CSE 484 / CSE M 584: Computer Security and Privacy

Crypto meets Web Security: Certificates and SSL/TLS

Fall 2016

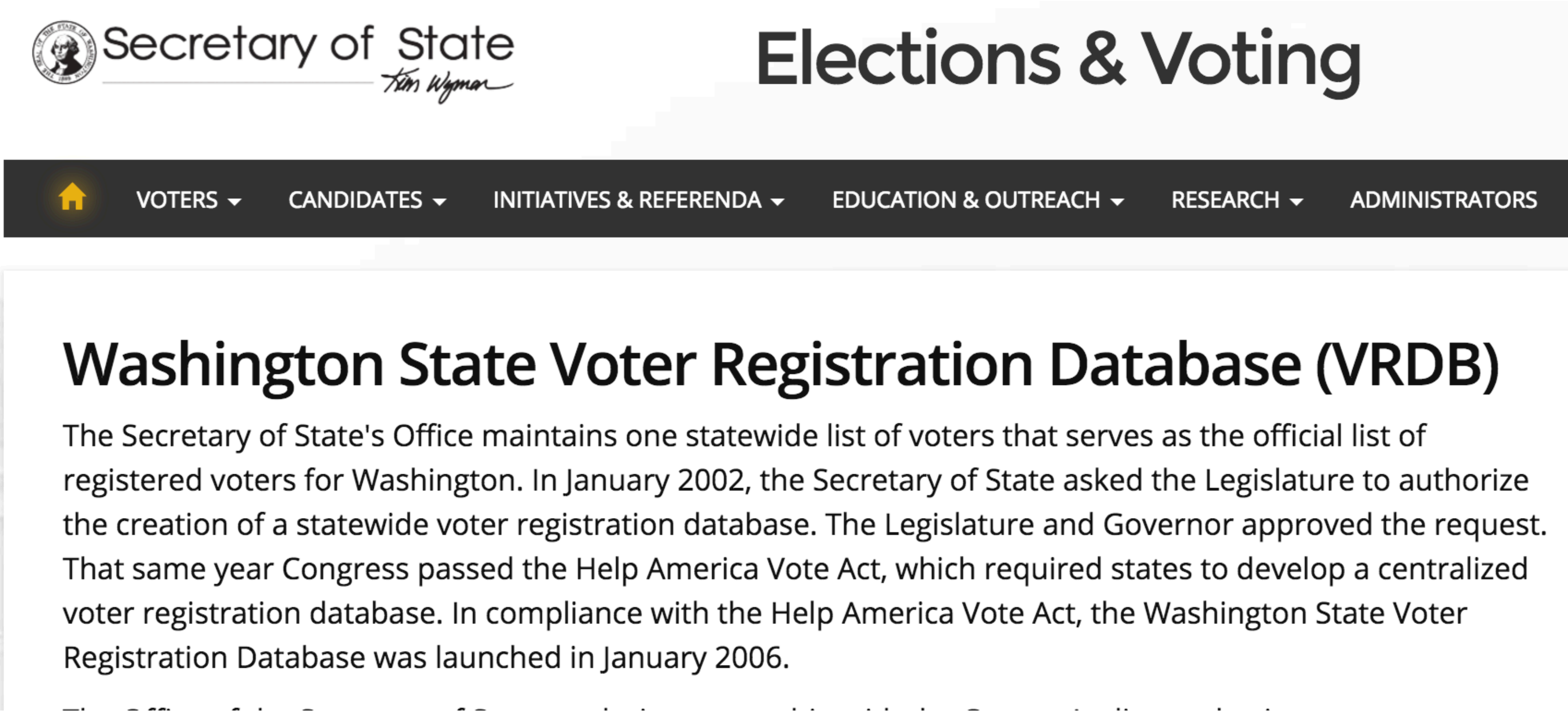
Ada (Adam) Lerner
lerner@cs.washington.edu

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

Security Mindset Anecdote

- Change voting registration information (e.g. change the address your ballot is mailed to)
  - First, last name
  - Birthday
  - Driver’s license number

Security Mindset Anecdote



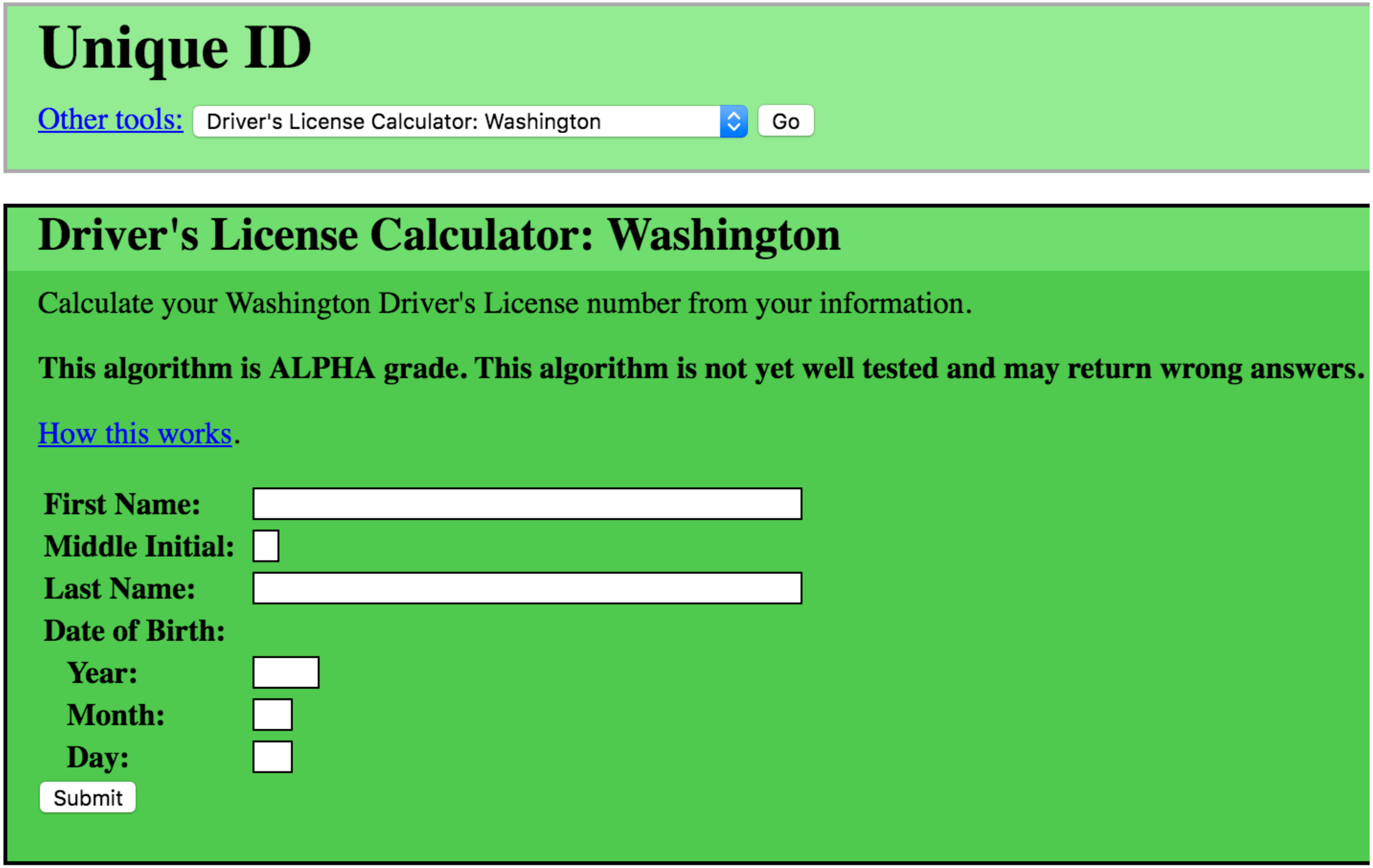
Washington State Voter Registration Database (VRDB)

The Secretary of State's Office maintains one statewide list of voters that serves as the official list of registered voters for Washington. In January 2002, the Secretary of State asked the Legislature to authorize the creation of a statewide voter registration database. The Legislature and Governor approved the request. That same year Congress passed the Help America Vote Act, which required states to develop a centralized voter registration database. In compliance with the Help America Vote Act, the Washington State Voter Registration Database was launched in January 2006.

Security Mindset Anecdote

- Change voting registration information (e.g. change the address your ballot is mailed to)
  - First, last name
  - Birthday
  - Driver's license number

Security Mindset Anecdote



Unique ID

Other tools: Driver's License Calculator: Washington

Driver's License Calculator: Washington

Calculate your Washington Driver's License number from your information.

This algorithm is ALPHA grade. This algorithm is not yet well tested and may return wrong answers.

How this works.

First Name:

Middle Initial:

Last Name:

Date of Birth:

Year:

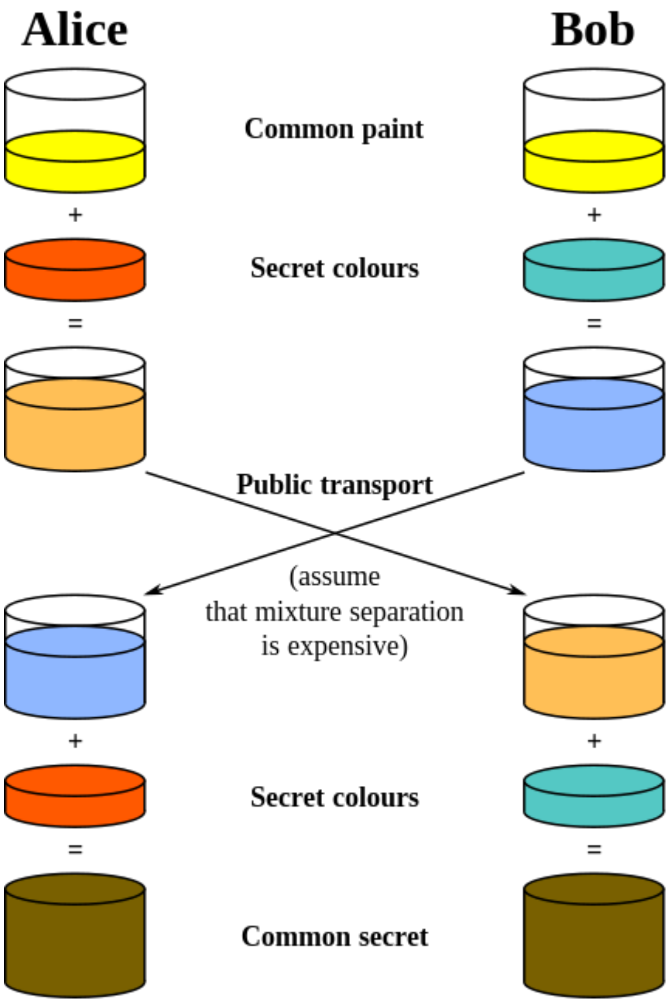
Month:

