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

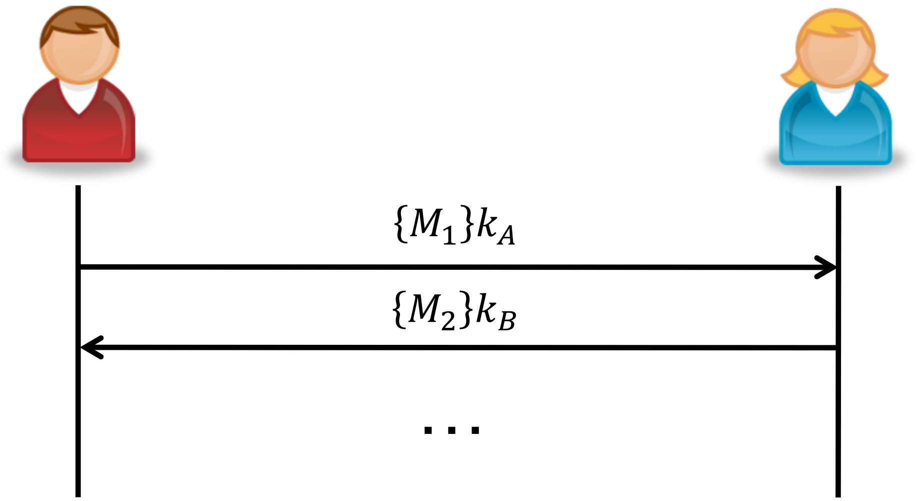
- Perfect forward secrecy
- Forward secrecy
- Backward secrecy

Deniable authentication protocols

Denial of service

- Computational puzzles
- Guided tour puzzles

Not all seemingly reasonable key exchange protocols provide perfect forward secrecy



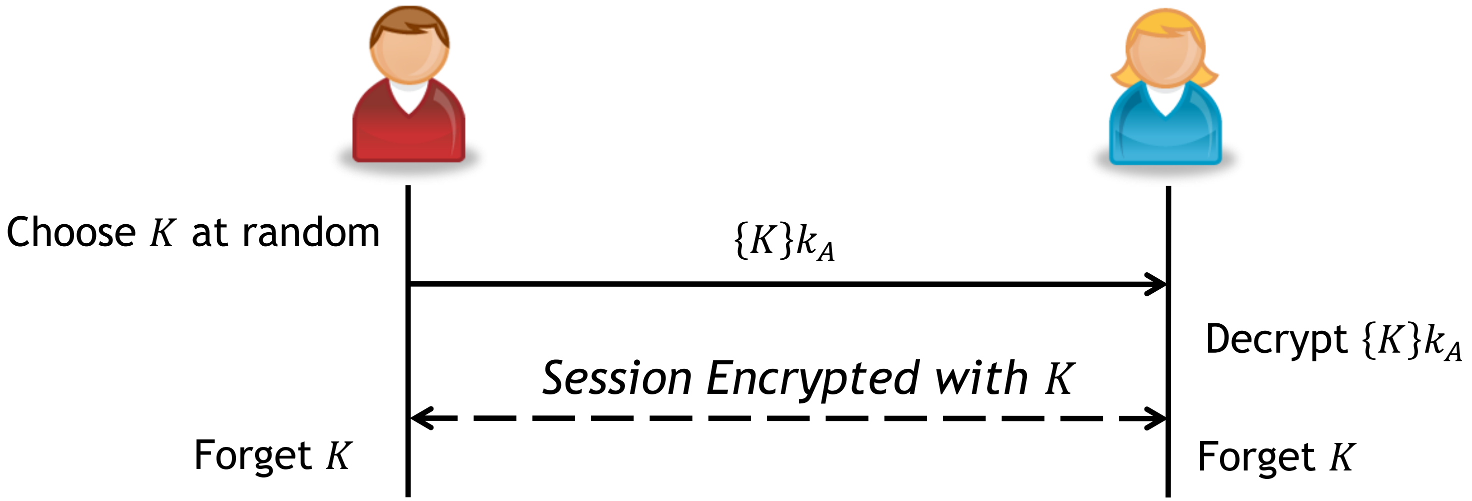
This protocol is inefficient and fails to provide perfect forward secrecy

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!

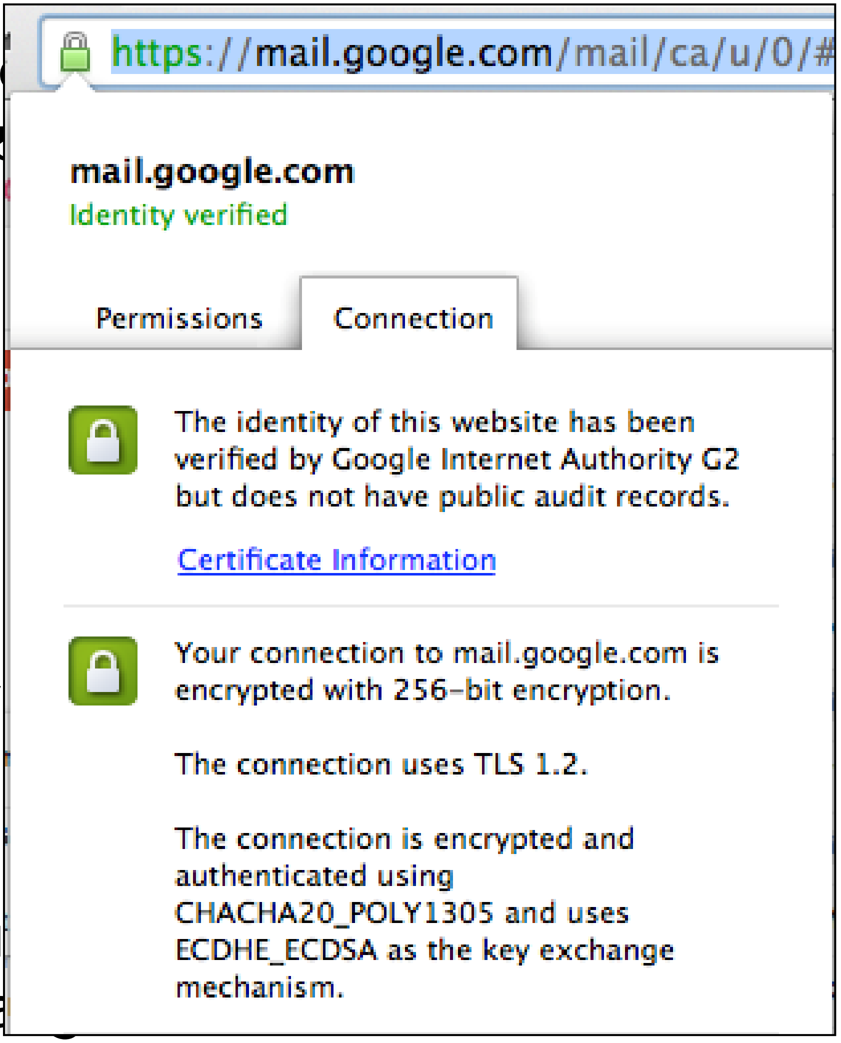
The previous protocol is similar to the “RSA key exchange” method supported in SSL/TLS

Many TLS cipher suites use RSA key

- TLS_RSA_EXPORT_WITH_RC4_40_MD
- 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



https://mail.google.com/mail/ca/u/0/#

mail.google.com
Identity verified

The identity of this website has been verified by Google Internet Authority G2 but does not have public audit records.

Certificate Information

Your connection to mail.google.com is encrypted with 256-bit encryption.

The connection uses TLS 1.2.

The connection is encrypted and authenticated using CHACHA20_POLY1305 and uses ECDHE_ECDSA as the key exchange mechanism.

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.

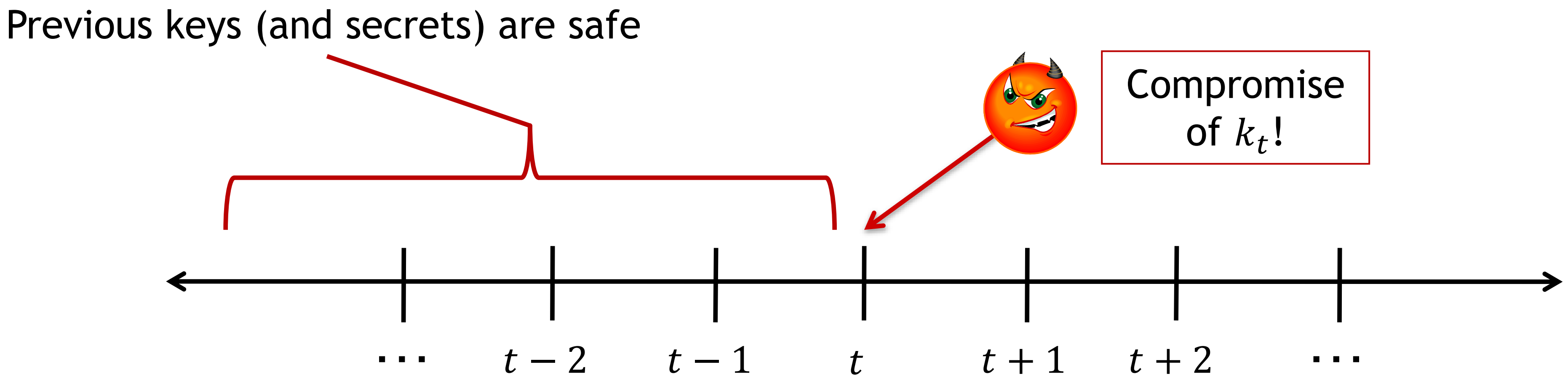
Simple idea: Set up a shared key for the chat room, and encrypt all messages using this key

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 $i < t$.

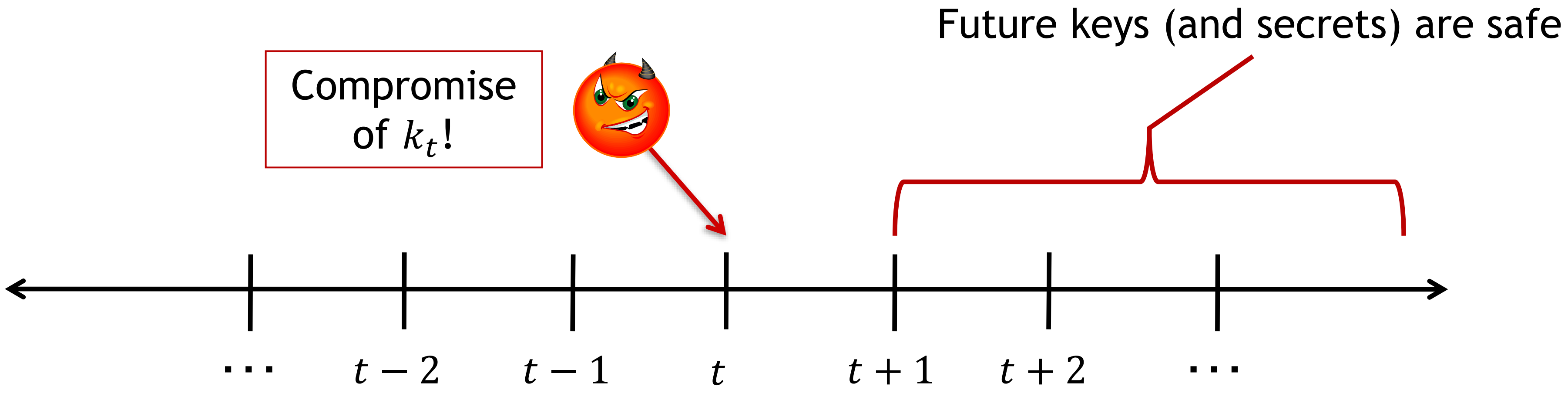


Informally: Later compromise does not reveal earlier keys

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 $i > t$.



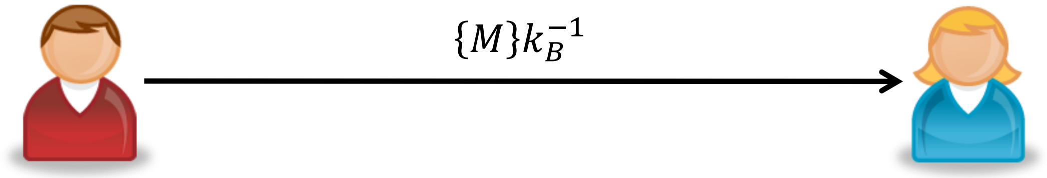
Informally: Earlier compromise does not reveal later keys

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!

