Lecture -09 – SSL/TLS

Ryan Cunningham
University of Illinois
ECE 422/CS 461 – Fall 2017

Security News

- Apple using face recognition for authentication
- Editorials (eg. NYTimes & Financial Times) discussing cryptocurrency “bubble”

Security on the web

- We don't want a network adversary to **modify our pages** or **eavesdrop**

- Defense: HTTPS

Threat model

- Controls infrastructure (routers, DNS, wireless access points)
- Passive attacker: only eavesdrops
- Active attacker: eavesdrops, injects, blocks, and modifies packets
- What examples?
  - Internet Cafe, hotel, ECE/Siebel, fake web site
- Does not protect against:
  - Intruder on server
  - Spyware on client
  - SQL injection, XSS, CSRF

Recap: Public-key cryptography

- Bob generates: k_{Pub}, k_{Priv}
- Alice can encrypt messages to Bob:
  - Using k_{Pub} encrypts messages, only Bob can decrypt with k_{Priv}
- Bob can sign messages that Alice can verify:
  - Using k_{Priv} signs message, anyone can verify
- Once we have a shared secret, we *encrypt* & *MAC*
- What was the hard problem we haven't yet solved about this?

Digital Signature

Signing

1. Hash the data
2. Encrypt the hash using k_{Priv} (signature)
3. Attach signature to data

Verifying

1. Hash the data
2. Decrypt the signature using k_{Pub}
3. Check that they match

Certificates

- Make use of trusted **"Certificate Authorities" (CA)**

- "This public key with SHA-256 hash (XXX) belongs to the site (name, e.g., Amazon.com)"
  - **Digitally signed** by a certificate authority

- Your browsers (e.g., Firefox, Chrome) trust a specific set of CAs as root CAs
  - Shipped with the public keys of the root CAs
  - Why do we need more than 1?

Certificates

How does Alice (web browser) obtain Bob's public key?

[Think of like a notary]

Browser (Alice)	**Server (Bob)**	**Certificate Authority (CA)**
(Knows $\mathbf{CA_{Pub}}$)		(Keeps $\mathbf{CA_{Priv}}$ Secret)
	1. Choose ($\mathbf{Bob_{Priv}}$,$\mathbf{Bob_{Pub}}$) $\mathbf{Bob_{Pub}}$ and evidence he is "Bob"	
		2. Checks evidence Signs certificate with $\mathbf{CA_{Priv}}$ "Bob's key is $\mathbf{Bob_{Pub}}$ Signed, CA"
	3. Keeps certificate on file	
1. Initiates a connection		
	2. Sends cert to Alice "Bob's key is $\mathbf{Bob_{Pub}}$ Signed, CA"	
3. Verifies CA's signature with $\mathbf{CA_{Pub}}$ and creates a secure channel using $\mathbf{Bob_{Pub}}$		

How the CA verifies your identity

- Typically 'DV' (**domain** validated)
  - –Proves you are in control of DNS registration
  - –Just an email based challenge to the address in the domain registration records
    - •Or some default email address, admin@domain.com
    - •Minimally secure [Why?]
  - –Alternately a web-based challenge
    - –Include challenge response in a <meta> tag
- Can also get 'EV' certs (extended validation)
- Cert has an expiration date

  (e.g., one year ahead) [Why?]

How to invalidate certificates?

- Expiration date of certs
- Certificate revocation
- What happens if a CA's secret key is leaked?
  - Can we trust the old certs from that CA?
- Interesting fact:
  - Google has instrumented Chrome such that when it observes a certificate for Google.com that it doesn't recognize, it panics…. (has happened several times)

Self-signed Certificates

- Issuer signs their own certificate
  - A loop in the owner and signer
- Avoid CA fees, useful for testing
  - You can add yourself as a CA to your own browser
- Browsers display warnings that users have to override
- Protects only against passive attacker "optimistic encryption"

Example: https://www.pcwebshop.co.uk/

This Connection is Untrusted

You have asked Firefox to connect securely to [redacted], but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Get me out of here!

▸ **Technical Details**

▸ **I Understand the Risks**

How do we translate?

Cryptographic Primitives

Symmetric Encryption

RSA

HMAC

Certificate

Public Key

RC4

Diffie-Hellman

DSA

ECDSA

Asymmetric Encryption

How do we translate?

Cryptographic Primitives

Symmetric Encryption

RSA

HMAC

Certificate

Public Key

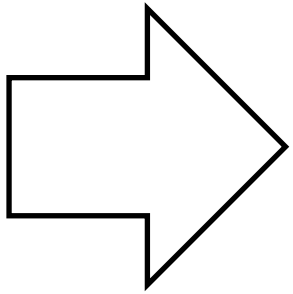
RC4

Diffie-Hellman

DSA

ECDSA

Asymmetric Encryption



Objectives

Message Integrity

Confidentiality

Authentication

How do we translate?