Day:

Security Mindset Anecdote

- Change voting registration information (e.g. change the address your ballot is mailed to)
  - First, last name
  - Birthday
  - Driver's license number
  - Driver's license issue date (added recently)

Diffie-Hellman: Conceptually



Common paint: p and g

Secret colors: x and y

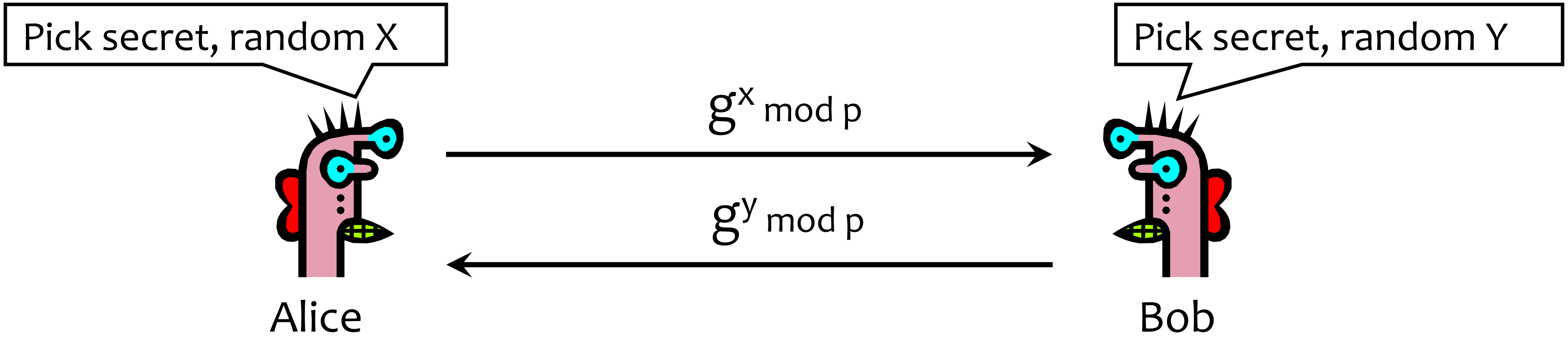
Send over public transport:
g^x mod p
g^y mod p

Common secret: g^{xy} mod p

[from Wikipedia]

Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: p and g
  - – p is a large prime number, g is a generator of Z_p^*
    - $Z_p^* = \{1, 2 \ldots p-1\}$; $\forall a \in Z_p^*$ $\exists i$ such that $a = g^i \bmod p$
    - Modular arithmetic: numbers "wrap around" after they reach p



Compute $k = (g^y)^x = g^{xy} \bmod p$

Compute $k = (g^x)^y = g^{xy} \bmod p$

Why is Diffie-Hellman Secure?

- Discrete Logarithm (DL) problem:
  given g^x *mod p*, it's hard to extract x
  - There is no known efficient algorithm for doing this
  - This is not enough for Diffie-Hellman to be secure!
- Computational Diffie-Hellman (CDH) problem:
  given g^x and g^y, it's hard to compute g^{xy} *mod p*
  - … unless you know x or y, in which case it's easy
- Decisional Diffie-Hellman (DDH) problem:

  given g^x and g^y, it's hard to tell the difference between g^{xy} *mod p* and g^r *mod p* where r is random

Properties of Diffie-Hellman

- Assuming DDH problem is hard (depends on choice of parameters!), Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  - Eavesdropper can't tell the difference between the established key and a random value
  - Can use the new key for symmetric cryptography
- Diffie-Hellman protocol (by itself) does not provide authentication

Choosing p

- In practice, we choose very large primes of the form

$$q = 2p + 1$$

(where p is prime)

RFC 3526

Smallest prime (1536-bit) standardized for DH is:

2^1536 - 2^1472 - 1 + 2^64 * { [2^1406 pi] + 741804 }

Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA237327 FFFFFFFF FFFFFFFF
```

Generator:

RFC 3526

Smallest prime (1536-bit) standardized for DH is:

2^1536 - 2^1472 - 1 + 2^64 * { [2^1406 pi] + 741804 }

Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA237327 FFFFFFFF FFFFFFFF
```

Generator: 2

RFC 3526

- Biggest prime given by RFC 3526 is 8192-bit

Some Number Theory Facts

- Euler totient function $\varphi(n)$ ($n \geq 1$) is the number of integers in the $[1,n]$ interval that are relatively prime to n
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
  - Easy to compute for primes: $\varphi(p) = p-1$
  - Note that if a and b are relatively prime, then $\varphi(ab) = \varphi(a)\,\varphi(b)$

Some Number Theory Facts

- Euler totient function $\varphi(n)$ ($n \geq 1$) is the number of integers in the [1,n] interval that are relatively prime to n
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
  - Easy to compute for primes: $\varphi(p) = p-1$
  - Note that if a and b are relatively prime, then $\varphi(ab) = \varphi(a)\ \varphi(b)$

- Euler's theorem: if $a \in Z_n^*$, then $a^{\varphi(n)} = 1 \bmod n$

  Z_n^*: integers relatively prime to n

RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate random large primes p, q
    - Say, 1024 bits each
  - Compute **n**=pq and **φ(n)**=(p-1)(q-1)
  - Choose small e, relatively prime to φ(n)
    - Typically, $e=2^{16}+1=65537$
  - Compute unique d such that ed = 1 mod φ(n)
    - Modular inverse: $d = e^{-1} \bmod \varphi(n)$
  - Public key = (e,n); private key = (d,n)
- Encryption of m: $c = m^e \bmod n$
- Decryption of c: $c^d \bmod n = (m^e)^d \bmod n = m$

Why RSA Decryption Works

$e \cdot d = 1 \bmod \varphi(n)$, thus $e \cdot d = 1 + k \cdot \varphi(n)$ for some k

Let m be any integer in Z_n^* (not all of Z_n)

$c^d \bmod n = (\mathbf{m}^e)^d \bmod n = \mathbf{m}^{1+k \cdot \varphi(n)} \bmod n$

$= (\mathbf{m} \bmod n) * (\mathbf{m}^{k \cdot \varphi(n)} \bmod n)$

Recall: Euler's theorem: if $a \in Z_n^*$, then $a^{\varphi(n)} = 1 \bmod n$

$c^d \bmod n = (\mathbf{m} \bmod n) * (\mathbf{1} \bmod n)$

$= \mathbf{m} \bmod n$

Proof omitted: True for all m in Z_n, not just m in Z_n^*

Why is RSA Secure?

- RSA problem: given c, n=pq, and e such that gcd(e, φ(n))=1, find m such that $m^e=c \bmod n$
  - In other words, recover m from ciphertext c and public key (n,e) by taking e^th root of c modulo n
  - There is no known efficient algorithm for doing this
- Factoring problem: given positive integer n, find primes $p_1, \dots, p_k$ such that $n=p_1^{e_1}p_2^{e_2}\dots p_k^{e_k}$
- If factoring is easy, then RSA problem is easy (knowing factors means you can compute d = inverse of e mod (p-1)(q-1))
  - It may be possible to break RSA without factoring n -- but if it is, we don't know how

RSA Encryption Caveats

- Encrypted message needs to be interpreted as an integer less than n
- Don't use RSA **directly** for privacy – output is deterministic! Need to pre-process input somehow
- Plain RSA also does not provide integrity
  - Can tamper with encrypted messages

Optimal Asymmetric Encryption Padding

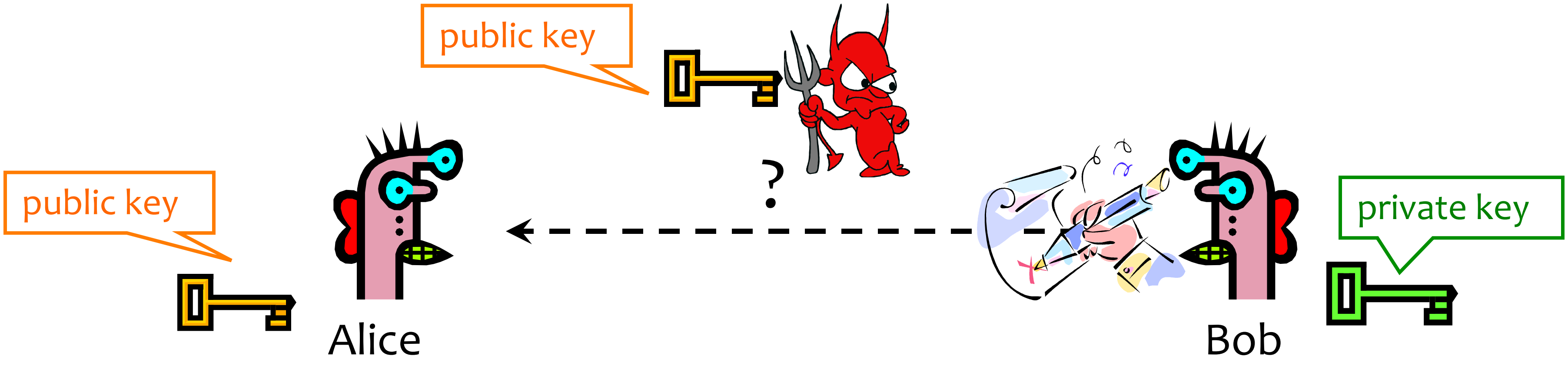
- Don’t use RSA **directly** for privacy – output is deterministic! Need to pre-process input somehow
- OAEP changes the plaintext randomly, creating a scheme which is secure under chosen plaintext attacks

OAEP: instead of encrypting M, encrypt

$M \oplus G(r) ; r \oplus H(M \oplus G(r))$

- r is random and fresh, G and H are hash functions

Digital Signatures: Basic Idea



Given: Everybody knows Bob's public key
Only Bob knows the corresponding private key

Goal: Bob sends a "digitally signed" message

1. To compute a signature, must know the private key
2. To verify a signature, only the public key is needed

RSA Signatures

- Public key is **(n,e)**, private key is **(n,d)**
- To sign message m: s = m^d mod n
  - Signing & decryption are same **underlying** operation in RSA
  - It's infeasible to compute **s** on **m** if you don't know **d**
- To verify signature s on message m:

  verify that s^e mod n = $(m^d)^e$ mod n = m
  - Just like encryption (for RSA primitive)
  - Anyone who knows **n** and **e** (public key) can verify signatures produced with d (private key)
- In practice, also need padding & hashing
  - Standard padding/hashing schemes exist for RSA signatures

DSS Signatures

- Digital Signature Standard (DSS)
  - U.S. government standard (1991, most recent rev. 2013)
- Public key: (p, q, g, $y=g^x$ mod p), private key: x
- Security of DSS requires hardness of discrete log
  - If could solve discrete logarithm problem, would extract x (private key) from g^x mod p (public key)

