CSE 484 (Winter 2011)

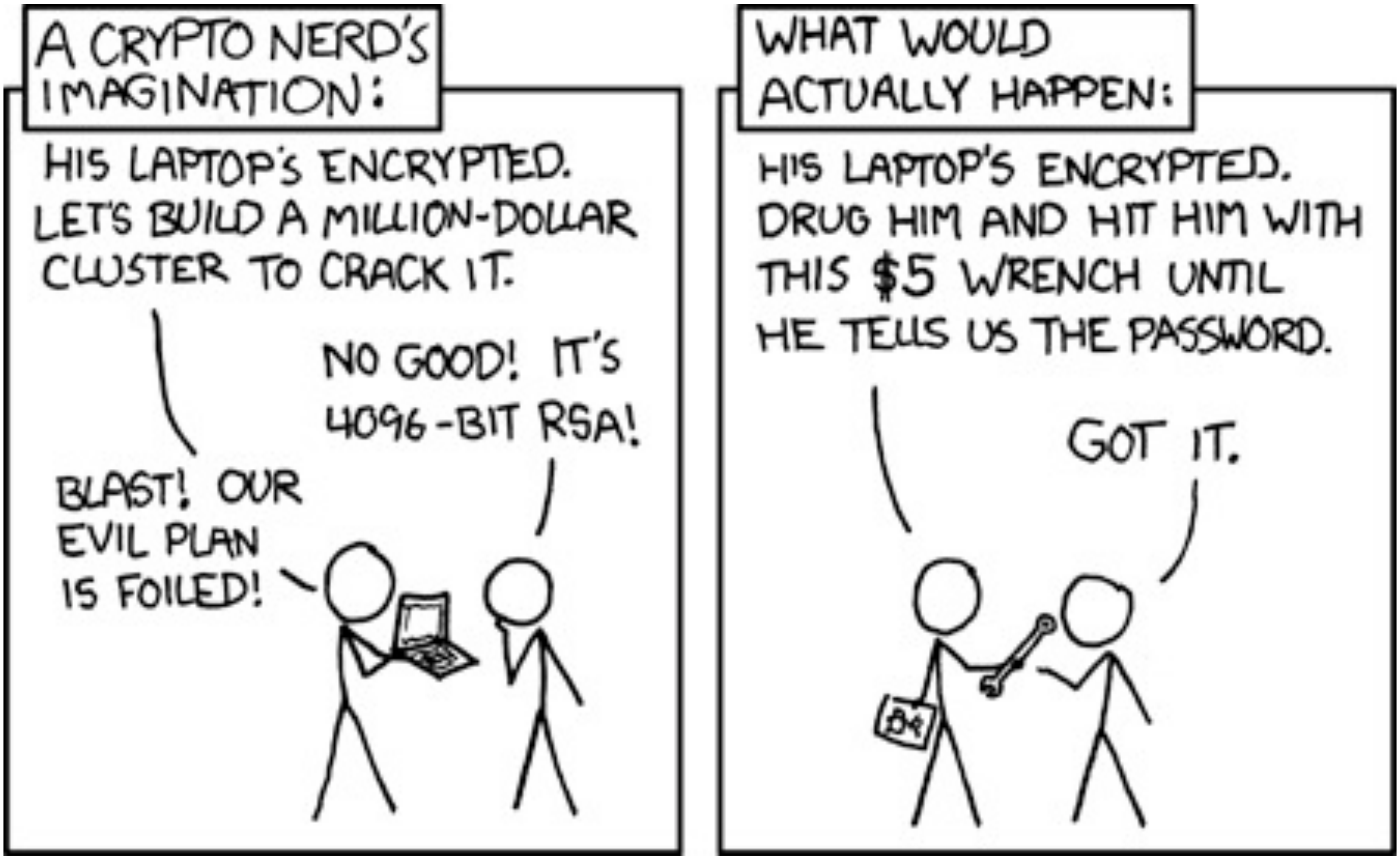
Asymmetric Cryptography

Tadayoshi Kohno

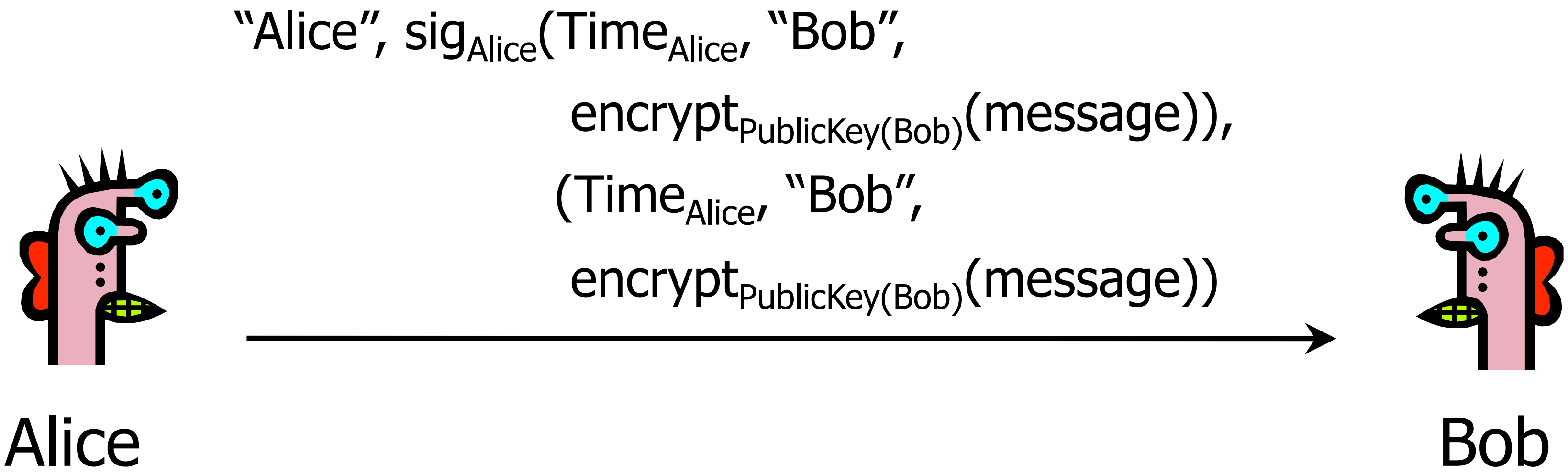
Thanks to Dan Boneh, Dieter Gollmann, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Goals for Today

- Asymmetric Cryptography

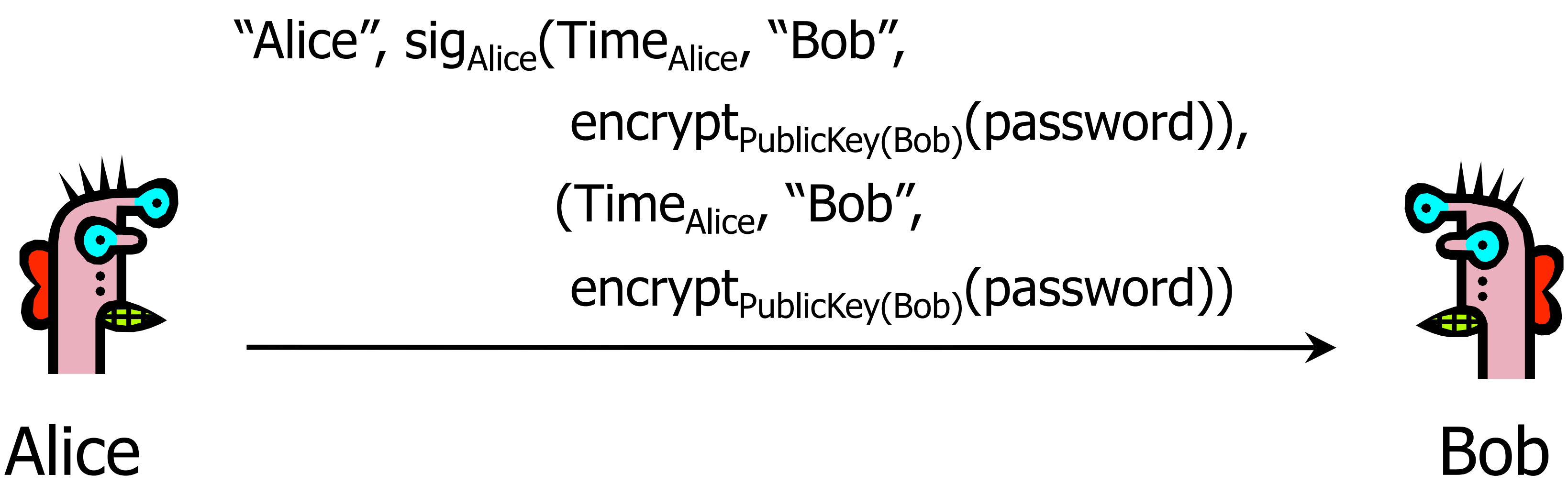

A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

X.509 Version 1



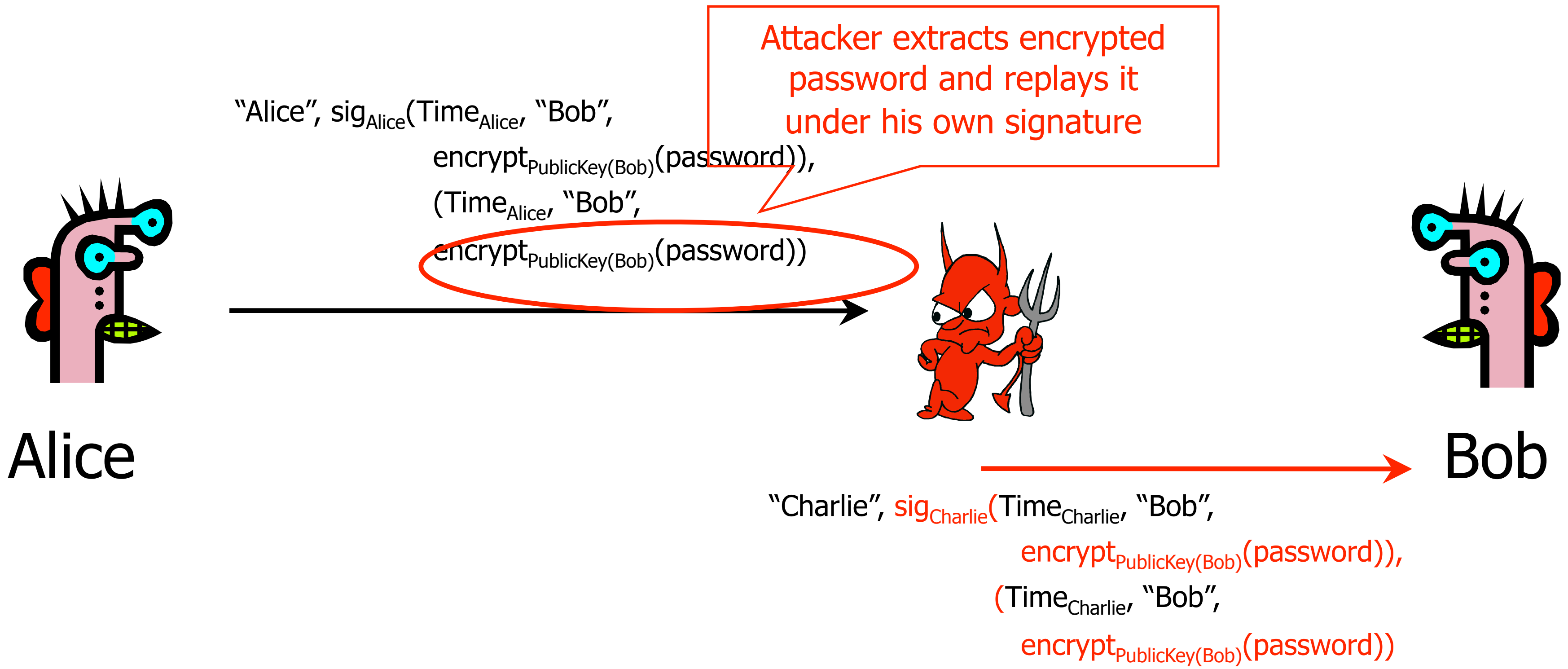
- Encrypt, then sign
  - Goal: achieve both confidentiality and authentication
  - E.g., encrypted, signed password for access control (for next slide: assume one password for whole system)
- Does this work?

