CSE 127 Computer Security
Stefan Savage, Fall 2023, Lecture 10

Cryptography II: PKI
Using Cryptography (review)

- Alice wants to send (a plaintext) $m$ to Bob, via a channel that is controlled by Eve.
Cryptographic Primitives (review)

- **Confidentiality**
  - Symmetric Encryption
    - $c = E_k(m)$, $m = D_k(c)$
  - Asymmetric Encryption
    - $c = E_K(m)$, $m = D_k(c)$
  - Combining Asymmetric with Symmetric
    - $k' \leftarrow r, E_K(k')||E_k(m)$
  - You can rely on plaintext remaining secret.
    - Ciphertext reveals nothing about plaintext contents
  - You **cannot** rely on plaintext remaining unmodified.

- **Integrity and Authenticity**
  - Symmetric MAC
    - $a = MAC_k(m)$
  - Asymmetric Signature
    - $s = S_k(H(m))$
    - $V_K(s, H(m))$: returns true or false
  - You can rely that whoever generated the tag (MAC or signature) had the secret key.
  - You **cannot** rely on tag not leaking information about the message.
Using Cryptography

- Assume we encrypt and sign a message from Alice to Bob. Assume decryption is successful and the signature verifies.

- What can Alice and Bob assume?

- Bob knows that
  - Alice knows the plaintext
  - Alice signed the plaintext at some point in the past

- Alice knows that
  - Only Bob can extract the plaintext from the encrypted channel
  - Bob can prove that Alice signed the plaintext
    - True?
Using Cryptography

- Assume we encrypt and sign a message from Alice to Bob
  Assume decryption is successful and the signature verifies

- What can Alice and Bob assume?

- Bob knows that
  - Alice knows the plaintext (and anyone else she shared it with or copied it from)
  - Alice (or someone with her private key) signed the plaintext at some point in the past

- Alice knows that
  - Only Bob (or someone with his private key) can extract the plaintext from the encrypted channel
  - Bob (or anyone else) can prove that Alice (or someone with her private key) signed the plaintext
Using Cryptography

- Alice does not know:
  - Whether Bob received the message
  - When Bob received the message
  - How many times Bob received the message
  - Whether Bob keeps the message secret

- Bob does not know:
  - Did Alice address this message to Bob
  - Who sent this copy of the message
  - When the message was sent
  - Who else knows the plaintext
Quick Aside: Digital Signatures

- What Does Signing Mean?
  - Signing is a mechanical operation that has *no meaning* in itself.

- What cryptography promises:
  - Only someone who knows the private key can create a signature that verifies using the corresponding public key

- Meaning of a digital signature is a matter of convention
  - Code signing: signer attests software is authorized to be installed
  - Email signing: signer attests she wrote message
  - Certificate signing: (coming up next!)

- Both signer and verifier need to **agree** on meaning and trust that the meaning is enforced locally
Using Cryptography

- Alice wants to send (a plaintext) $m$ to Bob, via a channel that is controlled by Eve.
- Alice and Bob know each other’s public keys. (assume pk crypto)
- Goal: Alice and Bob establish a secure “pipe” (e.g., like https or ssh)
  - Sign and encrypt all content
- If successful, Eve cannot see plaintext contents inside the pipe, or modify them without detection.
Using Cryptography

- Alice and Bob got secrecy + integrity + authenticity and everyone lived happily ever after, right?
- Let’s try to understand exactly how we might achieve this, and the problems along the way
Public Key Infrastructure (PKI)
Using Cryptography

- Alice wants to send (a plaintext) $m$ to Bob, via a channel that is controlled by Eve.
- Alice and Bob know each other’s public keys.
- Alice and Bob establish a secure “pipe”.

[Diagram showing Alice, Eve, and Bob with a secure pipe between Alice and Bob, indicating Eve is outside the secure channel.]
Getting Public Keys

- Alice and Bob need a way to get each other’s public key.
- Alice can send an unencrypted message to Bob:
  - “Hey, send me your public key. Here’s mine.”
- Bob sends Alice his public key.
- They communicate securely ever after?
Getting Public Keys

- What they want to happen

- What might happen instead
Getting Public Keys

- If Eve has person-in-the-middle capability, she can impersonate Alice to Bob and Bob to Alice.
  - Eve becomes invisible gateway between them.
  - Alice and Bob have no idea Eve is there.
Getting Public Keys