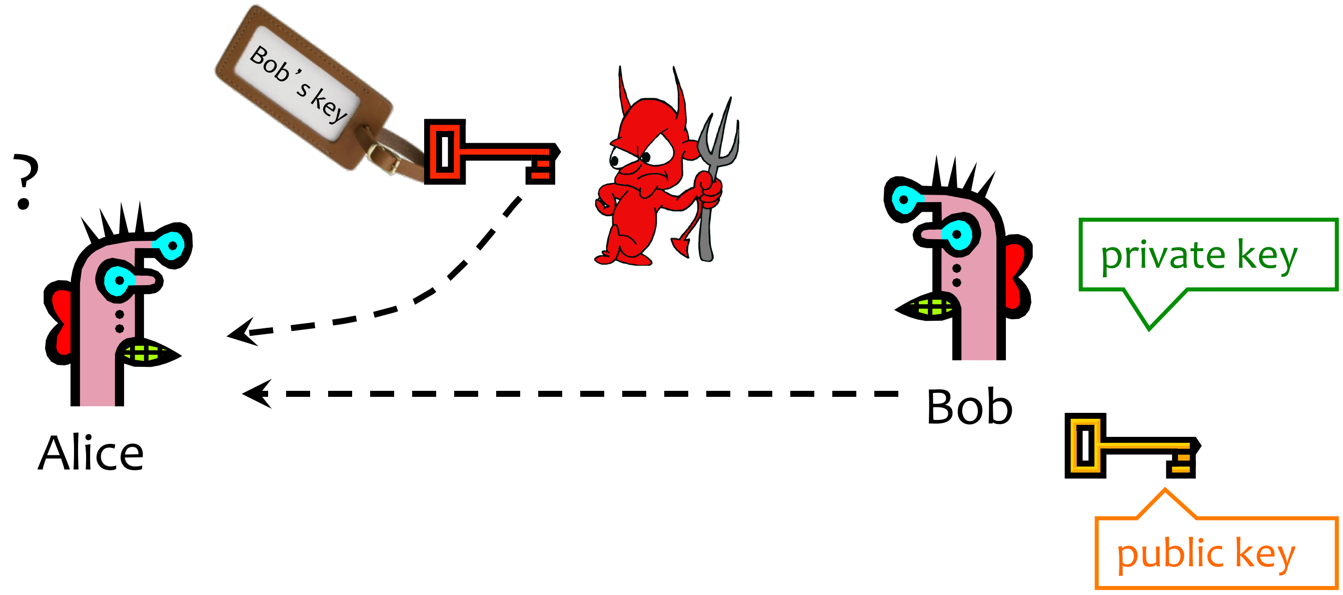
Advantages of Public Key Crypto

- Confidentiality without shared secrets
  - Very useful in open environments
  - Can use this for key establishment, with fewer "chicken-or-egg" problems
    - With symmetric crypto, two parties must share a secret before they can exchange secret messages
- Authentication without shared secrets
  - Use digital signatures to prove the origin of messages
- Encryption keys are public, but must be sure that Alice's public key is really *her* public key
  - This is a hard problem…

Disadvantages of Public Key Crypto

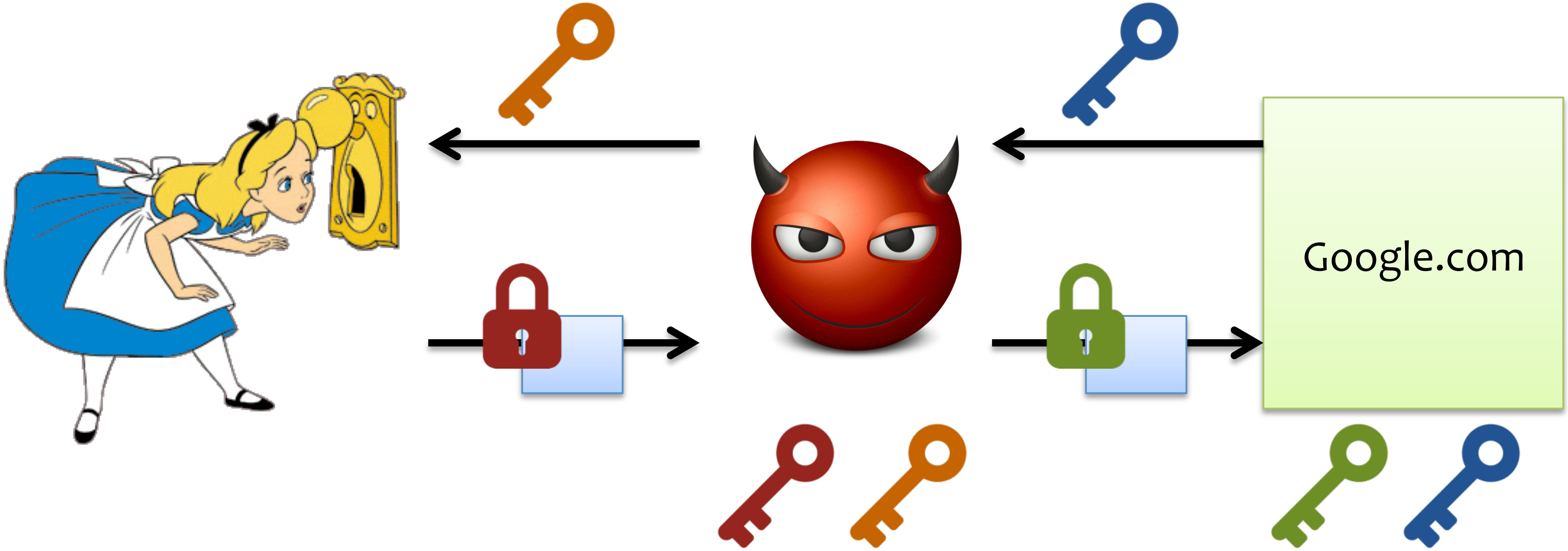
- Calculations are 2-3 orders of magnitude slower
  - Modular exponentiation is an expensive computation
  - Typical usage: use public-key cryptography to establish a shared secret, then switch to symmetric crypto
    - E.g., IPsec, SSL, SSH, …
- Keys are longer
  - 4096+ bits (RSA) rather than 128 bits (AES)
- Relies on unproven number-theoretic assumptions
  - What if factoring is easy?
    - Factoring is *believed* to be neither P, nor NP-complete
  - (Of course, symmetric crypto also rests on unproven assumptions…)

Authenticity of Public Keys



Problem: How does Alice know that the public key she received is really Bob's public key?

Threat: Man-In-The-Middle (MITM)



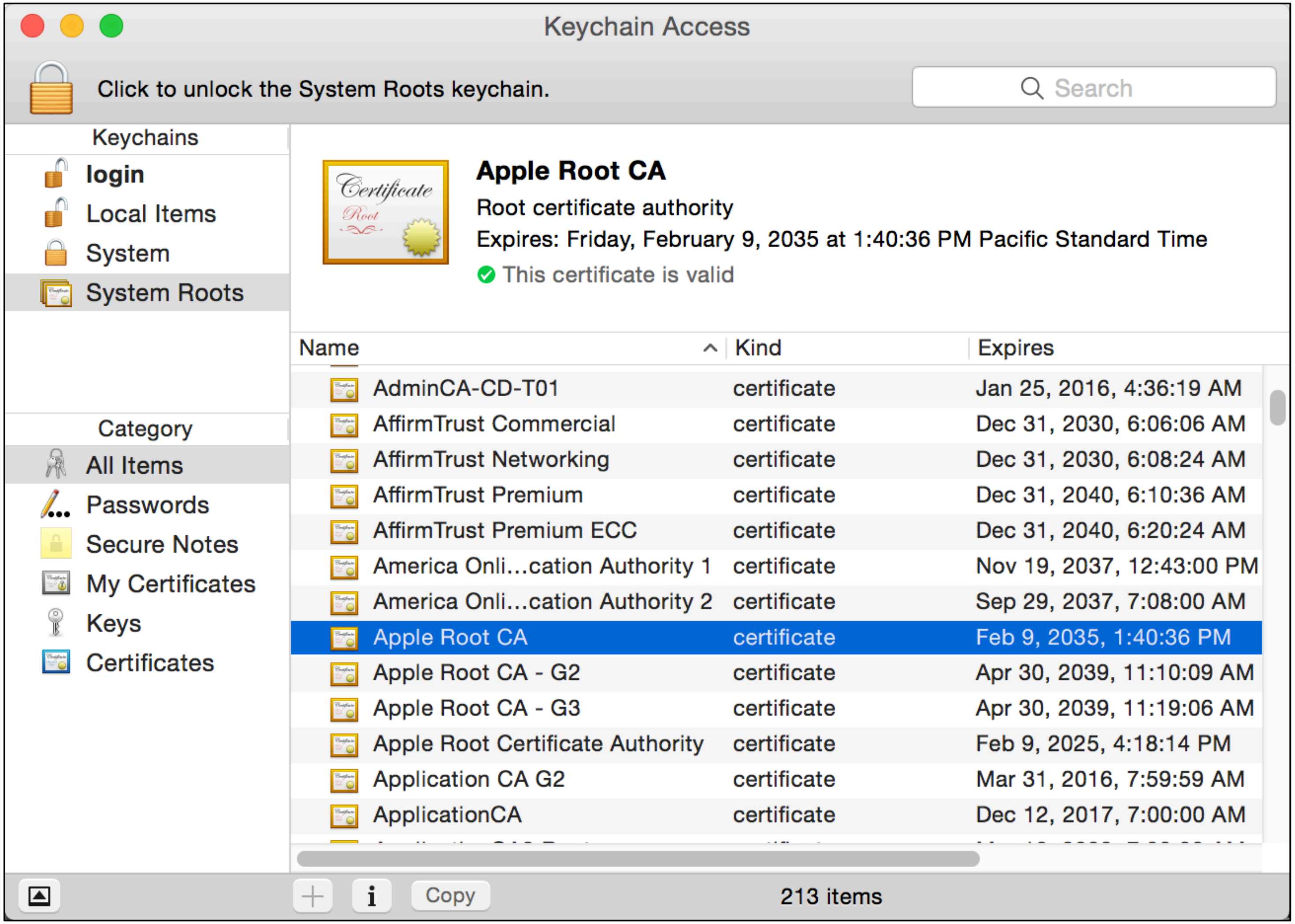
Certificates

- Public-key certificate
  - Signed statement specifying the key and identity
    - sig_{CA}(“Bob”, PK_B)

Distribution of Public Keys

- Public-key certificate
  - Signed statement specifying the key and identity
    - sig_{CA}(“Bob”, PK_B)
- Common approach: certificate authority (CA)
  - Single agency responsible for certifying public keys
  - After generating a private/public key pair, user proves his identity and knowledge of the private key to obtain CA’s certificate for the public key (offline)
  - Every computer is pre-configured with CA’s public key

