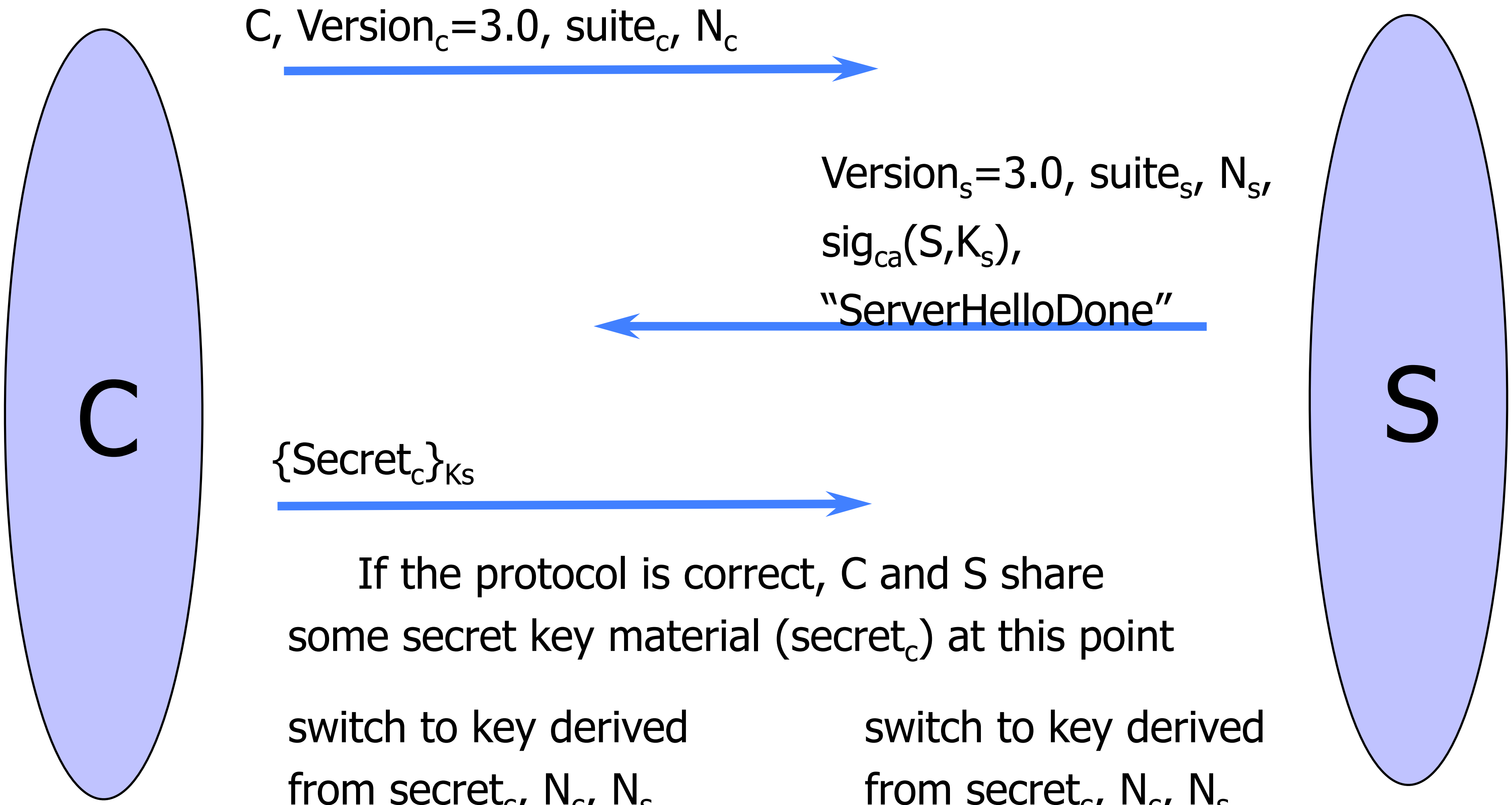
C, $\mathrm{Version}_c$, $\mathrm{suite}_c$, $N_c$

