University of Pittsburgh

Applied Cryptography and Network Security
CS 1653

Summer 2023

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Prof. Adam Lee's CS1653 slides.)

Announcements

- Homework 7 due this Friday @ 11:59 pm
- Project Phase 3 Due next Monday 7/17 @ 11:59 pm
  - Your team must schedule a meeting with me on or before this Thursday

Real-time communication security

Now we're going to talk about real-time security issues

Issues related to session key disclosure

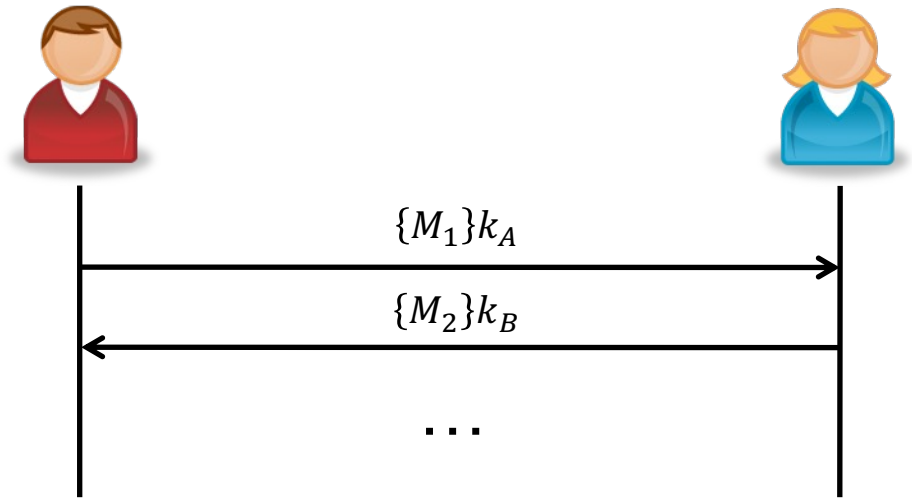
- Perfect forward secrecy
- Forward secrecy
- Backward secrecy

Deniable authentication protocols

Denial of service

- Computational puzzles
- Guided tour puzzles

Not all seemingly reasonable key exchange protocols provide perfect forward secrecy



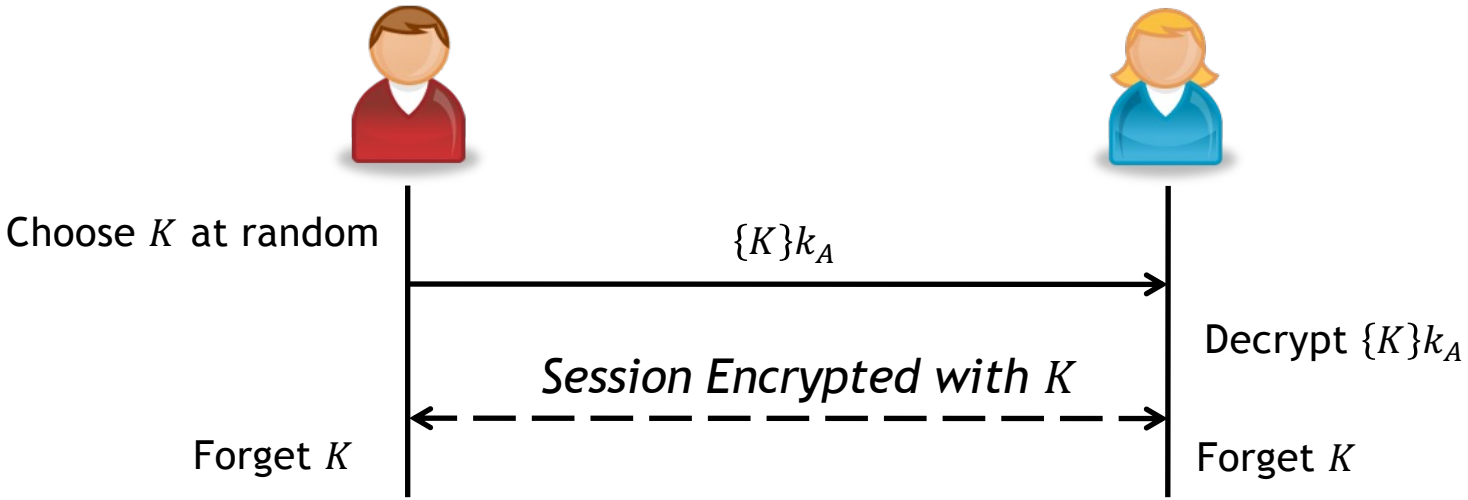
This protocol is inefficient and fails to provide perfect forward secrecy

What does the adversary learn?

- Monitoring: $\{M_1\}k_A$ and $\{M_2\}k_B$
- Compromising Alice and Bob: k_A^{-1} and k_B^{-1}

This compromises the protocol, as all messages can be recovered

Not all seemingly reasonable key exchange protocols provide perfect forward secrecy



This completely reasonable hybrid cryptosystem also fails to provide perfect forward secrecy (Why?)

What does the adversary learn?

- Monitoring: $\{K\}k_A$ and traffic encrypted with *K*
- Compromising Alice and Bob: k_A^{-1} and k_B^{-1}

Given k_A^{-1} and $\{K\}k_A$, the adversary can recover K even though Alice and Bob have both forgotten it!

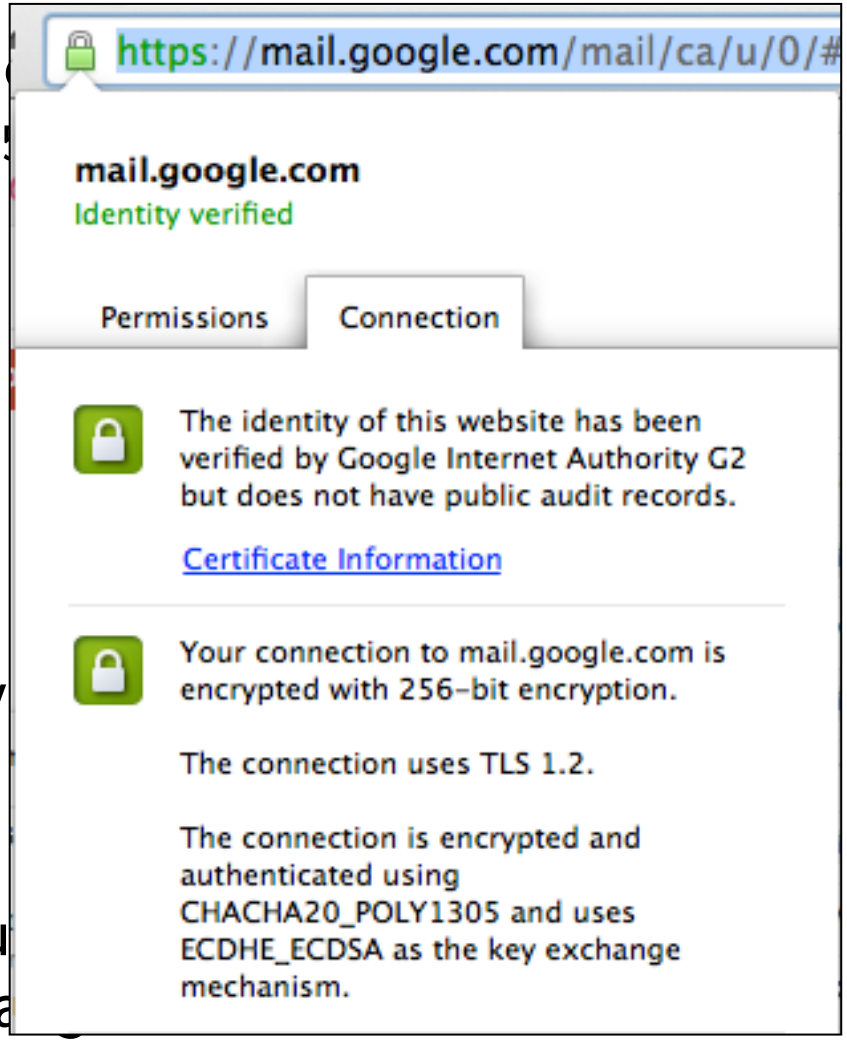
The previous protocol is similar to the “RSA key exchange” method supported in SSL/TLS

Many TLS cipher suites use RSA key

- TLS_RSA_EXPORT_WITH_RC4_40_MD5
- TLS_RSA_WITH_RC4_128_MD5
- TLS_RSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_IDEA_CBC_SHA
- TLS_RSA_WITH_DES_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- …

Using these cipher suites, one party
with the RSA key of the other party

In November 2011, Google added su
using the ECDHE family of key excha



So far, we've only talked about pairwise keys… What about group keys?