Trusted Certificate Authorities



Apple Root CA

Root certificate authority

Expires: Friday, February 9, 2035 at 1:40:36 PM Pacific Standard Time

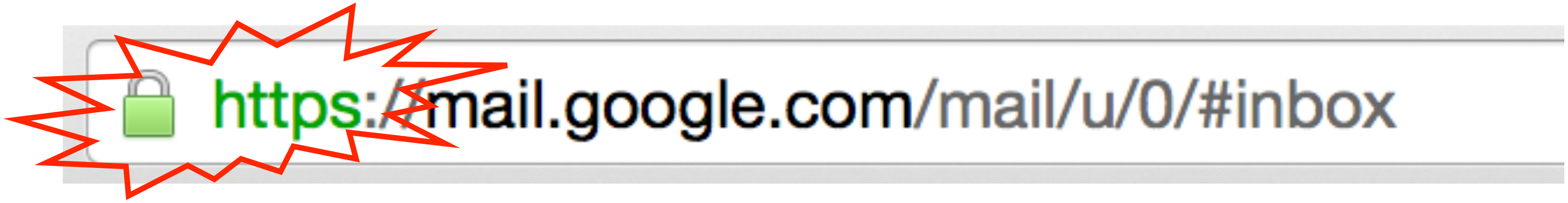
✓ This certificate is valid

Name	Kind	Expires
AdminCA-CD-T01	certificate	Jan 25, 2016, 4:36:19 AM
AffirmTrust Commercial	certificate	Dec 31, 2030, 6:06:06 AM
AffirmTrust Networking	certificate	Dec 31, 2030, 6:08:24 AM
AffirmTrust Premium	certificate	Dec 31, 2040, 6:10:36 AM
AffirmTrust Premium ECC	certificate	Dec 31, 2040, 6:20:24 AM
America Onli…cation Authority 1	certificate	Nov 19, 2037, 12:43:00 PM
America Onli…cation Authority 2	certificate	Sep 29, 2037, 7:08:00 AM
Apple Root CA	certificate	Feb 9, 2035, 1:40:36 PM
Apple Root CA - G2	certificate	Apr 30, 2039, 11:10:09 AM
Apple Root CA - G3	certificate	Apr 30, 2039, 11:19:06 AM
Apple Root Certificate Authority	certificate	Feb 9, 2025, 4:18:14 PM
Application CA G2	certificate	Mar 31, 2016, 7:59:59 AM
ApplicationCA	certificate	Dec 12, 2017, 7:00:00 AM

Hierarchical Approach

- Single CA certifying every public key is impractical
- Instead, use a trusted root authority
  - For example, Verisign
  - Everybody must know the public key for verifying root authority's signatures
- Root authority signs certificates for lower-level authorities, lower-level authorities sign certificates for individual networks, and so on
  - Instead of a single certificate, use a certificate chain
    - $\text{sig}_{\text{Verisign}}$(“AnotherCA”, $\text{PK}_{\text{AnotherCA}}$), $\text{sig}_{\text{AnotherCA}}$(“Alice”, PK_{A})
  - What happens if root authority is ever compromised?

You encounter this every day…

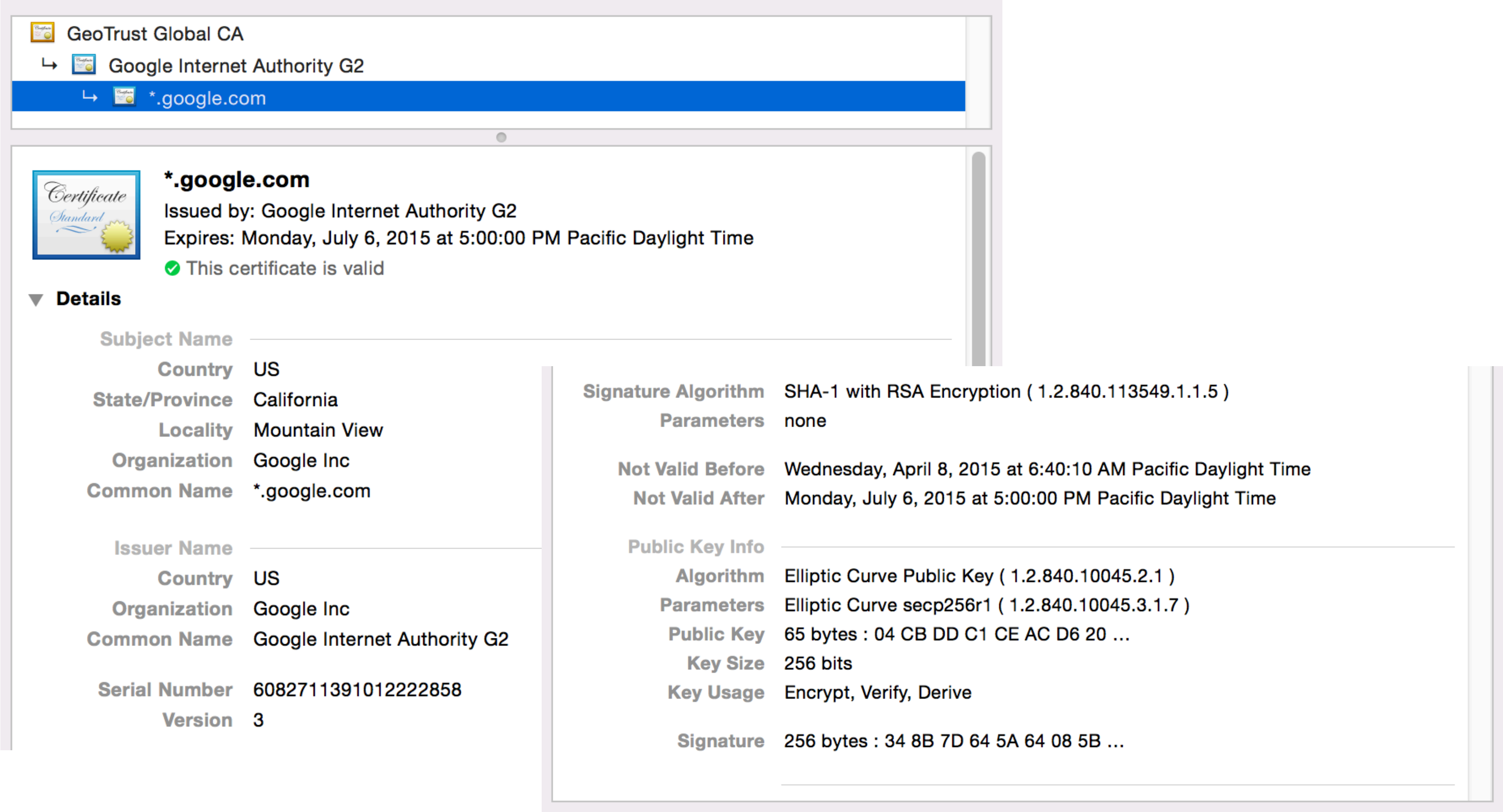


https://mail.google.com/mail/u/0/#inbox

SSL/TLS: Encryption & authentication for connections

(More on this later!)

Example of a Certificate



GeoTrust Global CA
↳ Google Internet Authority G2
↳ *.google.com

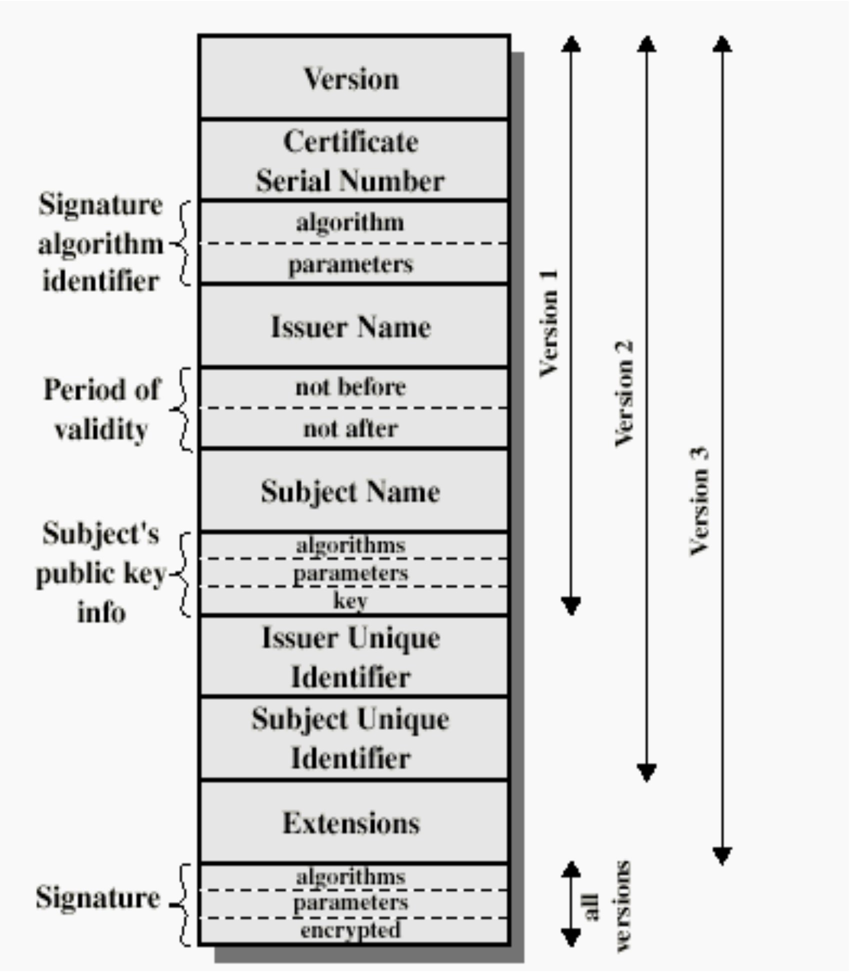
***.google.com**
Issued by: Google Internet Authority G2
Expires: Monday, July 6, 2015 at 5:00:00 PM Pacific Daylight Time
✓ This certificate is valid

▼ **Details**

Subject Name	
Country	US
State/Province	California
Locality	Mountain View
Organization	Google Inc
Common Name	*.google.com
Issuer Name	
Country	US
Organization	Google Inc
Common Name	Google Internet Authority G2
Serial Number	6082711391012222858
Version	3

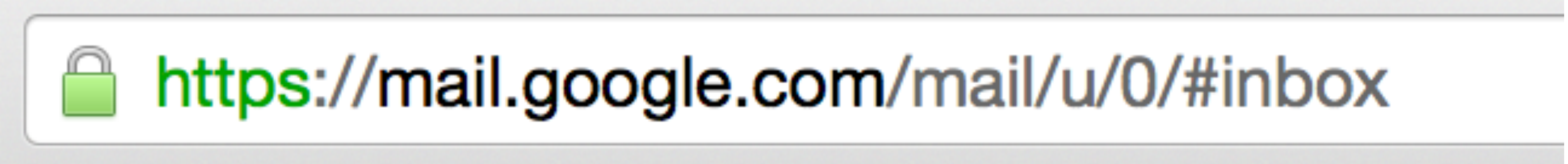
Signature Algorithm	SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)
Parameters	none
Not Valid Before	Wednesday, April 8, 2015 at 6:40:10 AM Pacific Daylight Time
Not Valid After	Monday, July 6, 2015 at 5:00:00 PM Pacific Daylight Time
Public Key Info	
Algorithm	Elliptic Curve Public Key (1.2.840.10045.2.1)
Parameters	Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)
Public Key	65 bytes : 04 CB DD C1 CE AC D6 20 …
Key Size	256 bits
Key Usage	Encrypt, Verify, Derive
Signature	256 bytes : 34 8B 7D 64 5A 64 08 5B …