Cryptographic Primitives

Symmetric Encryption

RSA

HMAC

Certificate

Public Key

AES

Diffie-Hellman

DSA

ECDSA

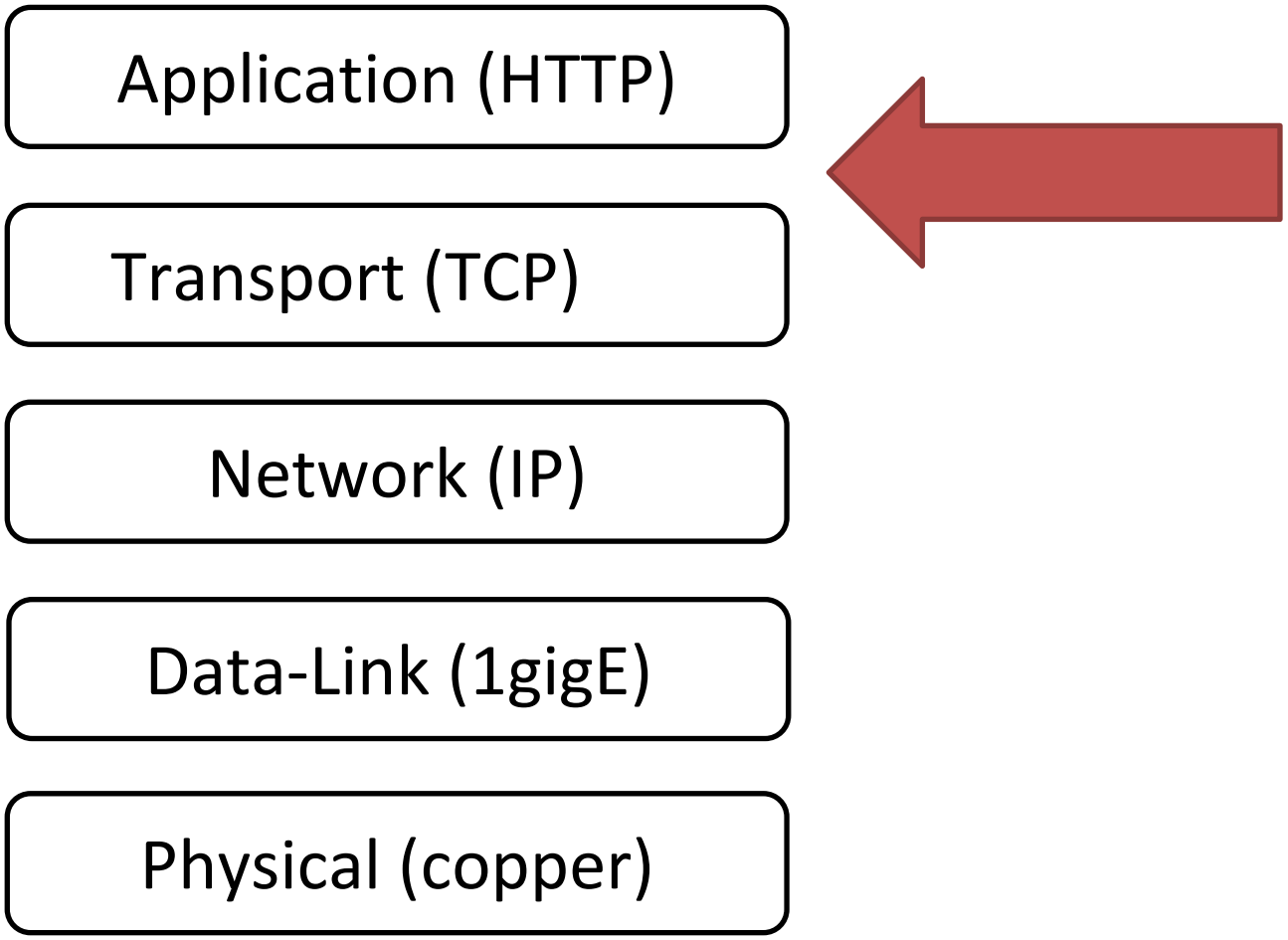
Asymmetric Encryption

Typical HTTPS Connection

Case Study: SSL/TLS

- Arguably the most important (and widely used) cryptographic protocol on the Internet
- Almost all encrypted protocols (minus SSH) use SSL/TLS for transport encryption
- HTTPS, POP3, IMAP, SMTP, FTP, NNTP, XMPP (Jabber), OpenVPN, SIP (VoIP), …

Where does TLS live?



Goals

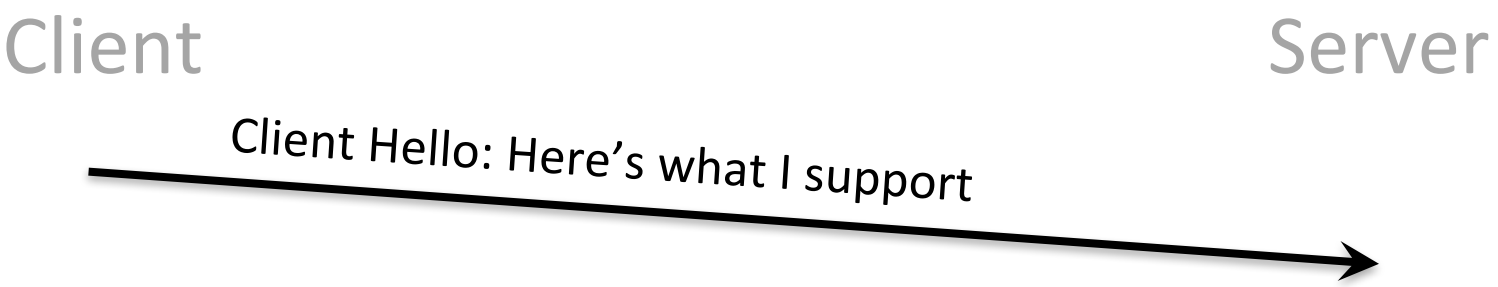
Confidentiality (Symmetric Crypto)

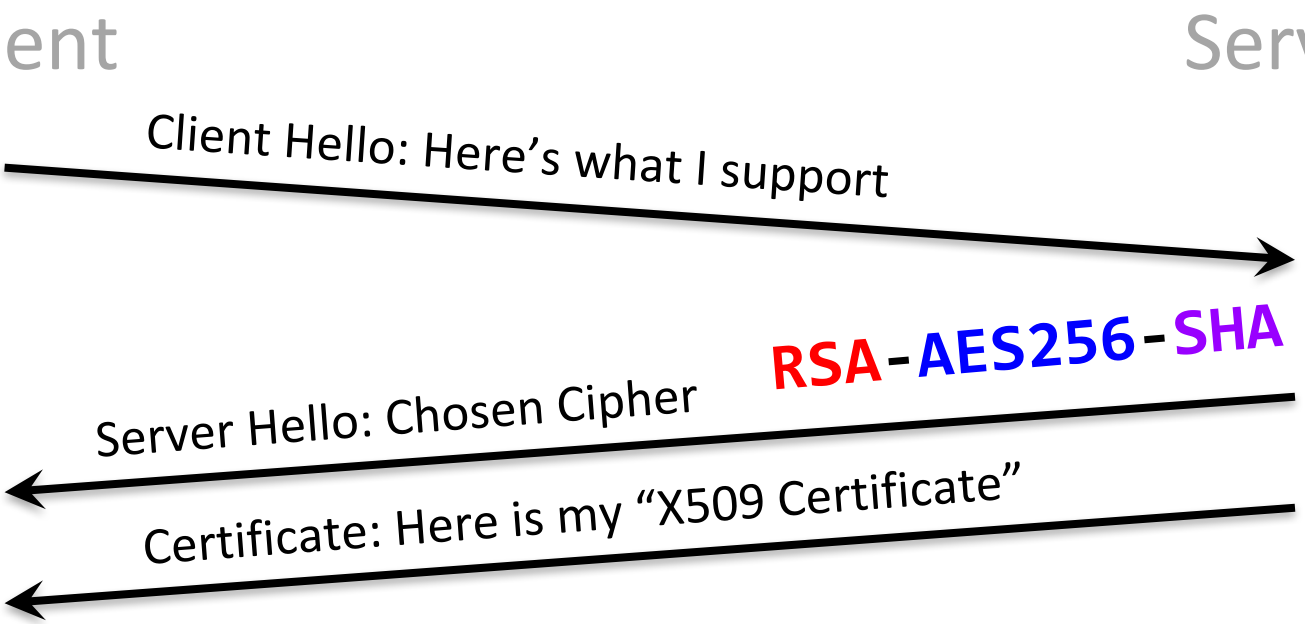
Message Integrity (HMACs)