Scenario: Secure chat rooms

Assume that we want a way for geographically distributed people to carry out secure conversations.

By secure, we mean that only parties that are part of the chat room can understand what is being said, even though all messages are exchanged over the public Internet.

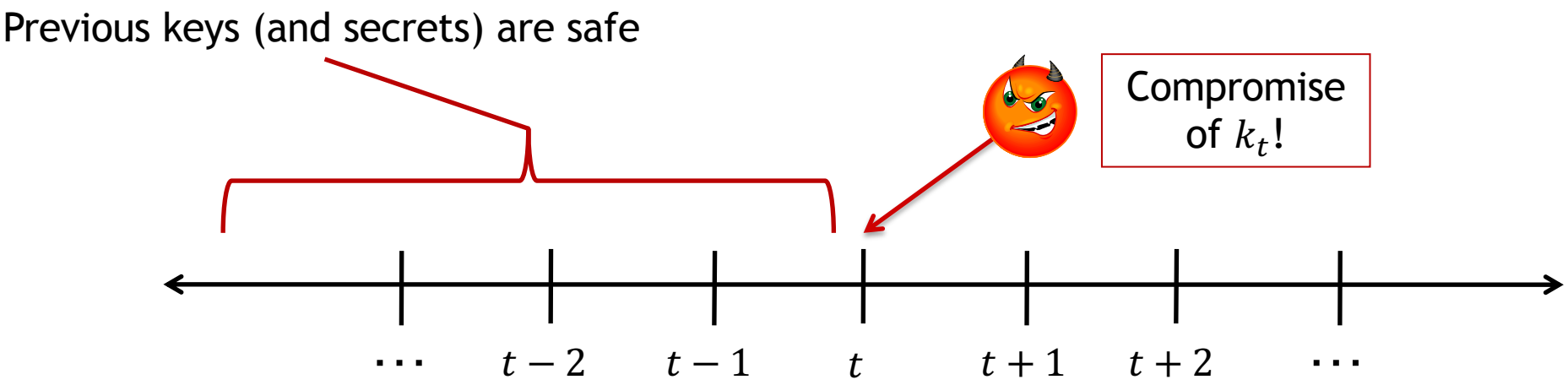
Simple idea: Set up a shared key for the chat room, and encrypt all messages using this key

Questions:

- Should people that join the group be able to decrypt old messages?
- Should people that leave the group be able to decrypt new messages?

Forward secrecy protects old information

A group communication scheme has forward secrecy if learning the key k_t for time t does not reveal any information about keys k_i for all $i < t$.

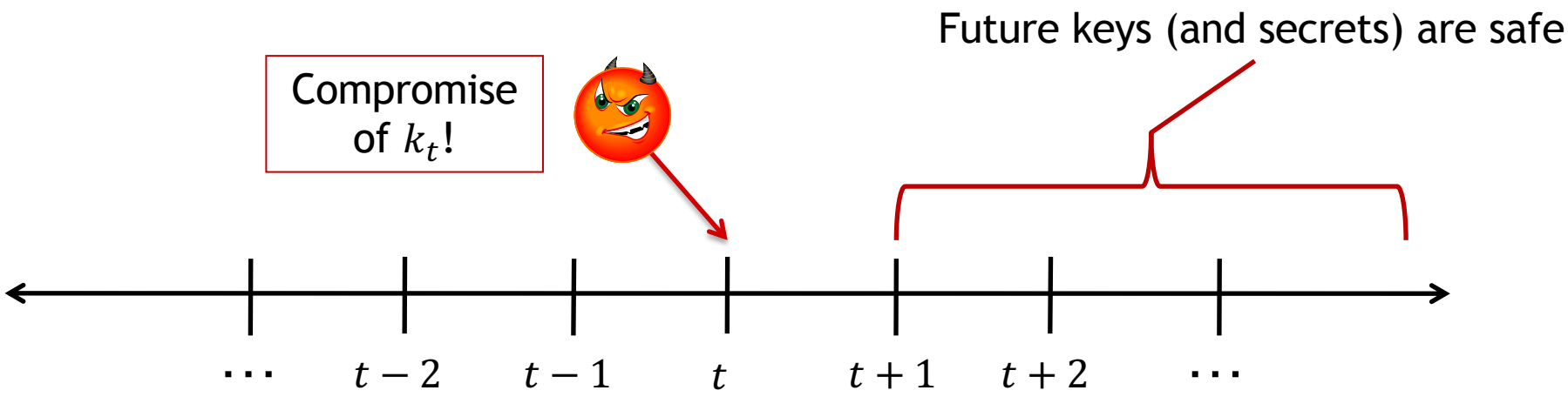


Informally: Later compromise does not reveal earlier keys

How can we achieve forward secrecy?

Backward secrecy is the complement of forward secrecy

A group communication scheme has backward secrecy if learning the key k_t for time t does not reveal any information about keys k_i for all $i > t$.



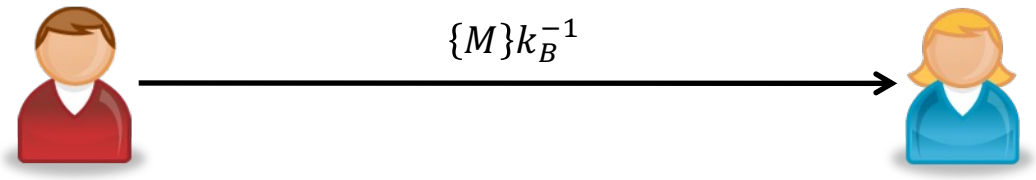
Informally: Earlier compromise does not reveal later keys

Question: Does our previous protocol also provide backward secrecy?

Deniability is another interesting property of authentication/key exchange protocols

Recall that a digitally signed message provides non-repudiability

- Alice can use k_B to verify that Bob signed the message
- But so can anyone else!

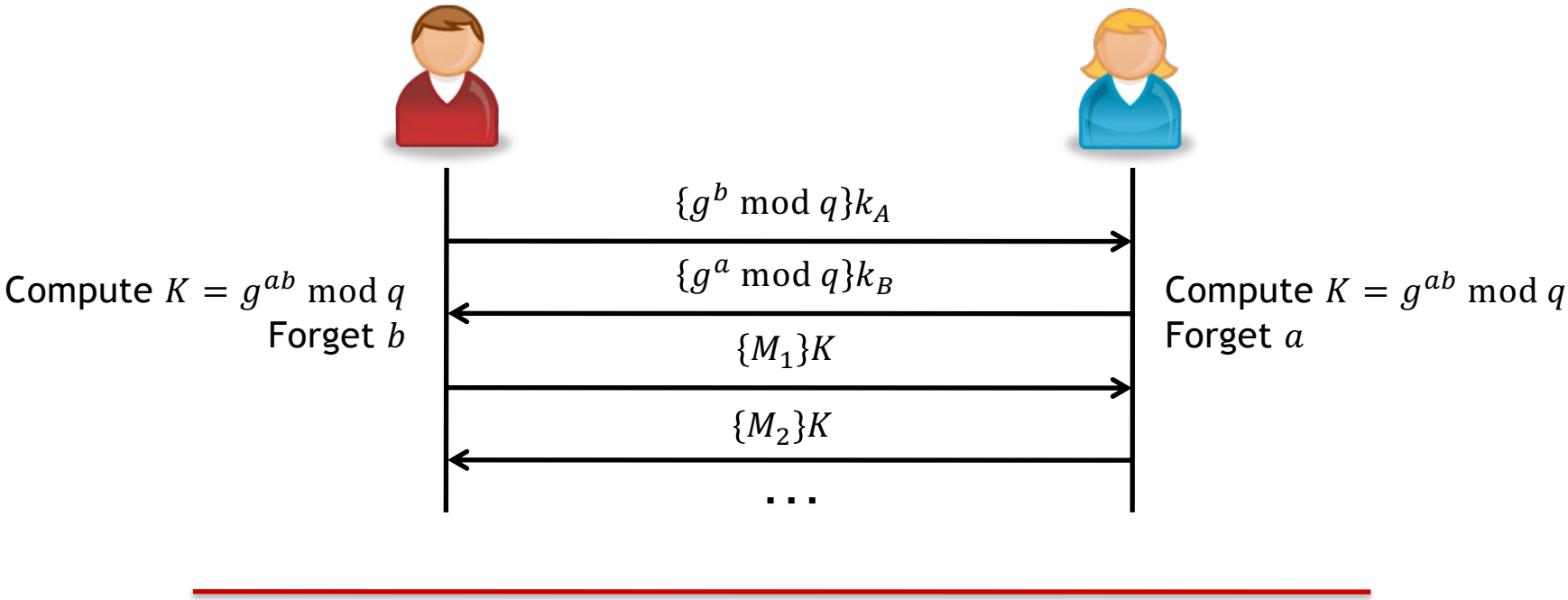


Sometimes, we would like Alice to be able to verify the authenticity of a message without being able to convince a third party that the message came from Bob