X.509 Certificate



Many Challenges…
[more examples in section]

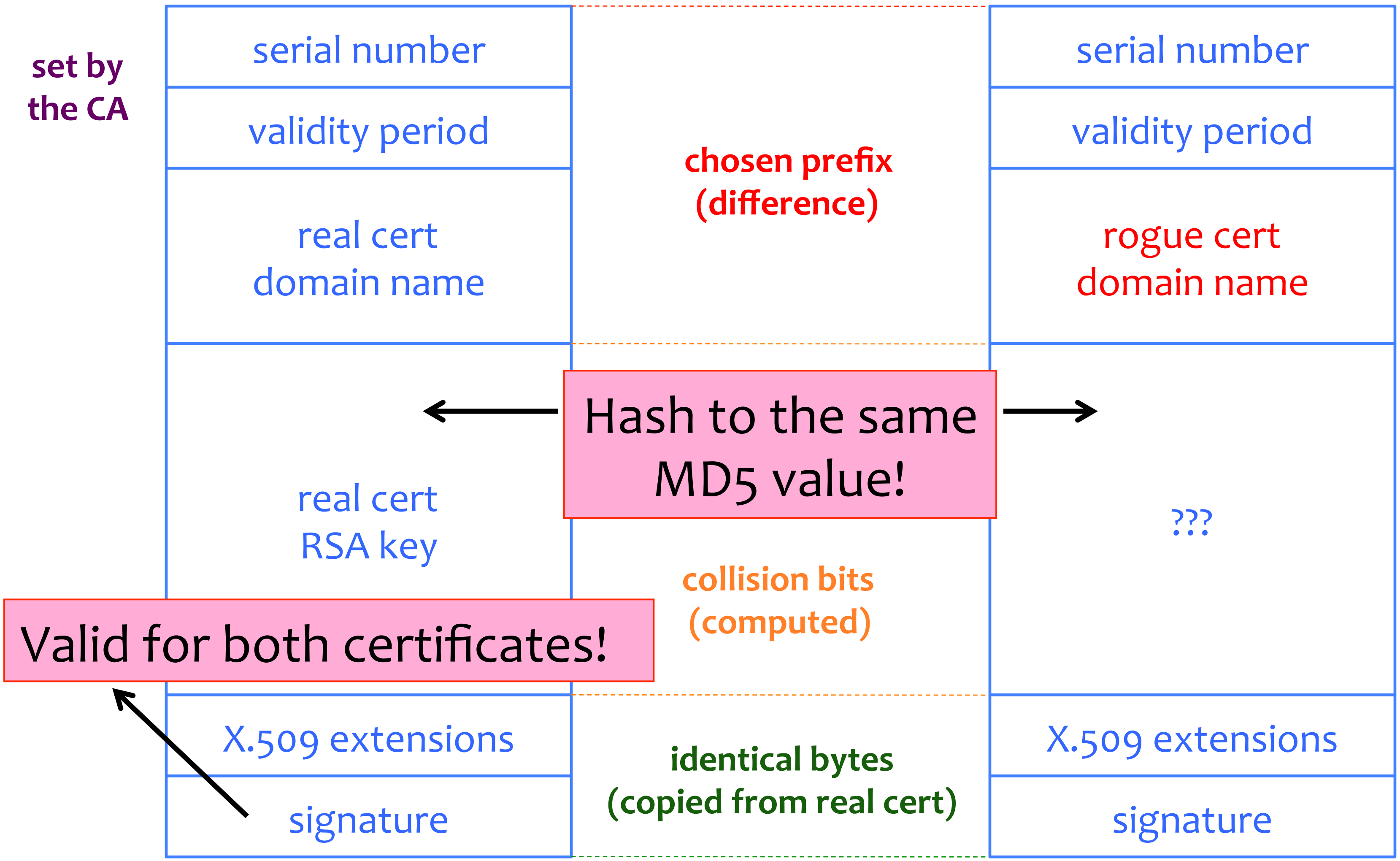
- Hash collisions
- Weak security at CAs
  - Allows attackers to issue rogue certificates
- Users don’t notice when attacks happen
  - We’ll talk more about this later
- Etc…



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

[Sotirov et al. “Rogue Certificates”]

Colliding Certificates



set by the CA

serial number

validity period

real cert domain name

chosen prefix (difference)

serial number

validity period

rogue cert domain name

Hash to the same MD5 value!

real cert RSA key

collision bits (computed)

???

Valid for both certificates!

X.509 extensions

signature

identical bytes (copied from real cert)

X.509 extensions

signature

Attacking CAs

DigiNotar Hacked by Black.Spook and Iranian Hackers

Posted by Mikko @ 09:05 GMT | Comments

DigiNotar is a Dutch Certificate Authority. They sell SSL certificates.

DigiNotar B.V. (0034104947) [NL] https://www.diginotar.nl

DigiNotar®, Internet Tru

Dé onafhankelijke partij voor

Somehow, somebody managed to get a rogue SSL certificate from them on **July 10th, 2011**. This certificate was issued for domain name **.google.com**.

What can you do with such a certificate? Well, you can impersonate Google — assuming you can first reroute Internet traffic for google.com to you. This is something that can be done by a government or by a rogue ISP. Such a reroute would only affect users within that country or under that ISP.

Security of DigiNotar servers:

- All core certificate servers controlled by a single admin password (Prod@dm1n)
- Software on public-facing servers out of date, unpatched
- No anti-virus (could have detected attack)

Consequences

- Attacker needs to first divert users to an attacker-controlled site instead of Google, Yahoo, Skype, but then…
  - For example, use DNS to poison the mapping of mail.yahoo.com to an IP address
- … "authenticate" as the real site
- … decrypt all data sent by users
  - Email, phone conversations, Web browsing

More Rogue Certs

- In Jan 2013, a rogue *.google.com certificate was issued by an intermediate CA that gained its authority from the Turkish root CA TurkTrust
  - TurkTrust accidentally issued intermediate CA certs to customers who requested regular certificates
  - Ankara transit authority used its certificate to issue a fake *.google.com certificate in order to filter SSL traffic from its network
- This rogue *.google.com certificate was trusted by every browser in the world

Certificate Revocation

- Revocation is very important
- Many valid reasons to revoke a certificate
  - Private key corresponding to the certified public key has been compromised
  - User stopped paying his certification fee to this CA and CA no longer wishes to certify him
  - CA's private key has been compromised!
- Expiration is a form of revocation, too
  - Many deployed systems don't bother with revocation
  - Re-issuance of certificates is a big revenue source for certificate authorities

Certificate Revocation Mechanisms

- Certificate revocation list (CRL)
  - CA periodically issues a signed list of revoked certificates
    - Credit card companies used to issue thick books of canceled credit card numbers
  - Can issue a "delta CRL" containing only updates
- Online revocation service
  - When a certificate is presented, recipient goes to a special online service to verify whether it is still valid
    - Like a merchant dialing up the credit card processor

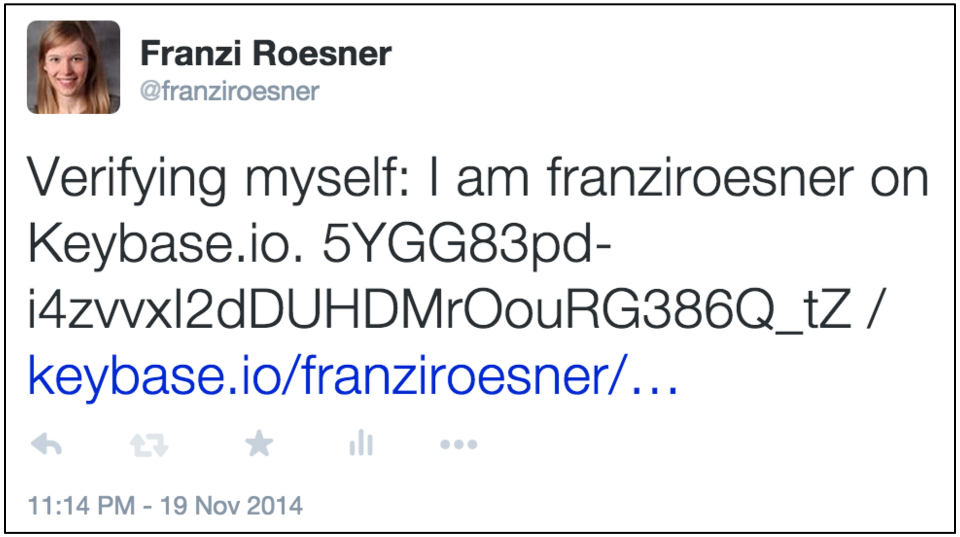
Attempt to Fix CA Problems: Convergence

- Background observation:
  - Attacker will have a hard time mounting man-in-the-middle attacks against **all** clients around the world
- Basic idea:
  - Lots of nodes around the world obtaining SSL/TLS certificates from servers
  - Check responses across servers, and also observe unexpected changes from existing certificates

http://convergence.io/

Keybase

- Basic idea:
  - Rely on existing trust of a person's ownership of other accounts (e.g., Twitter, GitHub, website)
  - Each user publishes signed proofs to their linked account



Franzi Roesner
@franziroesner

Verifying myself: I am franziroesner on Keybase.io. 5YGG83pd-i4zvvxl2dDUHDMrOouRG386Q_tZ / keybase.io/franziroesner/…