Authentication (Public Key Crypto)

Client Server

“the handshake”


Client
Server
Client Hello: Here's what I support

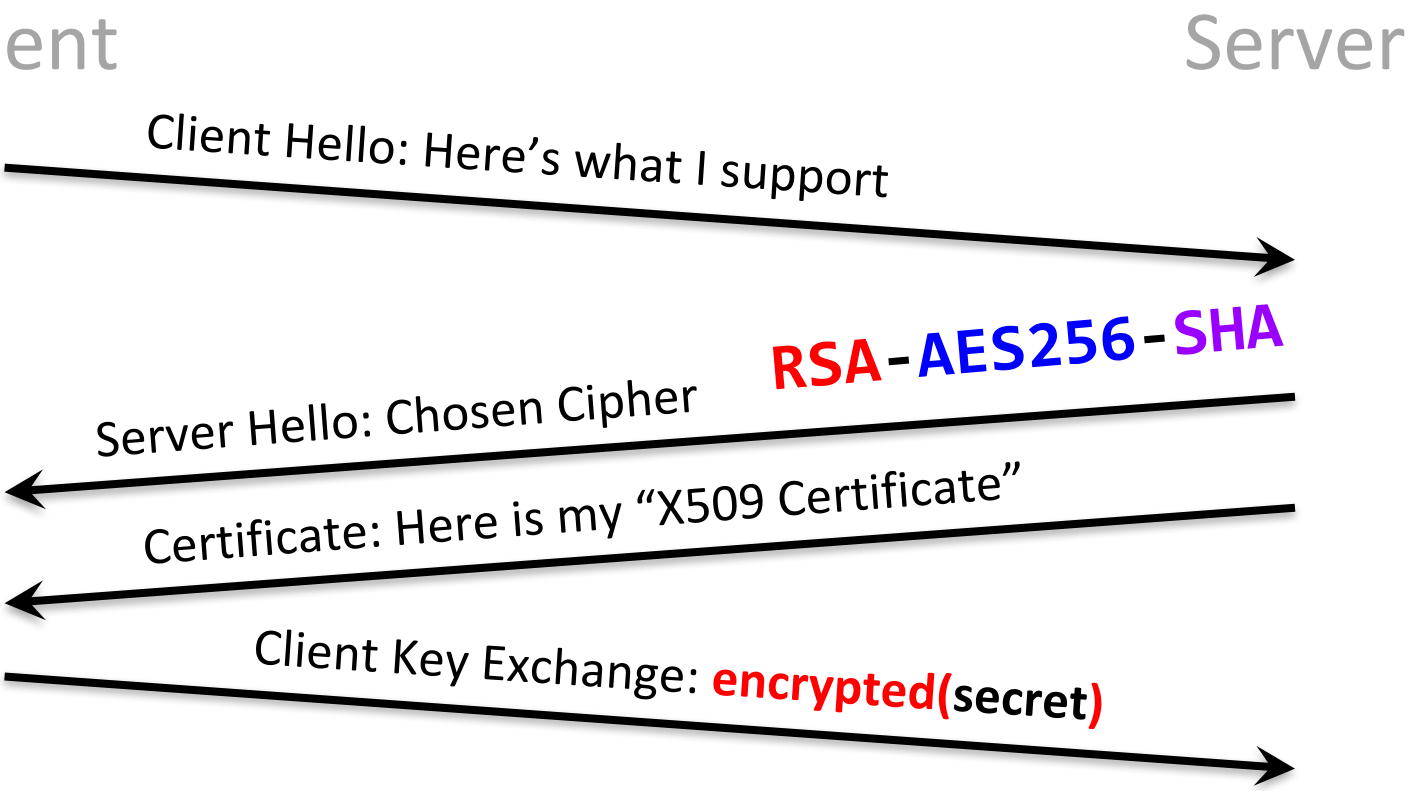


RSA-AES256-SHA

Key Exchange

Data Transfer Cipher

Message Digest / Authentication Code

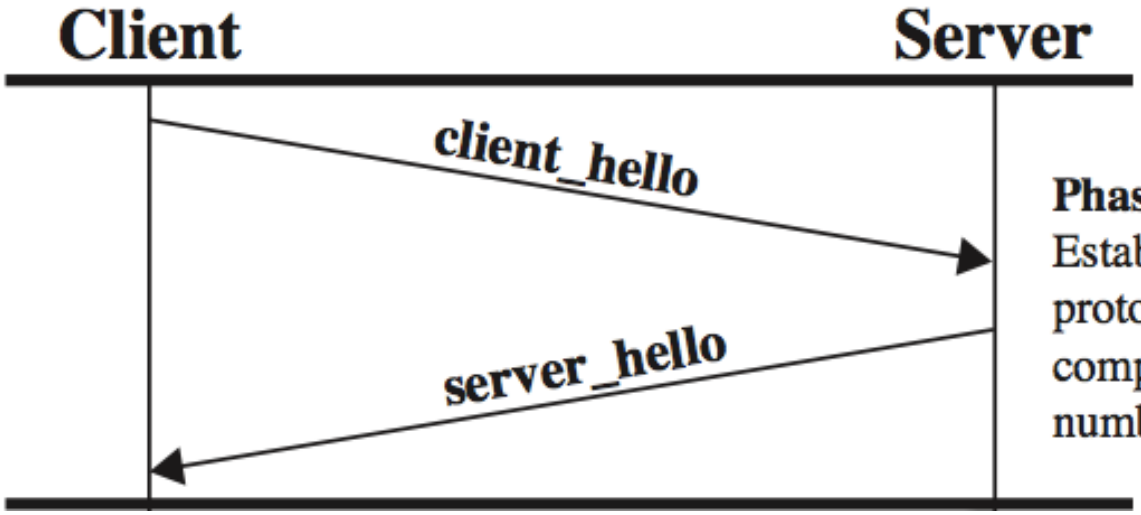


Encrypted using Server's public key
(The same public key included in the Cert)
This means: only the server can decrypt the secret! (Avoids MitM)

Shared **secret** is encrypted using Server's RSA public key
(The same RSA public key included in the Cert)
This means: only the server can decrypt the secret! (Avoids MitM)