In essence, Bob would like some form of plausible deniability

More formally, a protocol between two parties provides plausible deniability if the transcript of messages exchanged could have been generated by a single party

How can we provide plausible deniability?



This protocol mutually authenticates Alice and Bob

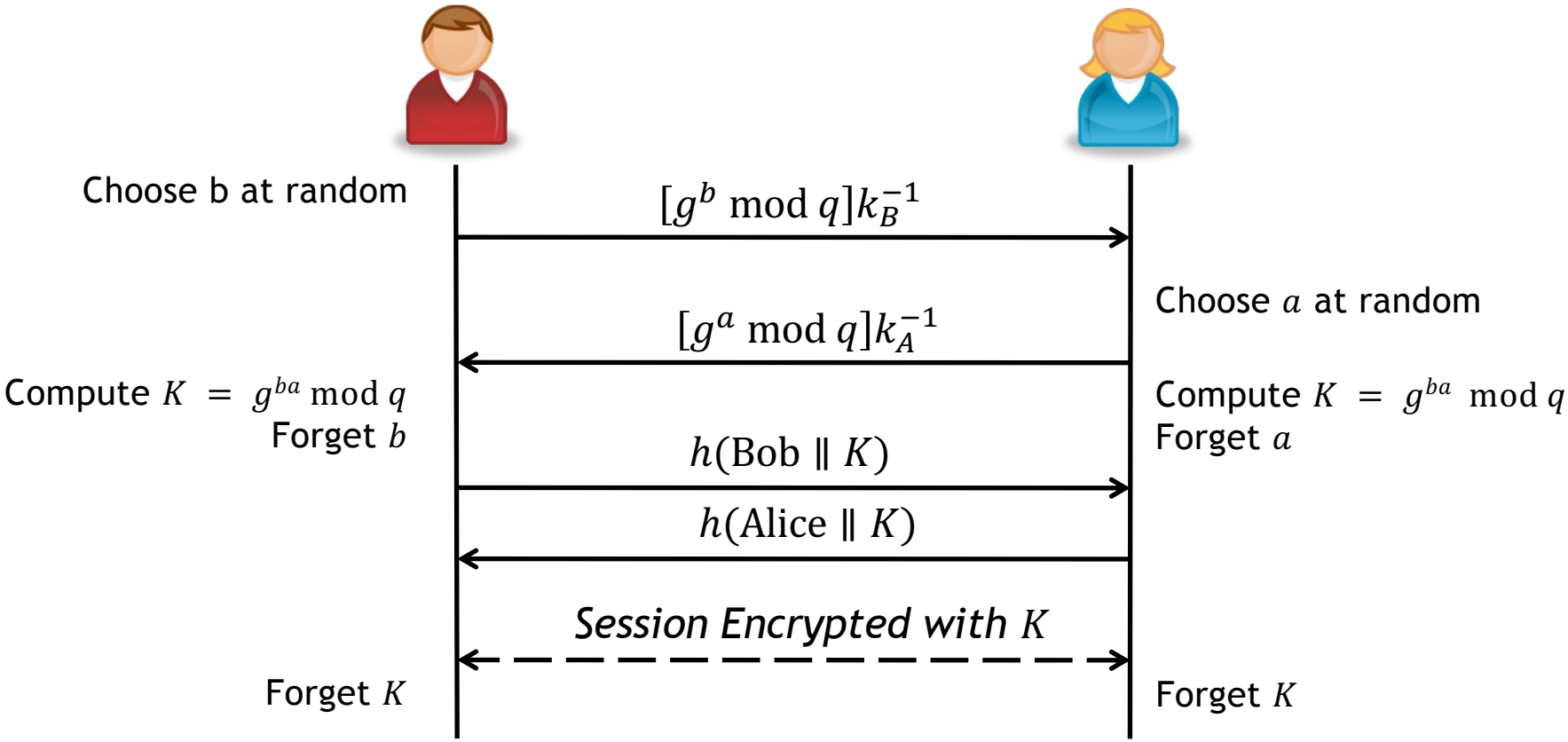
- Alice knows that only Bob could have derived K
- Bob knows that only Alice could have derived K
- Each party knows which M_is they sent

Anyone could have generated the transcript!

- Only need k_A and k_B, which are public
- The rest is random numbers!

Lesson: Causality is important. Notions of "I sent" or "I received" are *not* part of the transcript, but are part of real life.

What about the Signed Diffie-Hellman key exchange?



Computer security is typically defined with respect to three types of properties

How do I ensure that my secrets remain secret?

Confidentiality

Can I trust the services that I use?

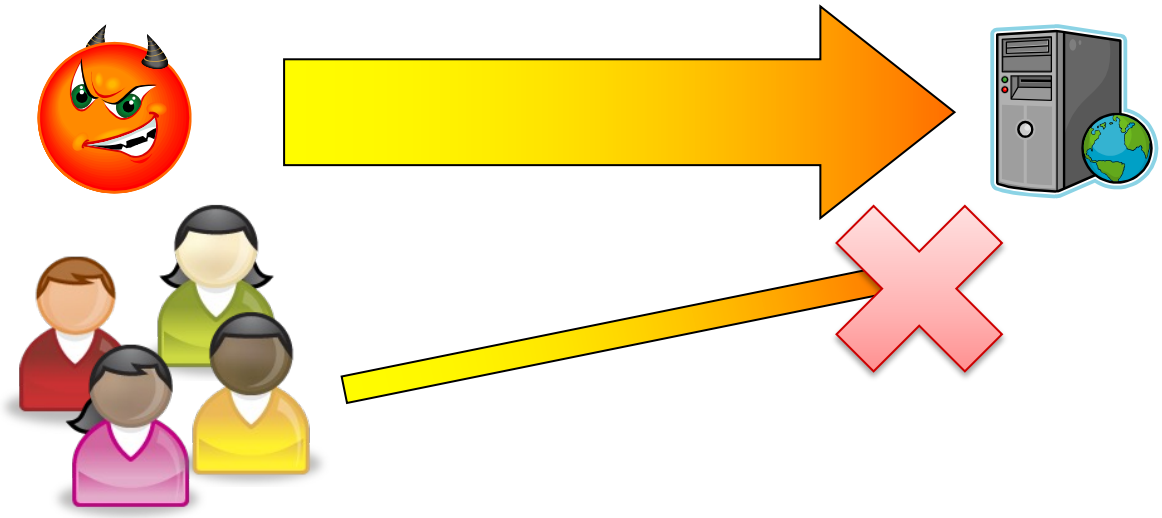
Integrity

Availability

Am I able to do what I need to do?