11:14 PM - 19 Nov 2014

https://keybase.io/

SSL/TLS

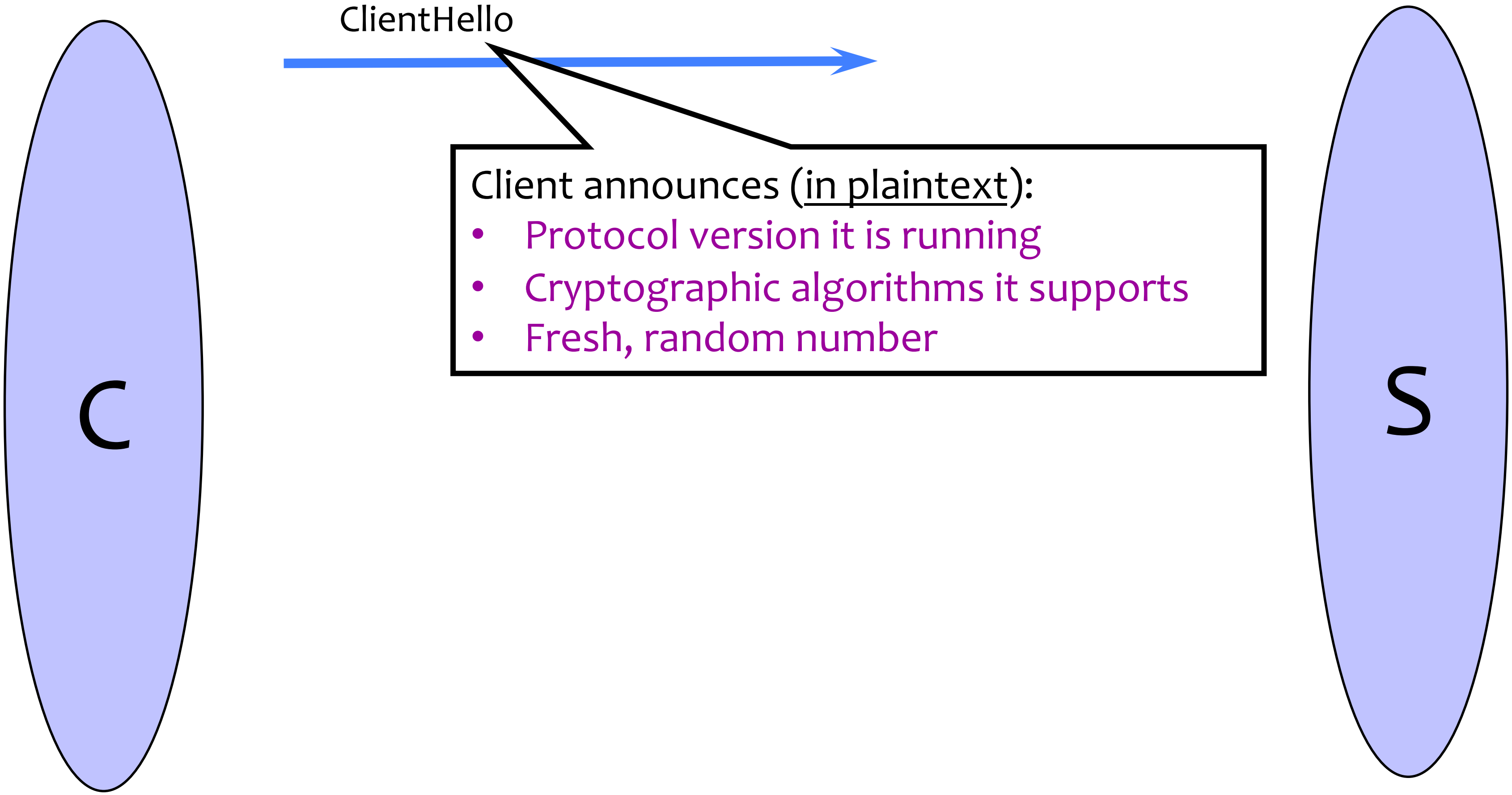
https://mail.google.com/mail/u/0/#inbox

- Secure Sockets Layer and Transport Layer Security protocols
  - Same protocol design, different crypto algorithms
- De facto standard for Internet security
  - "The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications"
- Deployed in every Web browser; also VoIP, payment systems, distributed systems, etc.

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 symmetric key established in the handshake protocol to protect communication between the client and the server

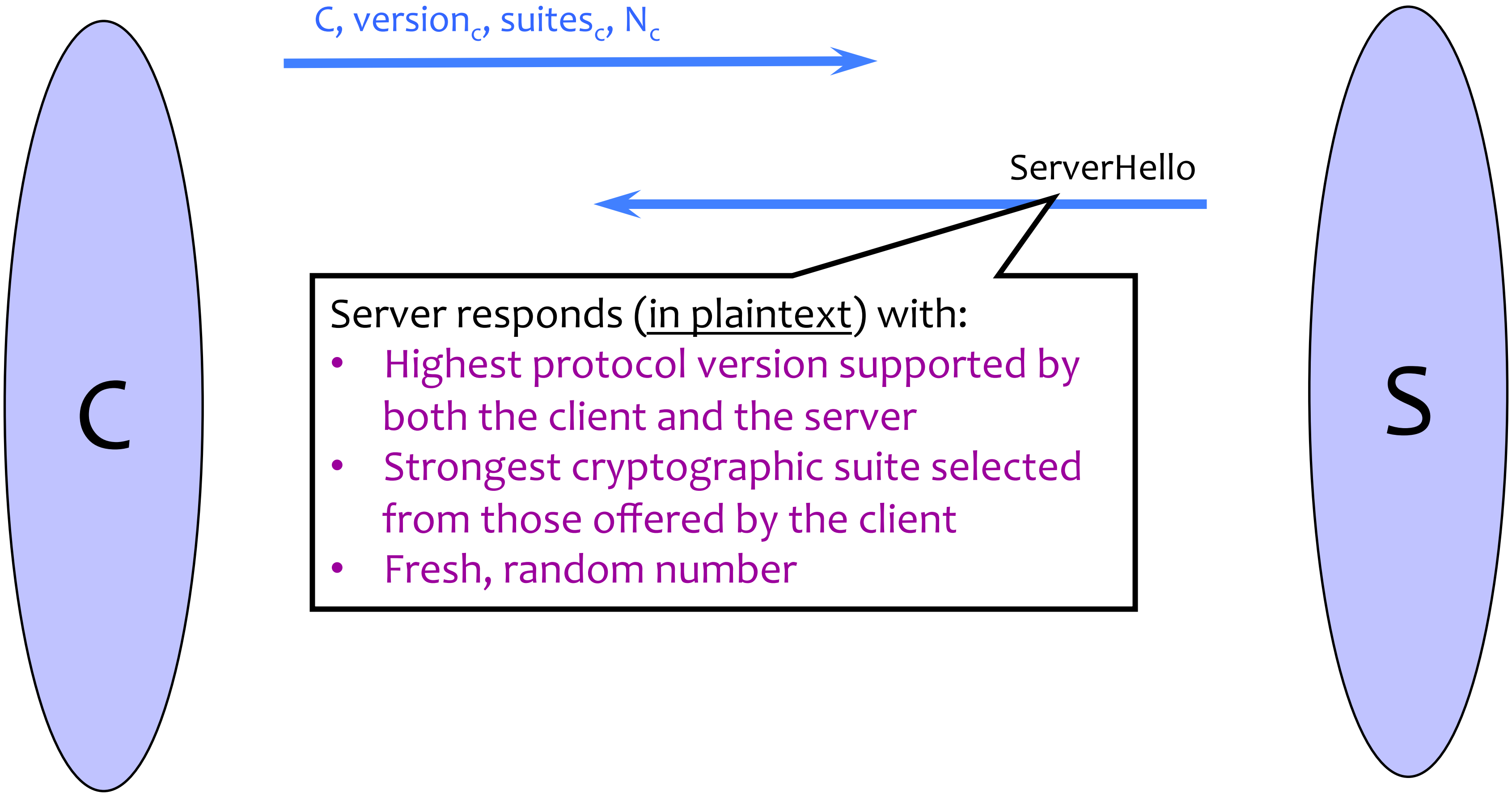
Basic Handshake Protocol



Client announces (in plaintext):

- Protocol version it is running
- Cryptographic algorithms it supports
- Fresh, random number

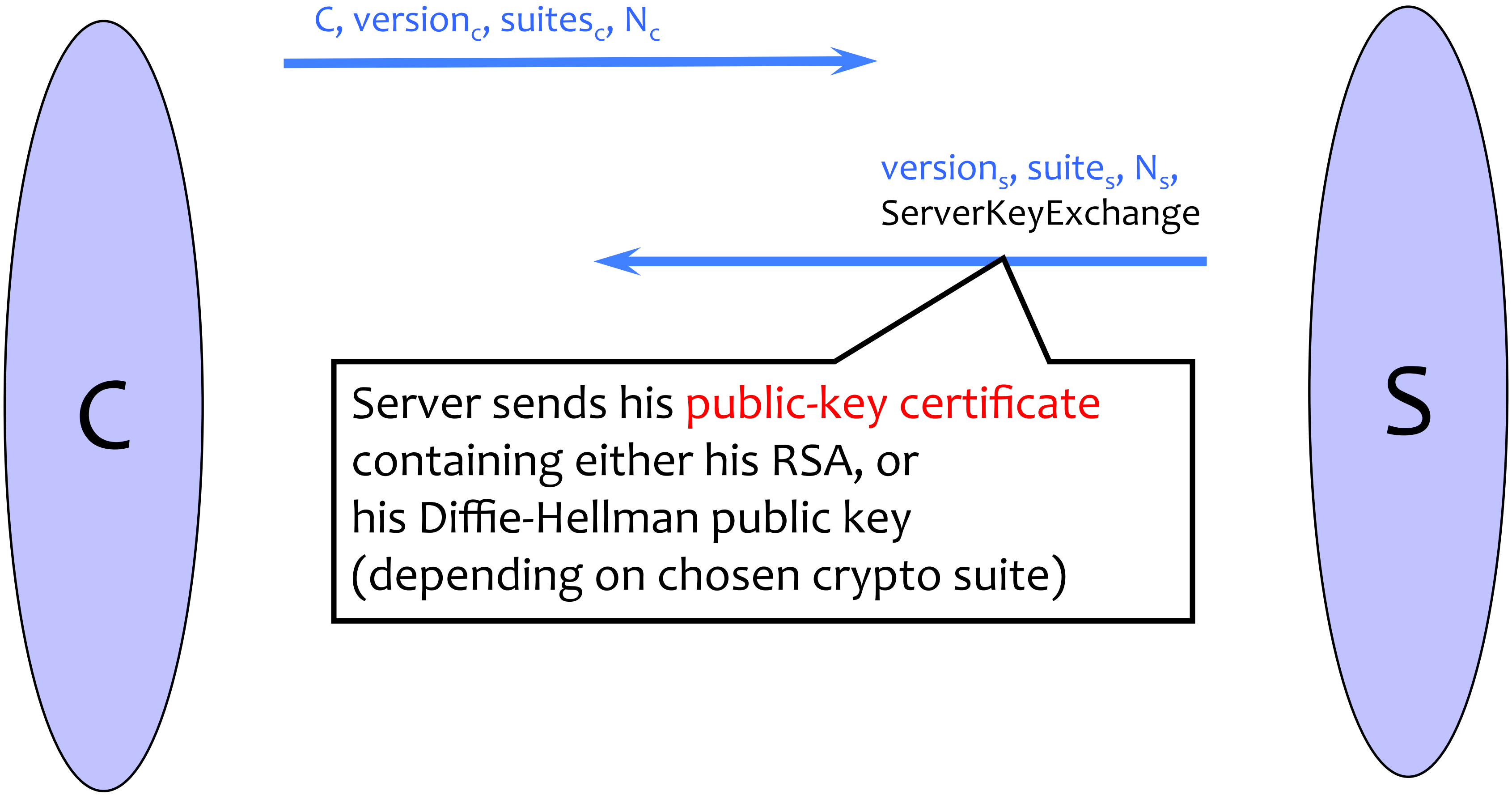
Basic Handshake Protocol



Server responds (in plaintext) with:

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

Basic Handshake Protocol

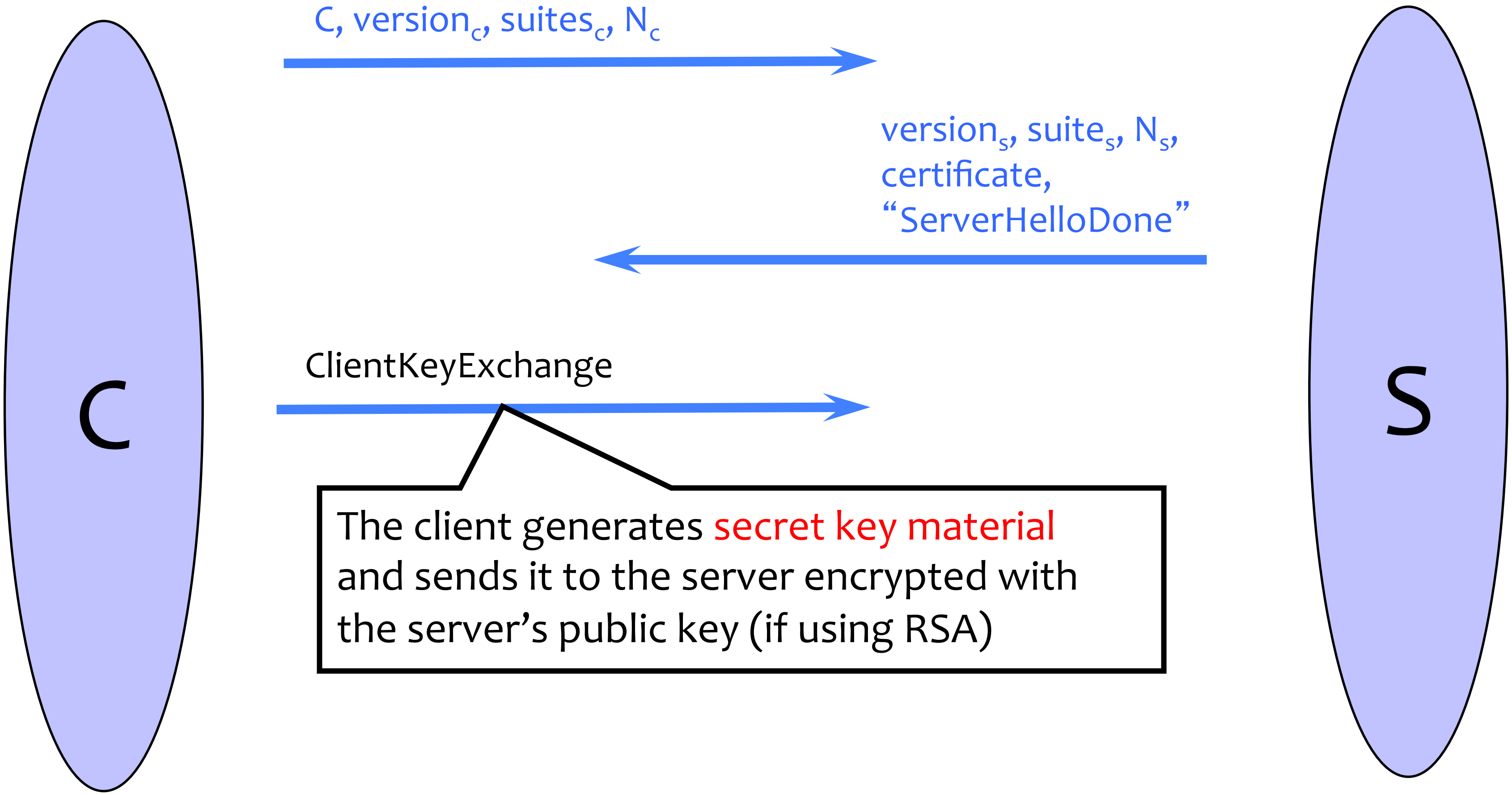


C, version$_c$, suites$_c$, N$_c$

version$_s$, suite$_s$, N$_s$, ServerKeyExchange

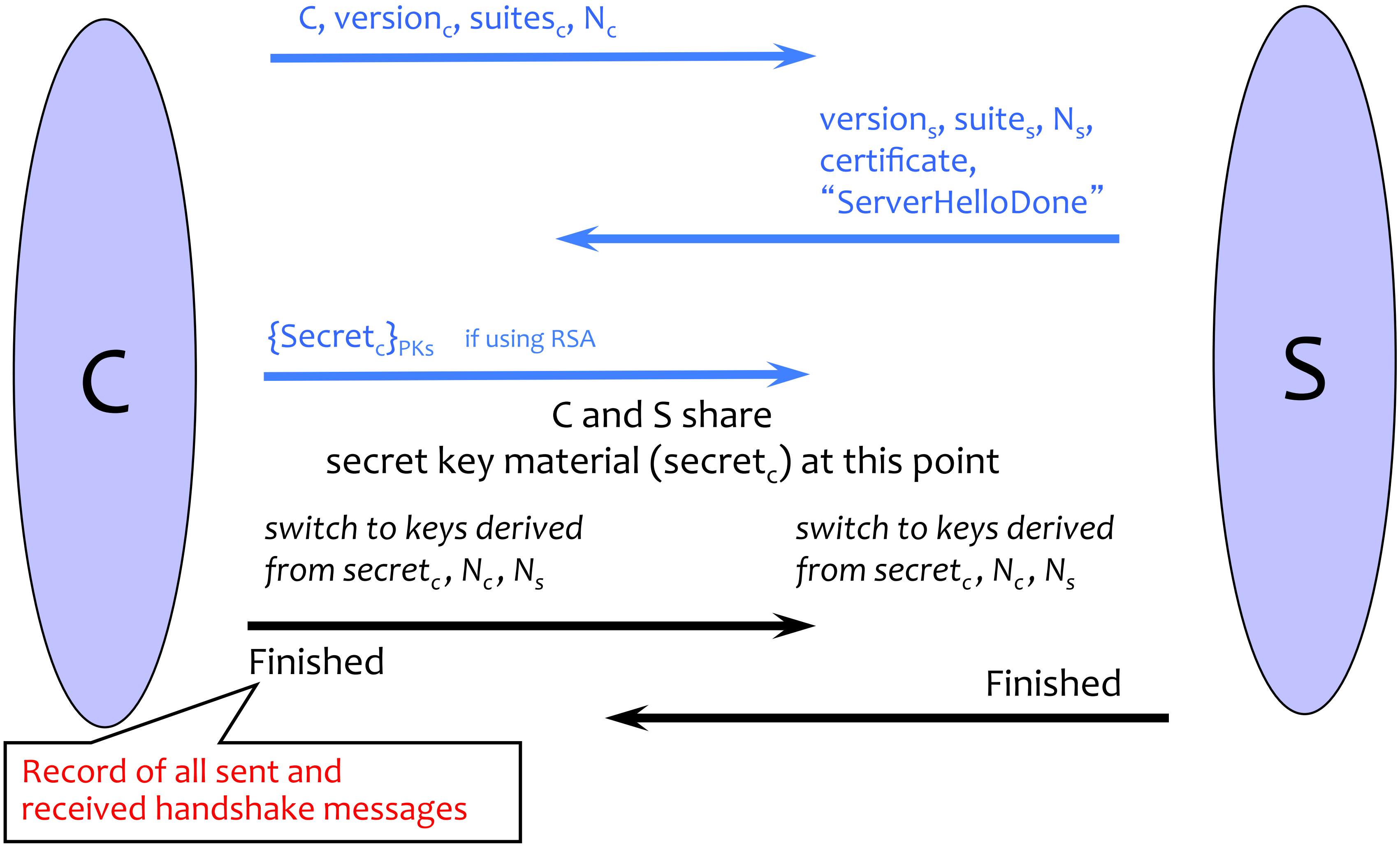
Server sends his public-key certificate containing either his RSA, or his Diffie-Hellman public key (depending on chosen crypto suite)

Basic Handshake Protocol


C
C, version$_c$, suites$_c$, N$_c$
version$_s$, suite$_s$, N$_s$,
certificate,
"ServerHelloDone"
ClientKeyExchange
S

The client generates secret key material and sends it to the server encrypted with the server's public key (if using RSA)

Basic Handshake Protocol



C, version_c, suites_c, N_c

version_s, suite_s, N_s, certificate, “ServerHelloDone”

$\{\text{Secret}_c\}_{PKs}$ if using RSA

C and S share secret key material (secret_c) at this point

switch to keys derived from secret_c, N_c, N_s

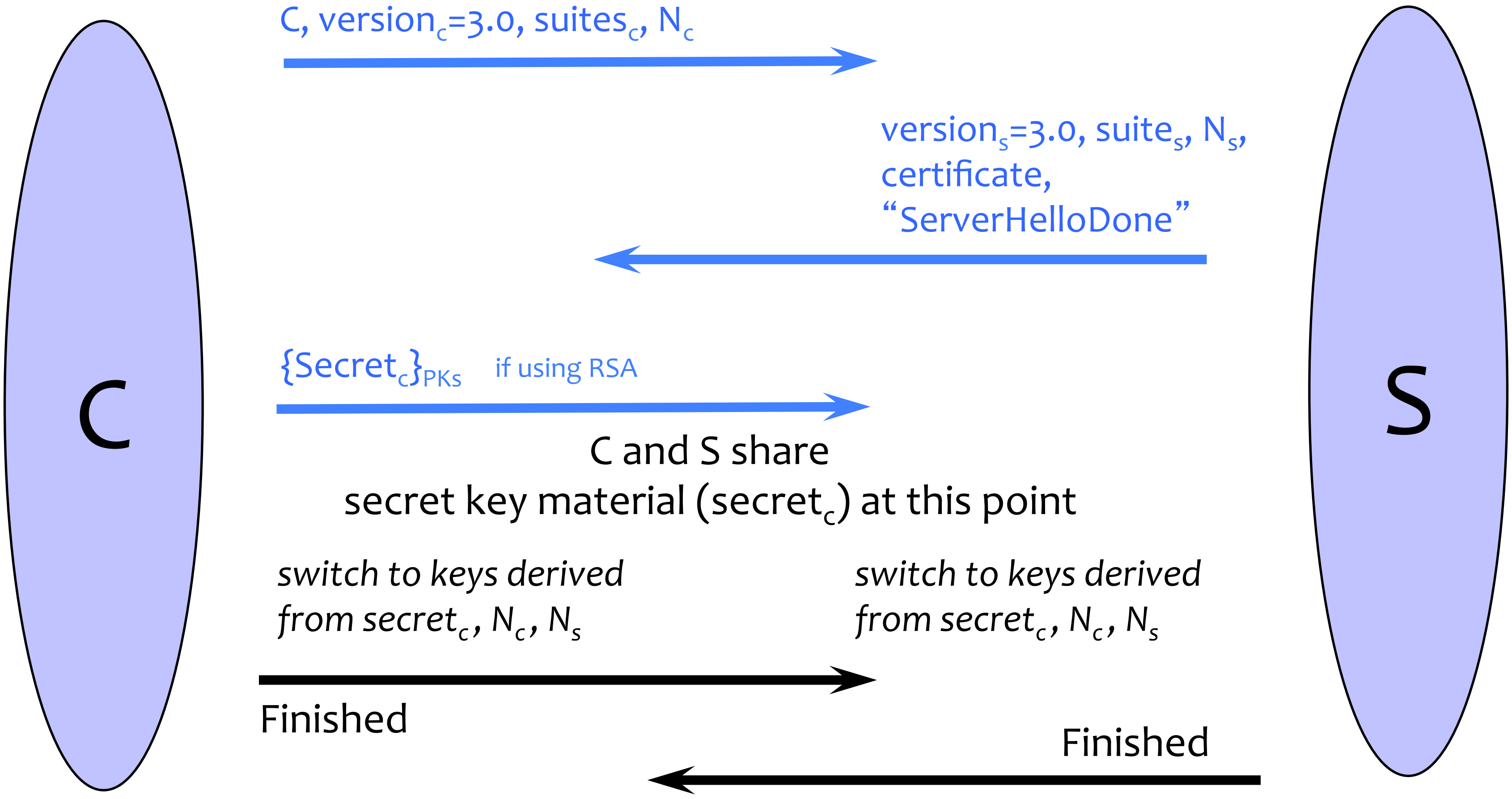
switch to keys derived from secret_c, N_c, N_s