X.509 Version 1 (message is passwd)



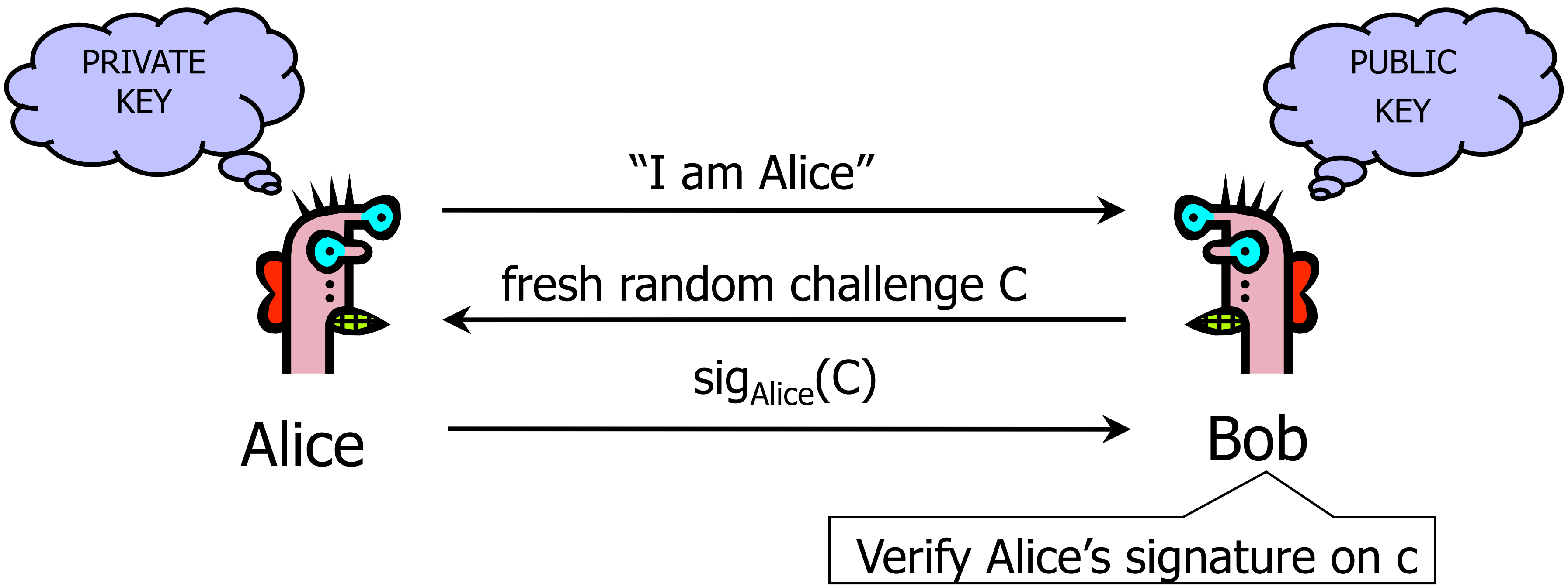
- Encrypt, then sign
  - Goal: achieve both confidentiality and authentication
  - E.g., encrypted, signed password for access control (for next slide: assume one password for whole system)
- Does this work?

Attack on X.509 Version 1



- Receiving encrypted password under signature does <u>not</u> mean that the sender actually knows the password!

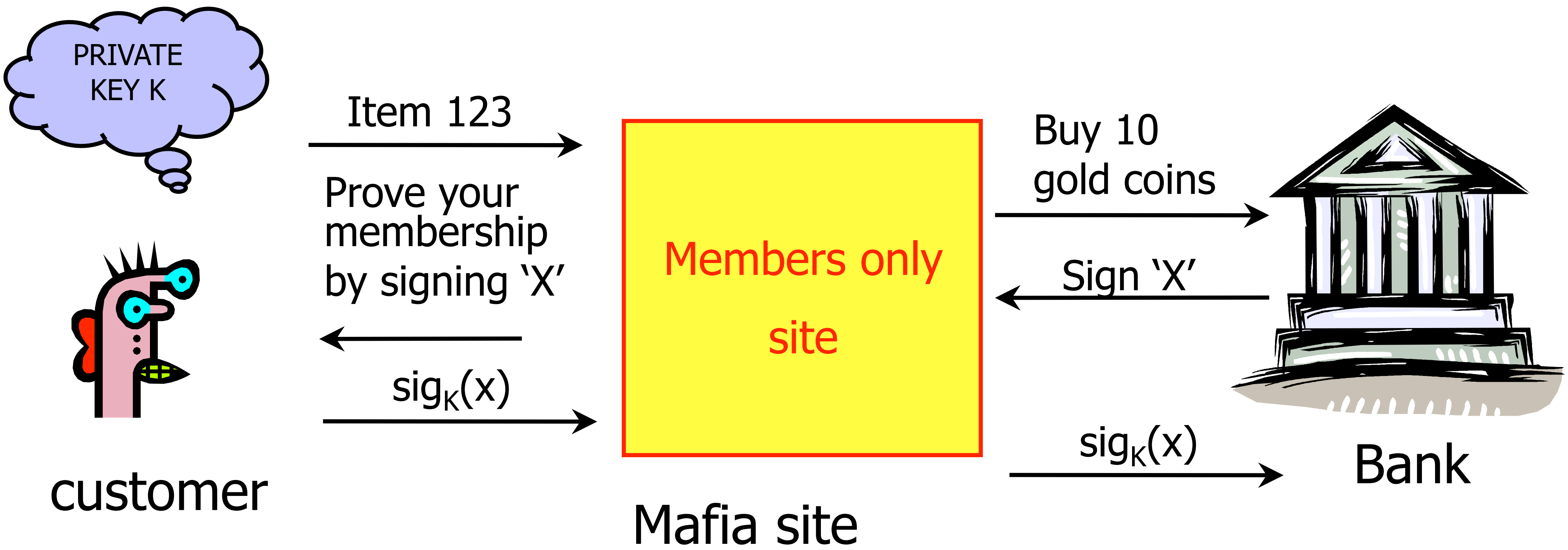
Authentication with Public Keys



1. Only Alice can create a valid signature
2. Signature is on a fresh, unpredictable challenge

Potential problem: Alice will sign anything

Mafia-in-the-Middle Attack [from Anderson's book]



One key recommendation: Don't use same public key / secret key pair for multiple applications. (Or make sure messages have different formats across applications.)

Secure Sessions

- Secure sessions are among the most important applications in network security
  - Enable us to talk securely on an insecure network
- Goal: secure bi-directional communication channel between two parties
  - The channel must provide confidentiality
    - Third party cannot read messages on the channel
  - The channel must provide authentication
    - Each party must be sure who the other party is
  - Other desirable properties: integrity, protection against denial of service, anonymity against eavesdroppers

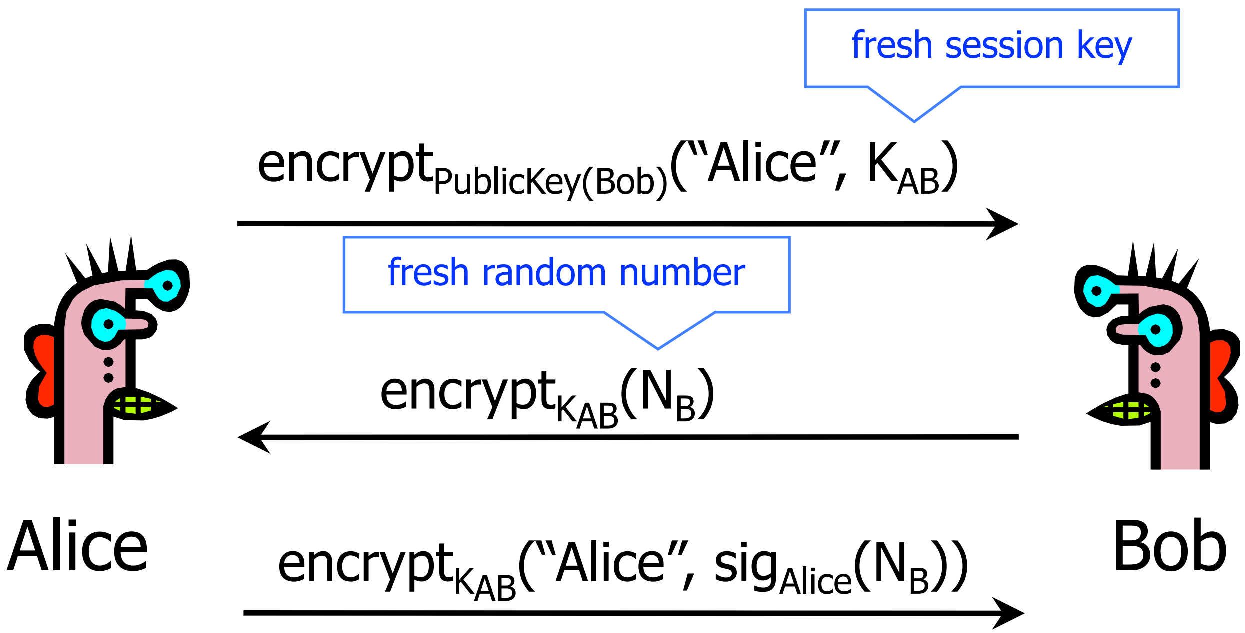
Key Establishment Protocols

- Common implementation of secure sessions:
  - Establish a secret key known only to two parties
  - Then use block ciphers for confidentiality, HMAC for authentication, and so on
- Challenge: how to establish a secret key
  - Using only <u>public</u> information?
  - Even if the two parties share a long-term secret, a fresh key should be created for each session
    - Long-term secrets are valuable; want to use them as sparingly as possible to limit exposure and the damage if the key is compromised
    - (Background: For N parties, there are N choose 2 = N*(N-1)/2 pairs of parties.)