$\mathrm{Version}_s$, $\mathrm{suite}_s$, $N_s$,
$\mathrm{sig}_{ca}(S,K_s)$,
“ServerHelloDone”

ClientKeyExchange

Client generates some secret key material and sends it to the server encrypted with the server’s public key (if using RSA)

# “Core” SSL 3.0 Handshake (Not TLS)



C, Version$_c$=3.0, suite$_c$, $N_c$

Version$_s$=3.0, suite$_s$, $N_s$,
sig$_{ca}$(S,$K_s$),
“ServerHelloDone”

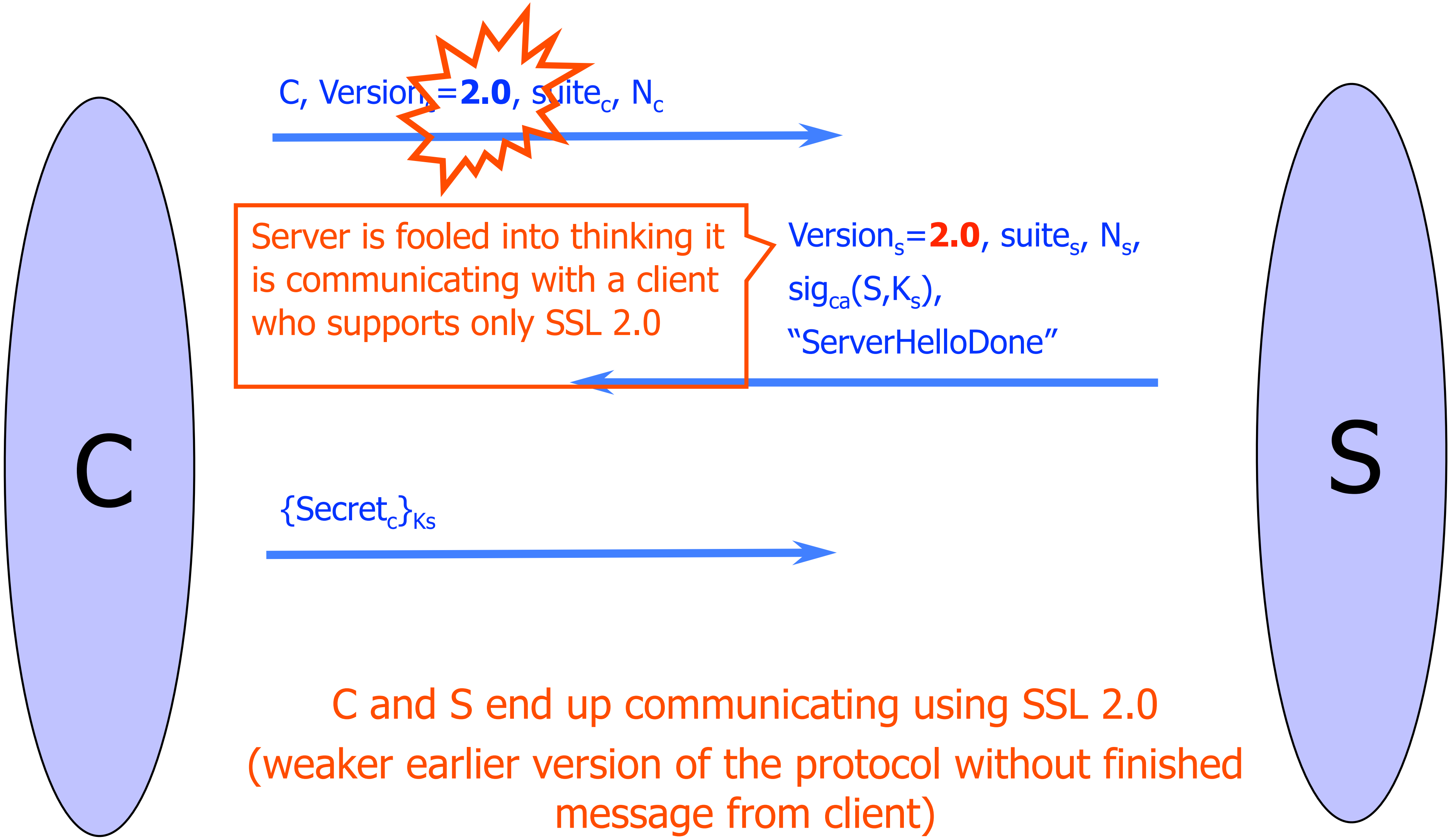
{Secret$_c$}$_{Ks}$

If the protocol is correct, C and S share some secret key material (secret$_c$) at this point