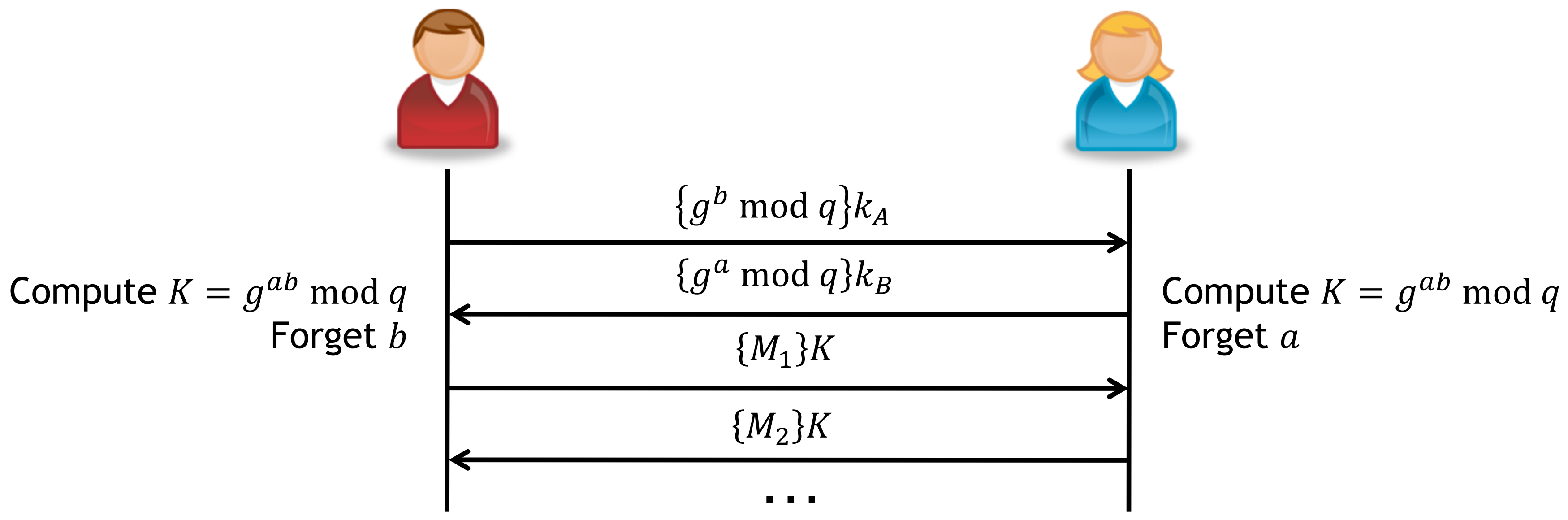


Sometimes, we would like Alice to be able to verify the authenticity of a message without being able to convince a third party that the message came from Bob

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

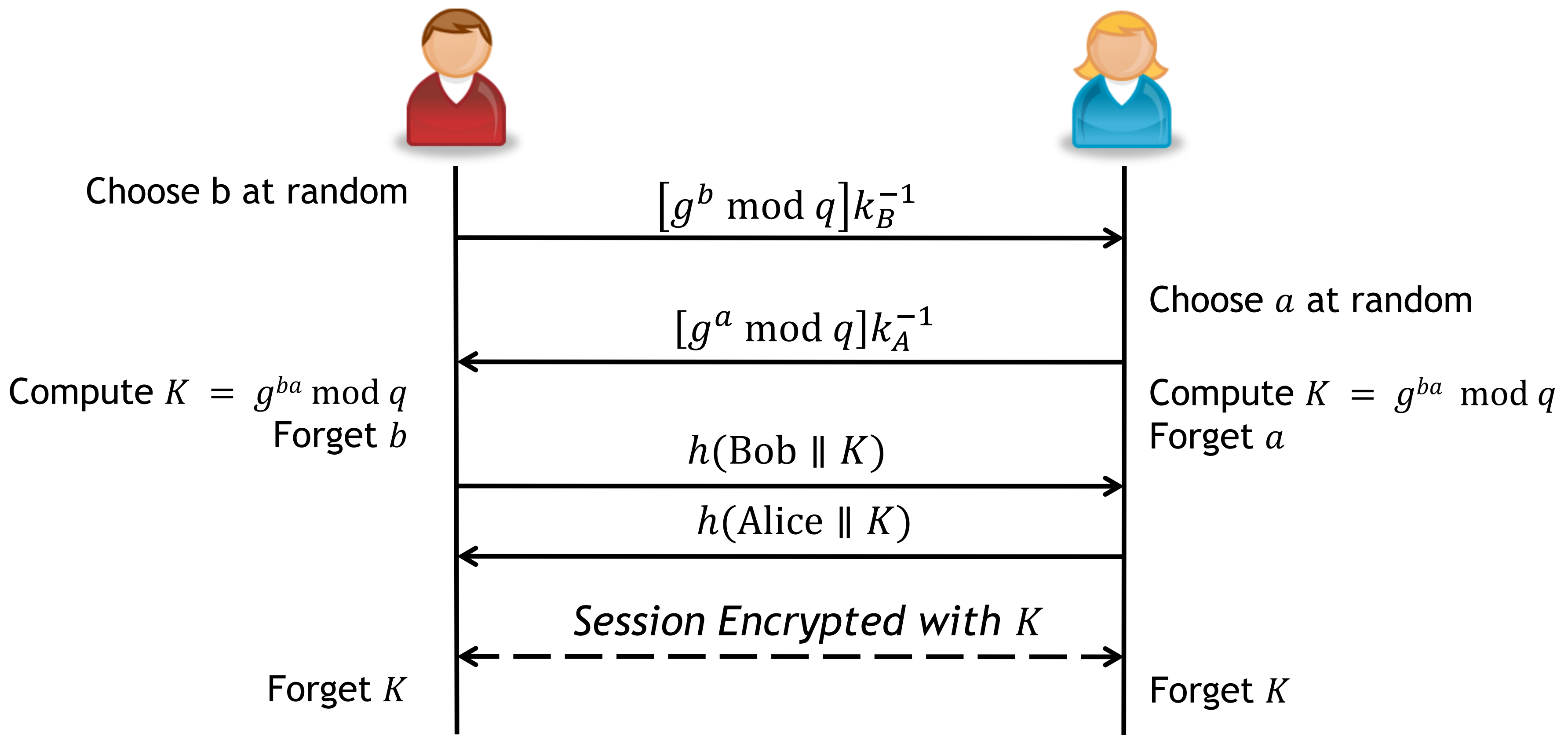
- Alice knows that only Bob could have derived K
- Bob knows that only Alice could have derived K
- Each party knows which M_is they sent

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?


Choose b at random
$[g^b \bmod q]k_B^{-1}$
Choose a at random
$[g^a \bmod q]k_A^{-1}$
Compute $K = g^{ba} \bmod q$
Forget b
Compute $K = g^{ba} \bmod q$
Forget a
$h(\text{Bob} \parallel K)$
$h(\text{Alice} \parallel K)$
Session Encrypted with K
Forget K
Forget K

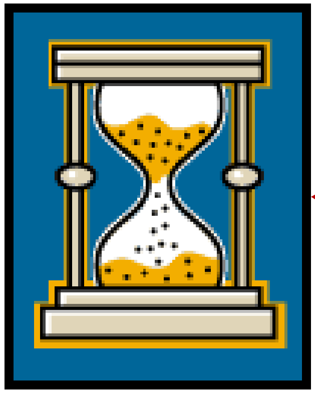
Computer security is typically defined with respect to three types of properties

How do I ensure that my secrets remain secret?

Confidentiality

Can I trust the services that I use?

Integrity

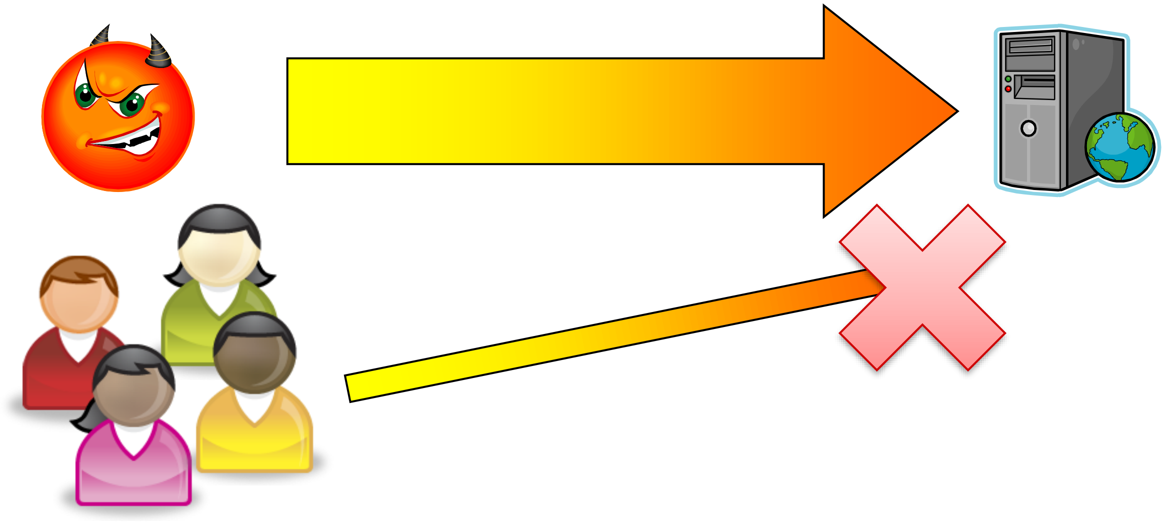


Availability

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



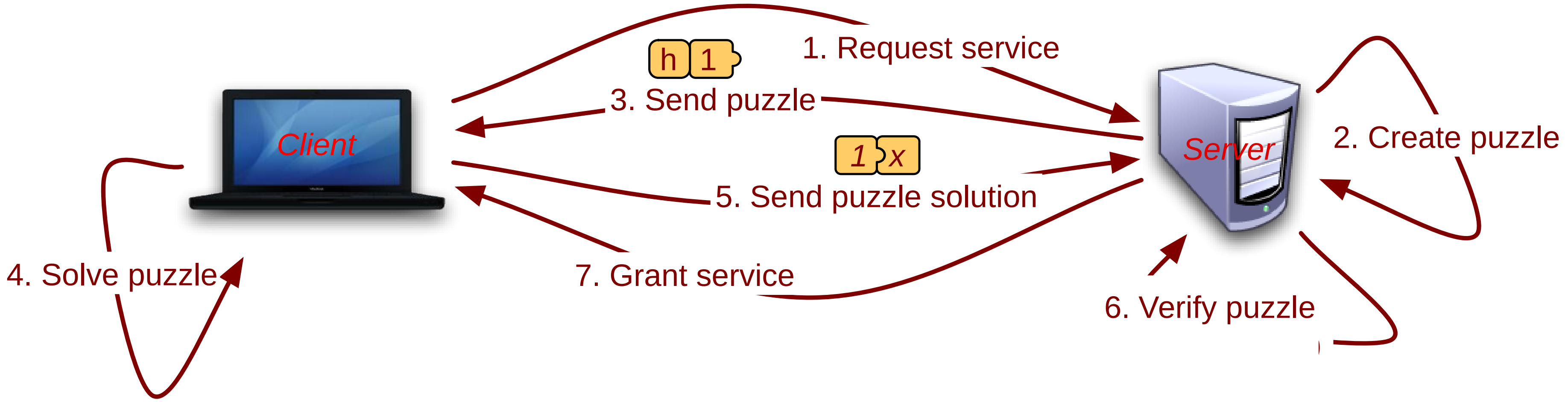
Denial of service attacks are typically abusing a resource disparity

- 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

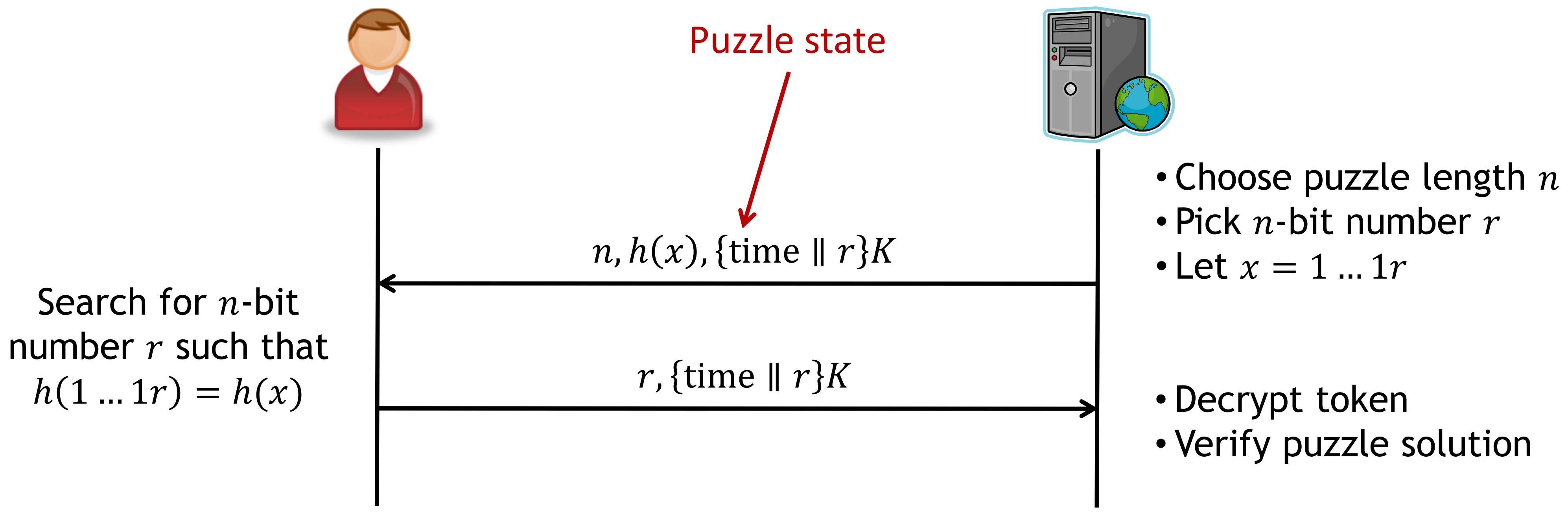
Question: Why do computational puzzles have these requirements?

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
$n, h(x), \{\text{time} \parallel r\}K$
$r, \{\text{time} \parallel r\}K$
Search for n-bit number r such that $h(1 \ldots 1r) = h(x)$
• Choose puzzle length n
• Pick n-bit number r
• Let $x = 1 \ldots 1r$
• 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?

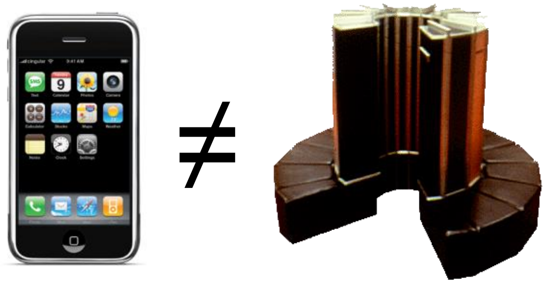
Strengths and weaknesses of computational puzzles

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!