Key Establishment Techniques

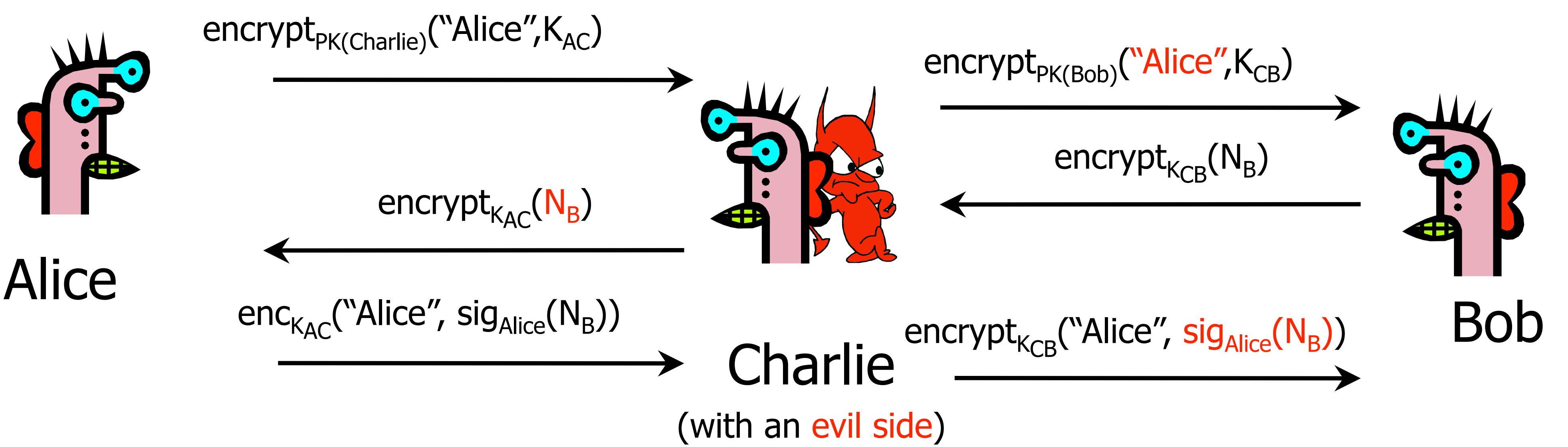
- Use a trusted key distribution center (KDC)
  - Every party shares a pairwise secret key with KDC
  - KDC creates a new random session key and then distributes it, encrypted under the pairwise keys
    - Example: Kerberos
- Use public-key cryptography
  - Diffie-Hellman authenticated with signatures
    - Example: IKE (Internet Key Exchange)
  - One party creates a random key, sends it encrypted under the other party's public key
    - Example: TLS (Transport Layer Security)

Early Version of SSL (Simplified)



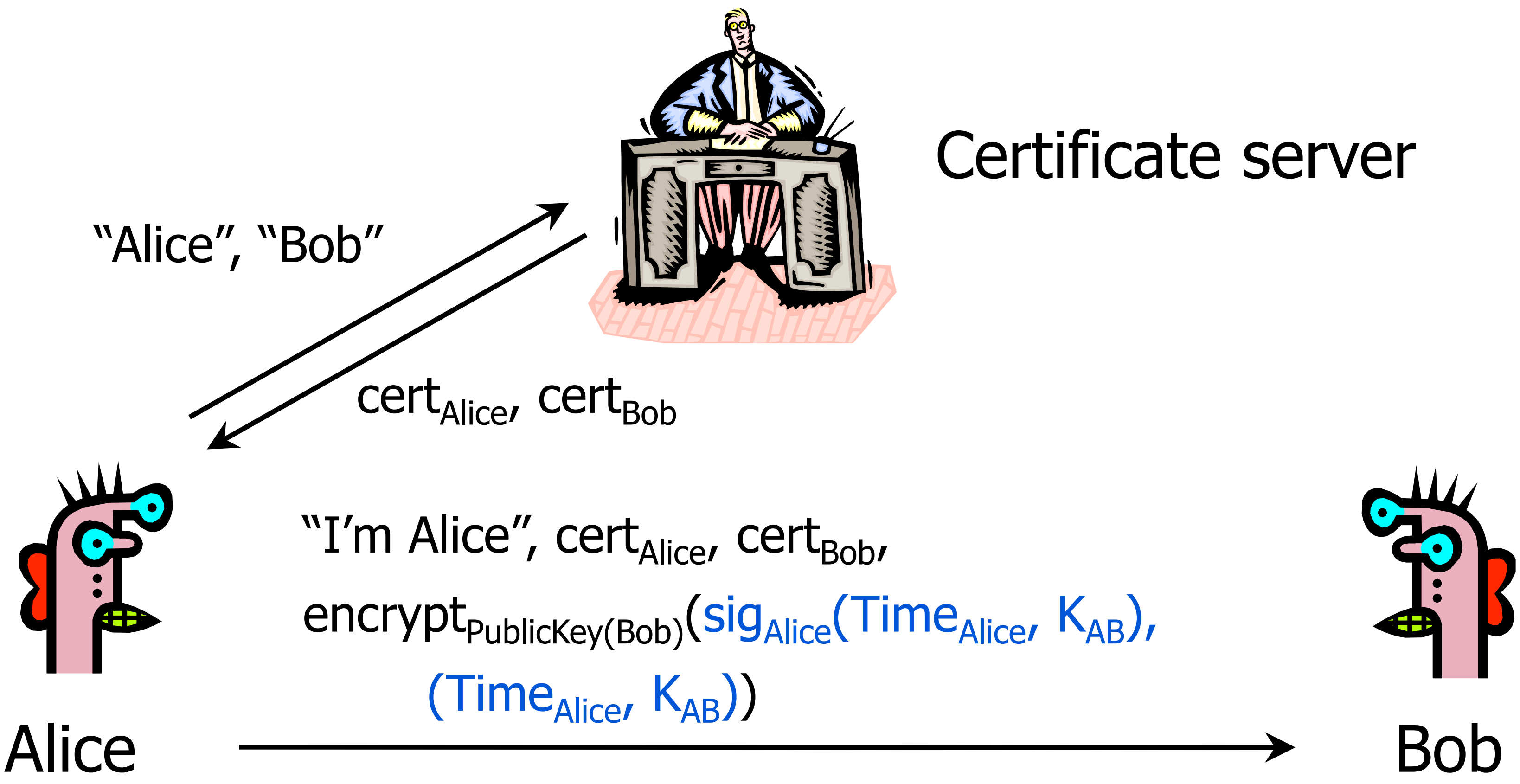
- Bob's reasoning: I must be talking to Alice because…
  - Whoever signed N_B knows Alice's private key… Only Alice knows her private key… Alice must have signed N_B… N_B is fresh and random and I sent it encrypted under K_{AB}… Alice could have learned N_B only if she knows K_{AB}… She must be the person who sent me K_{AB} in the first message...

Breaking Early SSL



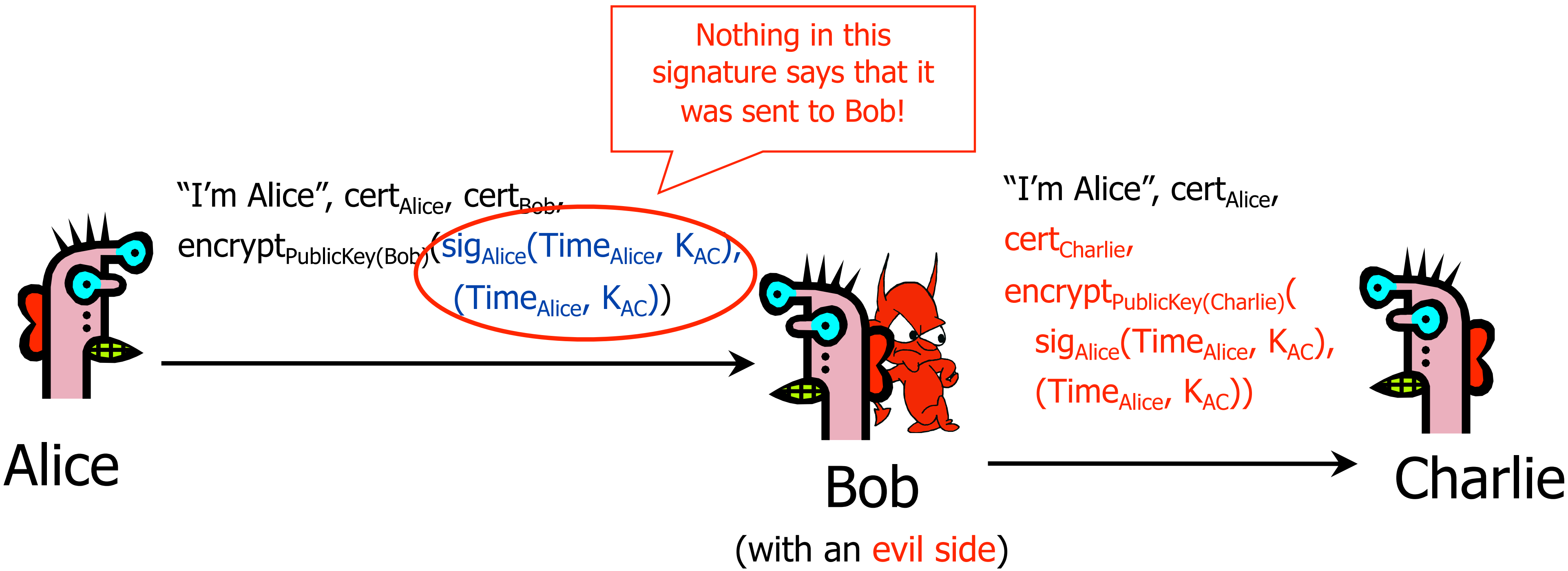
- Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob
  - Information signed by Alice is not sufficiently explicit