- Alice and Bob need a way to know that each has the real public key of the other.
- Ideal solution: Alice and Bob meet in person and exchange public keys
- Roughly equivalent: Alice and Bob meet in person and exchange public key fingerprints
  - Key fingerprint: cryptographic hash of public key
  - Public key itself can be sent in the open
  - (aside: this is what Signal does)
Getting Public Keys

- Problem with ideal:
  - We are back to pair-wise key establishment
  - Alice and Bob need to meet
  - Impractical to meet and verify key of everyone you talk to

- Many security problems can be solved with a trusted third party
Getting Public Keys

- Using a trusted intermediary
  - Alice and Bob have already exchanged keys with Charlie
  - Charlie sends signed message with Alice’s key to Bob
  - Charlie sends signed message with Bob’s key to Alice
  - Alice and Bob trust Charlie to send the real public keys
  - Alice and Bob now have each other’s public key
Getting Public Keys

- But every transaction is centralized through Charlie! We can do better...
- Charlie creates a certificate:
  - “I, Charlie, verified that Alice’s key is …”
- Charlie signs the message and gives it to Alice
  - Alice now has certificate attesting to her public key
- Alice sends Bob her public key and Charlie’s certificate
- Bob verifies the signature on certificate
- Bob trusts Charlie, accepts public key from Alice
Who is Charlie?

- Two common models:
  - PGP: Charlie is any other person you trust.
  - Almost everywhere else: Charlie is a *Certificate Authority*. 
PGP Web of Trust

- Pretty Good Privacy (PGP) is an application (and associated protocols) used for signing, encrypting, and decrypting texts, e-mails, files, directories, etc.

- PGP allows one user to attest to the trustworthyness of another user’s public key — *key signing*
  - PGP does not use the term “certificate”, but that’s because its old...
  - Public key has set of attestation signatures (certificates)

- A user can indicate how much she trusts another user’s signature on a key
PGP Web of Trust

- Alice’s signature on Bob’s PGP key means Alice claims that this is really Bob’s key (and *ideally* has verified this)
  - Email address and name associated with key are really his

- Other people who trust Alice can use her signature on Bob’s key to be sure it is Bob’s key

- How to decide?

https://xkcd.com/364/
Certificate Authorities

- An alternative to PGP-like web of trust is to rely on centralized Certificate Authorities: trusted signers of public keys.
- CA model used to sign certificates used on Web.
- Your browser has a set of public keys of trusted CAs.
  - Who makes this list?
  - How many CAs are on the list?
  - Who are these CAs?
## Certificate Authorities

<table>
<thead>
<tr>
<th>Certificate Authority</th>
<th>Name of Certificate Authority</th>
<th>Valid From (GMT)</th>
<th>Valid To (GMT)</th>
<th>Public Key Algorithm</th>
<th>Signature Key Algorithm</th>
<th>Trust Bits</th>
<th>Issuer</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate Authority 1</td>
<td>Name of Certificate Authority 1</td>
<td>2023-01-01 00:00:00</td>
<td>2023-12-31 23:59:59</td>
<td>RSA</td>
<td>RSA</td>
<td>1</td>
<td>Issuer 1</td>
<td>Country 1</td>
</tr>
<tr>
<td>Certificate Authority 2</td>
<td>Name of Certificate Authority 2</td>
<td>2023-01-01 00:00:00</td>
<td>2023-12-31 23:59:59</td>
<td>ECC</td>
<td>ECC</td>
<td>2</td>
<td>Issuer 2</td>
<td>Country 2</td>
</tr>
<tr>
<td>Certificate Authority 3</td>
<td>Name of Certificate Authority 3</td>
<td>2023-01-01 00:00:00</td>
<td>2023-12-31 23:59:59</td>
<td>ECDSA</td>
<td>ECDSA</td>
<td>3</td>
<td>Issuer 3</td>
<td>Country 3</td>
</tr>
</tbody>
</table>

Count: 153
Certificate Authorities (2021)

- Mozilla
  - ~143 root certificates
  - https://wiki.mozilla.org/CA/Included_Certificates

- iOS
  - ~203 root certificates

- Microsoft
  - ~417 root certificates
  - http://aka.ms/RootCert
Certificate Authorities

- Certificate semantics:
  - Subject (name, domain)
  - Issuing CA
  - Validity period
  - Limitations on use
Using Certificates: Transport Layer Security (TLS)

- This is what makes https:// work
- "When secured by TLS, connections between a client (e.g., a web browser) and a server (e.g., wikipedia.org) have one or more of the following properties:
  - The connection is private (or secure) because symmetric cryptography is used to encrypt the data transmitted…
  - The identity of the communicating parties can be authenticated using public-key cryptography…
  - The connection ensures integrity because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission."