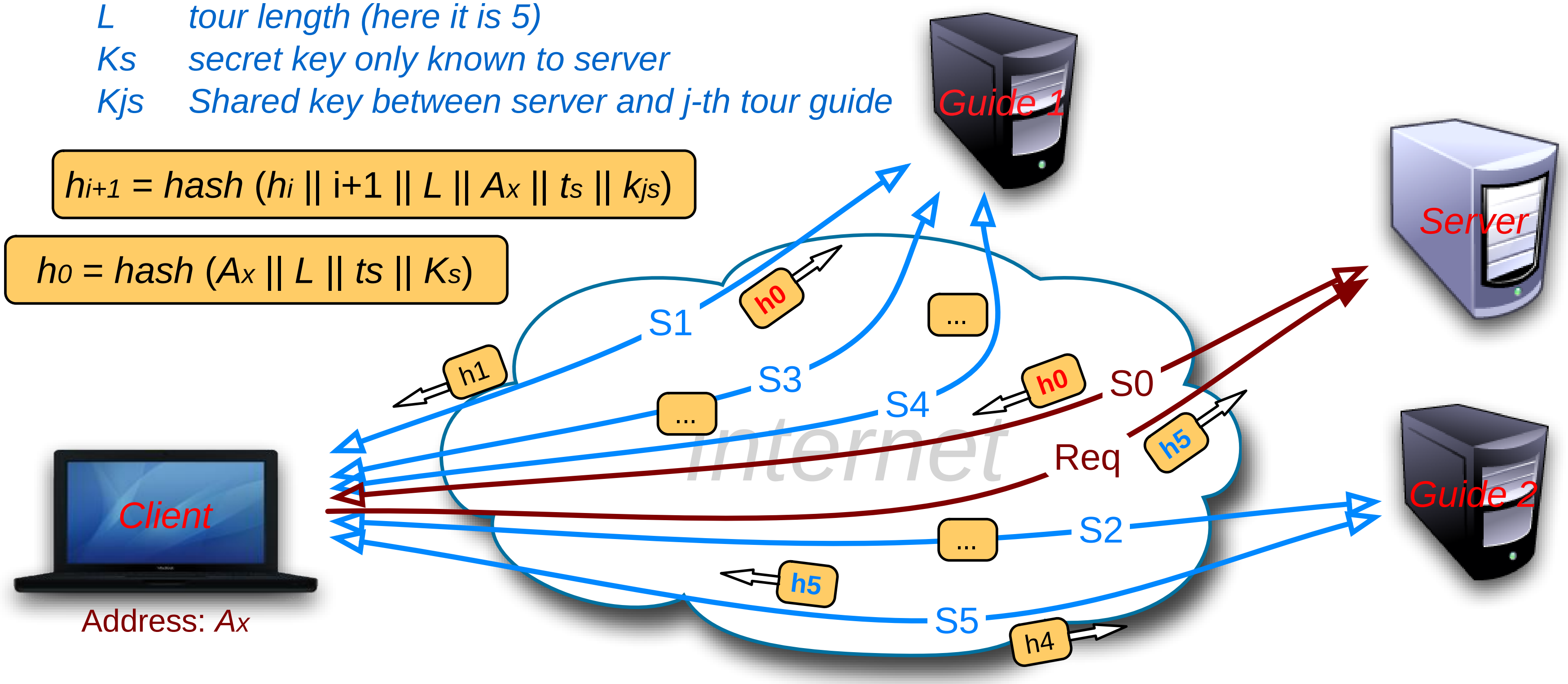
- Puzzle: An ordered list of tour guides to contact
- Solution: A serial computation carried out by these nodes in order

How do guided tours work?

L	*tour length (here it is 5)*
Ks	*secret key only known to server*
Kjs	*Shared key between server and j-th tour guide*

$h_{i+1} = hash\ (h_i \,||\, i{+}1 \,||\, L \,||\, A_x \,||\, t_s \,||\, k_{js})$

$h_0 = hash\ (A_x \,||\, L \,||\, t_s \,||\, K_s)$



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

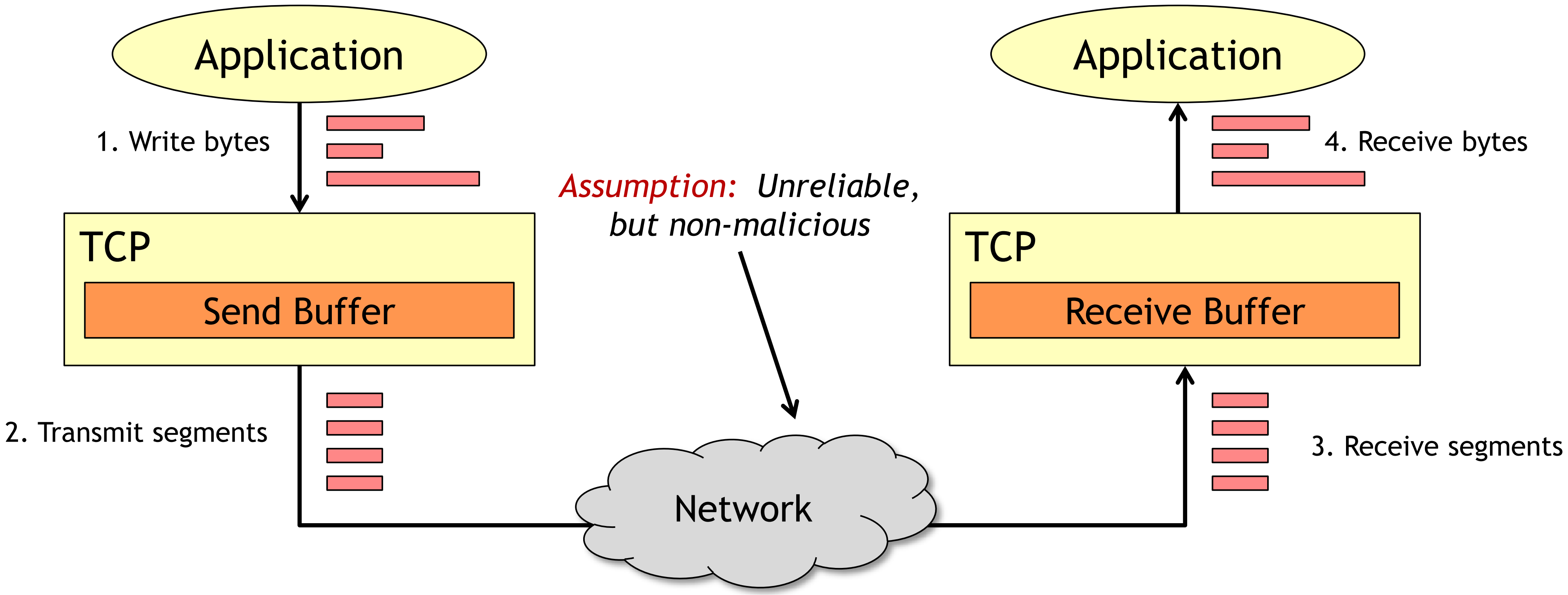
TCP is reliable, but not secure

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



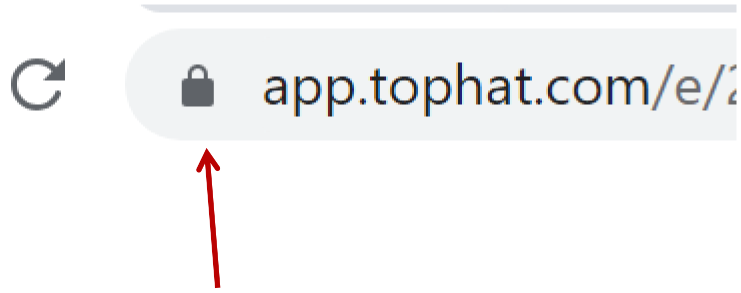
SSL and TLS were developed to provide applications with a secure means of utilizing TCP

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

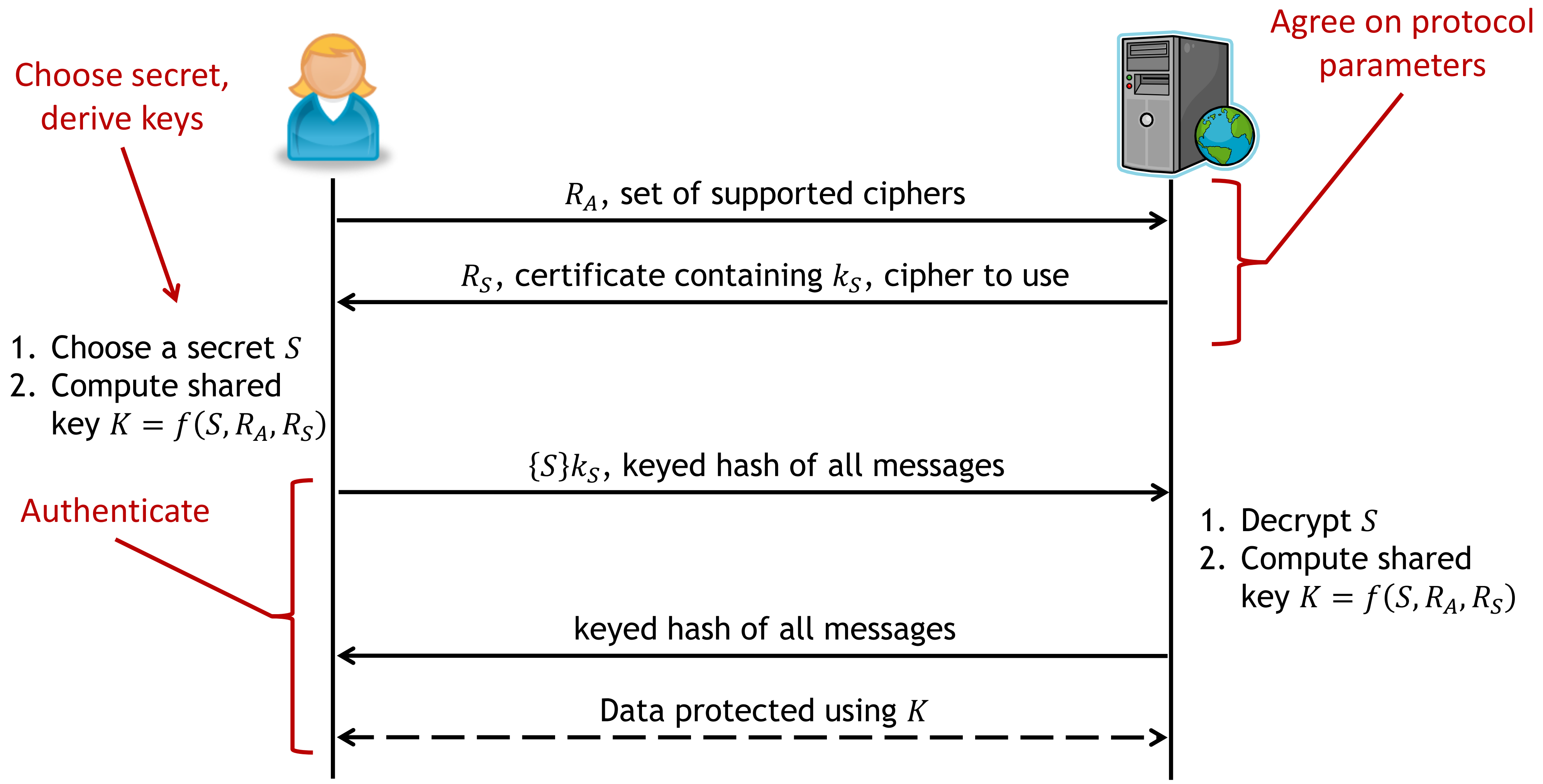
Microsoft developed PCT by tweaking SSL v2

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

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


Choose secret, derive keys
Agree on protocol parameters
R_A, set of supported ciphers
R_S, certificate containing k_S, cipher to use
1. Choose a secret S
2. Compute shared key $K = f(S, R_A, R_S)$
$\{S\}k_S$, keyed hash of all messages
Authenticate
1. Decrypt S
2. Compute shared key $K = f(S, R_A, R_S)$
keyed hash of all messages
Data protected using K

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

Symmetric key cryptography will be done using AES

Example: `TLS_RSA_WITH_AES_128_CBC_SHA`

SHA-1 will be used to hash

This cipher suite is defined in the TLS specification

Public key operations will be carried out using RSA

AES will use a 128-bit key

AES will operate in CBC mode

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

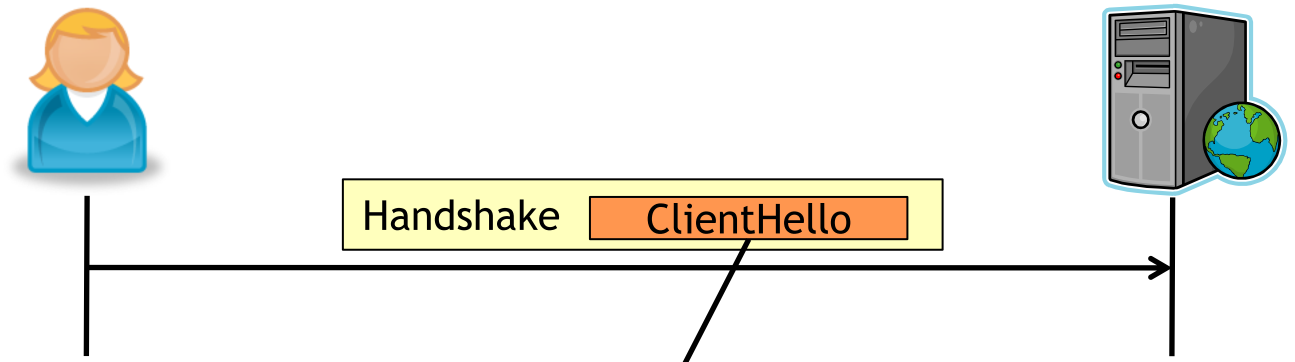
Note that a single record may contain multiple messages inside of it