Denning-Sacco Protocol



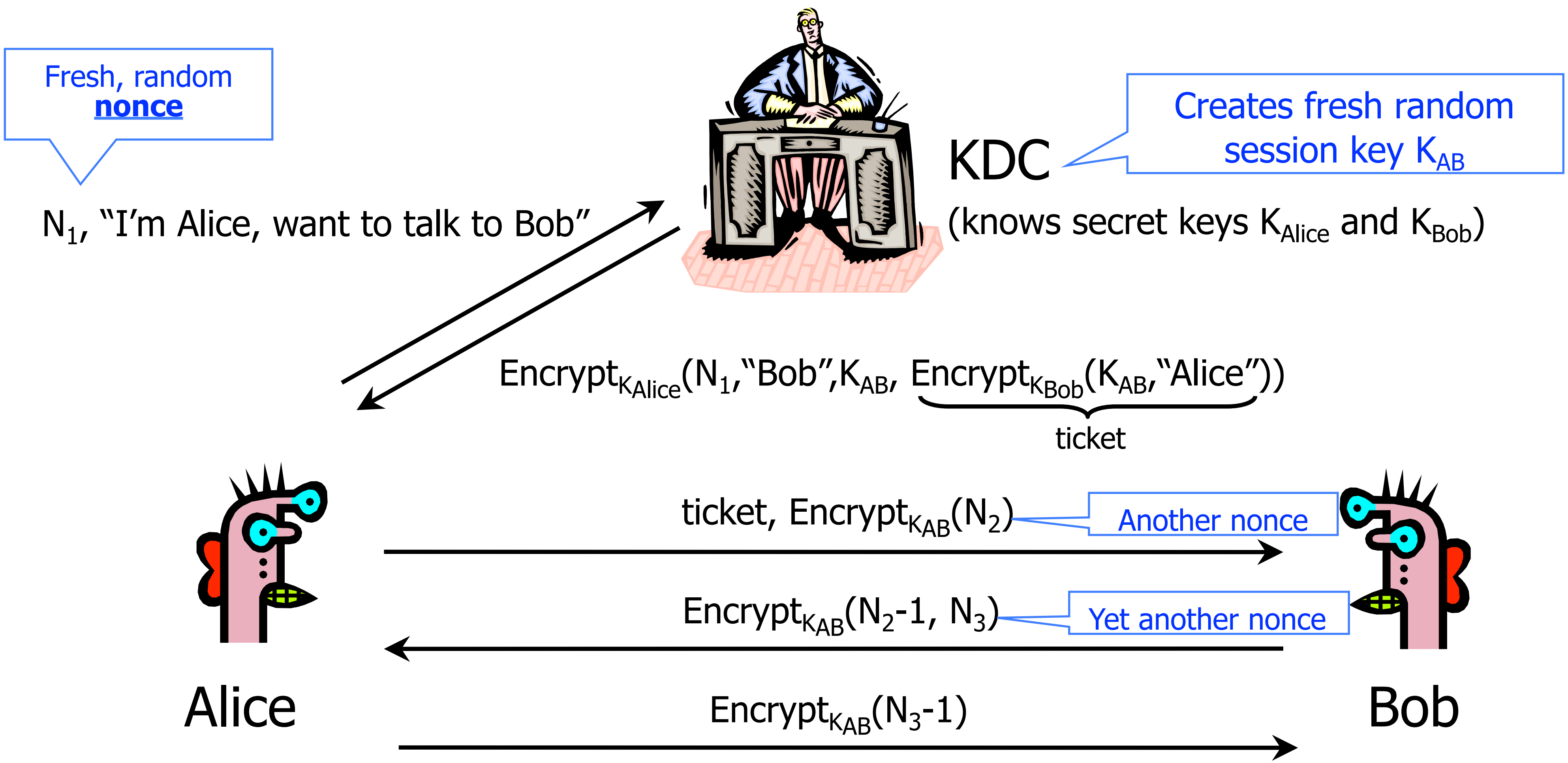
- Goal: establish a new shared key K_{AB} with the help of a trusted certificate service

Attack on Denning-Sacco



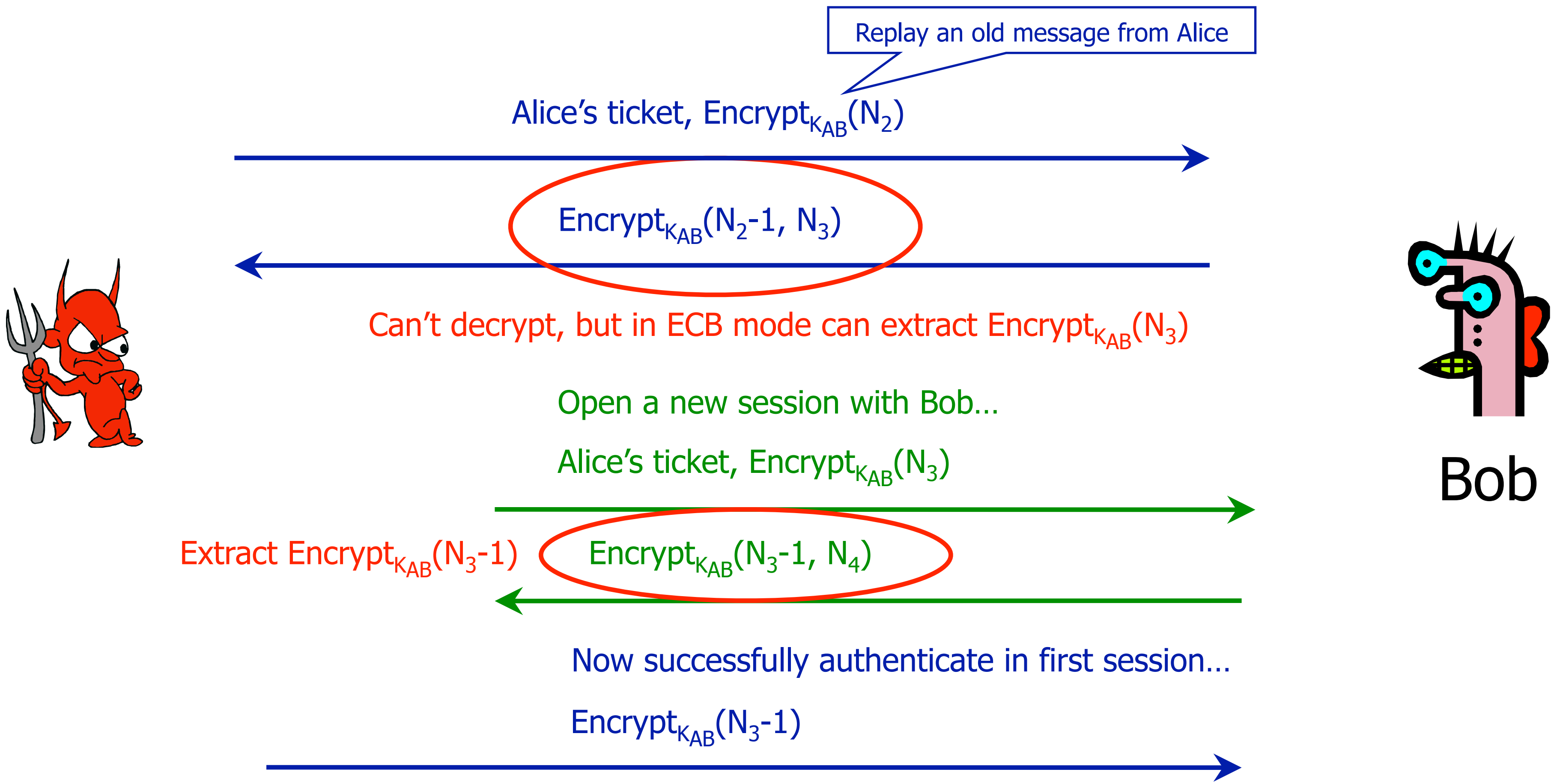
- Alice's signature is insufficiently explicit
  - Does not say to whom and why it was sent
- Alice's signature can be used to impersonate her