- Details of protocol are complex, but the basic idea isn’t:
  - Browser gets and verifies server’s certificate, and extracts PK
  - Use PK to encrypt random symmetric session key
  - Use session key to encrypt session and to key HMAC for integrity

SSL Client

1. "client hello"
   - Cryptographic information
2. "server hello"
   - CipherSuite
   - Server certificate
   - "client certificate request" (optional)
3. Verify server certificate. Check cryptographic parameters
4. Client key exchange
   - Send secret key information (encrypted with server public key)
5. Send client certificate
6. Verify client certificate (if required)
7. Client “finished”
8. Server “finished”
9. Exchange messages
   - (encrypted with shared secret key)

SSL Server
A quick step back...

- Using TLS (i.e., https) what security is being claimed?
  - Authenticity
    - That the server is who they say they are (e.g., www.amazon.com)
    - That the client is who they say they are?
  - Confidentiality
    - That no one but the client and server can read the messages sent between them
  - Integrity
    - That no one can alter messages between client and server without being detected

- What are we depending on?
  - Crypto works
  - The attestations of the CAs are reliable and their services are secure
Certificate Authorities

- Which CA can issue a certificate for mycompany.com?
- For fbi.gov?

- How do site owners prove they are authorized to get a certificate for example.com?
  - Traditionally
    - Provide tangible evidence of ownership (e.g., corp documents via FAX) and pay a fee
  - Today, a new model dominates
    - Provide evidence of control over site domain name
Let’s Encrypt

Web Server
Admin
Software

example.com

Put ed98 at https://example.com/8303
Sign 9cf0b331

https://letsencrypt.org/how-it-works/
Let’s Encrypt

[Diagram showing the process of Let’s Encrypt]

https://letsencrypt.org/how-it-works/
Certificate Authorities

- What if we take a **Trusted Third Party** and combine it with **another Layer of Indirection**?

- **Certificate Hierarchy**

- **Root CA** signs keys for **Intermediate CAs**, which in turn sign keys for users (or other intermediate CAs)
Certificate Authorities

- Certificate hierarchy for ucsd.edu
Aside: other uses of certificates

- Certificates also used in code signing
  - https://source.android.com/security/apksigning/
  - https://docs.microsoft.com/en-us/windows-hardware/drivers/install/driver-signing

- Who is the CA?

- What is the meaning of the signature?
  - Alice released this app?
  - Alice authorizes this app to run?
  - Alice authorizes this app to access privileged resources?
Certificate Revocation

- What happens if someone steals your private key?
  - They can impersonate you and read messages encrypted to you

- Certificate expiration helps with this but not enough

- CA and PGP PKIs support revocation
  - Owner says: “I, Alice, revoke my public key ... do not use it.”
  - Signs revocation with her private key
  - Others can verify Alice’s signature, stop using key
Certificate Revocation

- How does Bob know if Alice’s key has been revoked?
- Bob asks Alice: “Has your key been revoked?”
- Alice sends signed message: “No.”
- Does not work: if Alice’s key has been compromised, then Eve could have forged the message “No.”
- Availability of trusted revocation list critical
Certificate Revocation

- In PGP model, only Alice can revoke her own key
  - If Alice loses her private key, she can’t revoke
    - Do not lose private PGP key
  - Option: generate revocation transaction with key, store in secure place

- In CA model, Alice asks CA to revoke certificate
  - Alice does not need private key to do this, can authenticate herself through other means
Certificate Revocation

- Two Mechanisms: CRL and OCSP

- **Certificate Revocation List (CRL):**
  - Certificate says where to get CRL
  - Clients periodically download updated CRLs
  - What if CRL server is down?
Certificate Revocation

- Two Mechanisms: CRL and OCSP
- **Online Certificate Status Protocol (OCSP):**
  - Query CA about status of cert before trusting it
  - “You said I can trust this key, but are you still sure?”
- OCSP Stapling
  - Server includes recent OCSP status (signed by CA)
- Aside: Certificate Pinning
  - Remember which certificate was used for a particular domain and raise an alert if a different one is used later
- Visit [https://revoked-isrgrootx1.letsencrypt.org/](https://revoked-isrgrootx1.letsencrypt.org/) with your browser
Some additional complexity: CDNs

- HTTPS is secured by TLS
Some additional complexity: CDNs

- HTTPS is secured by TLS
Content Delivery Networks (CDNs)