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

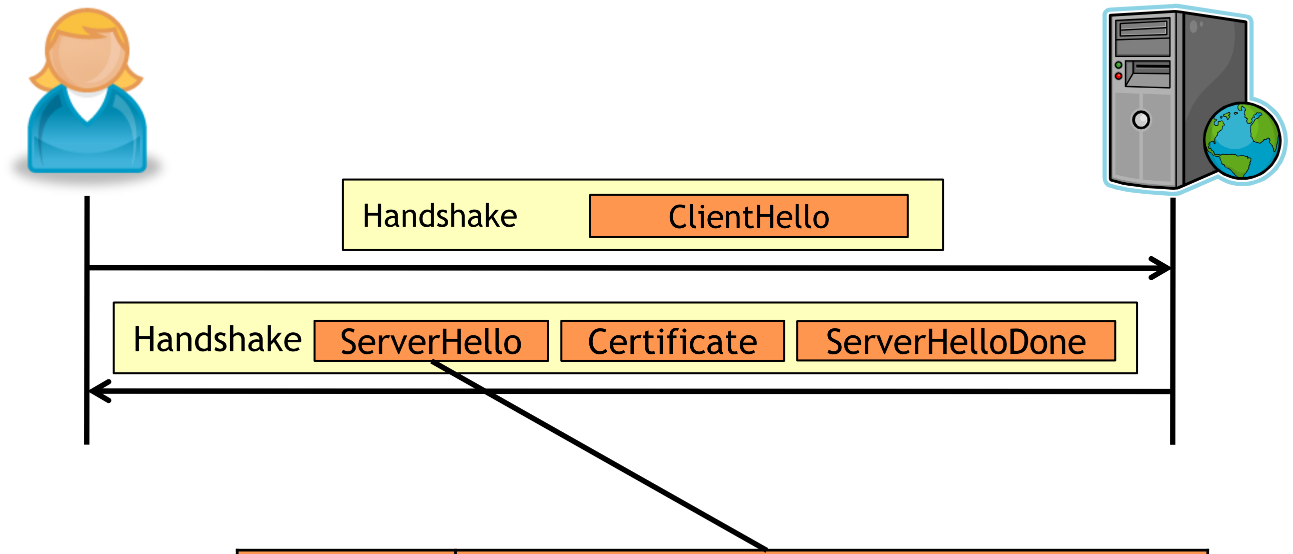


Bytes	*Content*
1	Message type: ClientHello
3	Length
2	Version number
32	R_A
1	Length of session ID
var	Session ID
2	Length of cipher suite list
var	List of supported cipher suites
1	Length of compression mode list
var	List of supported compression modes

Used to resume sessions. More on this later.

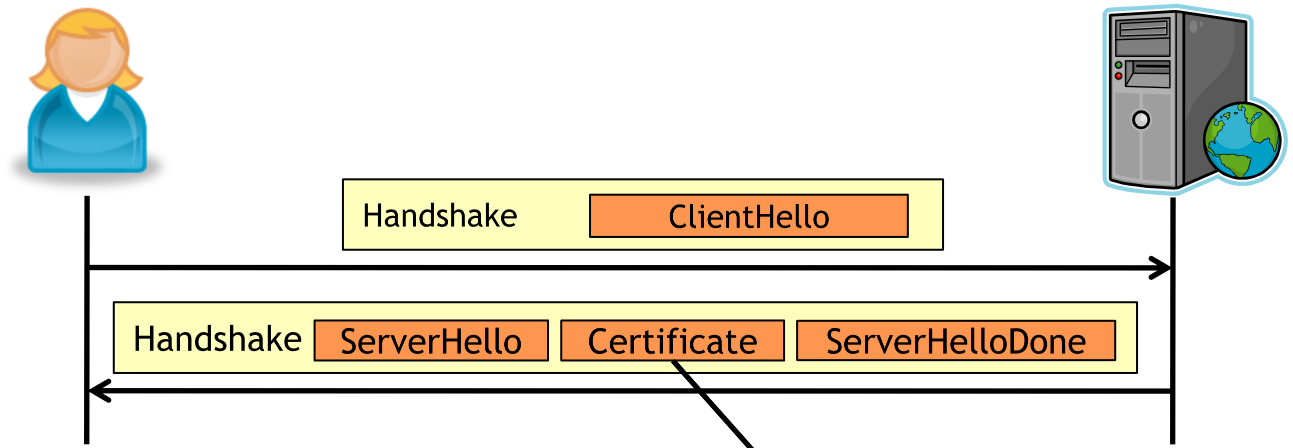
For efficiency reasons, TLS compresses the data that it transmits.

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

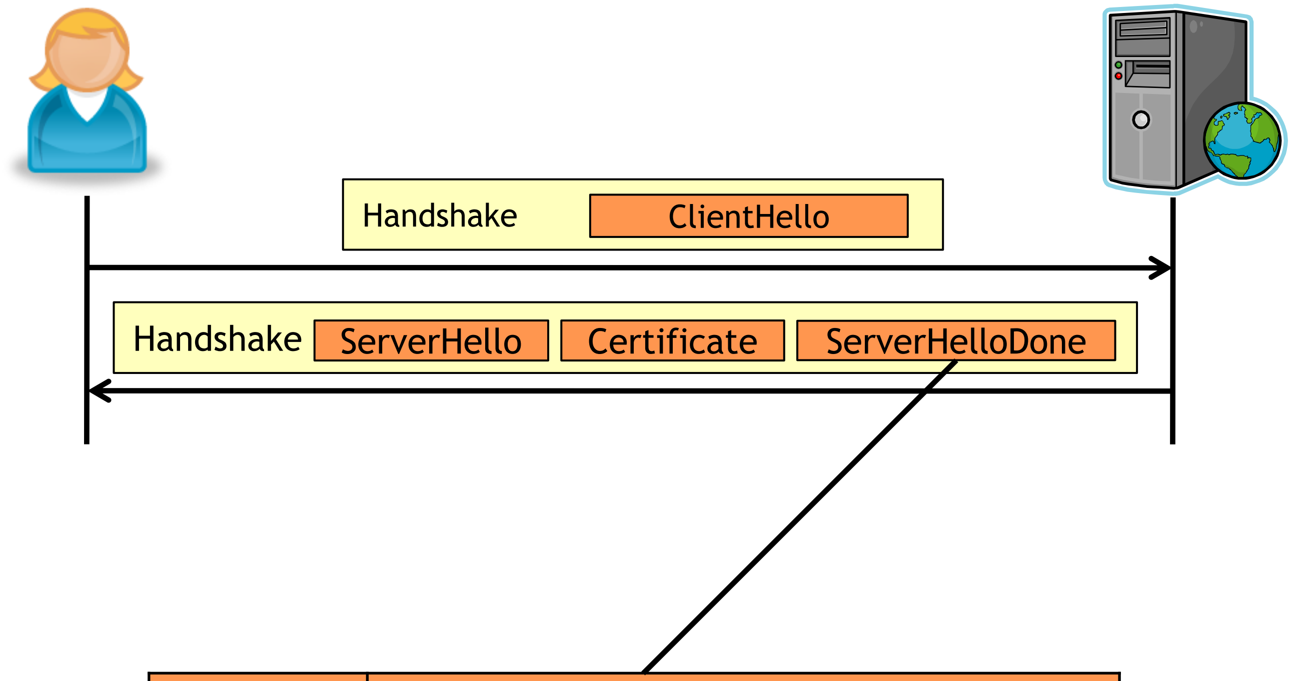
Handshake: Message 2



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

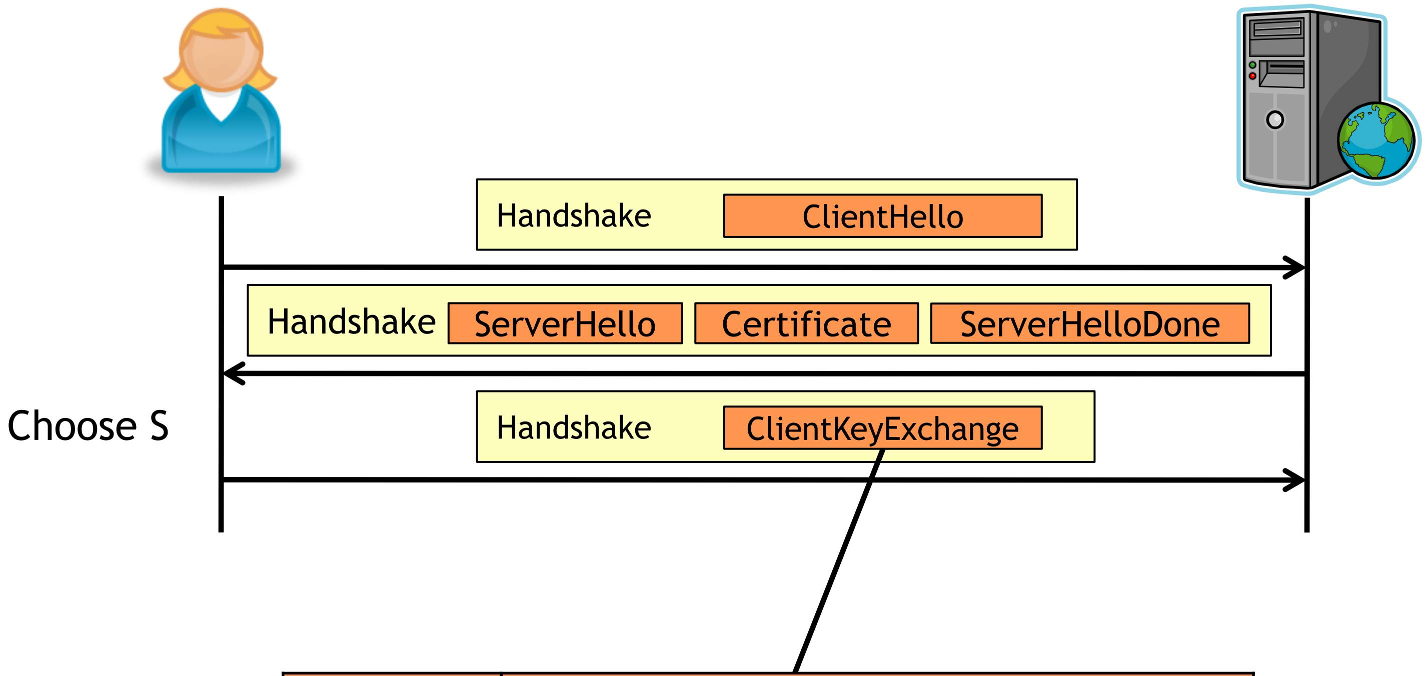
Why?

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

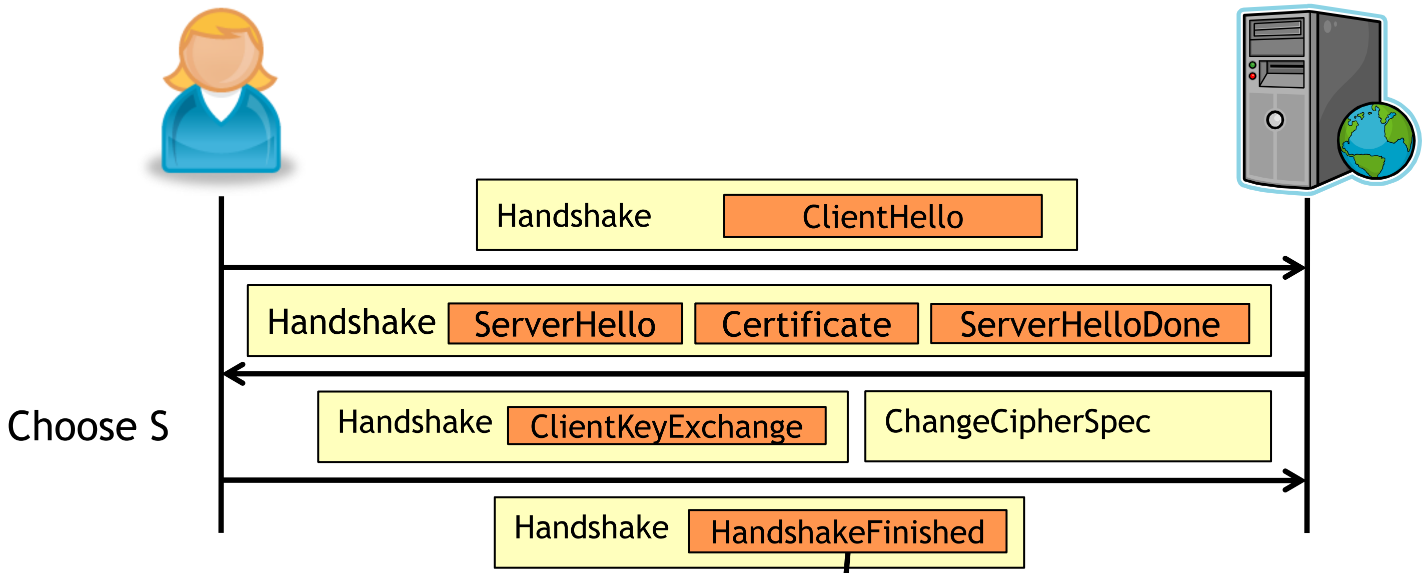
- Client → server MAC key
- Server → client MAC key
- Client → server symmetric encryption key
- Server → client symmetric encryption key
- Client → server IV
- Server → client IV

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

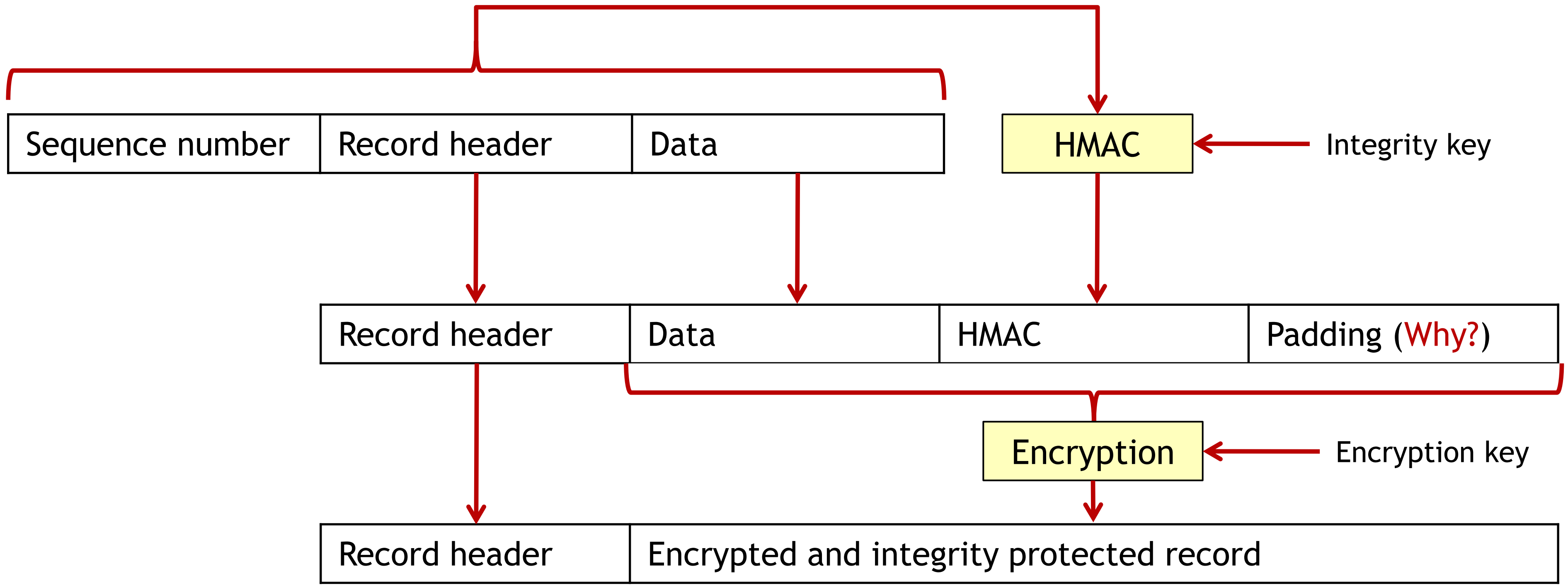
All records sent after ChangeCipherSpec are encrypted and integrity protected

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!