switch to key derived from secret$_c$, $N_c$, $N_s$

switch to key derived from secret$_c$, $N_c$, $N_s$

# Version Rollback Attack



C, Version$_c$=**2.0**, suite$_c$, $N_c$

Server is fooled into thinking it is communicating with a client who supports only SSL 2.0

Version$_s$=**2.0**, suite$_s$, $N_s$,
sig$_{ca}$(S,$K_s$),
"ServerHelloDone"

{Secret$_c$}$_{Ks}$

C and S end up communicating using SSL 2.0
(weaker earlier version of the protocol without finished message from client)

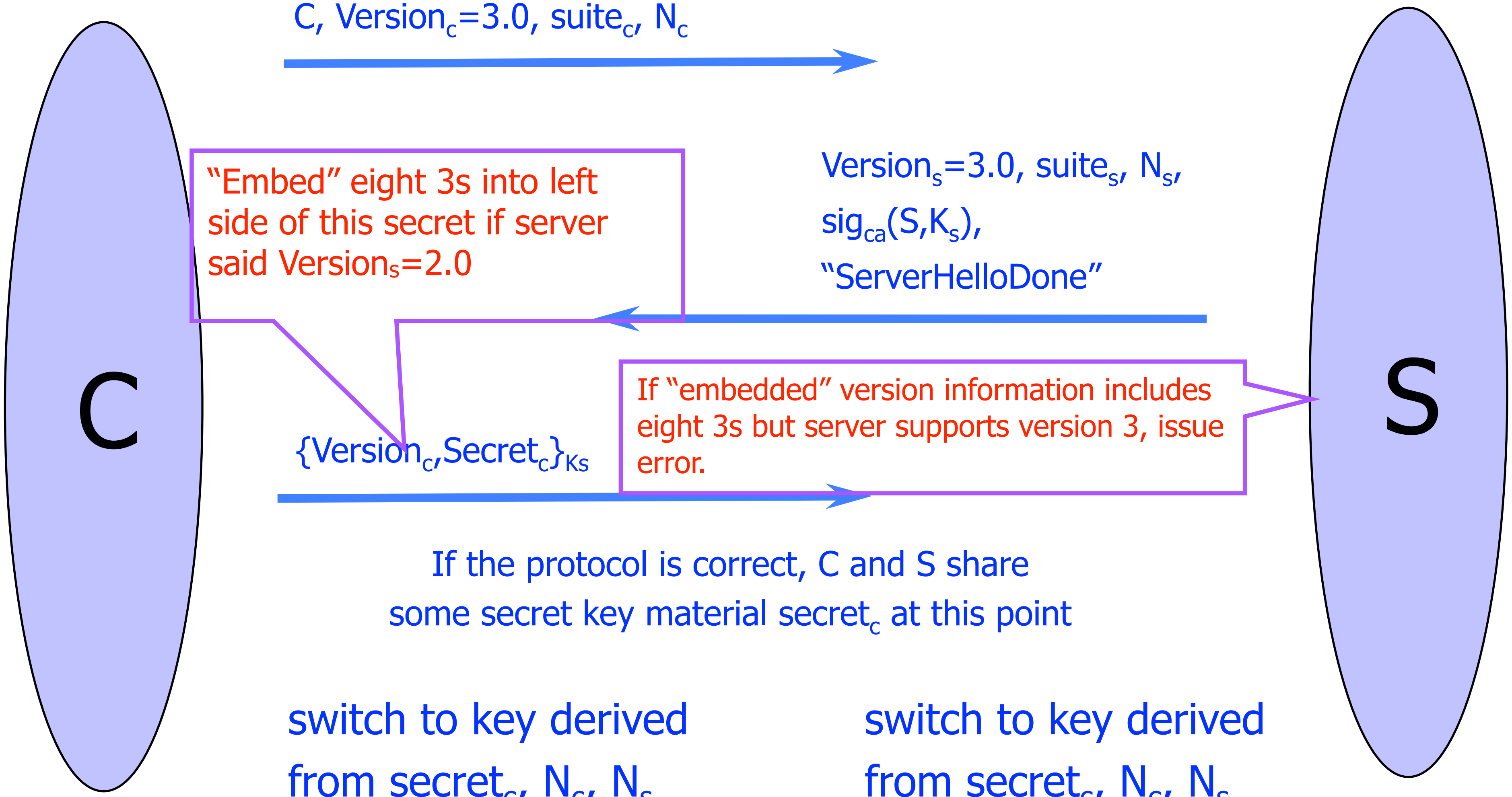
# SSL 2.0 Weaknesses (Fixed in 3.0)