TLS Handshake

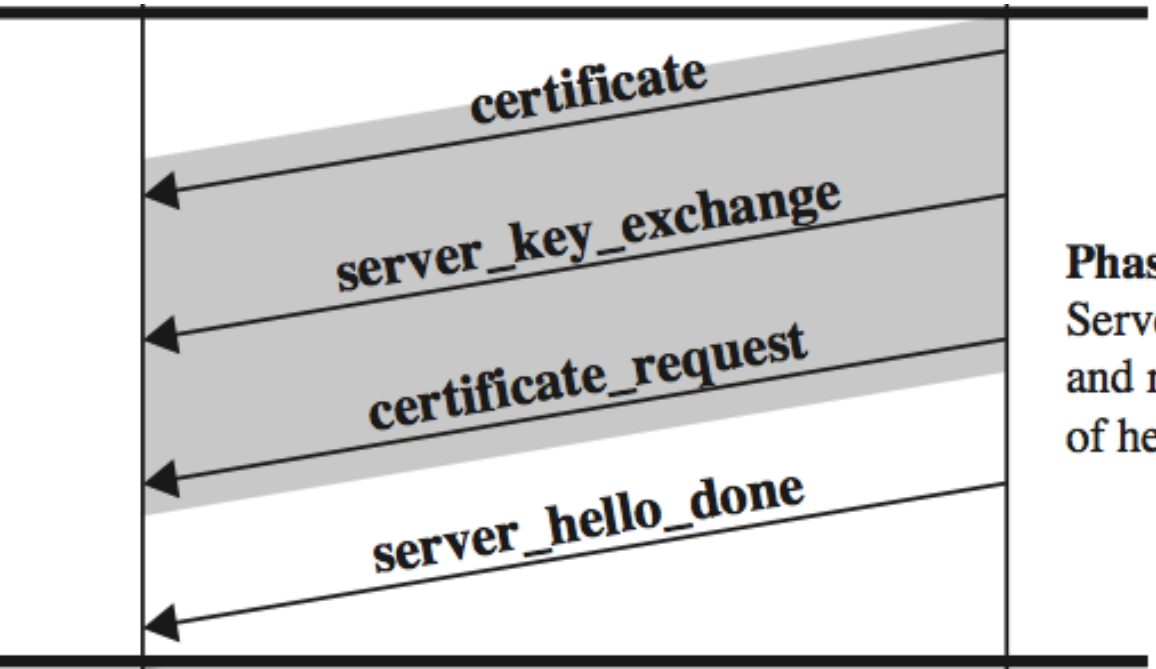
- Phase 1: establish capabilities
  - Which version of TLS?
  - What our session ID?
  - What is our cipher suite?
  - Are we compressing data?



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

TLS Handshake

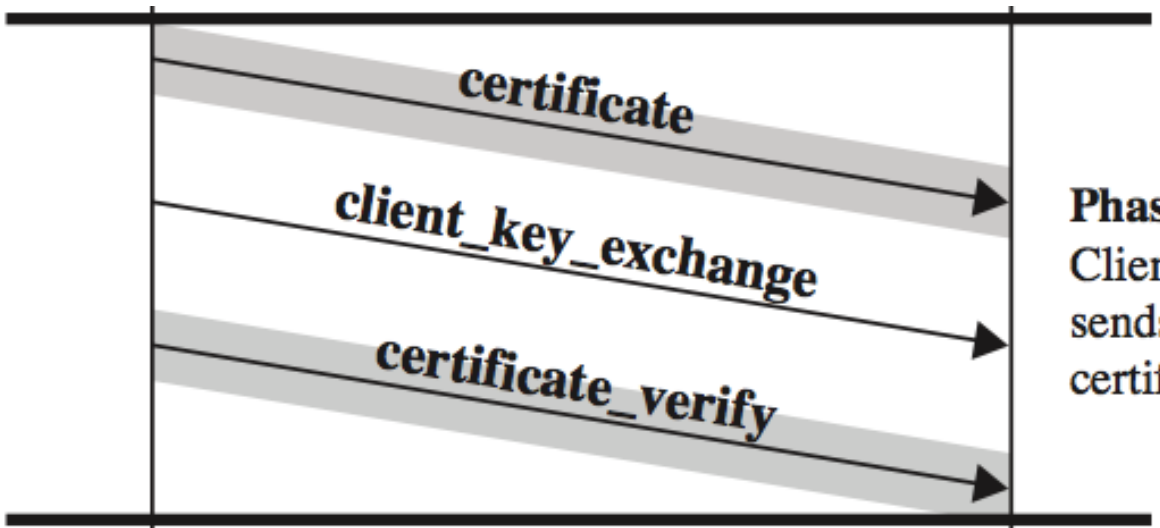
- Phase 2: Server Authentication
  - Server sends certificate



Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

TLS Handshake

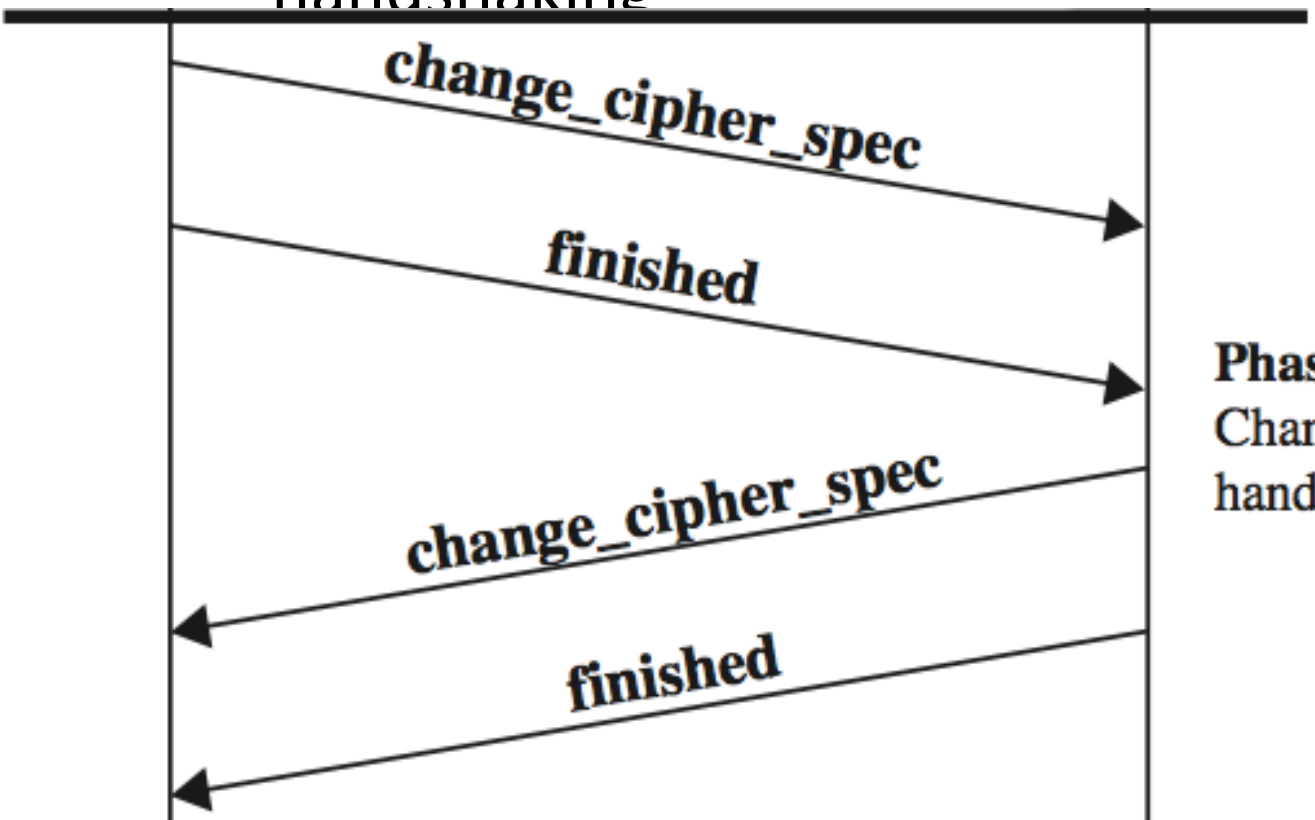
- Phase 3: Client Authentication
  - Client sends certificate (maybe)
  - Client exchanges key
  - Client sends verification of server cert



Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

TLS Handshake

- Phase 4: Switch to Secure Connection
  - Change to agreed upon cipher suite and stop handshaking



Phase 4
Change cipher suite and finish handshake protocol.

Cipher Suites

The following CipherSuite definitions require that the server provide an RSA certificate that can be used for key exchange. The server may request any signature-capable certificate in the certificate request message.

```
CipherSuite TLS_RSA_WITH_NULL_MD5                = { 0x00,0x01 };
CipherSuite TLS_RSA_WITH_NULL_SHA                = { 0x00,0x02 };
CipherSuite TLS_RSA_WITH_NULL_SHA256             = { 0x00,0x3B };
CipherSuite TLS_RSA_WITH_RC4_128_MD5             = { 0x00,0x04 };
CipherSuite TLS_RSA_WITH_RC4_128_SHA             = { 0x00,0x05 };
CipherSuite TLS_RSA_WITH_3DES_EDE_CBC_SHA        = { 0x00,0x0A };
CipherSuite TLS_RSA_WITH_AES_128_CBC_SHA         = { 0x00,0x2F };
CipherSuite TLS_RSA_WITH_AES_256_CBC_SHA         = { 0x00,0x35 };
CipherSuite TLS_RSA_WITH_AES_128_CBC_SHA256      = { 0x00,0x3C };
CipherSuite TLS_RSA_WITH_AES_256_CBC_SHA256      = { 0x00,0x3D };
```

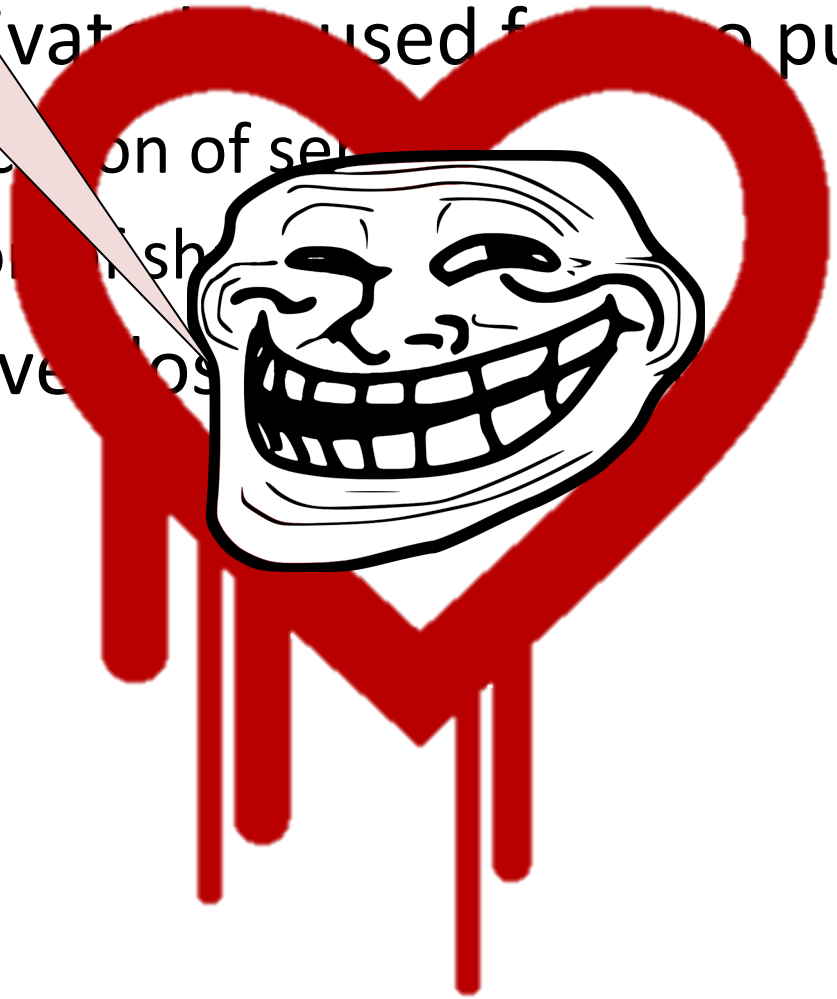
Forward secrecy

Using RSA for key exchange/authentication

- Get server's public key through authentication method
  - Server gives us certificate
  - We validate certificate
  - Certificate contains public key
- Encrypt secret key and send it to the server
- Future communication is
  - confidential - only server can decrypt our key
  - authentic - only server can respond using key
- NO MAN IN THE MIDDLE!

Problem?