- CDN: geographically distributed network of proxy servers
  - Cache static content closer to the requester
  - Improve latency
  - Decrease network congestion
  - Improve reliability and availability
    - DDOS protection
  - Cloudflare, Akamai, CloudFront, etc

- Mess up our nice security abstractions
  - Now Alice deliberately wants her CDN to impersonate her to Bob!
Content Delivery Networks (CDNs)

- Bob wants to connect to www.fbi.gov
- Bob’s browser attempts to get the corresponding IP address via DNS
- Because FBI used Cloudflare CDN, DNS resolves to a Cloudflare server
- But Bob’s browser thinks it’s talking to fbi.gov
- Cloudflare needs to convince Bob’s browser that it’s really FBI
Content Delivery Networks (CDNs)

- Deputized via “Subject Alternate Name” field
  - “Yeah, I’m cloudflaressl.com, but I’m authorized to communicate on behalf fbi.gov”

- Who decides whether a CDN can get a given Subject Alternate Name in its cert?
### cse.ucsd.edu

- **Issued by:** InCommon RSA Server CA
- **Expires:** Monday, January 4, 2021 at 3:59:59 PM Pacific Standard Time
- **This certificate is valid**

#### Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject Name</td>
<td></td>
</tr>
<tr>
<td>Country</td>
<td>US</td>
</tr>
<tr>
<td>Postal Code</td>
<td>92093</td>
</tr>
<tr>
<td>State/Province</td>
<td>CA</td>
</tr>
<tr>
<td>Locality</td>
<td>La Jolla</td>
</tr>
<tr>
<td>Street Address</td>
<td>9500 Gilman Drive</td>
</tr>
<tr>
<td>Organization</td>
<td>University of California, San Diego</td>
</tr>
<tr>
<td>Organizational Unit</td>
<td>UCSD</td>
</tr>
<tr>
<td>Common Name</td>
<td>cse.ucsd.edu</td>
</tr>
<tr>
<td>Issuer Name</td>
<td></td>
</tr>
<tr>
<td>Country</td>
<td>US</td>
</tr>
<tr>
<td>State/Province</td>
<td>MI</td>
</tr>
<tr>
<td>Locality</td>
<td>Ann Arbor</td>
</tr>
<tr>
<td>Organization</td>
<td>Internet2</td>
</tr>
<tr>
<td>Organizational Unit</td>
<td>InCommon</td>
</tr>
<tr>
<td>Common Name</td>
<td>InCommon RSA Server CA</td>
</tr>
<tr>
<td>Common Name</td>
<td>cse.ucsd.edu</td>
</tr>
<tr>
<td>Serial Number</td>
<td>36 F6 DC 47 6F 09 25 8E 94 EF BF 36 65 4F</td>
</tr>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Signature Algorithm</td>
<td>SHA-256 with RSA Encryption</td>
</tr>
<tr>
<td></td>
<td>(1.2.840.113549.1.1.11)</td>
</tr>
</tbody>
</table>
Who are we trusting?
Who is this cert for?

Who are we trusting?
<table>
<thead>
<tr>
<th>Key ID</th>
<th>1E 05 A3 77 8F 8C 96 E2 5B 87 4B A6 B4 86 AC 71 00 0C E7 3B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension</td>
<td>Subject Alternative Name (2.5.29.17)</td>
</tr>
<tr>
<td>Critical</td>
<td>NO</td>
</tr>
<tr>
<td>DNS Name</td>
<td>cse.ucsd.edu</td>
</tr>
<tr>
<td>DNS Name</td>
<td>cs.ucsd.edu</td>
</tr>
<tr>
<td>DNS Name</td>
<td>www-cs.ucsd.edu</td>
</tr>
<tr>
<td>DNS Name</td>
<td><a href="http://www.cs.ucsd.edu">www.cs.ucsd.edu</a></td>
</tr>
<tr>
<td>DNS Name</td>
<td><a href="http://www.cs.ucsd.edu">www.cs.ucsd.edu</a></td>
</tr>
</tbody>
</table>

Who is this cert for?