This handshake procedure is fairly heavyweight

Public key cryptography is used by both parties

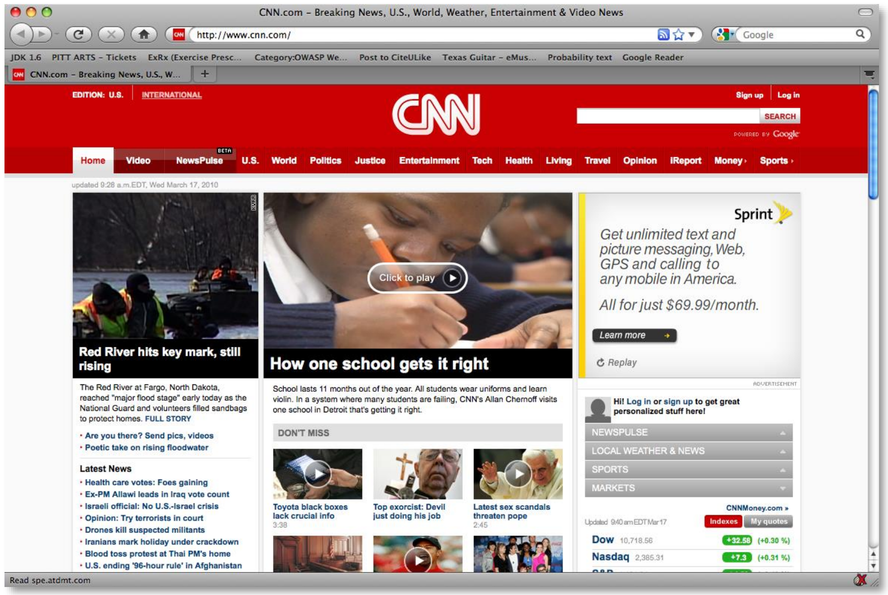
- Alice encrypts her pre-master secret using the server's public key
- The server decrypts this pre-master secret

So what! Aren't connections long lived?

Example: Visiting http://www.cnn.com

Visiting this single web page triggers over 130 separate HTTP connections!

This is less than optimal. Can we do better?


CNN.com – Breaking News, U.S., World, Weather, Entertainment & Video News

http://www.cnn.com/

updated 9:28 a.m.EDT, Wed March 17, 2010

Red River hits key mark, still rising

The Red River at Fargo, North Dakota, reached "major flood stage" early today as the National Guard and volunteers filled sandbags to protect homes. FULL STORY

- Are you there? Send pics, videos
- Poetic take on rising floodwater

Latest News

- Health care votes: Foes gaining
- Ex-PM Allawi leads in Iraq vote count
- Israeli official: No U.S.-Israel crisis
- Opinion: Try terrorists in court
- Drones kill suspected militants
- Iranians mark holiday under crackdown
- Blood toss protest at Thai PM's home
- U.S. ending '96-hour rule' in Afghanistan

How one school gets it right

School lasts 11 months out of the year. All students wear uniforms and learn violin. In a system where many students are failing, CNN's Allan Chernoff visits one school in Detroit that's getting it right.

DON'T MISS

Toyota black boxes lack crucial info 3:38

Top exorcist: Devil just doing his job

Latest sex scandals threaten pope 2:45

Get unlimited text and picture messaging, Web, GPS and calling to any mobile in America.

All for just $69.99/month.

Hi! Log in or sign up to get great personalized stuff here!

Updated 9:40 am EDT Mar 17

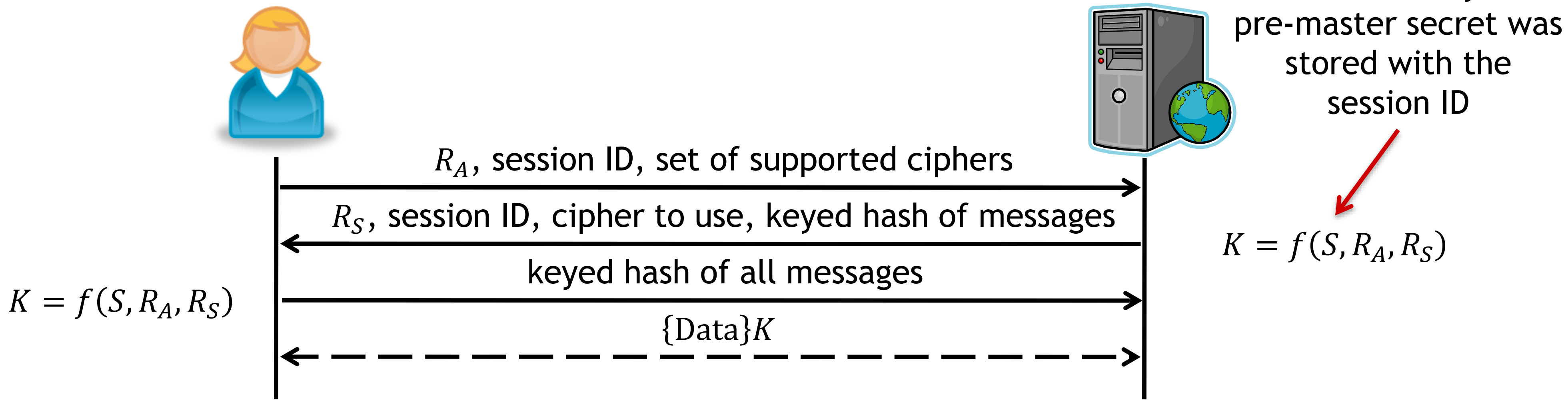
Dow	10,718.56	+32.58	(+0.30 %)
Nasdaq	2,385.31	+7.3	(+0.31 %)

Read spe.atdmt.com

Sessions vs. Connections

In TLS, sessions are assumed to be long-lived entities that may involve many smaller connections

Connections can be spawned from an existing session using a streamlined session resumption protocol:



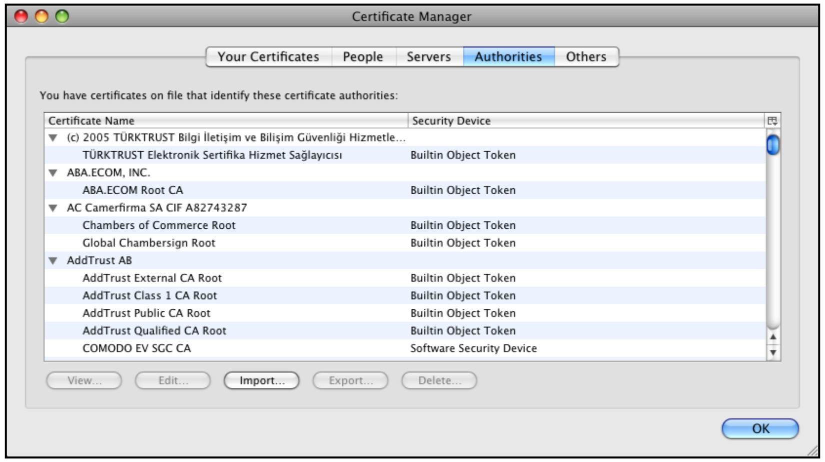
Note: Previously used pre-master secret was stored with the session ID
R_A, session ID, set of supported ciphers
R_S, session ID, cipher to use, keyed hash of messages
keyed hash of all messages
$\{\text{Data}\}K$
$K = f(S, R_A, R_S)$
$K = f(S, R_A, R_S)$

In this model, a single session could be set up with CNN and connections can be spawned as needed to retrieve content

Using HTTP/2, this concept can be taken even farther: server can respond with data for queries before they're even requested

Servers authenticate themselves using certificates. How do we authenticate the certificates?

Most TLS deployments use an oligarchy PKI model



You have certificates on file that identify these certificate authorities:

Certificate Name	Security Device
▼ (c) 2005 TÜRKTRUST Bilgi İletişim ve Bilişim Güvenliği Hizmetle...	
TÜRKTRUST Elektronik Sertifika Hizmet Sağlayıcısı	Builtin Object Token
▼ ABA.ECOM, INC.	
ABA.ECOM Root CA	Builtin Object Token
▼ AC Camerfirma SA CIF A82743287	
Chambers of Commerce Root	Builtin Object Token
Global Chambersign Root	Builtin Object Token
▼ AddTrust AB	
AddTrust External CA Root	Builtin Object Token
AddTrust Class 1 CA Root	Builtin Object Token
AddTrust Public CA Root	Builtin Object Token
AddTrust Qualified CA Root	Builtin Object Token
COMODO EV SGC CA	Software Security Device

That is, as long as the server presents a certificate chain that uses one of our trusted roots, we're happy

What about naming?

- Servers are usually known by their DNS names
- X.509 is not set up for DNS naming
- Usually the CN field of the X.509 certificate contains the DNS name

Example: C = US, ST = Washington, L = Seattle, O = Amazon.com Inc., CN = www.amazon.com ⟵ Why is this safe?

Summary of TLS

Although TCP provides a reliable data transfer protocol, it is not secure

- TCP can recover from bit-flips and dropped packets
- But malicious adversaries can alter data undetected

TLS provides cryptographic confidentiality and integrity protection for data sent over TCP

The security afforded by TLS is defined by using cipher suites

- Developers to easily incorporate new algorithms
- Security professionals can tune the level of security offered
- Breaking a cipher does not break the protocol

What is a hash function?

Recall: A hash function is a function that maps a variable-length input to a fixed-length code

Hash functions should possess the following 3 properties:

- Preimage resistance: Given a hash output value z, it should be infeasible to calculate a message x such that $H(x) = z$
- Collision resistance: It is infeasible to find two messages x and y such that $H(x) = H(y)$
- Second preimage resistance: Given a message x, it is infeasible to calculate a second message y such that $H(x) = H(y)$

Question: Why are cryptographic hash functions important to PKIs?

Digital signature operations are expensive to compute! Instead of signing a certificate c, we actually sign $H(c)$.

What happens if we don’t have these properties?

Attack 1: No preimage resistance

- One attack is being able to recover a password from $H(\text{password})$
- Not critical to the security of PKIs, but would cause issues in general

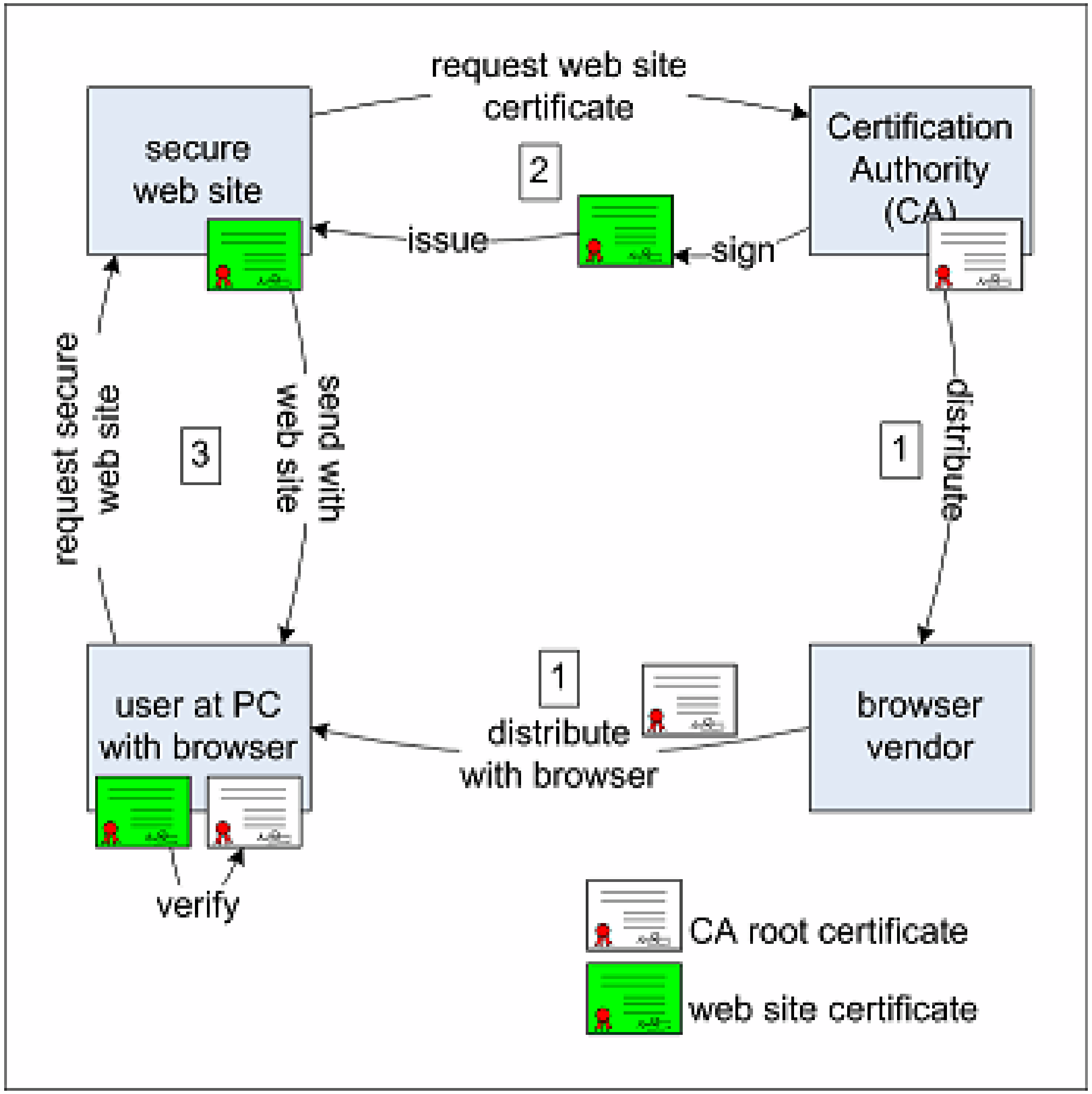
Attack 2: No second preimage resistance

- Assume that we have a message m_1 with a signature computed over $H(m_1)$
- If we don’t have second preimage resistance, we can find a message m_2 such that $H(m_1) = H(m_2)$
- The result: It looks like the signer signed m_2!