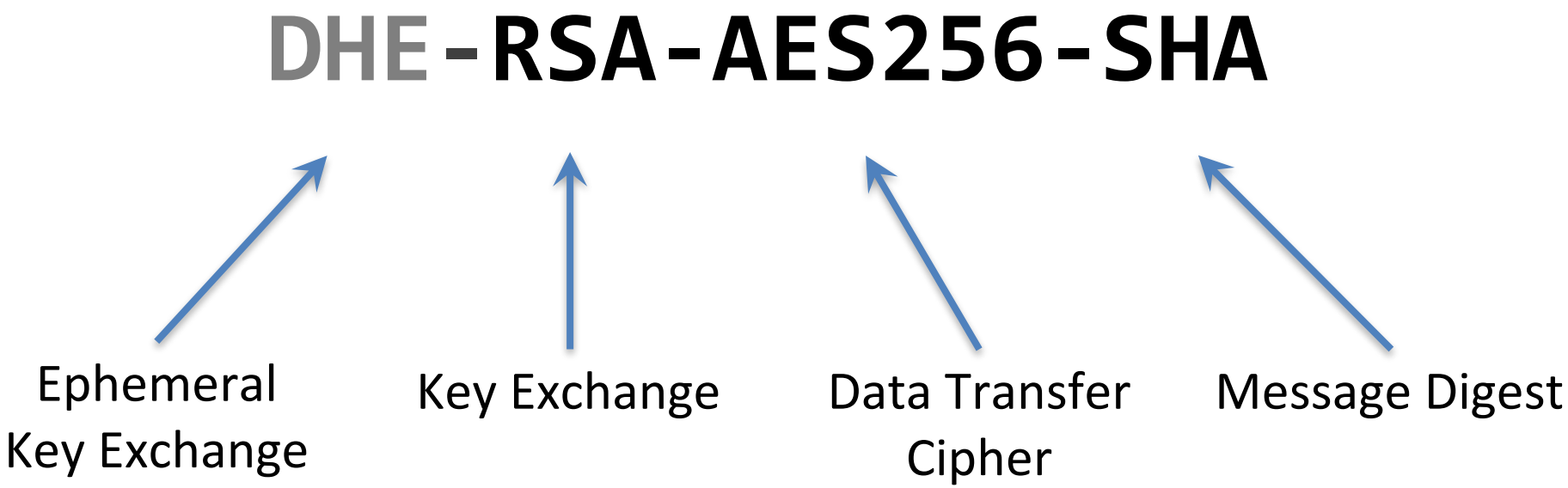
- Server's priva[illegible] used f[illegible] purposes
  - –Authentic[illegible]on of se[illegible]
  - –Encryptio[illegible]sh[illegible]
- What if serve[illegible]los[illegible]



Solution

- Use private key *only* for authentication
- Use Diffie-Hellman to exchange secret key
  - Called “Diffie-Hellman Ephemeral”
- Cipher suit we should use:
  - ECDHE-RSA-RC4-SHA
- Ensures *Forward Secrecy*
  - Protect ourselves in the present from attacks in the future

Cipher Suites



DH and DHE


```
CipherSuite TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA      = { 0x00,0x0D };
CipherSuite TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA      = { 0x00,0x10 };
CipherSuite TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA     = { 0x00,0x13 };
CipherSuite TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA     = { 0x00,0x16 };
CipherSuite TLS_DH_DSS_WITH_AES_128_CBC_SHA       = { 0x00,0x30 };
CipherSuite TLS_DH_RSA_WITH_AES_128_CBC_SHA       = { 0x00,0x31 };
CipherSuite TLS_DHE_DSS_WITH_AES_128_CBC_SHA      = { 0x00,0x32 };
CipherSuite TLS_DHE_RSA_WITH_AES_128_CBC_SHA      = { 0x00,0x33 };
CipherSuite TLS_DH_DSS_WITH_AES_256_CBC_SHA       = { 0x00,0x36 };
CipherSuite TLS_DH_RSA_WITH_AES_256_CBC_SHA       = { 0x00,0x37 };
CipherSuite TLS_DHE_DSS_WITH_AES_256_CBC_SHA      = { 0x00,0x38 };
CipherSuite TLS_DHE_RSA_WITH_AES_256_CBC_SHA      = { 0x00,0x39 };
CipherSuite TLS_DH_DSS_WITH_AES_128_CBC_SHA256    = { 0x00,0x3E };
CipherSuite TLS_DH_RSA_WITH_AES_128_CBC_SHA256    = { 0x00,0x3F };
CipherSuite TLS_DHE_DSS_WITH_AES_128_CBC_SHA256   = { 0x00,0x40 };
CipherSuite TLS_DHE_RSA_WITH_AES_128_CBC_SHA256   = { 0x00,0x67 };
CipherSuite TLS_DH_DSS_WITH_AES_256_CBC_SHA256    = { 0x00,0x68 };
CipherSuite TLS_DH_RSA_WITH_AES_256_CBC_SHA256    = { 0x00,0x69 };
CipherSuite TLS_DHE_DSS_WITH_AES_256_CBC_SHA256   = { 0x00,0x6A };
CipherSuite TLS_DHE_RSA_WITH_AES_256_CBC_SHA256   = { 0x00,0x6B };
```

HTTPS key exchange

At the end of the exchange, a secret is used to generate 4 keys (2 for MAC, 2 for encryption)

1. RSA key exchange
   - Use RSA for encryption to achieve confidentiality
   - Use RSA for signature to achieve authentication
2. Ephemeral Diffie Hellman (EDH)
   - For *forward secrecy* guarantees
3. Fixed Diffie Hellman
   - For packet inspection within the server's network

SSL Certificates

- A trusted authority vouches that a certain public key belongs to a particular site
- Format called x.509 (complicated)
- Browsers ship with CA public keys for large number of trusted CAs [accreditation process]
- Important fields:
    - Common Name (CN) [e.g., *.google.com]
      Expiration Date [e.g. 2 years from now]
      Subject's Public Key
      Issuer -- e.g., Verisign
      Issuer's signature
- Common Name field
    - Explicit name, e.g. ece.illinois.edu
    - Or wildcard, e.g. *.illinois.edu

X509 Certificates

Subject: C=US/O=Google Inc/CN=www.google.com
Issuer: C=US/O=Google Inc/CN=Google Internet Authority
Serial Number: 01:b1:04:17:be:22:48:b4:8e:1e:8b:a0:73:c9:ac:83
Expiration Period: Jul 12 2010 - Jul 19 2012
Public Key Algorithm: rsaEncryption
Public Key: 43:1d:53:2e:09:ef:dc:50:54:0a:fb:9a:f0:fa:14:58:ad:a0:81:b0:3d
7c:be:b1:82:19:b9:7c3:8:04:e9:1e5d:b5:80:af:d4:a0:81:b0:b0:68:5b:a4:a4
:ff:b5:8a:3a:a2:29:e2:6c:7c3:8:04:e9:1e5d:b5:7c3:8:04:e9:39:23:46