Private-Key Needham-Schroeder

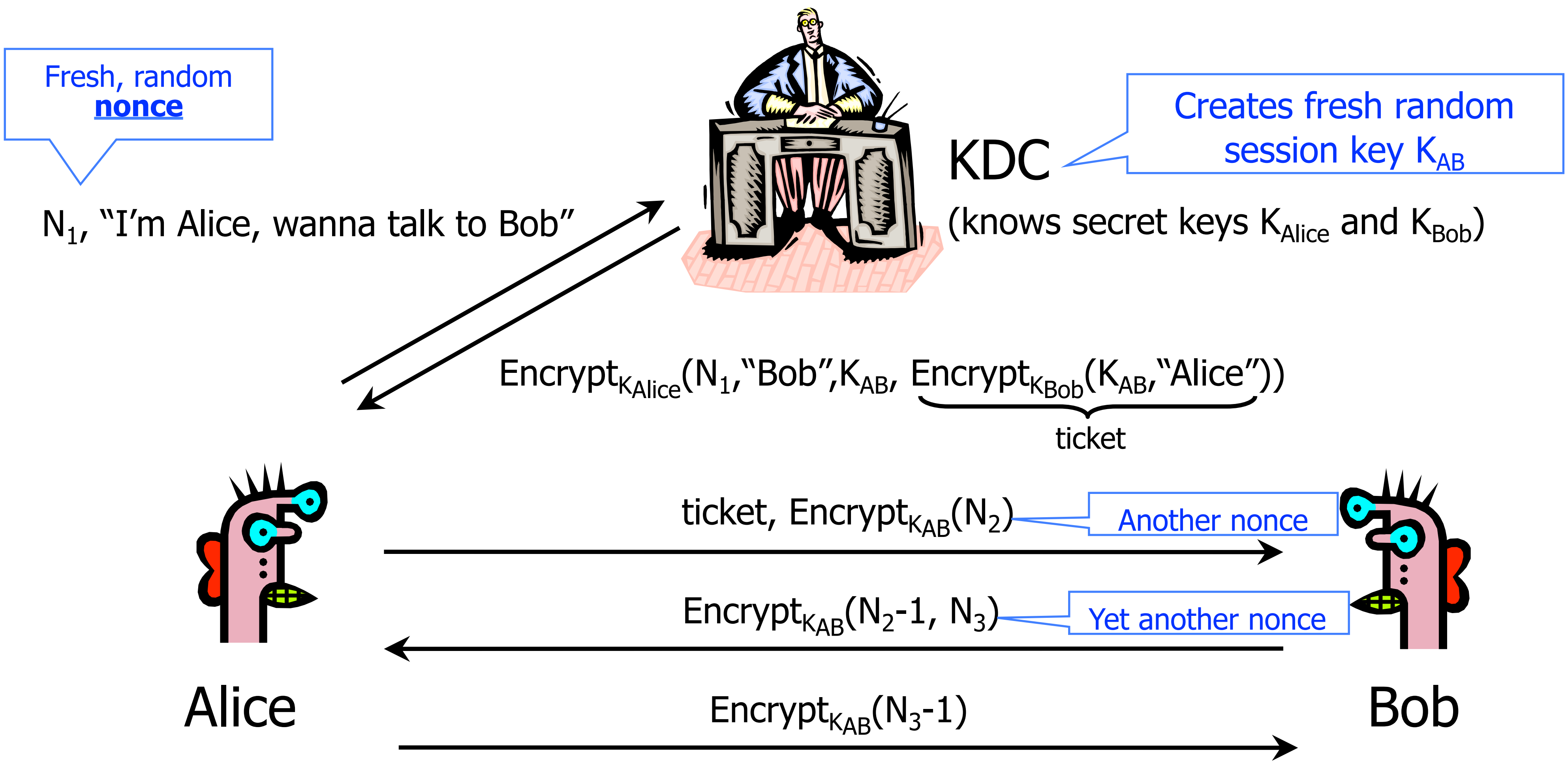


Reflection Attack

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

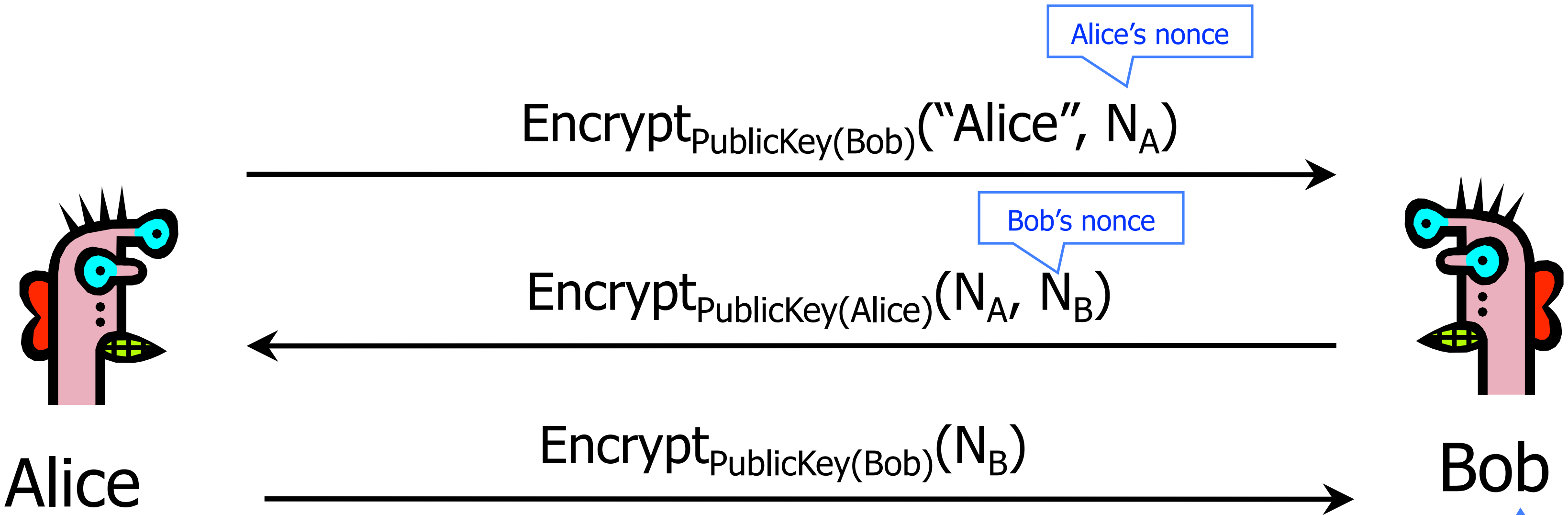


Private-Key Needham-Schroeder



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

Public-Key Needham-Schroeder



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!

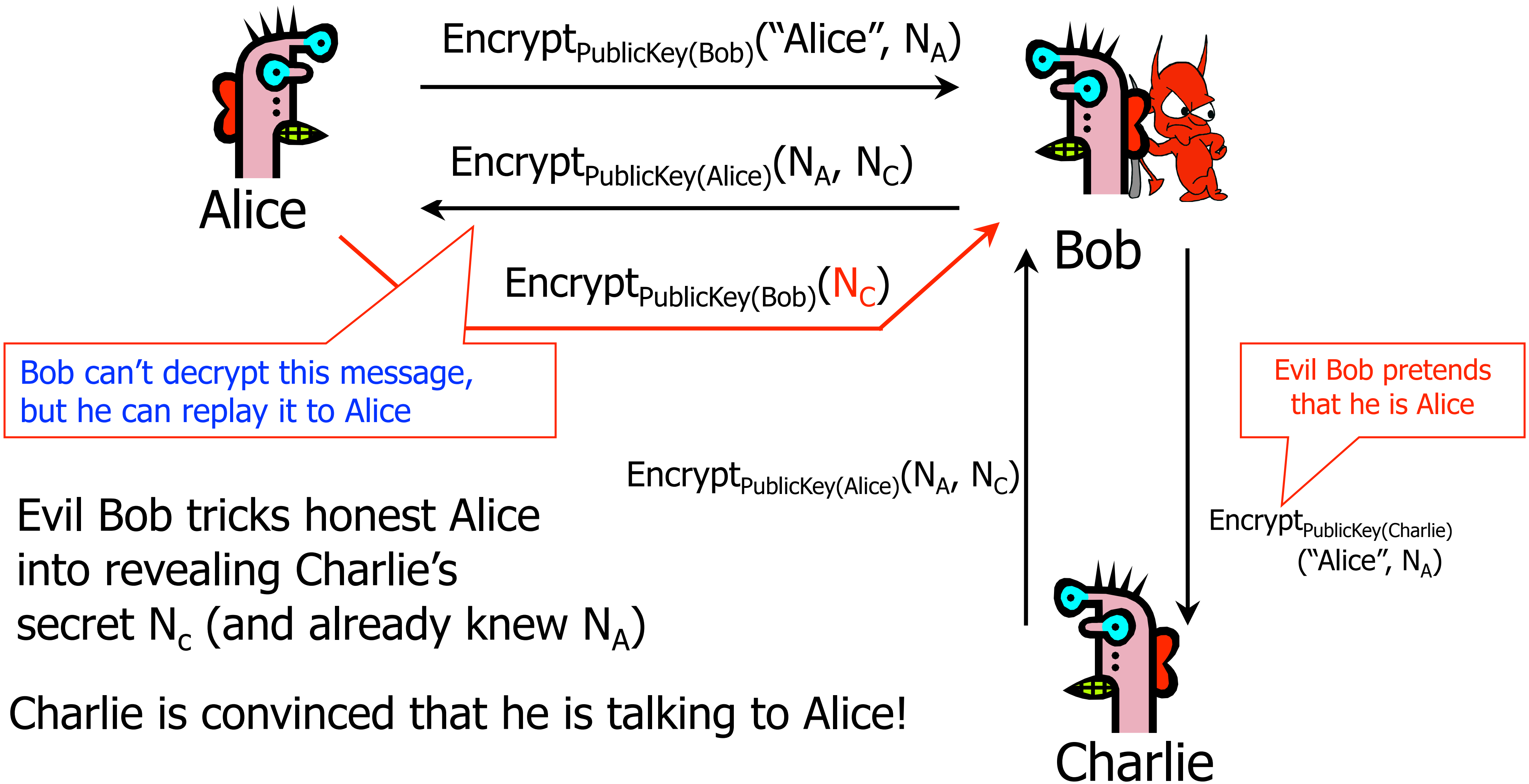
Bob's reasoning:

- 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]



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

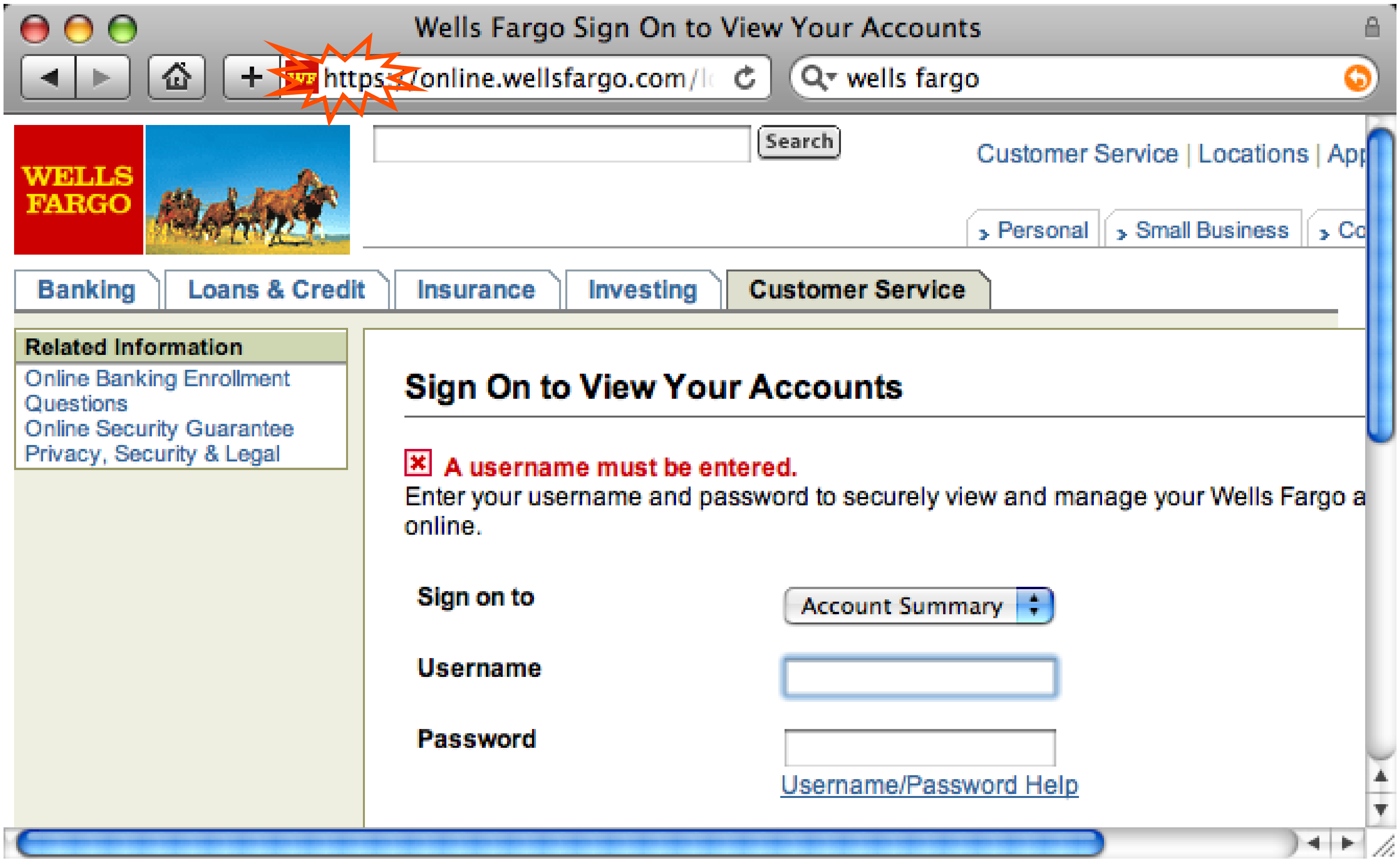
- This is yet another example of design challenges
  - Alice is correct that Bob must have decrypted $Encrypt_{PublicKey(Bob)}$("Alice", N_A), but this does not mean that $Encrypt_{PublicKey(Alice)}(N_A, N_B)$ came from Bob
- It is important to realize limitations of protocols
  - The attack requires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