- Cipher suite preferences are not authenticated
  - "Cipher suite rollback" attack is possible
- SSL 2.0 uses padding when computing MAC in block cipher modes, but padding length field is not authenticated
  - Attacker can delete bytes from the end of messages
- MAC hash uses only 40 bits in export mode
- No support for certificate chains or non-RSA algorithms, no handshake while session is open

# Protocol Rollback Attacks

- Why do people release new versions of security protocols? Because the old version got broken!
- New version must be backward-compatible
  - Not everybody upgrades right away
- Attacker can fool someone into using the old, broken version and exploit known vulnerability
  - Similar: fool victim into using weak crypto algorithms
- Defense is hard: must authenticate version in early designs
- Many protocols had "version rollback" attacks
  - SSL, SSH, GSM (cell phones)

# Version Check in SSL 3.0 (Approximate)



C, $\text{Version}_c$=3.0, $\text{suite}_c$, $N_c$

$\text{Version}_s$=3.0, $\text{suite}_s$, $N_s$,
$\text{sig}_{ca}(S,K_s)$,
"ServerHelloDone"

"Embed" eight 3s into left side of this secret if server said $\text{Version}_S$=2.0

$\{\text{Version}_c,\text{Secret}_c\}_{Ks}$

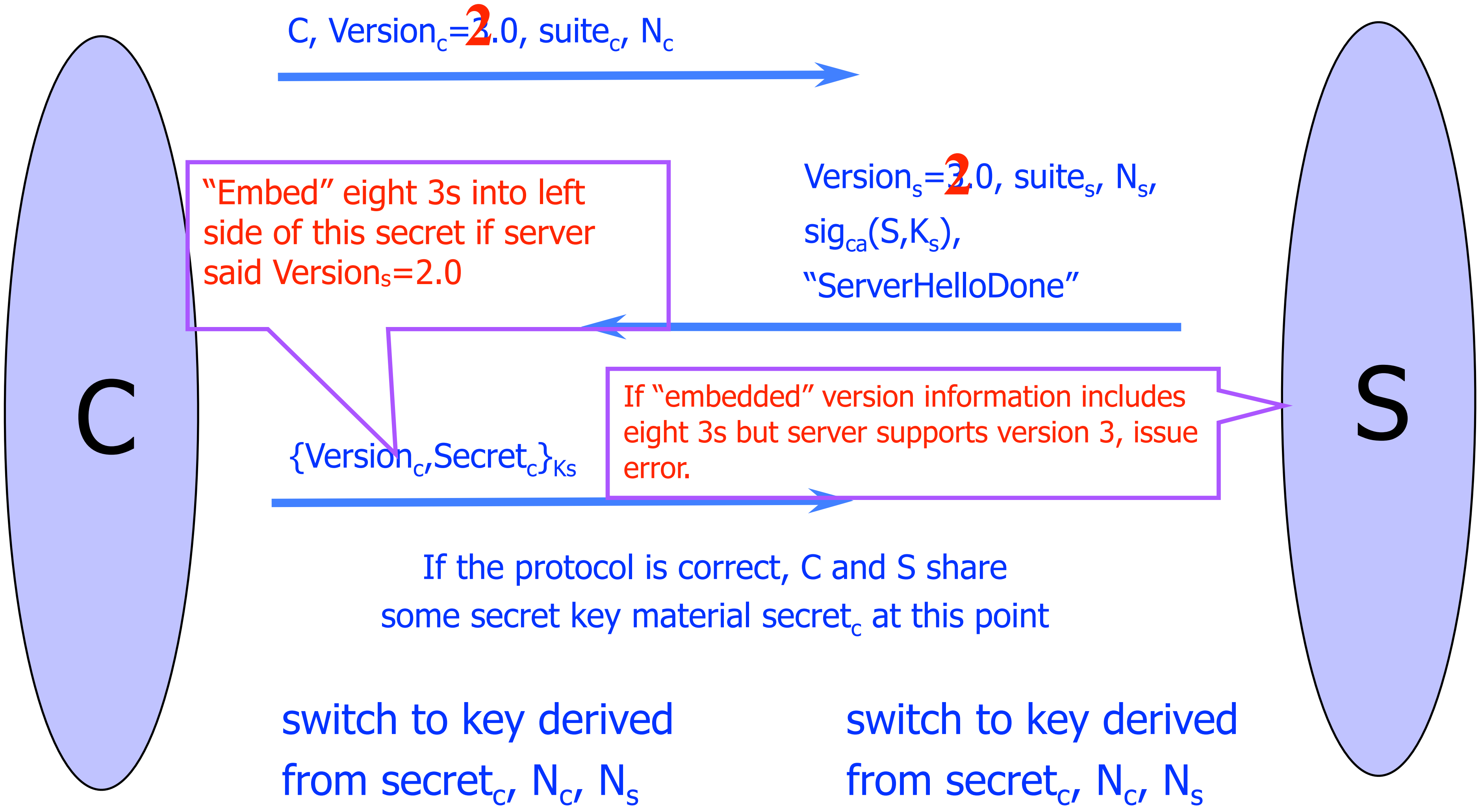
If "embedded" version information includes eight 3s but server supports version 3, issue error.

If the protocol is correct, C and S share some secret key material $\text{secret}_c$ at this point

switch to key derived from $\text{secret}_c$, $N_c$, $N_s$

switch to key derived from $\text{secret}_c$, $N_c$, $N_s$

# Version Check in SSL 3.0 (Approximate)



C, Version$_c$=~~3~~**2**.0, suite$_c$, N$_c$

Version$_s$=~~3~~**2**.0, suite$_s$, N$_s$, sig$_{ca}$(S,K$_s$), "ServerHelloDone"