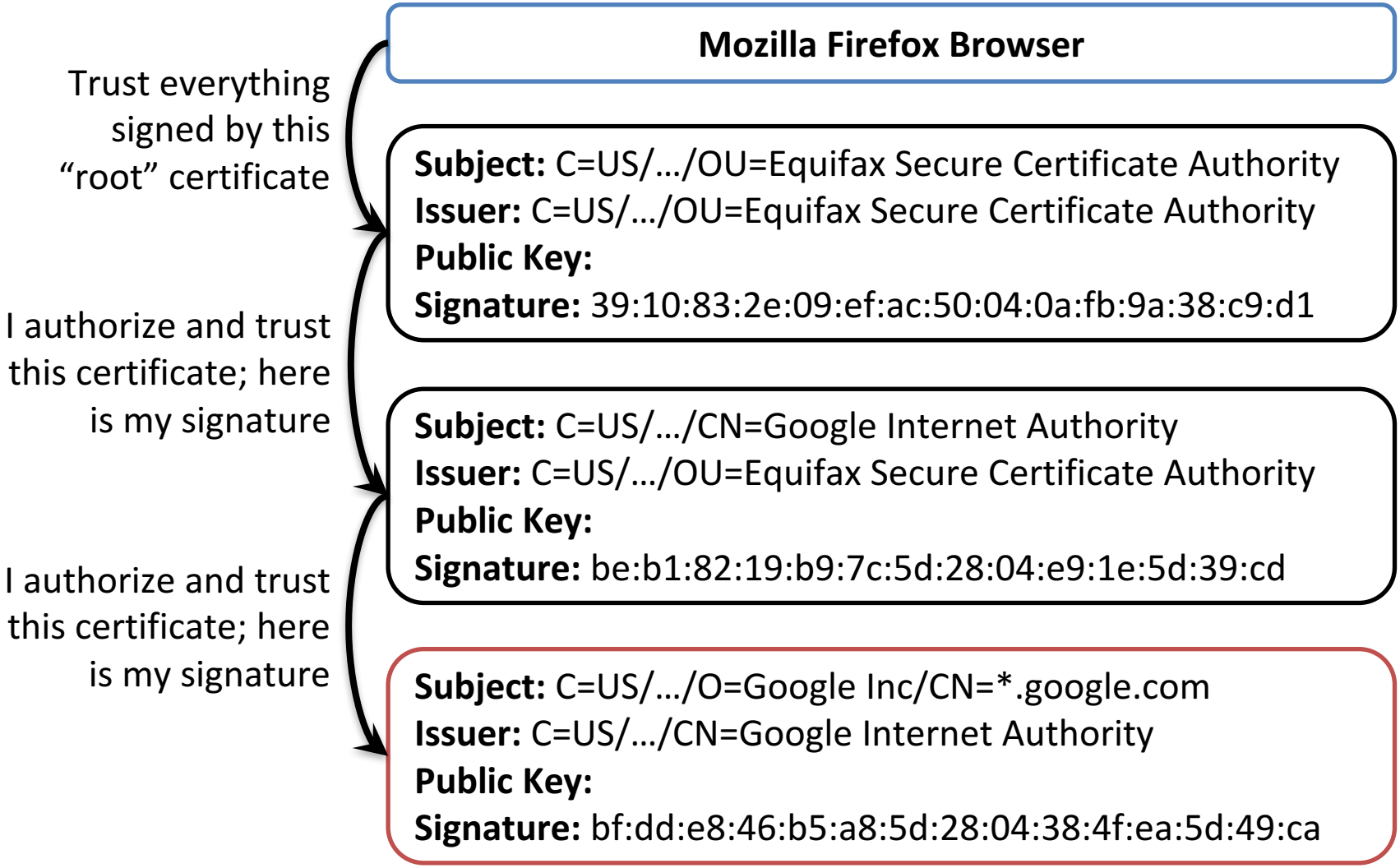
Signature Algorithm: sha1WithRSAEncryption

Signature: 39:10:83:2e:09:ef:ac:50:04:0a:fb:9a:f0:fa:14:58:ad:a0:81:b0:3d
7c:be:b1:82:19:b9:7c3:8:04:e9:1e5d:b5:80:af:d4:a0:81:b0:b0:68:5b:a4:a4
:ff:b5:8a:3a:a2:29:e2:6c:7c3:8:04:e9:1e5d:b5:7c3:8:04:e9:1e:5d:b5

Certificate Chains

- CA can delegate ability to generate certificates for certain names: Intermediate CAs
- Root CA signs "certificate issuing certificate" for delegated authority
- Delegated authority signs cert for ”ece.illinios.edu”
  - Delegated CA certificate: "pubkey=.... is allowed to sign certs for *.illinois.edu"
- Browser that trusts root can examine certs to establish validity -- "Chain of trust”
- How to find out about all the CAs?
- More than 1000 trusted parties today, can sign for any domain – huge problem!

Certificate Chains



Certificate Authority Ecosystem

Each browser trusts a set of CAs

- CAs can sign certificates for new CAs
- CAs can sign certificates for any web site

If a single CA is compromised, then the entire system is compromised

We ultimately place our complete trust of the Internet in the weakest CA

Immediate Concerns

- Nobody has any idea who all these CAs are…
- 1,733 *umich*-known browser trusted CAs
- History of CAs being hacked (e.g. Diginotar)
- Oooops, Korea gave every elementary school, library, and agency a CA certificate (1,324)
  - Luckily invalid due to a higher-up constraint

Getting a Certificate

- Certificates are free (from LetsEncrypt!)
  - –Identity validated by challenge to website
- Certificates are cheap elsewhere too
  - –Identity is validated via e-mail to the default e-mail addresses
- Setting up SSL is hard. People are terrible at it.