SSL

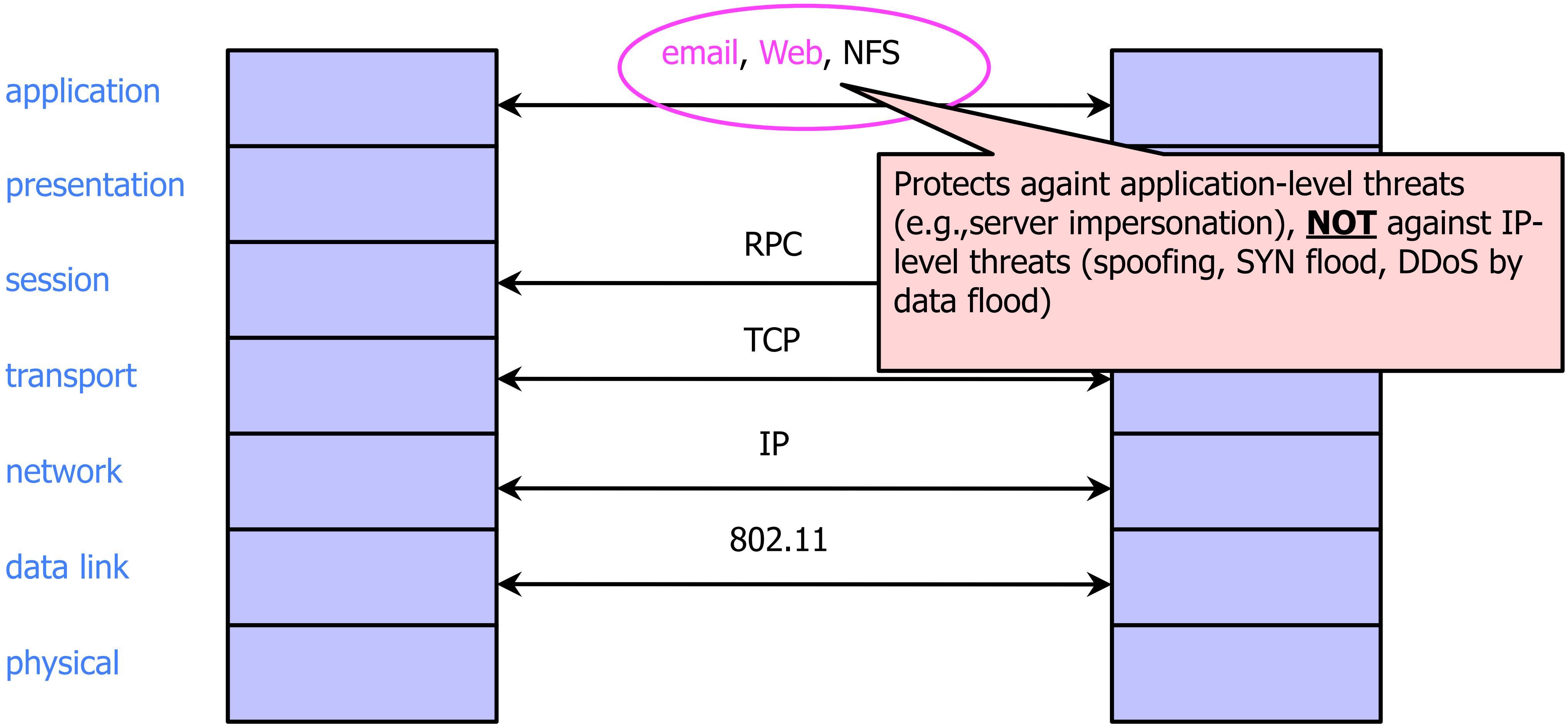
What is SSL / TLS?