"Embed" eight 3s into left side of this secret if server said Version$_S$=2.0

{Version$_c$,Secret$_c$}$_{Ks}$

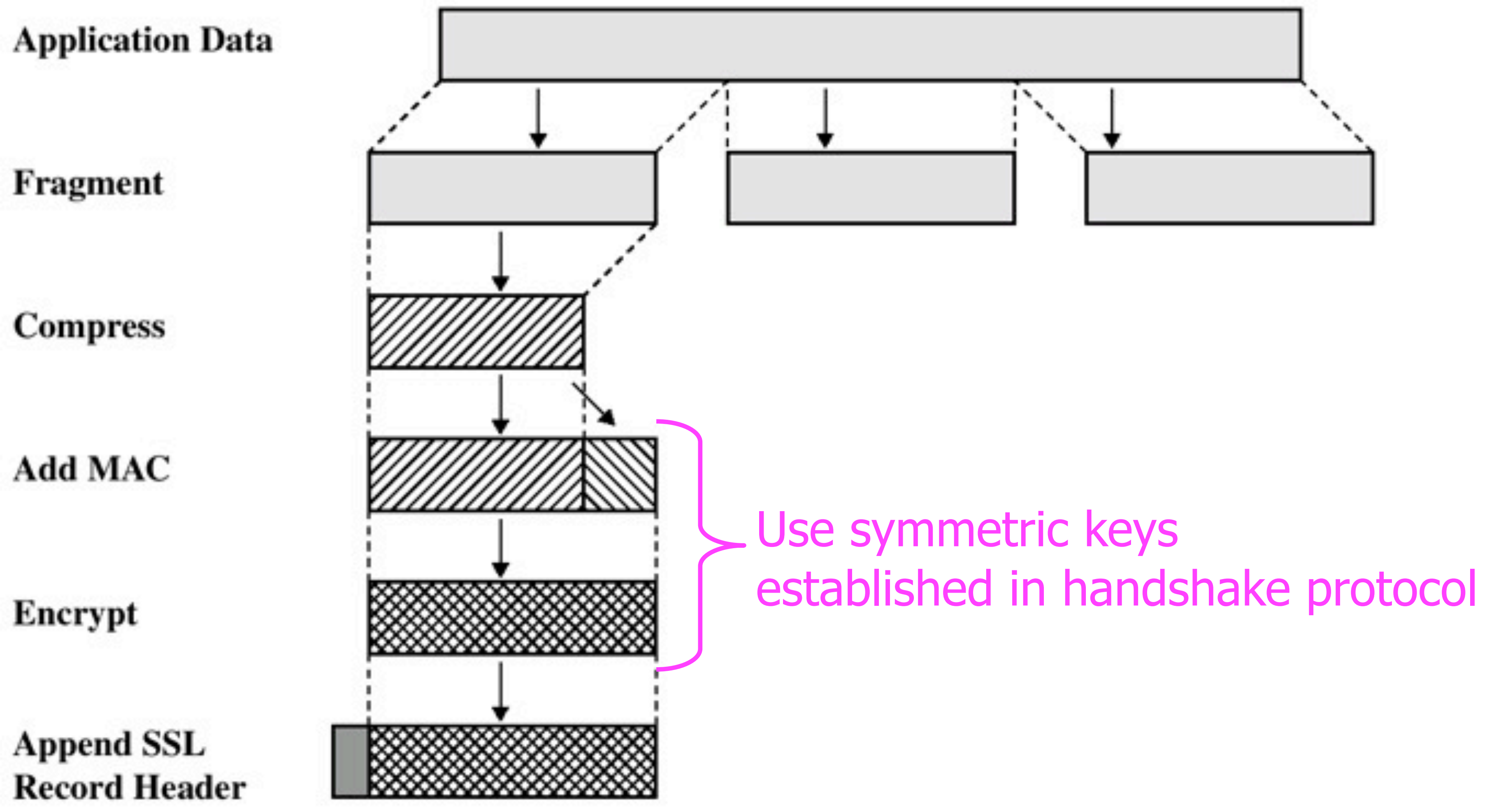
If "embedded" version information includes eight 3s but server supports version 3, issue error.

If the protocol is correct, C and S share some secret key material secret$_c$ at this point

switch to key derived from secret$_c$, N$_c$, N$_s$

switch to key derived from secret$_c$, N$_c$, N$_s$

# SSL/TLS Record Protection



Use symmetric keys
established in handshake protocol