Attack 3: No collision resistance

- This means that we can find two messages that have the same hash
- The authors use a clever variant of this type of attack to construct two certificates that have the same MD5 hash
- This is bad news…

How TLS should work...

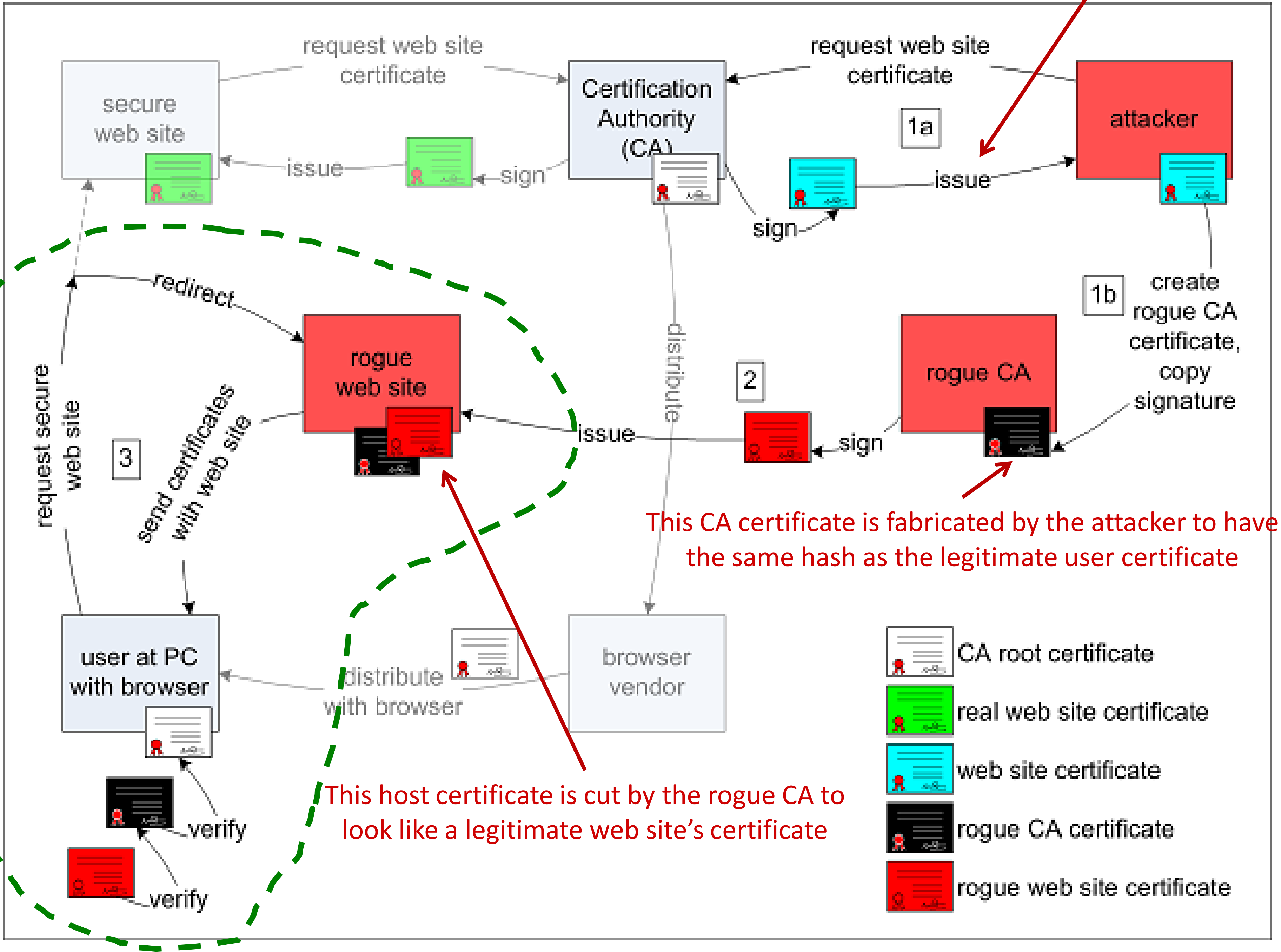


What’s the big deal?

Question: Can you describe a scenario in which having two certificates with the same hash would be problematic?

Attack Overview

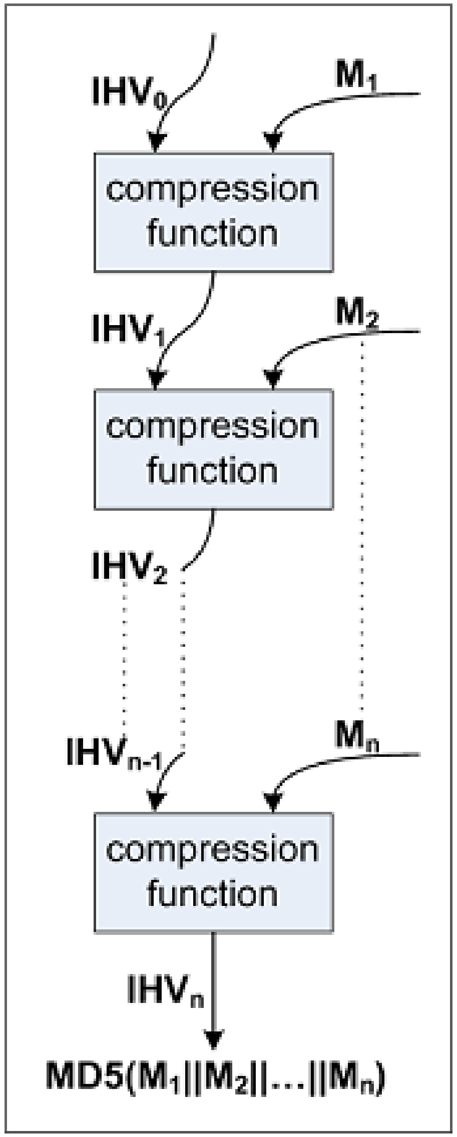
This is a legitimate host certificate obtained through standard channels



This CA certificate is fabricated by the attacker to have the same hash as the legitimate user certificate

This host certificate is cut by the rogue CA to look like a legitimate web site's certificate

How does MD5 work?



Given a variable length input string, MD5 produces a 128-bit hash value

MD5 uses a Merkle-Damgård iterative construction

- 128-bit intermediate hash value (IHV) and a 512-bit message chunk are fed into a compression function that generates a 128-bit output
- This output is concatenated to the next 512-bit message chunk and the process is repeated
- The final IHV is the hash value for the message

Note that:

- The initial message must be padded out to a multiple of 512 bits
- IHV_0 is a publicly-known fixed value (0x67452301 0xEFCDAB89 0x98BADCFE 0x10325476)

Early attacks against MD5

Early partial attacks on MD5 were suggestive of larger troubles

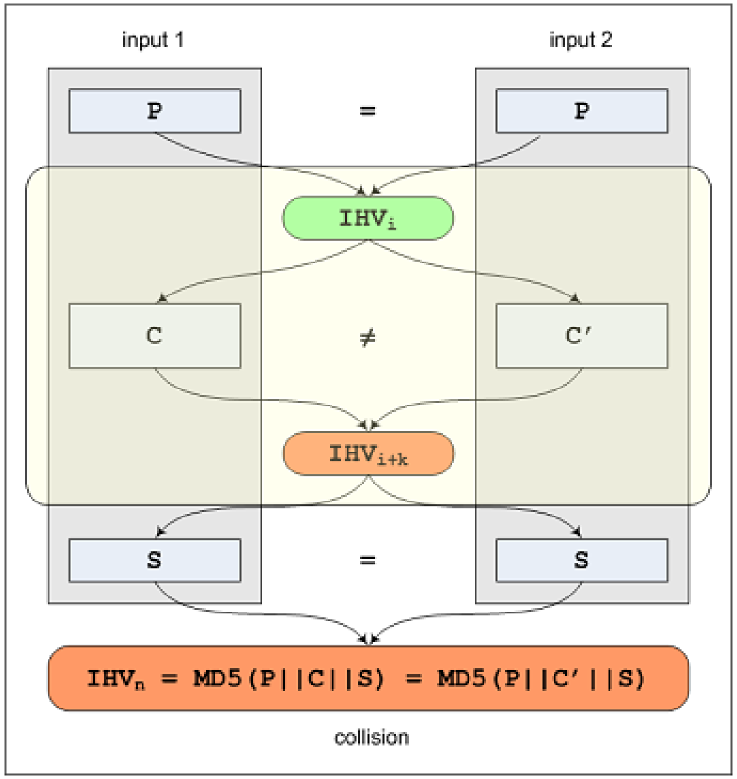
- [1993] Den Boer and Bosselaers found a partial collision of the MD5 compression function (two different IHVs lead to the same output value)
- [1996] Dobbertin found a full collision of the MD5 compression function

The first real attack on MD5 was found by Wang and Yu in 2004

Given any IHV, they showed how to compute two pairs $\{M_1, M_2\}$ and $\{M_1', M_2'\}$ such that:

- $IHV_1 = CF(IHV, M_1) \neq IHV_1' = CF(IHV, M_1')$
- $IHV_2 = CF(IHV_1, M_2) = CF(IHV_1', M_2')$

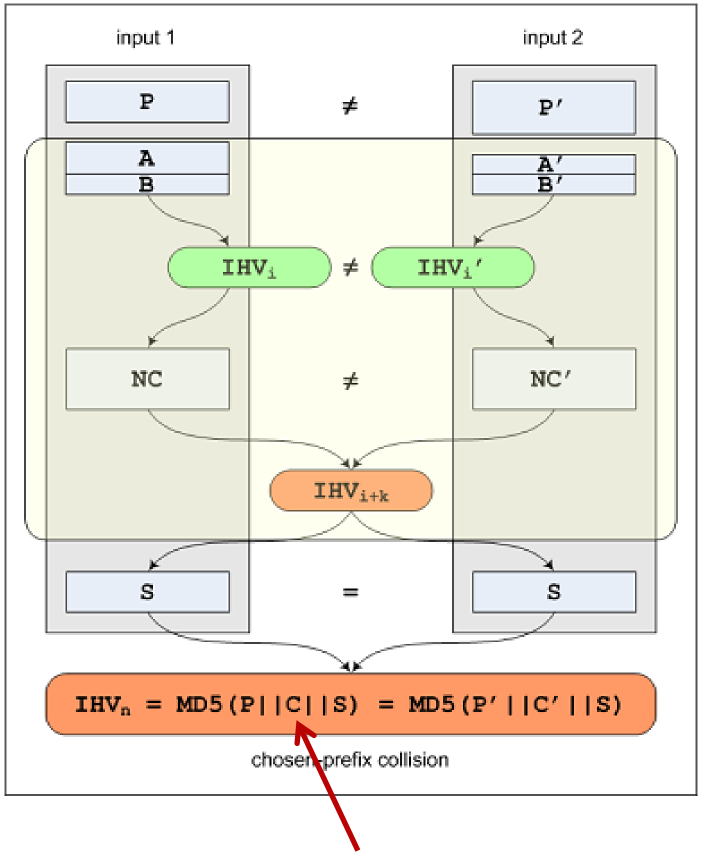
This yields a fairly scary generalization



In 2007 Stevens refined the Wang and Yu attack to allow differing message prefixes

How does this process work?

1. Choose P and P’ at will
2. Choose A and A’ s.t. (P || A) and (P’ || A’) are of the same bit length
3. In a “birthday step”, choose B and B’ such that (P || A || B) and (P’ || A’ || B’) are a multiple of 512 bits and the output IHVs have a special structure
4. This special structure allows the attacker to find two near collision blocks NC and NC’ such that the resulting MD5 function collides!



Note: C = A || B || NC

One use of this attack is generating different documents (plus some hidden content) that end up with the same hash value/signature!

This also opens the door to forged certificates…

How can we launch this attack against X.509?