- 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



Application-Level Protection



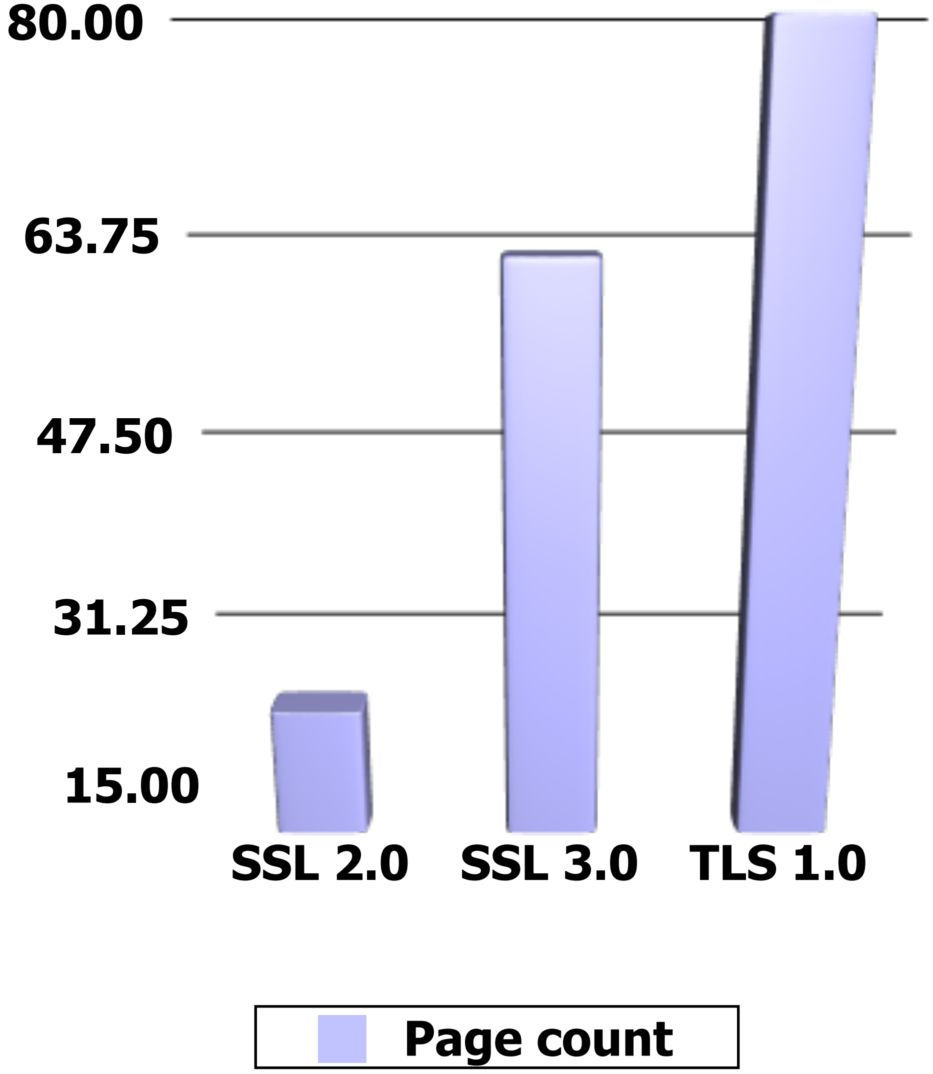
History of the Protocol

- 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

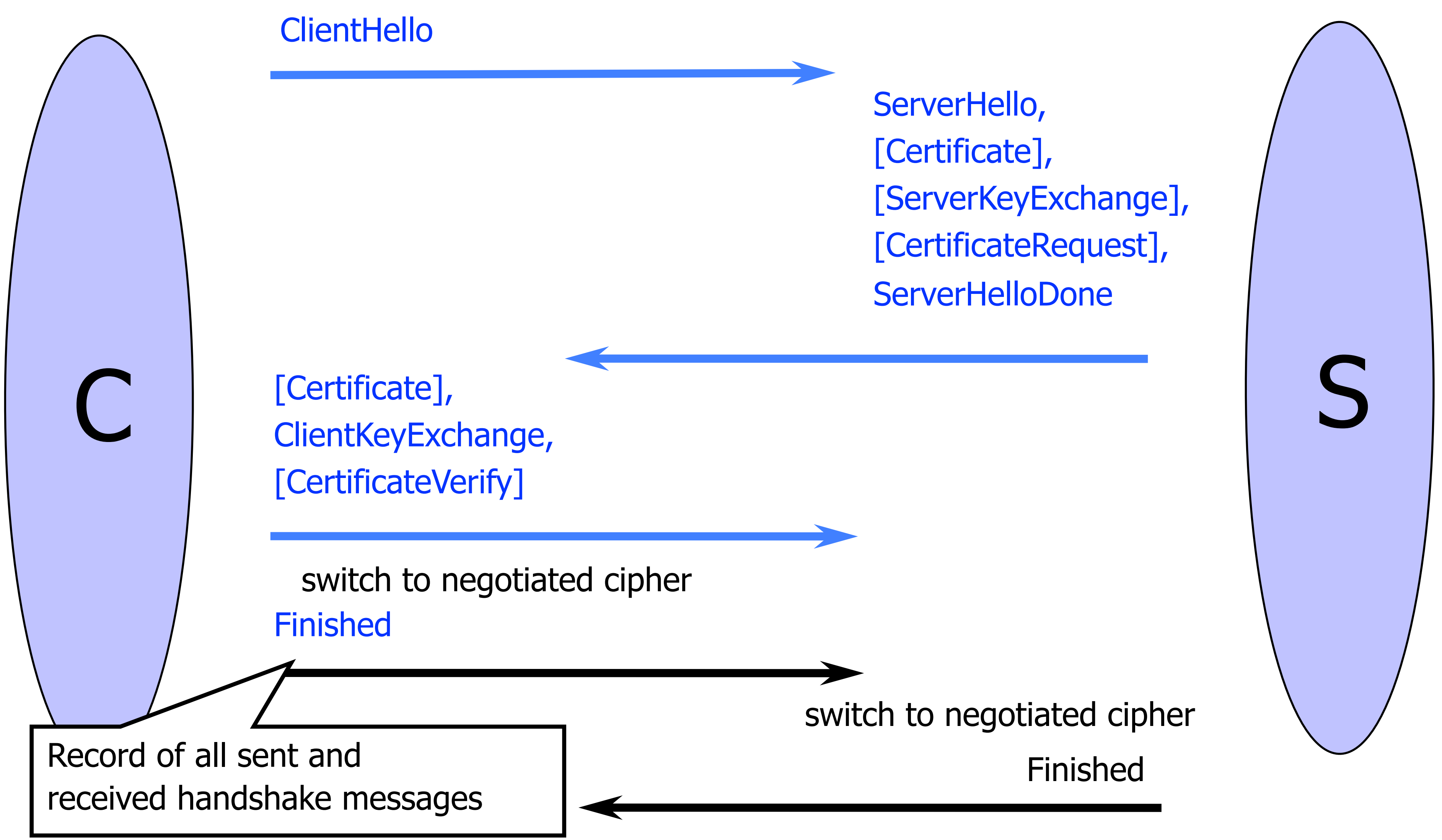
TLS Basics

- 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

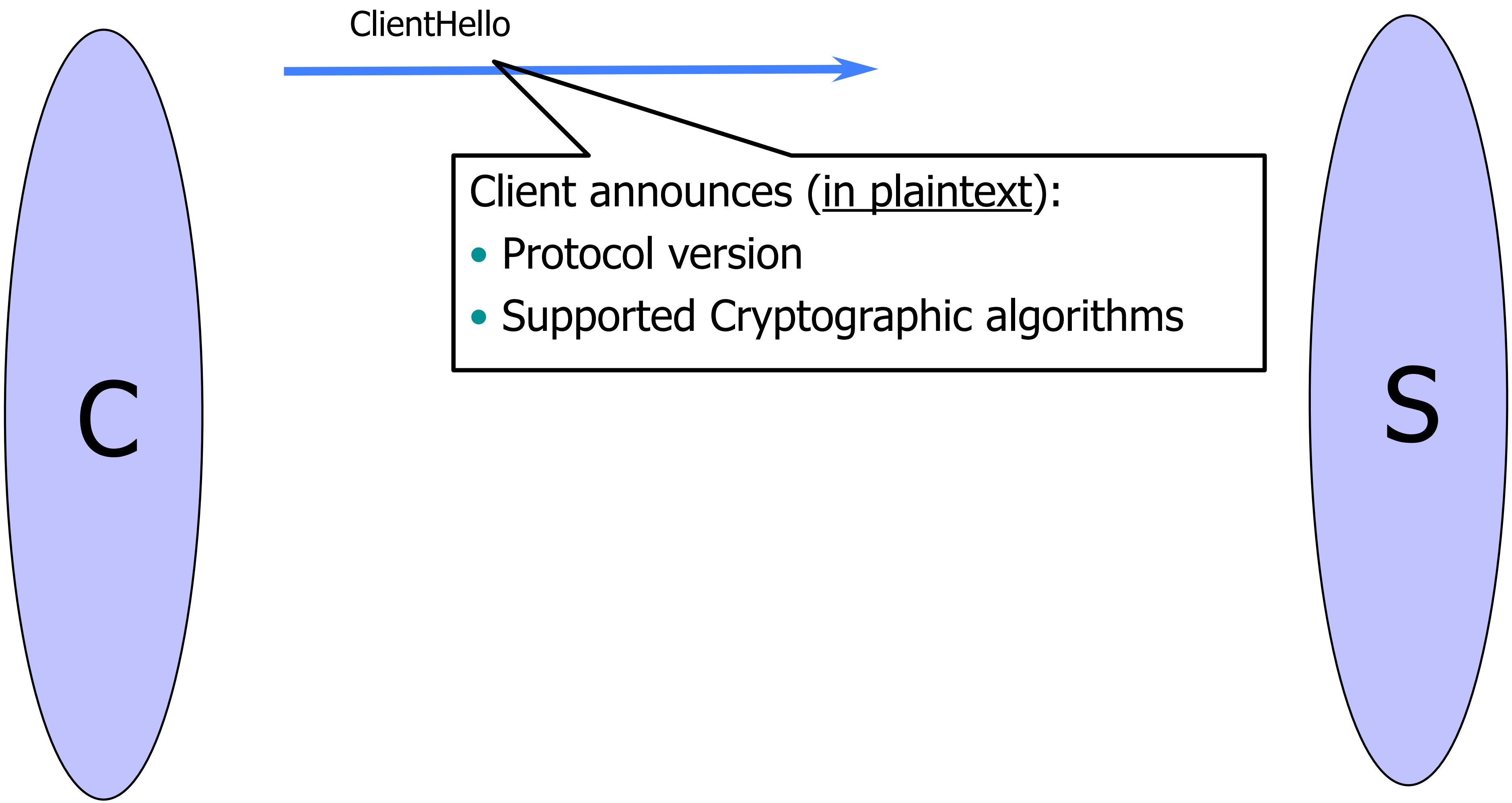
TLS Handshake Protocol

- 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



ClientHello



ClientHello (RFC)

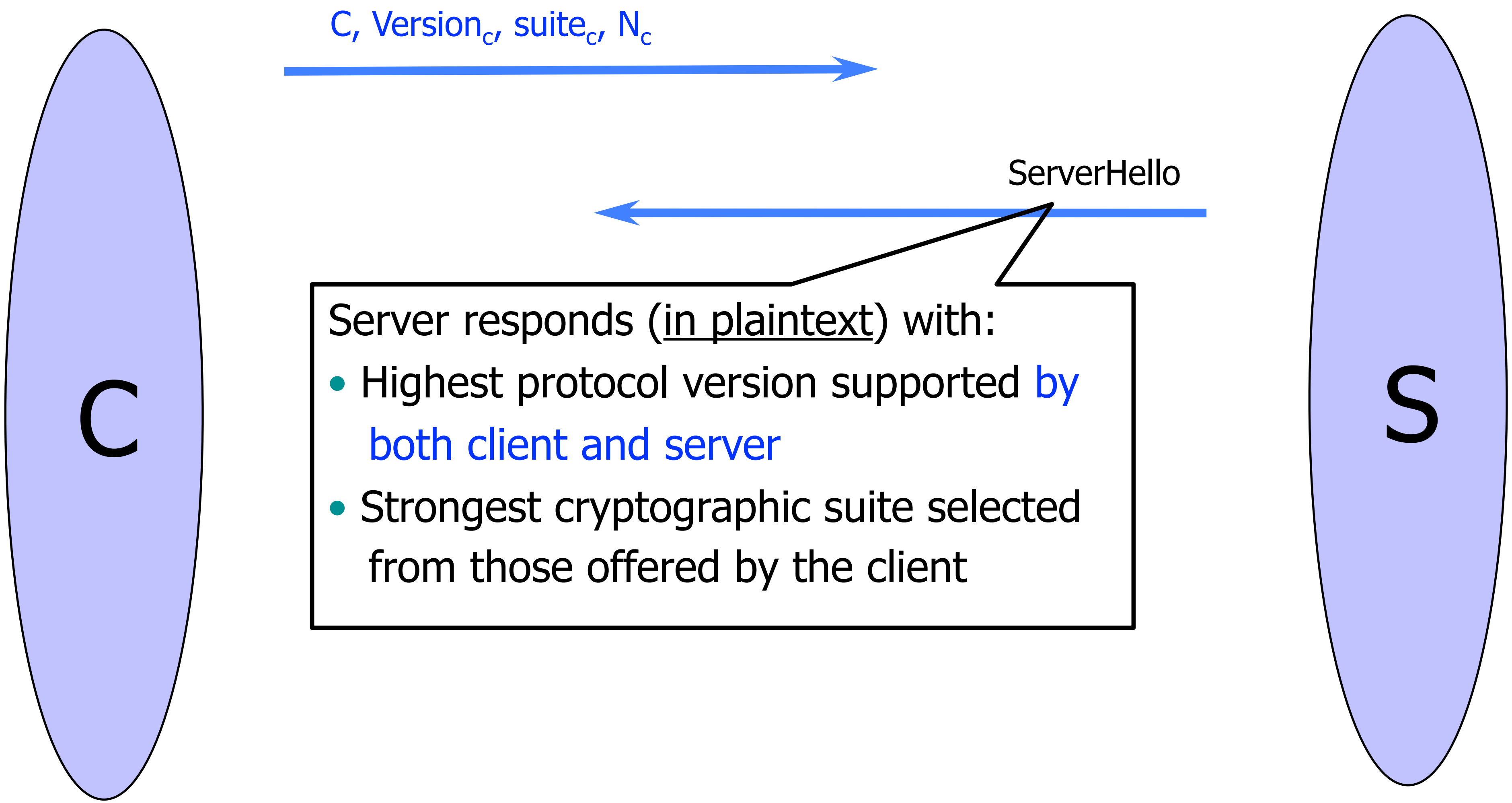
```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites;
    CompressionMethod compression_methods;
} ClientHello
```

Highest version of the protocol supported by the client

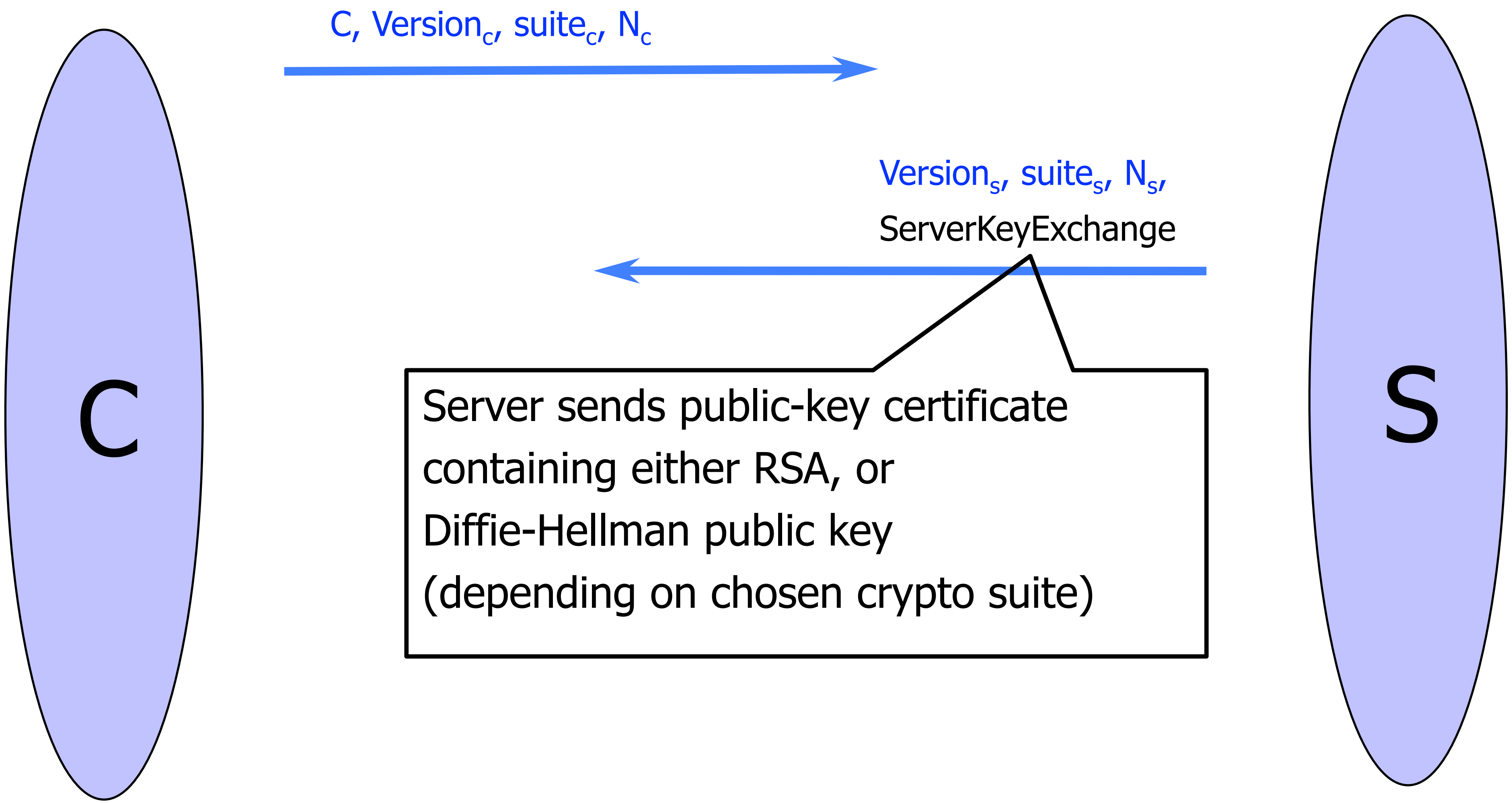
Session id (if the client wants to resume an old session)