Denial of Service (DoS)

A denial of service (DoS) attack occurs when a malicious client is able to overwhelm a server, thereby preventing service to legitimate clients



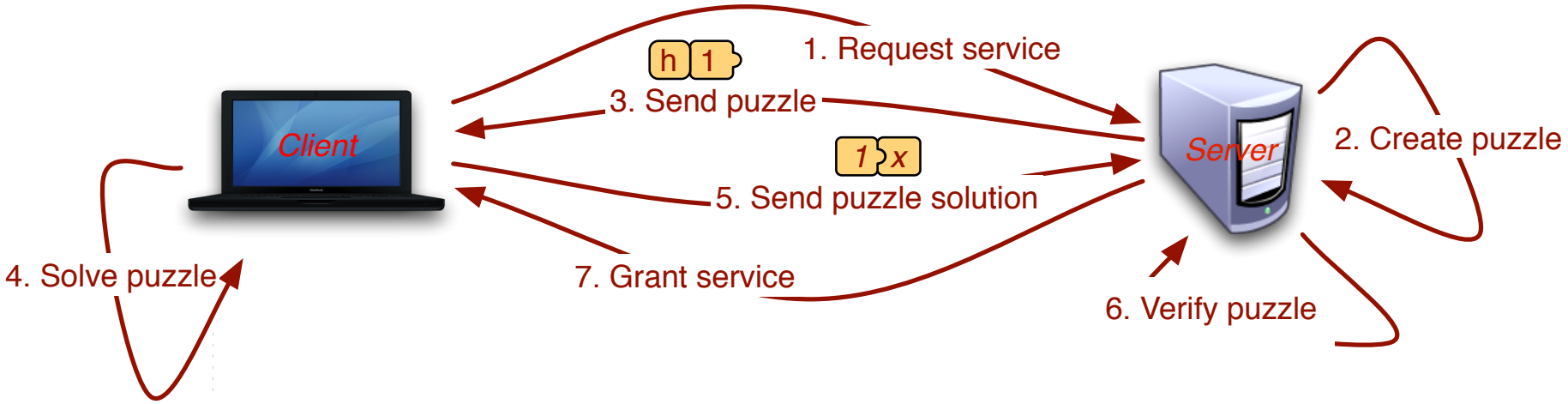
Denial of service attacks are typically abusing a resource disparity

- Easy to generate lots of network requests, hard to maintain server state
- Easy to create random application data, decryption takes time/cycles
- …

Avoiding resource disparity is a good design principle, but is not always easy to do in practice

Computational puzzles can be used to mitigate DoS attacks

Idea: Make clients pay for their requests by solving a hard puzzle first



Computational puzzles must satisfy three requirements:

1. Puzzles should be easy for the server to generate
2. Puzzles should be hard for the client to solve
3. Puzzle solutions should be easy for the server to verify

Question: Why do computational puzzles have these requirements?

Example: Hash inversion

Intuition: It should be very hard to invert a cryptographic hash function

- Recall that "hard" means $O(2^m)$ work where m is the hash bit length

Hash inversion puzzles work as follows:

Puzzle state

- Choose puzzle length n
- Pick n-bit number r
- Let $x = 1 \ldots 1r$

$n, h(x), \{\text{time} \parallel r\}K$

Search for n-bit number r such that $h(1 \ldots 1r) = h(x)$

$r, \{\text{time} \parallel r\}K$

- Decrypt token
- Verify puzzle solution

Note: Puzzle hardness can be tuned by using different length values (n)

Question: Why is the puzzle state offloaded to the client?

Discussion

When would computational puzzles be effective for mitigating DoS attacks? When are they ineffective?

Strengths and weaknesses of computational puzzles

When do computational puzzles work well?

- All clients have approximately similar computational ability
- Puzzles cannot be parallelized between multiple clients
- Puzzles can be efficiently generated

Unfortunately, computational puzzles are not a panacea…

- Not all clients are created equal
- Clients have better things to do than burn cycles solving puzzles
- Attackers often have a means of parallelizing puzzles

≠

Earlier work in our department is attempting to address this problem

Observation: Attackers can fairly easily control:

- Their own computational abilities
- A subset of nodes in the network (e.g., via compromise)
- The quality of their connection to the network

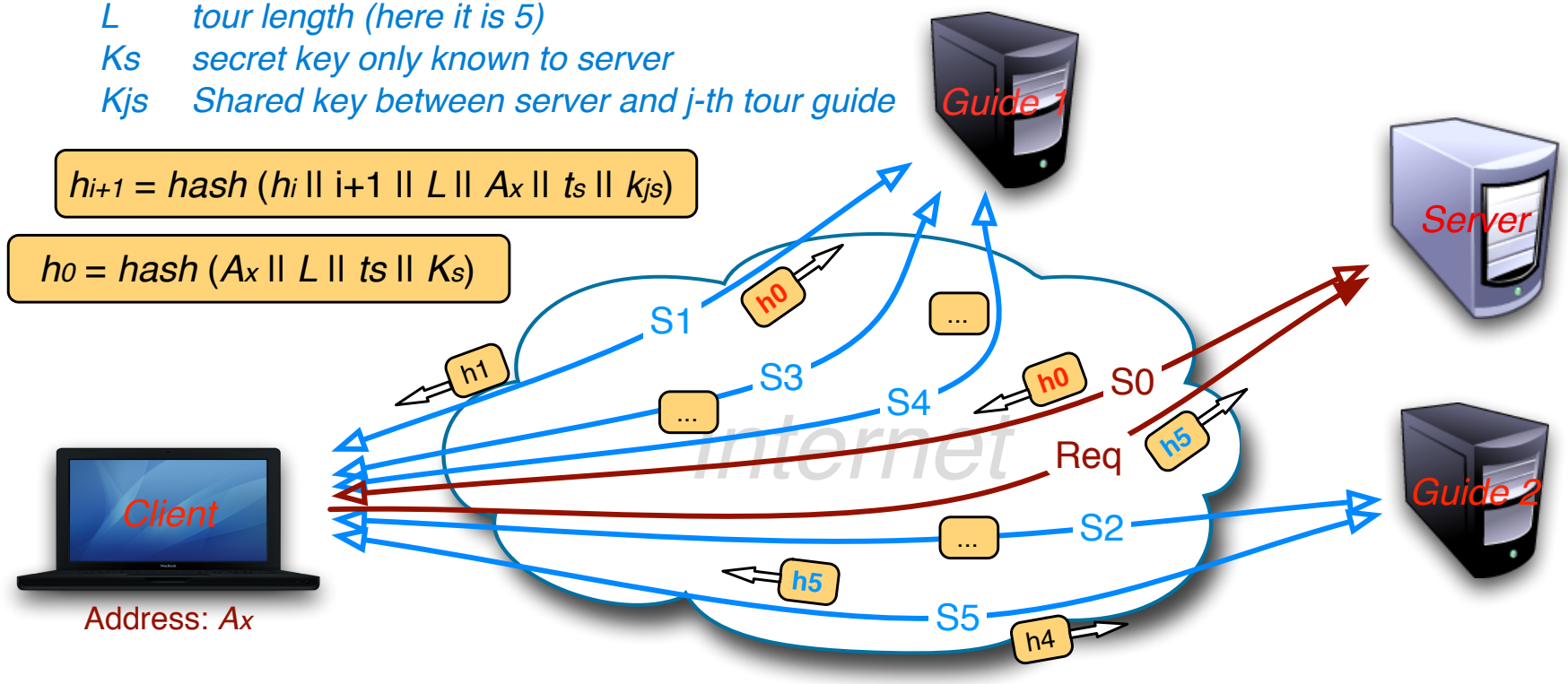
However, they cannot control the overall latency characteristics of routes that traverse segments of the Internet

Rather than asking clients to solve computational puzzles, we can ask them to provide evidence that they have carried out an ordered traversal of some number of nodes

Guided tour puzzles do exactly this!

- Puzzle: An ordered list of tour guides to contact
- Solution: A serial computation carried out by these nodes in order

How do guided tours work?



Guided tour puzzles are efficient to check

- Next tour guide chosen using a single hash operation
- A length n tour is checked using n hash operations

Cheating is hard

- Cannot guess next tour guide without knowing the secret key
- Cannot control the delay characteristics of the Internet

Real-time Communication Security

We talked about a variety of real-time issues

Session key security issues include:

- Perfect forward secrecy: Session keys safe even if long term keys compromised
- Forward secrecy: Compromising current key does not compromise previous keys
- Backward secrecy: Compromising current key does not lead to the compromise of future keys

Deniable protocols have completely forgeable transcripts, yet still provide authentication, confidentiality, and integrity protection

DoS attacks impact system availability

Puzzles (computational or network) can be used to mitigate DoS attacks

Transport Layer Security (TLS)

What is TLS and why do we need it?

How does TLS work?