  Certificate Signing Requests, eugh

  Integrating in a web server

SSL in the browser

- Lock icon
  - HTTPS cert must be issued by a CA trusted by browser
  - HTTPS cert is valid (e.g., not expired or revoked)
  - CommonName in cert matches domain in URL
- Extended Validation (EV) certificates
  - CA does extra work to verify identity -- expensive, but more secure
- Invalid certificate warnings

Attack Vectors

- Attack the weakest Certificate Authority
- Attack browser implementations
- Magically notice a bug in a key generation library that leads you to discovering all the private keys on the Internet
- Attack the cryptographic primitives
  - Math is hard

"-----BEGIN RSA PRIVATE KEY-----" -openssl

Search

About 274,000 results (0.24 seconds)

Everything
Images
Maps
Videos
News
Shopping
More

All results
Related searches
More search tools

-----BEGIN RSA PRIVATE KEY - Pastebin.com - #1 paste tool since ...
pastebin.com/TbaeU93m
19 Apr 2010 – ... the difference. Copied. -----**BEGIN RSA PRIVATE KEY**-----.
MIICXwIBAAKBpenis1ePqHkVN9IKaGBESjV6zBrIsZc+XQYTtSlVa9R/4SAXoYpl ...

-----BEGIN RSA PRIVATE KEY - Pastebin.com - #1 paste tool since ...
pastebin.com/sC7bGw30
18 Apr 2010 – ... difference. Copied. -----**BEGIN RSA PRIVATE KEY**-----.
MIIEogIBAAKCAQEAvxBalhzKMewLvmIr1ptID1gO7EWGFyudzOAHLqm3+0+gpPbk ...

site:pastebin.com "-----**BEGIN RSA PRIVATE KEY**-----" - Posterous
cdevers.posterous.com/sitepastebincom-**begin-rsa-private-key**-google
20 Apr 2010 – Apr 19, 2010 ... -----**BEGIN RSA PRIVATE KEY**-----
MIICXwIBAAKBpenis1ePqHkVN9IKaGBESjV6zBrIsZc+ XQYTtSlVa9R/4SAXoYpl .

help/en/howto/sftp – Cyberduck
trac.cyberduck.ch/wiki/help/en/howto/sftp
Private keys containing a DSA or RSA private key in PEM format are supported (look for -----BEGIN DSA PRIVATE KEY----- or -----**BEGIN RSA PRIVATE KEY**----- ...

SSH access with a private RSA key [Archive] - VanDyke Software For...
forums.vandyke.com/archive/index.php/t-2185.html
2 Sep 2011 – -----**BEGIN RSA PRIVATE KEY**-----
MIIEogIBAAKCAQBujdbtxyIX4KaQPeTf5F/
aOSBwSpZN4MjTixU2Yq8JkipjMYpYwpNj1TODzRJf ...

Attacking site design

- SSLstrip attack
  - Proxy through the content w/o HTTPS
- Defense
  - Default HTTPS for all web sites?
  - HSTS (hypertext strict transport security): header says: always expect HTTPS, enforced by browsers.
  - HTTPS Everywhere: browser extension
  - EV: Extended Validation (compared to DV: Domain Validation)

Attacking site design

- Mixed Content attack -- Page loads over HTTPS but contains content over HTTP
  - e.g. JavaScript, Flash
  - Active attacker can tamper with HTTP content to hijack session
- Defense: Browser warnings: ["This page contains insecure content"],
  - but inconsistent and often ignored

UI interface based attacks

- Invalid certs
  - Expired, Common Name != URL, unknown CA (e.g., self-signed)
- Defense: browser warnings, anti-usability to bypass…
- Picture-in-picture attack: spoof the user interface
  - Attacker page draws fake browser window with lock icon
- Defense: individualized image

Attacking the PKI: CA compromise
Example: DigiNotar

Attacking the PKI: CA compromise Example: DigiNotar

- DigiNotar ***was*** a Dutch Certificate Authority
- On June 10, 2011, **`*.google.com`** cert was issued to an attacker and subsequently used to orchestrate MITM attacks in Iran
- Nobody noticed the attack until someone found the certificate in the wild… and posted to *pastebin*

DigiNotar Contd.

- DigiNotar later admitted that dozens of fraudulent certificates were created
- Google, Microsoft, Apple and Mozilla all revoked the root Diginotar certificate
- Dutch Government took over Diginotar
- Diginotar went bankrupt and died

Attacking the PKI: Hash collisions

- MD5/SHA1 is known to be broken -- Can generate collisions
- In 2008, researchers showed that they could create a rogue CA certificate using an MD5 collision
- Attack: Make colliding messages A, B, with same MD5 hash:
  - A: Site certificate: "cn=attack.com, pubkey=...."
  - B: Delegated CA certificate: "pubkey=.... is allowed to sign certs for *"
  - Get CA to sign A -- Signature is Sign(MD5(message))
  - Signature also valid for B (same hash)
  - Attacker is now a CA!
  - Make a cert for any site, browsers will accept it

MD5 considered harmful

- MD5 CA certificates still exist, but CAs have stopped signing certificates with them
  - 879,705 certificates still have MD5 signatures

- SHA-1 should not be used either
  - 46,969,095 out of 146,442,087 certs ever seen by Censys use SHA1WithRSA (32%)

Attacking implementations: Null Termination Attack

- ASN.1 utilizes Pascal-style strings
- Web browsers utilize use C-style strings
- Announced by Moxie Marlinspike in 2009

`gmail.com\0.badguy.com`

Null Termination Attack

- www.attacker.com
  - [CAs verify cert by looking up who owns the last part of the domain via DNS record]
  - emails "webmaster@attack.com" --> "Click here to validate cert request”
- x.509 certs encode CN field as a Pascal string (length+data)
- Browsers copy it into a C string (data+\0)
- What if CA contains "\0"?
  - www.paypal.com\0.attacker.com?
  - CA contacts "attacker.com" to verify (last part of domain name)
  - Browsers copy to C string, terminates at "\0" -- see only paypal.com
  - Attacker now has a cert that works for Paypal!

Other implementation-based attacks

- Goto fail, Feb. 2014 (Apple SSL bug; skipped certificate check for almost a year!)
- Heartbleed, April 2014 (OpenSSL bug; leaked data, possibly including private key!)
- Mozilla BERserk vulnerability, Oct 2014 (Bug in verifying cert signatures, allowed spoofing certs, probably since the beginning….!)
- Logjam, Oct 2016 (TLS vulnerable to Man-in-the middle “Downgrade” attack)

Who controls the TLS endpoint?

Cloudbleed

- one of the most popular “content delivery networks”
- acts as the SSL endpoint for many servers
- a **buffer overflow** attack caused it to leak HTTPS data

Clientside HTTP Interception -

- Most antivirus software intercepts your HTTPS [How?]
- Introduces new vulnerabilities by implementing poorly
- Tavis Ormandy (again)

Takeaways

- Use HTTPS! It’s so much better than nothing

- SSL keeps breaking. Use it, but don't rely on it exclusively.