An X.509 certificate contains quite a bit of information

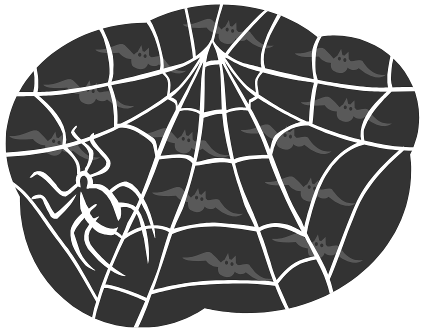
- Version
- Serial number: Must be unique amongst all certificates issued by the same CA
- Signature algorithm ID: What algorithm was used to sign this certificate?
- Issuer's distinguished name (DN): Who signed this certificate?
- Validity interval: Start and end of certificate validity
- Subject's DN: Who is this certificate for?
- Subject's public key information: The public key of the subject
- Issuer's unique ID: Used to disambiguate issuers with the same DN
- Subjects unique ID: Used to disambiguate subjects with the same DN
- Extensions: Typically used for key and policy information
- Signature: A digital signature of (a hash of) all other fields

---

Insight 1: Stevens's chosen prefix attack means that it might be possible to generate two certificates with different subjects, but the same signature!

Insight 2: Collision blocks can potentially be hidden is (part) of the public key block and/or in extension fields!

Successfully launching this attack means finding a CA that will grant a certificate that has a signature over an MD5 hash



To find such a server, the authors wrote a web crawler

- Crawler ran for about a week
- Found 100,000 SSL certificates
- 30,000 certificates signed by "trusted" CAs
- Of these, 6 issued certificates with MD5-based signatures signed in 2008

Of the certificates found with signatures over MD5 hashes, 97% were issued by RapidSSL (http://www.rapidssl.com/)

RapidSSL issues certificates in an online manner, so they actually made an ideal target for launching this attack

- Predictable timing
- Not human-based, so multiple requests would not seem strange

Question: Why attack a production server instead of a test server?

Unfortunately, we have no control over the certificate serial number or validity fields!

Question: Why might this pose a problem?

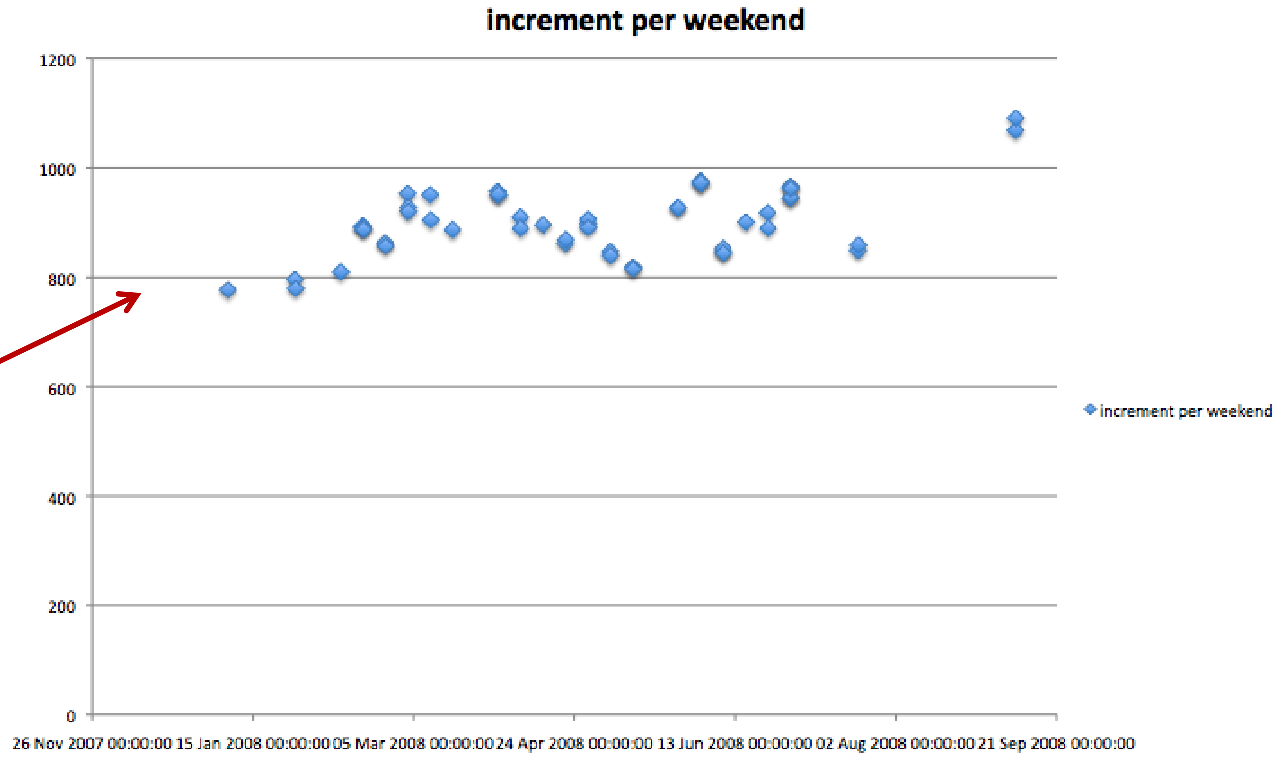
- Because these fields occur in the "chosen" prefix of the certificate and thus must be known before the collision blocks can be computed!

Predicting the validity period turned out to be trivial

- Certificate always issued 6 seconds after it was requested
- Valid for exactly one year

Furthermore, it turns out that RapidSSL uses a counter to populate the serial number field!

800-1000 certificates issued per weekend



Setting up a legitimate certificate request

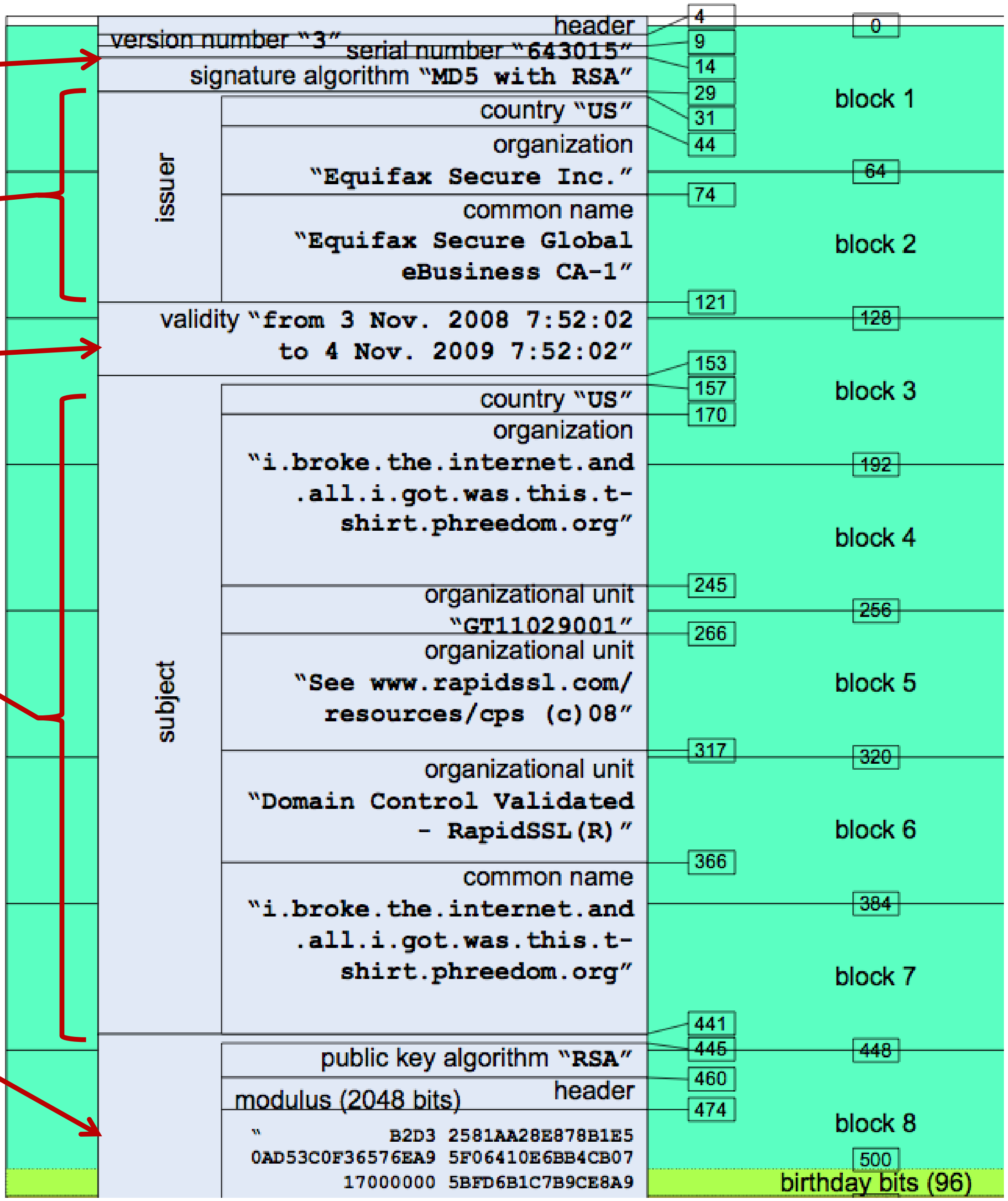
Predicted serial number

Info pulled from CA certificate

Derived validity period

Subject information completely chosen by the attacker

Question: How do you choose a public key (2048-bit modulus and an exponent) when part of it is the collision blocks needed to make the attack work?!?!


header
version number "3"
serial number "643015"
signature algorithm "MD5 with RSA"
issuer
country "US"
organization "Equifax Secure Inc."
common name "Equifax Secure Global eBusiness CA-1"
validity "from 3 Nov. 2008 7:52:02 to 4 Nov. 2009 7:52:02"
subject
country "US"
organization "i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom.org"
organizational unit "GT11029001"
organizational unit "See www.rapidssl.com/resources/cps (c)08"
organizational unit "Domain Control Validated - RapidSSL(R)"
common name "i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom.org"
public key algorithm "RSA"
modulus (2048 bits)
header
" B2D3 2581AA28E878B1E5 0AD53C0F36576EA9 5F06410E6BB4CB07 17000000 5BFD6B1C7B9CE8A9
4
9
14
29
31
44
74
121
153
157
170
245
266
317
366
441
445
460
474
0
64
128
192
256
320
384
448
500
block 1
block 2
block 3
block 4
block 5
block 6
block 7
block 8
birthday bits (96)

Answer: By being extremely clever

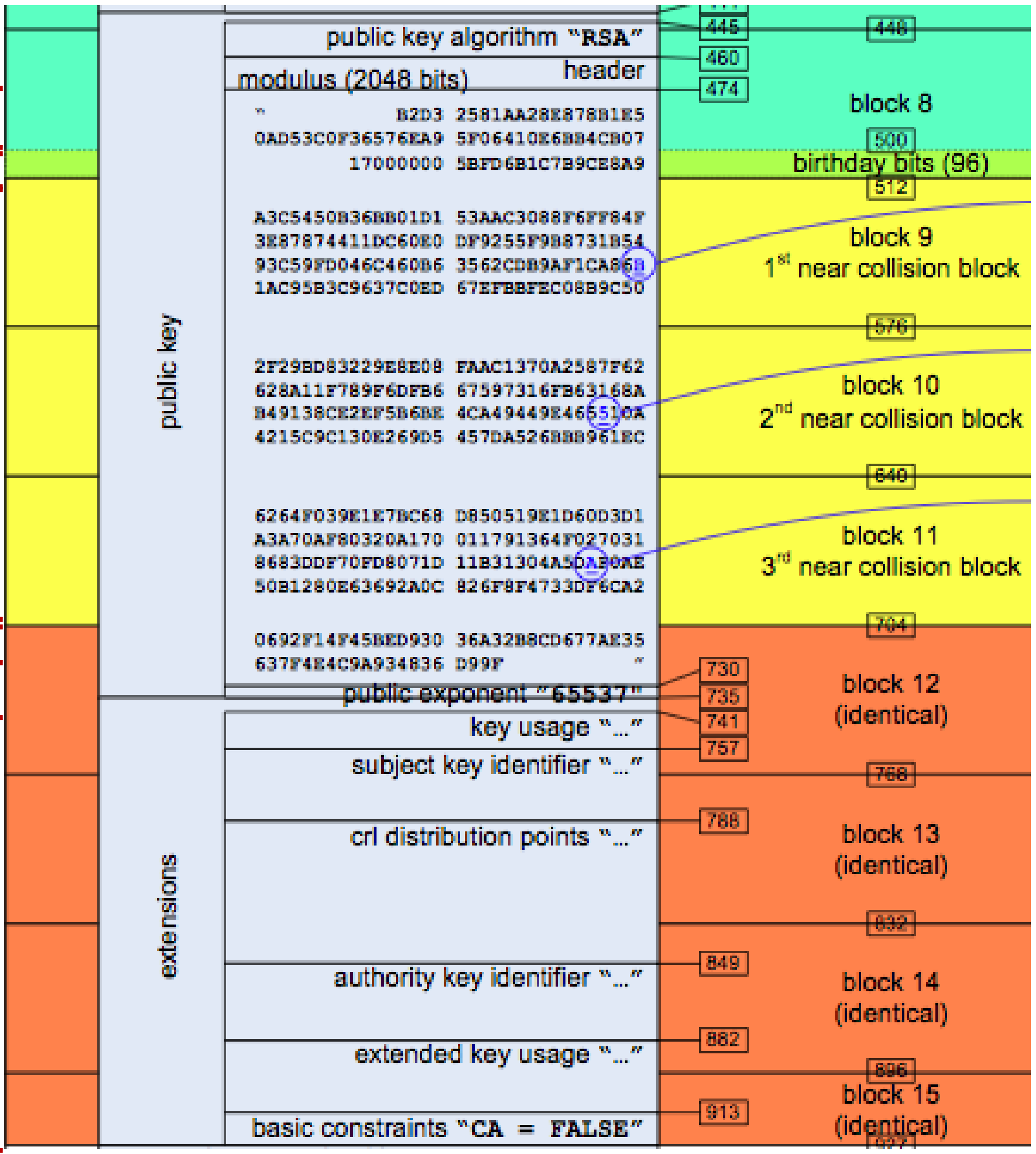
(a) chosen uniformly at random

(b) chosen to set up relationship between this certificate and the rogue CA cert to be generated next