- High-level intuitive explanation
- Packet-level details
- Key derivation
- Session resumption

PKI considerations when using TLS

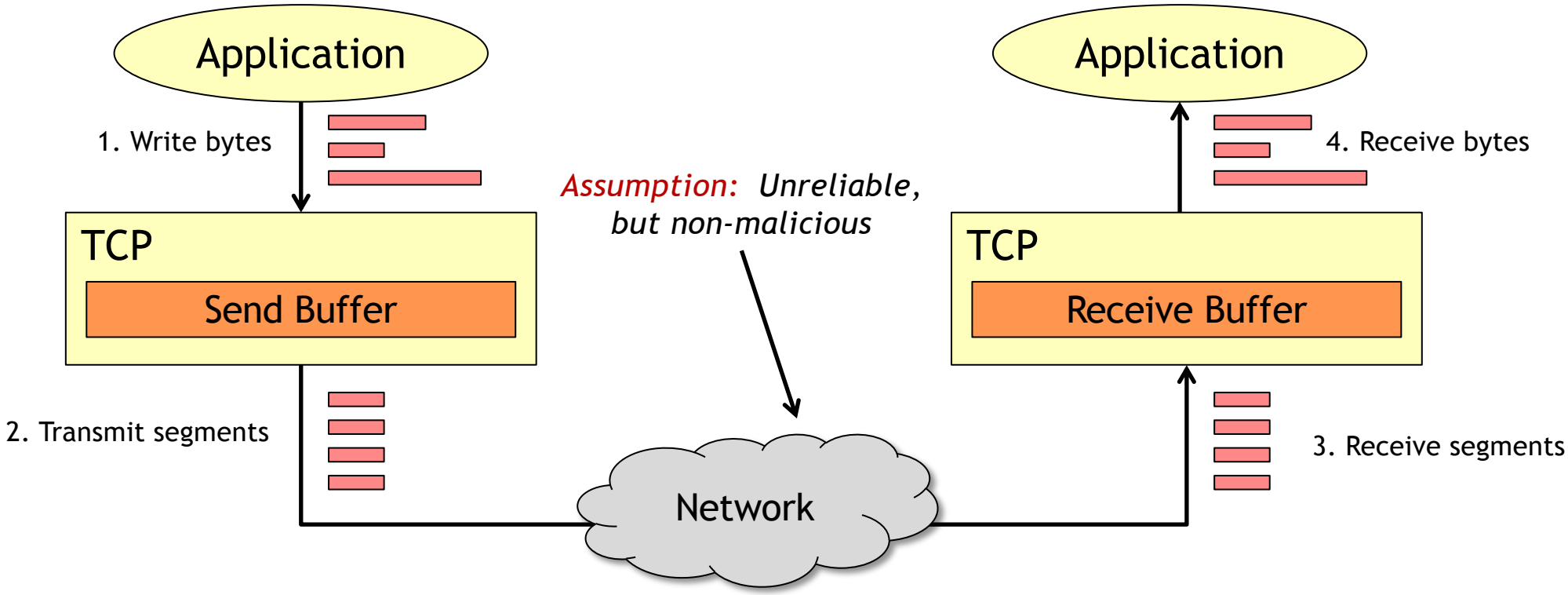
TCP is reliable, but not secure

TCP provides a reliable transport service

- Acknowledgements ensure that data is eventually delivered
- Checksums help protect packet integrity in the event of transient failure

Please note that:

- Active attackers can change the contents of packets
- Checksums are not cryptographic, so they can be forged



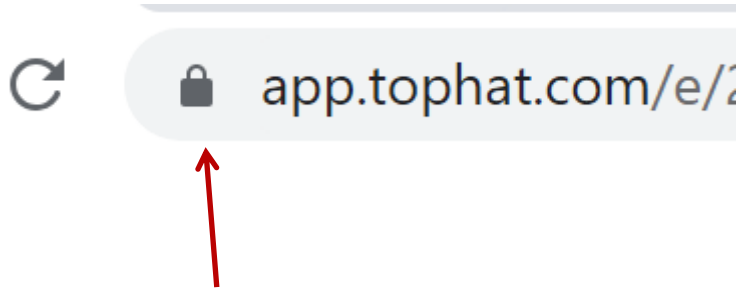
SSL and TLS were developed to provide applications with a secure means of utilizing TCP

Goal: Provide a generic means of authentication, confidentiality, and integrity protection to networked applications

That is, SSL/TLS were designed to simplify network security in the same way that Berkeley sockets simplified network programming

Where is SSL/TLS used?

- Web browsers (HTTPS)
- Protecting FTP (FTPS)
- POP/IMAP via STARTTLS
- VoIP security
- …



TLS protection in Chrome

Authentication in SSL can be one-way or mutual

- One way: Web browser authenticating
- Mutual: B2B web services transactions

Historical Context

Building a protocol suite for secure networking applications is a great idea. As such, there were many attempts made at this.

Secure Sockets Layer (SSL) was developed by Netscape

- Version 1 was never deployed
- Version 2 appeared in 1995
- Version 3 appeared in 1996

Microsoft developed PCT by tweaking SSL v2

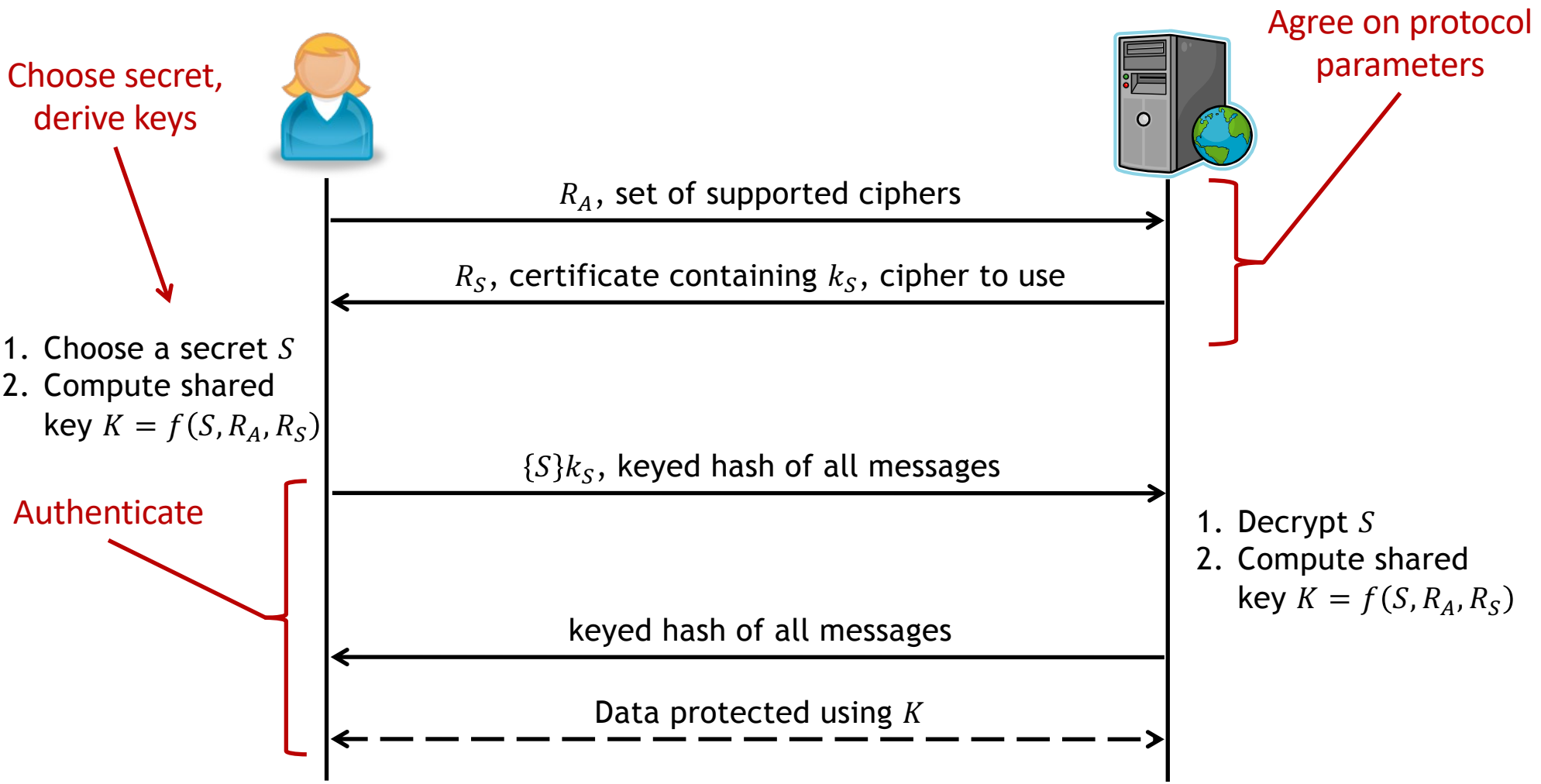
- This was mostly a political move
- Forced Netscape to turn control of SSL over to the IETF