Finished

Finished

Record of all sent and received handshake messages

“Core” SSL 3.0 Handshake (Not TLS)



C

S

C, version_c=3.0, suites_c, N_c

version_s=3.0, suite_s, N_s,
certificate,
“ServerHelloDone”

$\{\text{Secret}_c\}_{\text{PKs}}$ if using RSA

C and S share
secret key material (secret_c) at this point

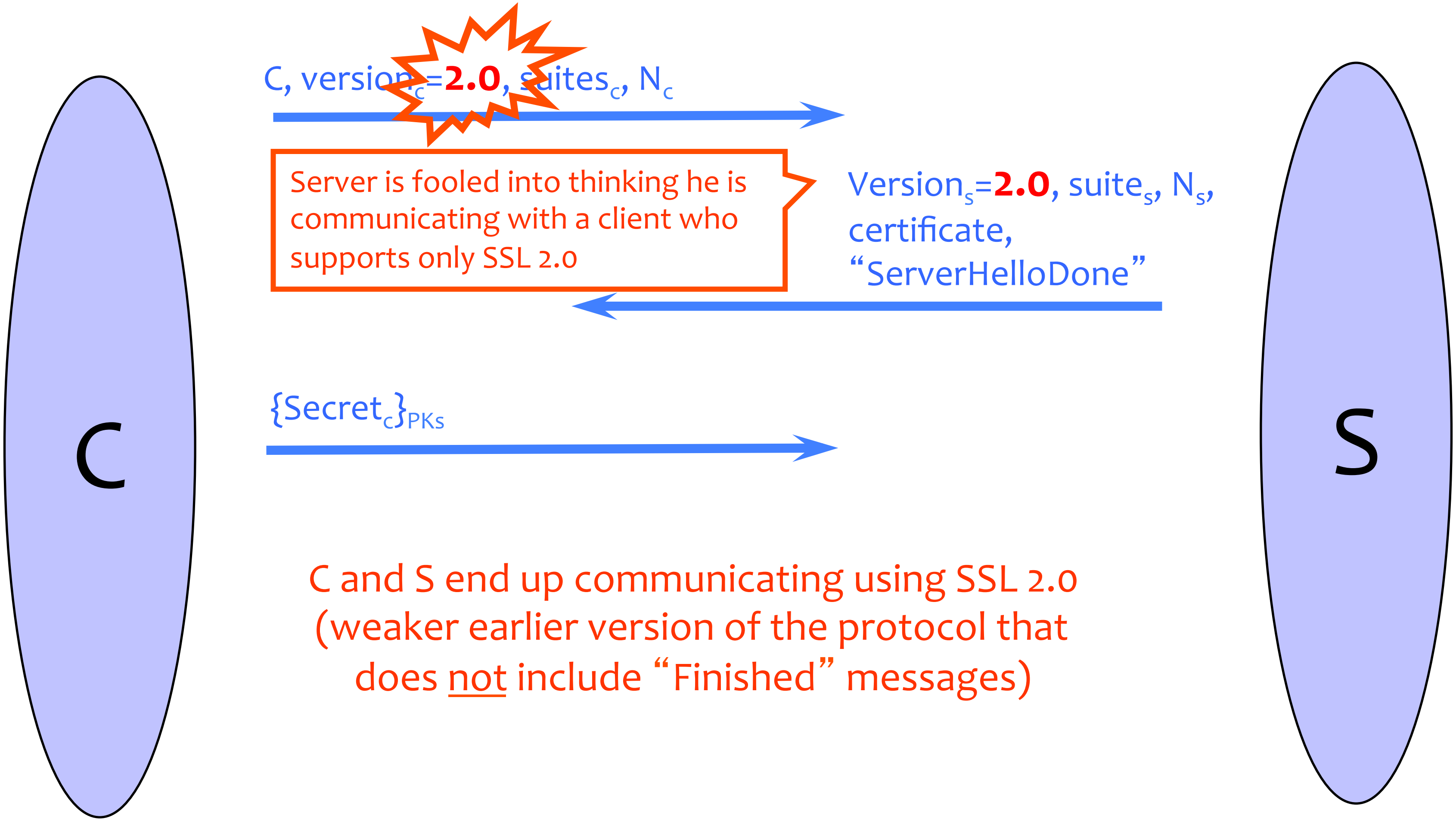
switch to keys derived from $\textit{secret}_c$*,* N_c*,* N_s

switch to keys derived from $\textit{secret}_c$*,* N_c*,* N_s

Finished

Finished

Version Rollback Attack



C, version$_c$=**2.0**, suites$_c$, N$_c$

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

Version$_s$=**2.0**, suite$_s$, N$_s$, certificate, “ServerHelloDone”

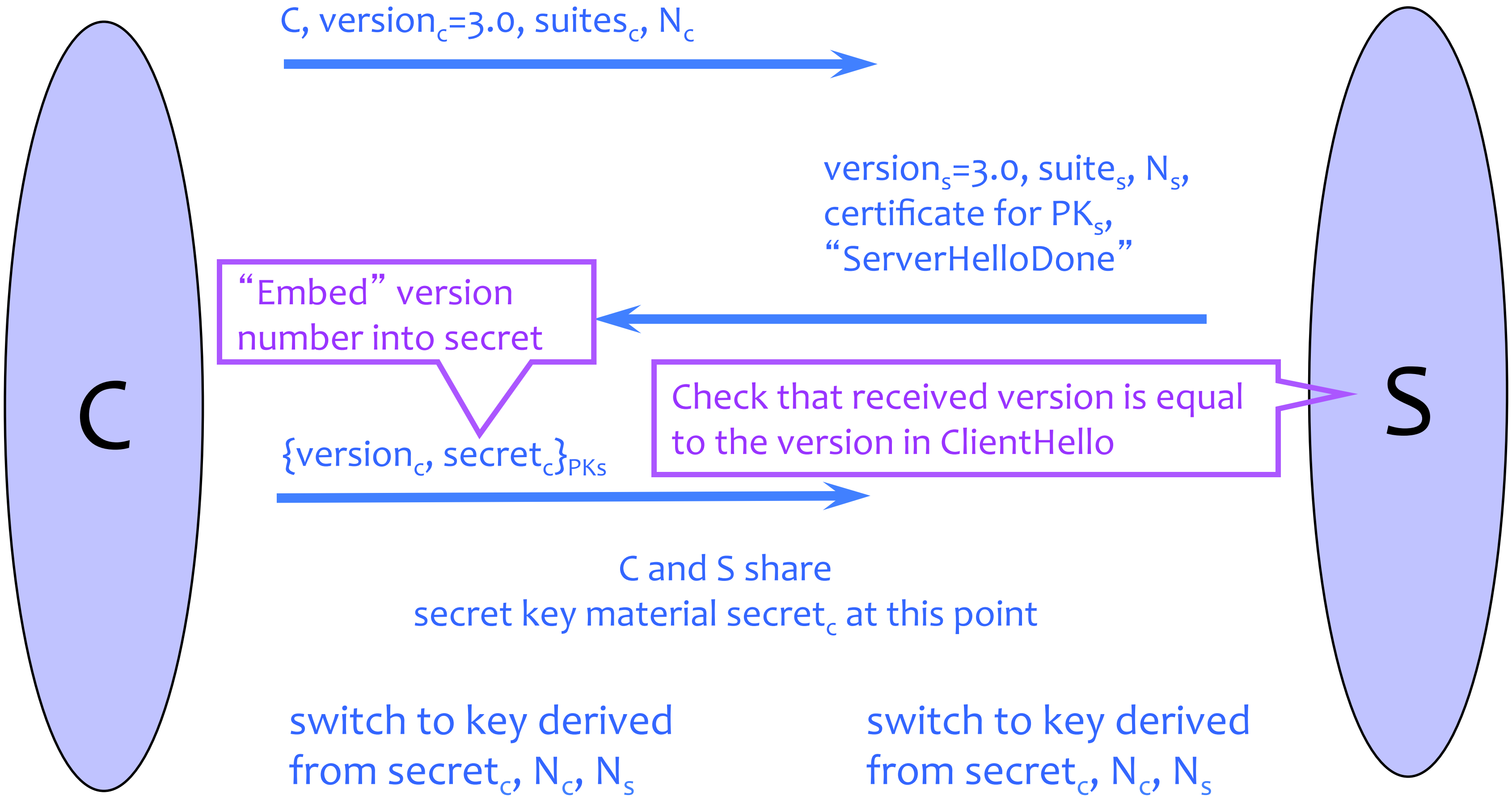
{Secret$_c$}$_{PKs}$

C and S end up communicating using SSL 2.0 (weaker earlier version of the protocol that does not include “Finished” messages)

“Chosen-Protocol” 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



C, version_c=3.0, suites_c, N_c

version_s=3.0, suite_s, N_s,
certificate for PK_s,
“ServerHelloDone”

“Embed” version number into secret

Check that received version is equal to the version in ClientHello

$\{\text{version}_c, \text{secret}_c\}_{PKs}$

C and S share
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