(c) output of the authors' collision-finding algorithm

(d) chosen by the attacker such that when the bits comprising parts (a)-(d) are interpreted as an integer, the result is a number "n" such that n = pq

This is just standard PKI junk…



```
"               B2D3  2581AA28E878B1E5
0AD53C0F36576EA9  5F06410E6BB4CB07
        17000000  5BFD6B1C7B9CE8A9

A3C5450B36BB01D1  53AAC3088F6FF84F
3E87874411DC60E0  DF9255F9B8731B54
93C59FD046C460B6  3562CDB9AF1CA86B
1AC95B3C9637C0ED  67EFBBFEC08B9C50

2F29BD83229E8E08  FAAC1370A2587F62
628A11F789F6DFB6  67597316FB63168A
B49138CE2EF5B6BE  4CA49449E465510A
4215C9C130E269D5  457DA526BBB961EC

6264F039E1E7BC68  D850519E1D60D3D1
A3A70AF80320A170  011791364F027031
8683DDF70FD8071D  11B31304A5DAF0AE
50B1280E63692A0C  826F8F4733DF6CA2

0692F14F45BED930  36A32B8CD677AE35
637F4E4C9A934836  D99F               "
```

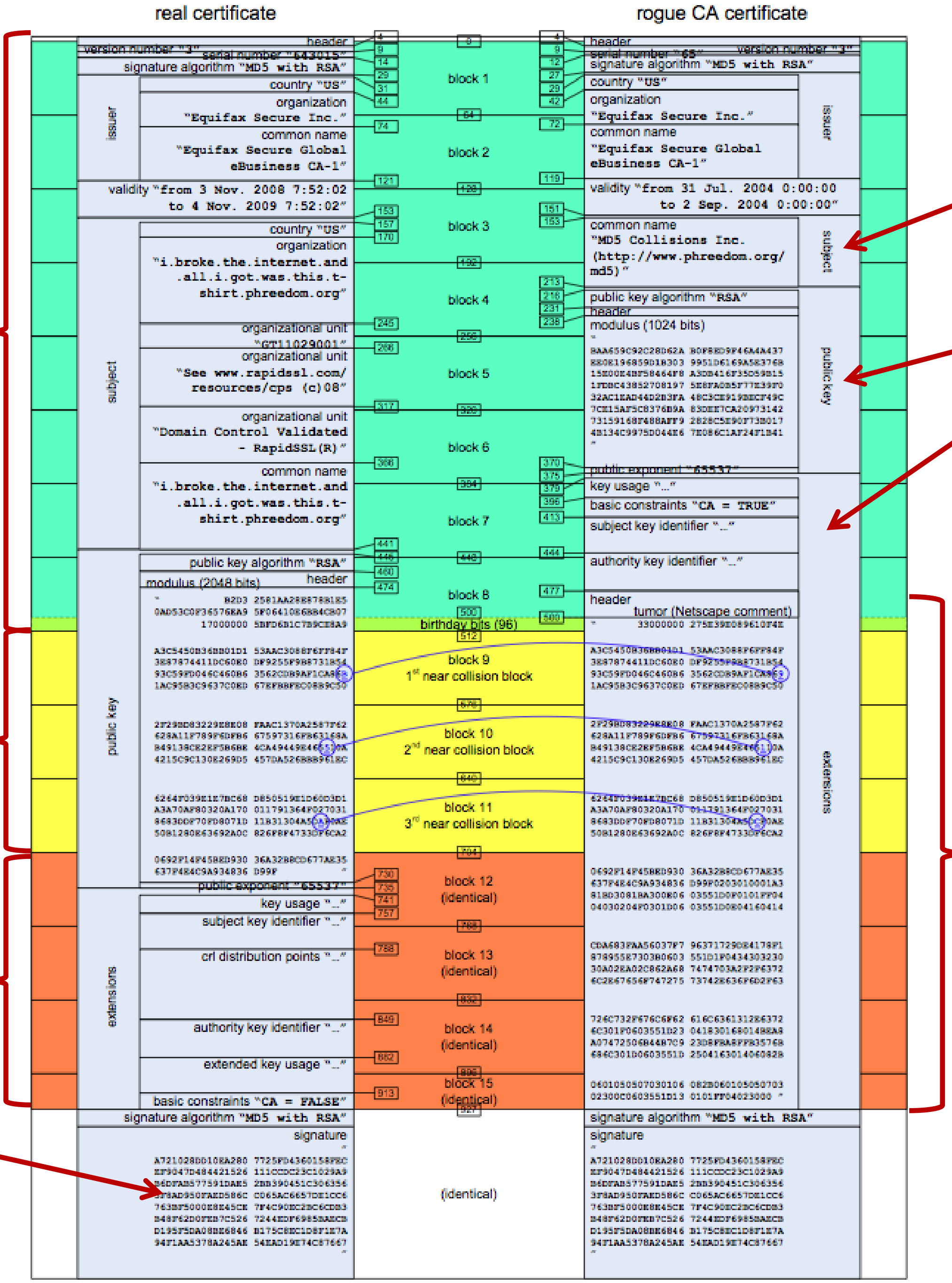
Creating the rogue CA certificate

Chosen prefix

Collision blocks (different but related)

Identical trailers

Signature of CA



real certificate
rogue CA certificate
header
version number "3"
serial number "643015"
signature algorithm "MD5 with RSA"
issuer
country "US"
organization "Equifax Secure Inc."
common name "Equifax Secure Global eBusiness CA-1"
validity "from 3 Nov. 2008 7:52:02 to 4 Nov. 2009 7:52:02"
subject
country "US"
organization "i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom.org"
organizational unit "GT11029001"
organizational unit "See www.rapidssl.com/resources/cps (c)08"
organizational unit "Domain Control Validated - RapidSSL(R)"
common name "i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom.org"
public key
public key algorithm "RSA"
modulus (2048 bits)
public exponent "65537"
extensions
key usage "..."
subject key identifier "..."
crl distribution points "..."
authority key identifier "..."
extended key usage "..."
basic constraints "CA = FALSE"
signature algorithm "MD5 with RSA"
signature
block 1
block 2
block 3
block 4
block 5
block 6
block 7
block 8
birthday bits (96)
block 9
1st near collision block
block 10
2nd near collision block
block 11
3rd near collision block
block 12 (identical)
block 13 (identical)
block 14 (identical)
block 15 (identical)
(identical)
header
serial number "65"
version number "3"
signature algorithm "MD5 with RSA"
country "US"
organization "Equifax Secure Inc."
common name "Equifax Secure Global eBusiness CA-1"
validity "from 31 Jul. 2004 0:00:00 to 2 Sep. 2004 0:00:00"
common name "MD5 Collisions Inc. (http://www.phreedom.org/md5)"
public key algorithm "RSA"
modulus (1024 bits)
public exponent "65537"
key usage "..."
basic constraints "CA = TRUE"
subject key identifier "..."
authority key identifier "..."
tumor (Netscape comment)
signature algorithm "MD5 with RSA"
signature

Shorter subject info

Small public key

Standard public key stuff (Note that this cert is for a CA!)

(Typically ignored) certificate extension to hold collision blocks and trailer from original certificate

Some specifics…

The attack used by the authors had two stages:

1. Finding the 96 “birthday bits”
2. Finding the three near collision blocks

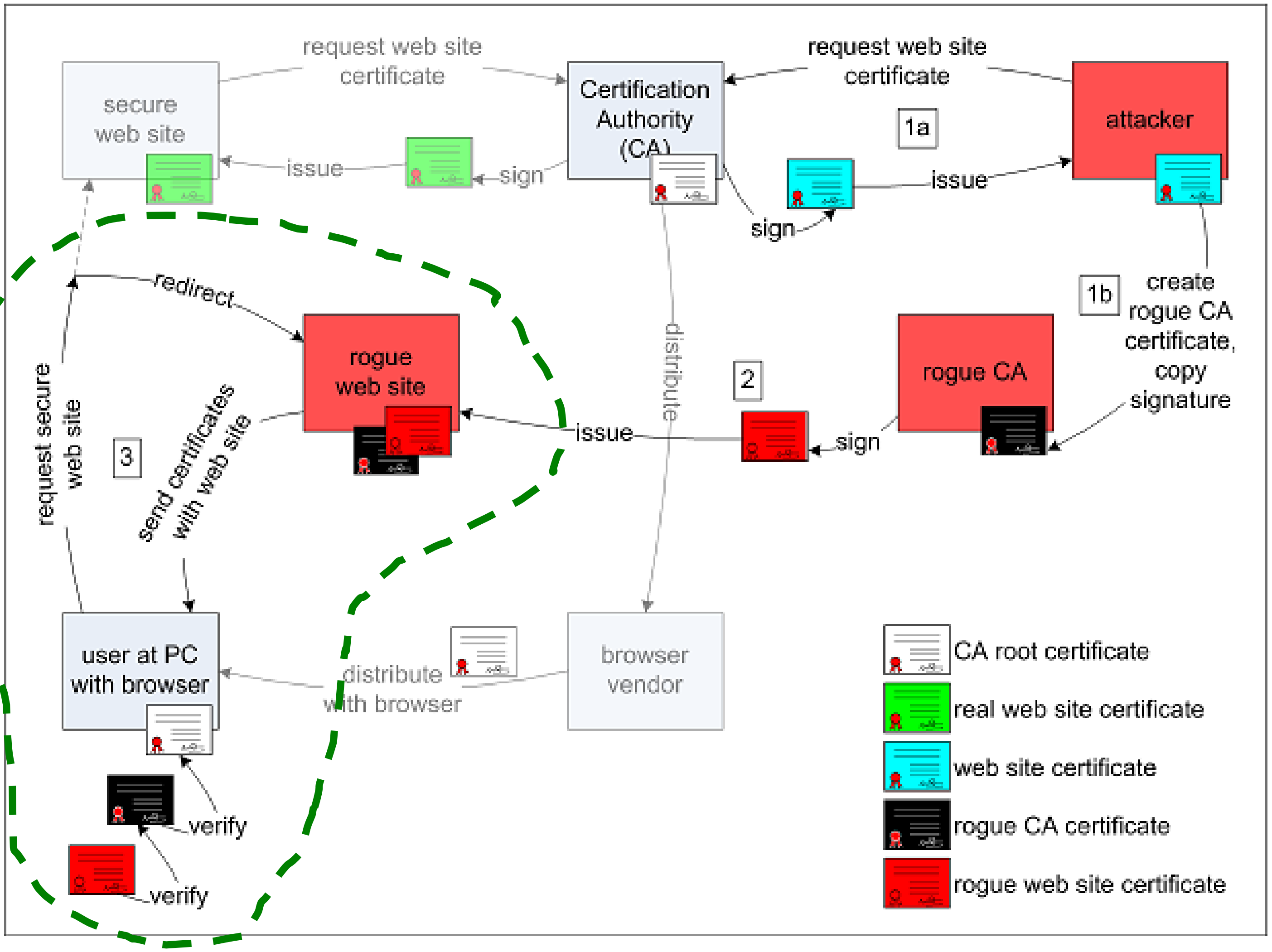
Stage 1 is computationally expensive, but well-suited for execution on the processors used by PlayStation3 game consoles. This took about 18 hours on a cluster of 200 PS3s.

Stage 2 is not terribly expensive, and is better suited to run on commodity PCs. This takes 3-10 hours on a single quad-core PC.

Attacking only on weekends (due to reduced CA load) the authors were able to carry out their attack in 4 weeks for a cost of $657.

This attack has very real implications…

In short, it is possible for a malicious principal to get a cheap X.509 host certificate, and leverage this to create a trusted CA certificate!



The Bottom Line: Consider MD5 broken!

Broken? What do you mean by broken?

Since MD5 does not have all three properties required of cryptographic hash functions, it is not considered to be "cryptographically strong"

While you could still use MD5 for non-cryptographic applications (such as?), it is better to avoid it altogether

---

SHA-1 is the cryptographic hash function that is now most often used

Attacks against SHA-1 have been announced, but still require a large amount of computing effort to mount

- e.g., Collision finding in 2^{63} steps, rather than 2^{80}

NIST recently ran a competition to design SHA-3, a new hash family to complement the SHA-{1,2} family of hash functions

- Welcome, Keccak

How did vendors react to the MD5 break?

December 30th: This work was presented at the Chaos Computing Congress

December 31st

- Verisign issues a statement, stops using MD5
- Microsoft issues Security Advisory (961509): “Research proves feasibility of collision attacks against MD5”
- Mozilla has a short item in the Mozilla Security Blog: “MD5 Weaknesses Could Lead to Certificate Forgery”

January 2nd

- RSA has an entry in the Speaking of Security blog: “A Real New Year’s Hash”
- US-CERT, the US Department of Homeland Security’s Computer Emergency Readiness Team, published Vulnerability Note VU#836068: “MD5 vulnerable to collision attacks”

January 15th

- Cisco published “Cisco Security Response: MD5 Hashes May Allow for Certificate Spoofing”

Discussion

Question 1: People have had evidence that MD5 was weak since the mid 1990s. Why did it take this long to finally convince vendors to stop using MD5? Do you think that this will change the response to future cryptographic vulnerabilities?

Question 2: In the 1980s, Adi Shamir and Eli Biham “discovered” differential cryptanalysis, which is a general means of attacking block ciphers. It was later revealed that NSA and IBM actually discovered this technique first, but kept it a secret. What if this MD5 attack was known before it was discovered in the public domain? What would the implications be?

The aftermath: Flame malware

“[A]rguably, it is the most complex malware ever found”

Developed by NSA and GCHQ for cyber espionage targeting middle eastern countries. Records:

- Screenshots
- Audio
- Network activity
- Keyboard activity
- Nearby devices’ contacts via bluetooth

Used components signed by Microsoft?!

Counterfeit certificate crafted from a terminal service certificate with code-signing enabled, signed using MD5

Previously unknown variant of the chosen-prefix collision attack

First detected in 2012…

… but its main component was first seen in 2007