Microsoft

The IETF then developed the Transport Layer Security (TLS) protocol

- TLS 1.0 released in 1999 (RFC 2246)
- Most current version is TLS 1.3, which was released in 2018 (RFC 8446)

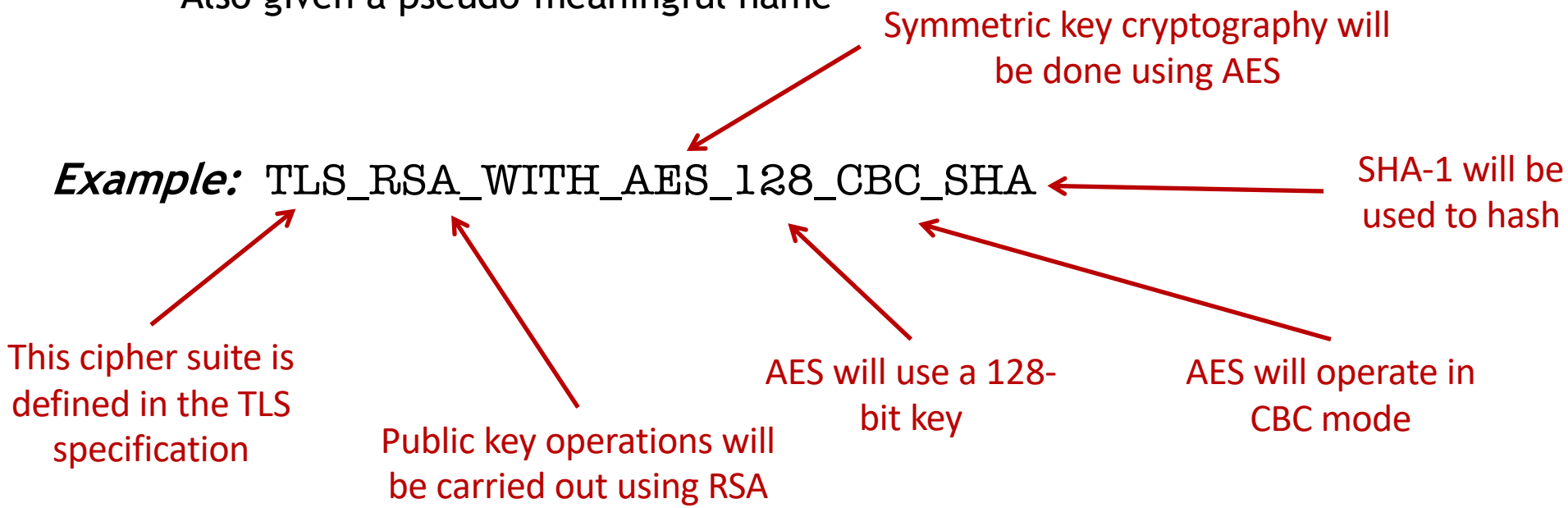
Viewed abstractly, TLS is actually a very easy to understand protocol



What is a cipher suite?

A cipher suite is a complete set of algorithmic parameters specifying exactly how the protocol will run

- Specified using a 16-bit identifier
- Also given a pseudo-meaningful name



In TLS, the client proposes a list of supported cipher specifications, and the server gets to choose which one will be used

Discussion

Why are cipher suites a good idea?

TLS transmits data one record at a time

TLS defines four specific message types

- Handshake records contain connection establishment/setup information
- The ChangeCipherSpec record is a flag used to indicate when cryptographic changes will go into effect
- Alert records contain error messages or other notifications
- Application_Data records are used to transport protected data

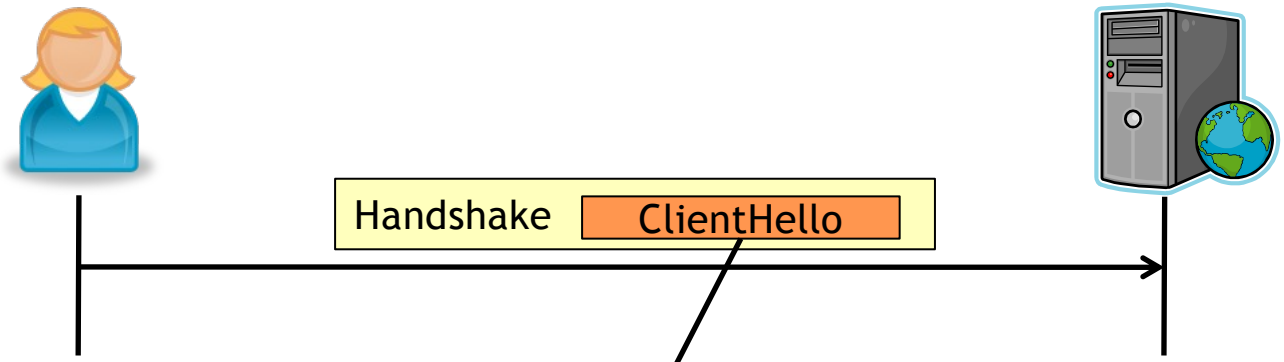
Note that a single record may contain multiple messages inside of it

Each record is delineated using a 5 byte record header

Bytes	*Content*
1	Record type code
2	Version number
2	Length

To illustrate this record/message sending process, let's look into the details of the protocol…

Handshake: Message 1

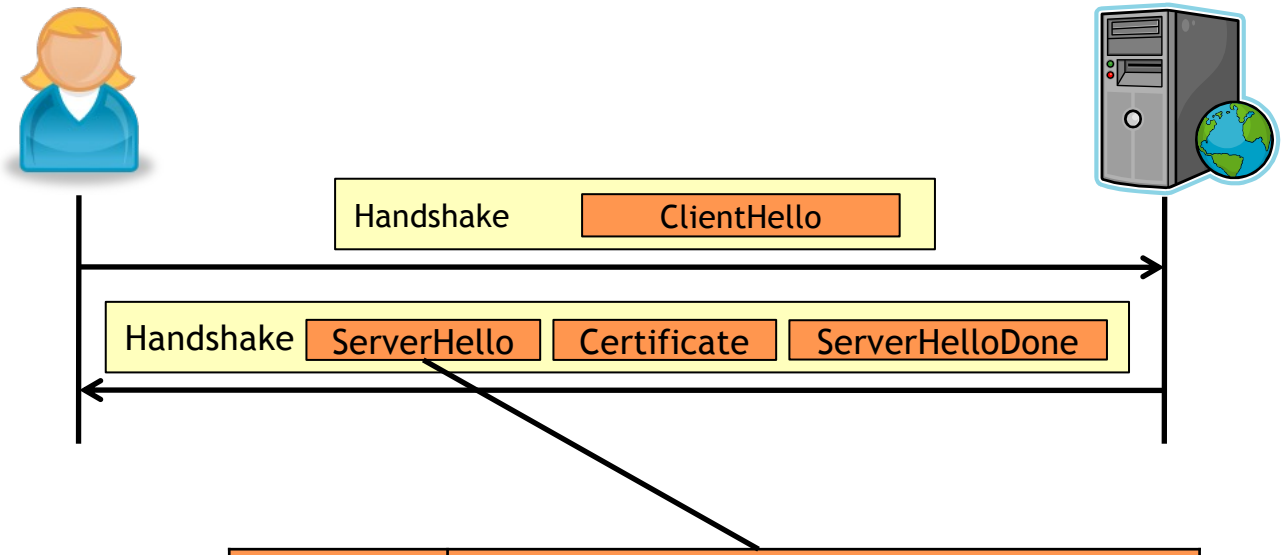


Bytes	*Content*
1	Message type: ClientHello
3	Length
2	Version number
32	R_A
1	Length of session ID
var	Session ID
2	Length of cipher suite list
var	List of supported cipher suites
1	Length of compression mode list
var	List of supported compression modes

Used to resume sessions. More on this later.