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

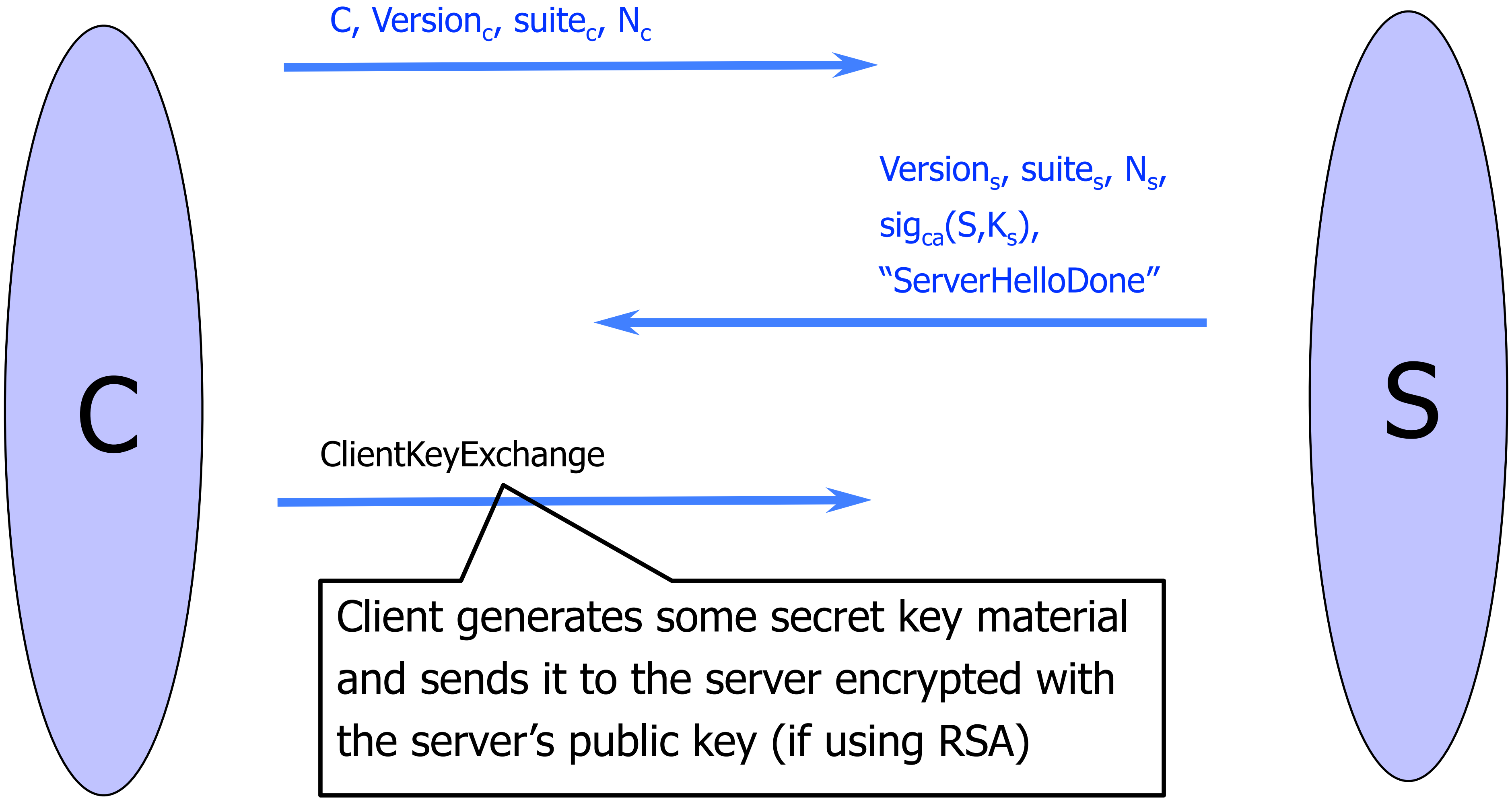
ServerHello



ServerKeyExchange



ClientKeyExchange



"Core" SSL 3.0 Handshake (Not TLS)

C

C, $Version_c$=3.0, $suite_c$, N_c →

← $Version_s$=3.0, $suite_s$, N_s, $sig_{ca}(S,K_s)$, "ServerHelloDone"

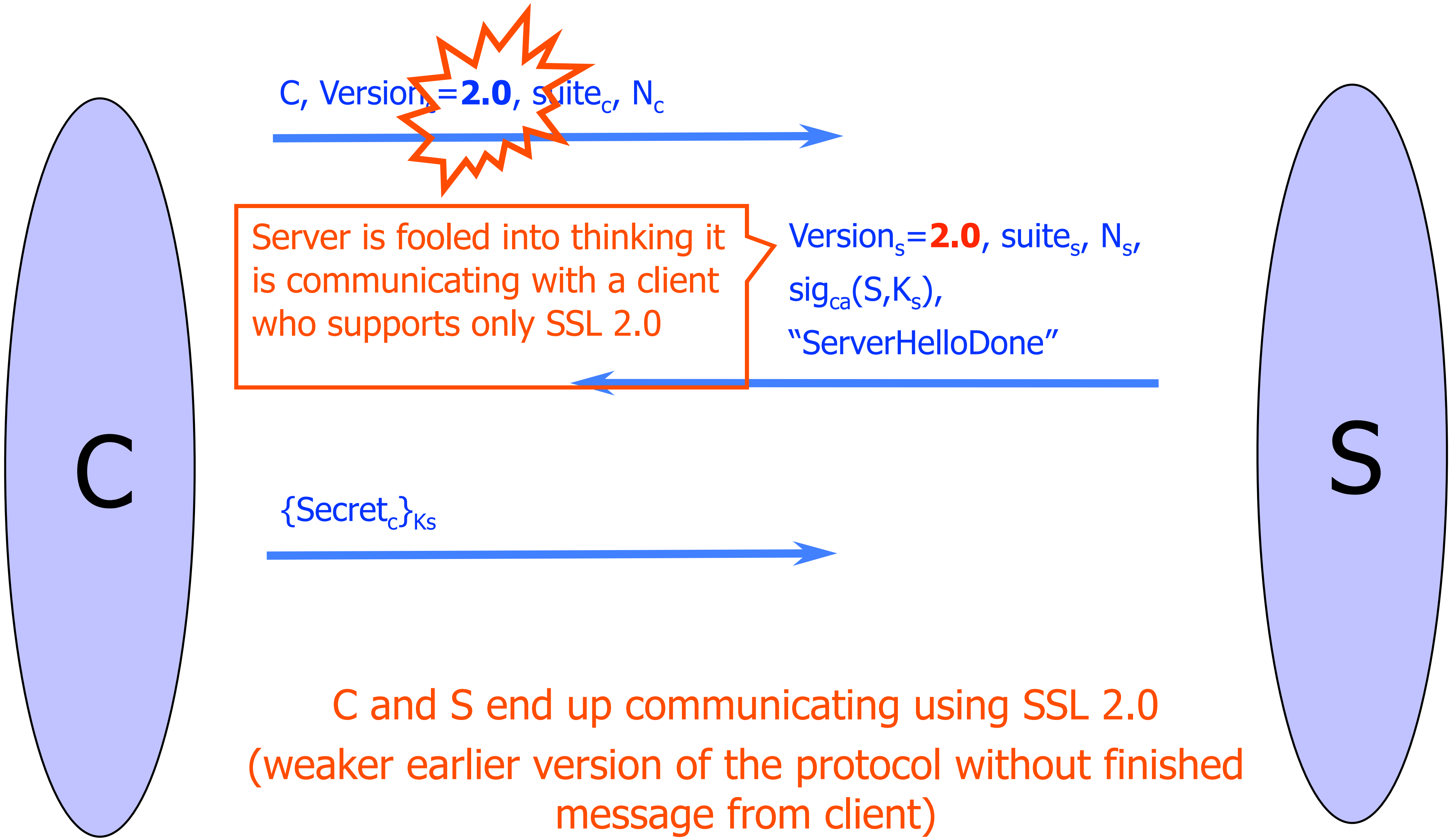
$\{Secret_c\}_{Ks}$ →

S

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



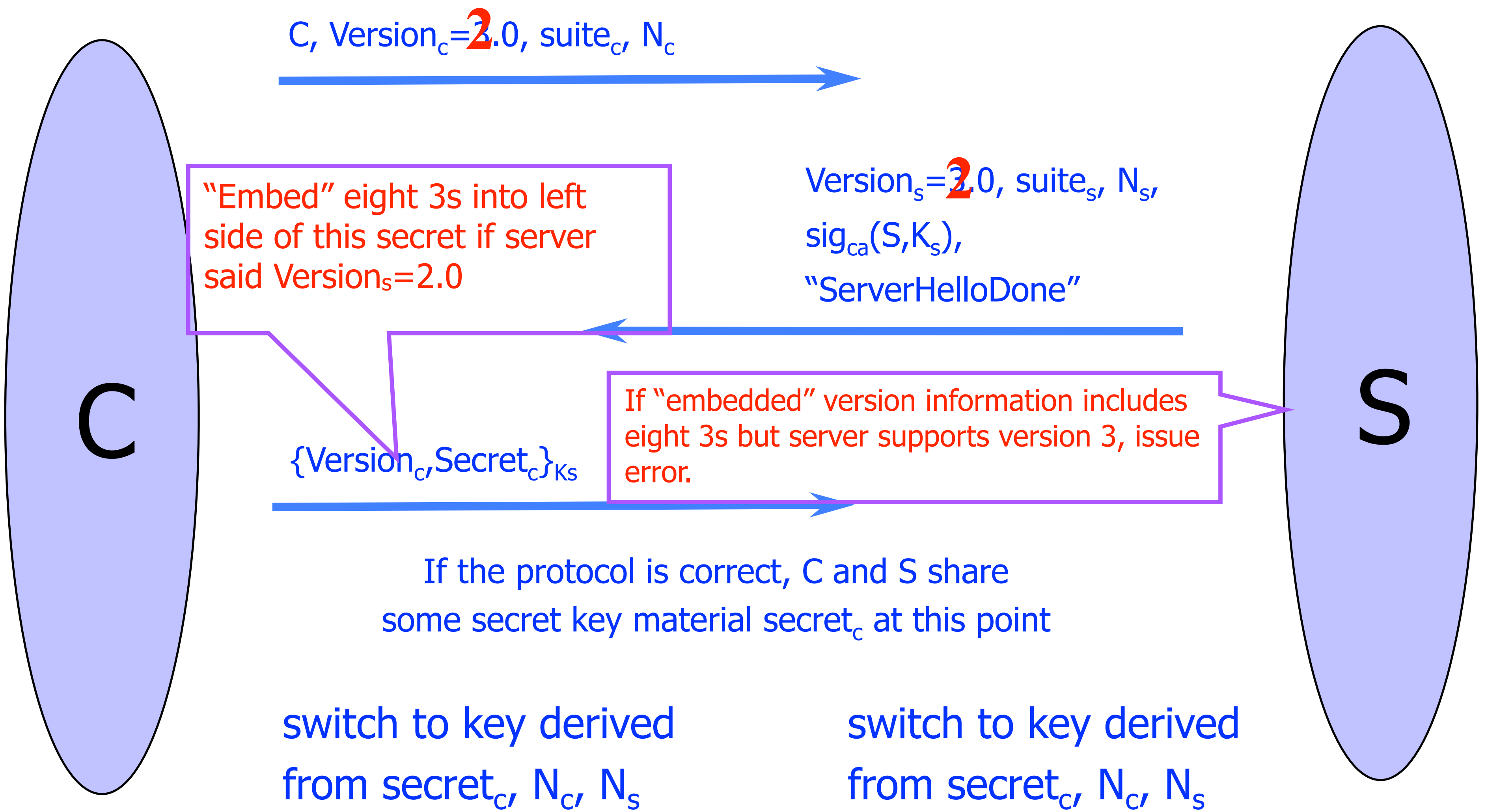
SSL 2.0 Weaknesses (Fixed in 3.0)

- 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)



SSL/TLS Record Protection