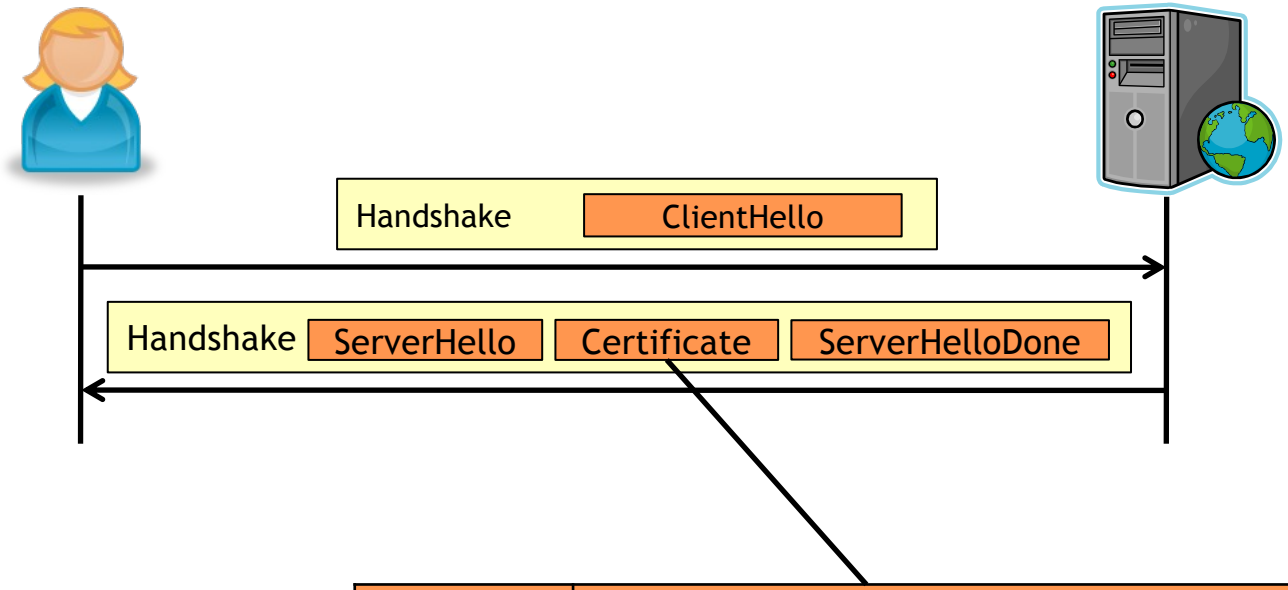
For efficiency reasons, TLS compresses the data that it transmits.

Handshake: Message 2



Bytes	*Content*
1	Message type: ServerHello
3	Length
2	Version number
32	R_S
1	Length of session ID
var	Session ID
2	Chosen cipher suite
1	Chosen compression method

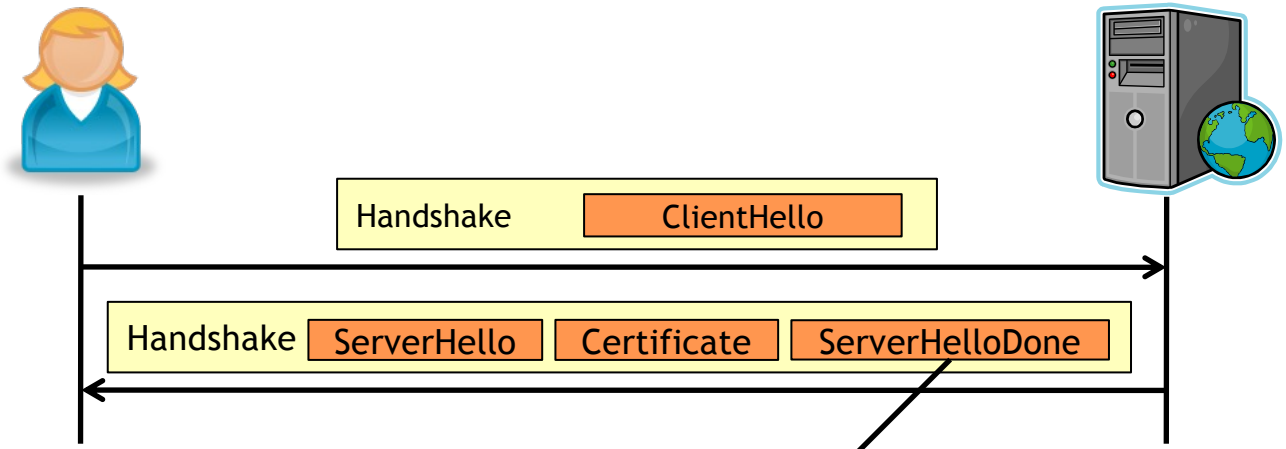
Handshake: Message 2



Bytes	*Content*
1	Message type: Certificate
3	Length
3	(Redundant) length
3	Length of first certificate
var	First certificate
var	More (length, certificate) pairs

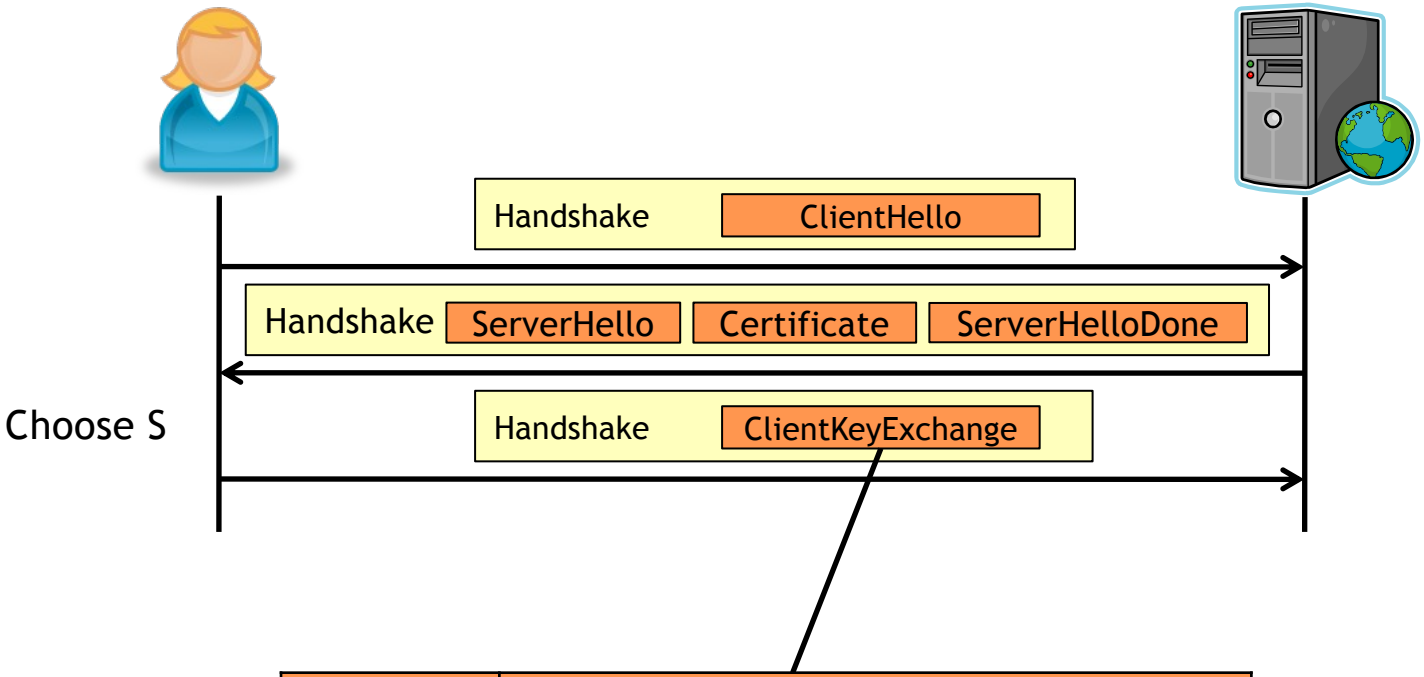
Why?

Handshake: Message 2



Bytes	*Content*
1	Message type: ServerHelloDone
3	Length = 0

Handshake: Message 3



Bytes	*Content*
1	Message type: ClientKeyExchange
3	Length = 0
2	(Redundant) length
var	Encrypted pre-master secret $\{S\}k_S$

How are keys computed?

How are session keys computed?

RFC 5246 defines a pseudorandom function that is used to expand the master secret into an randomized key stream:

```
PRF(secret, seed) =
   HMAC(secret, A(1) || seed) ||
   HMAC(secret, A(2) || seed) ||
   HMAC(secret, A(3) || seed) || ...
```

The sequence A is defined as follows:

- $A(0) = \text{seed}$
- $A(i) = \text{HMAC}\big(\text{secret}, A(i-1)\big)$

Note: The version of HMAC that is used depends on the cipher suite!

- E.g., `TLS_RSA_WITH_AES_128_CBC_SHA` uses HMAC-SHA-1

Key derivation, continued

First, the master secret is computed:

- master_secret = PRF(pre_master_secret, "master secret" || R_A || R_S)[0..47]

Next, a stream of random bytes is generated from the master secret

- key_block = PRF(master_secret, "key expansion" || R_S || R_A)

Keys are taken from the above block of key material in the following order

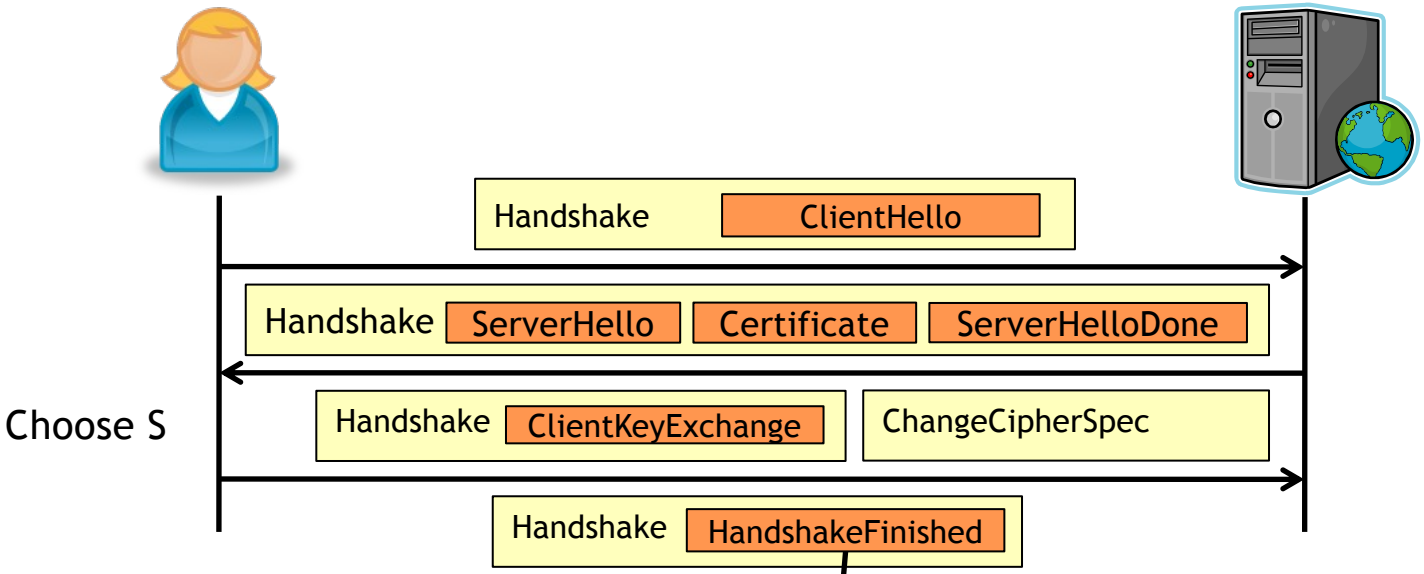
- Client → server MAC key
- Server → client MAC key
- Client → server symmetric encryption key
- Server → client symmetric encryption key
- Client → server IV
- Server → client IV

This process is called key expansion. Note that the amount of key material generated depends on the cipher suite chosen. (Why?)

Discussion

In TLS, the client provides the pre master secret, S, to the server. Why are R_A and R_S included when computing the master secret from the pre master secret?

Finishing the Handshake



Bytes	*Content*
1	Message type: HandshakeFinished
3	Length of digest
var	Keyed digest of message exchange

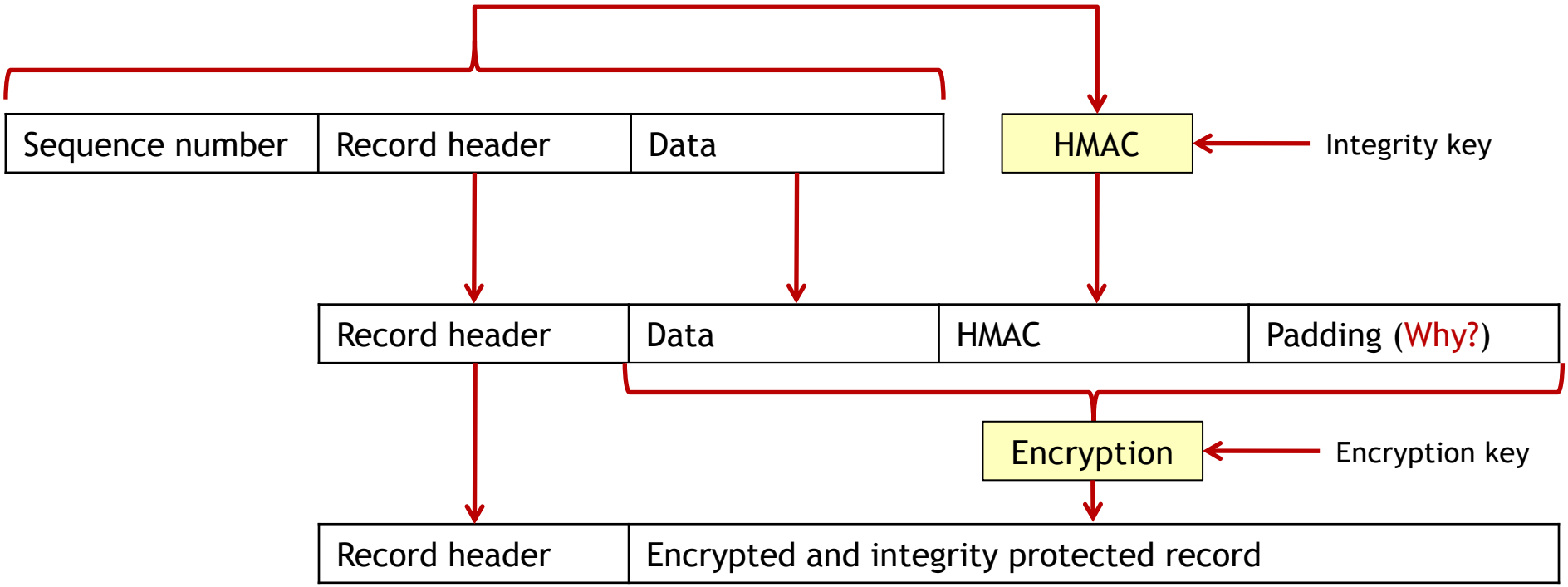
All records sent after ChangeCipherSpec are encrypted and integrity protected

All of this protection is afforded by the algorithms identified in the cipher suite chosen by the server

Example: `TLS_RSA_WITH_AES_128_CBC_SHA`

- Encryption provided using 128 bit AES in CBC mode
- Integrity protection provided by HMAC-SHA-1

Note: Data is protected one record at a time



Protocol summary

At a high level, this protocol proceeds in four phases

- Setup and parameter negotiation
- Key exchange/derivation
- Authentication
- Data transmission

For security reasons, both parties participate in (almost) all phases

- Setup: Client proposes, server chooses
- Key derivation: Randomness contributed by both parties
- Authentication: Usually, just the server is authenticated (Why?)
- Data transmission: Both parties can encrypt and integrity check

The low level details are not much more complicated than that!