| Extension | Certificate Policies (2.5.29.32)                           |
| Critical  | NO                                                          |
| Policy ID #1 | (1.3.6.1.4.1.5923.1.4.3.1.1)                        |
| Qualifier ID #1 | Certification Practice Statement (1.3.6.1.5.5.72.1) |
| CPS URI   | https://www.incommon.org/cert/repository/cps_ssl.pdf      |
| Policy ID #2 | (2.23.140.1.2.2)                                          |

| Extension | CRL Distribution Points (2.5.29.31)                         |
| Critical  | NO                                                          |
| URI       | http://crl.incommon-rsa.org/InCommonRSAserverCA.crl       |

| Extension | Certificate Authority Information Access (1.3.6.1.5.5.71.1) |
| Critical  | NO                                                          |
| Method #1 | CA Issuers (1.3.6.1.5.5.7.48.2)                             |
| URI       | http://cert.usertrust.com/InCommonRSAserverCA_2.crt        |
| Method #2 | Online Certificate Status Protocol (1.3.6.1.5.5.7.48.1)    |
| URI       | http://ocsp.usertrust.com                                  |
CSE’s pub key info

Issuer Name
Country US
State/Province MI
Locality Ann Arbor
Organization Internet2
Organizational Unit InCommon
Common Name InCommon RSA Server CA
Serial Number 36 F8 DC 47 6F 09 25 8E 94 EF BF 36 65 4F EB 98
Version 3
Signature Algorithm SHA-256 with RSA Encryption
(1.2.840.113549.1.1.1)
Parameters None

Not Valid Before Thursday, January 4, 2018 at 4:00:00 PM Pacific Standard Time

Public Key Info
Algorithm RSA Encryption (1.2.840.113549.1.1.1)
Parameters None
Public Key 256 bytes: FA F9 1A 08 92 86 9C 7B ...
Exponent 65537
Key Size 2048 bits
Key Usage Encrypt, Verify, Wrap, Derive
Signature 256 bytes: 6F 62 36 46 B7 43 28 04 ...

Extension Key Usage (2.5.29.15)
Critical YES
Usage Digital Signature, Key Encipherment
<table>
<thead>
<tr>
<th>Key ID</th>
<th>1E 05 A3 77 8F 6C 96 E2 5B 87 4B A6 B4 86 AC 71 00 0C E7 3B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension</td>
<td>Subject Alternative Name (2.5.29.17)</td>
</tr>
<tr>
<td>Critical</td>
<td>NO</td>
</tr>
<tr>
<td>DNS Name</td>
<td>cse.ucsd.edu</td>
</tr>
<tr>
<td>DNS Name</td>
<td>cs.ucsd.edu</td>
</tr>
<tr>
<td>DNS Name</td>
<td>www-cs.ucsd.edu</td>
</tr>
<tr>
<td>DNS Name</td>
<td>www-cse.ucsd.edu</td>
</tr>
<tr>
<td>DNS Name</td>
<td><a href="http://www.cs.ucsd.edu">www.cs.ucsd.edu</a></td>
</tr>
<tr>
<td>DNS Name</td>
<td><a href="http://www.cse.ucsd.edu">www.cse.ucsd.edu</a></td>
</tr>
<tr>
<td>Extension</td>
<td>Certificate Policies (2.5.29.32)</td>
</tr>
<tr>
<td>Critical</td>
<td>NO</td>
</tr>
<tr>
<td>Policy ID #1</td>
<td>(1.3.6.1.4.1.5923.14.3.1.1)</td>
</tr>
<tr>
<td>Qualifier ID#1</td>
<td>Certification Practice Statement (1.3.6.1.5.5.7.2.1)</td>
</tr>
<tr>
<td>CPS URI</td>
<td><a href="https://www.incommon.org/cert/repository/cps_ssl.pdf">https://www.incommon.org/cert/repository/cps_ssl.pdf</a></td>
</tr>
<tr>
<td>Policy ID #2</td>
<td>(2.23.140.1.2.2)</td>
</tr>
</tbody>
</table>

| Extension   | CRL Distribution Points (2.5.29.31)                         |
| Critical    | NO                                                          |
| URI         | http://crl.incommon-rsa.org/InCommonRASAServerCA.crl       |
| Extension   | Certificate Authority Information Access                    |
| Critical    | NO                                                          |
| Method #1   | CA Issuers (1.3.6.1.5.5.7.48.2)                            |
| URI         | http://crt.usertrust.com/InCommonRASAServerCA_2.crt        |
| Method #2   | Online Certificate Status Protocol                          |
| Critical    | NO                                                          |
| URI         | http://ocsp.usertrust.com                                  |

Where we should check for revocation information
Another approach: Secure Shell (SSH)

- "Secure Shell (SSH) provides a secure channel over an unsecured network, connecting an SSH client application with an SSH server. Common applications include remote command-line login and remote command execution, but any network service can be secured with SSH."

![Diagram of SSH process]

1. Client initiates the connection by contacting server
2. Sends server public key
3. Negotiate parameters and open secure channel
4. User login to server host operating system
Another approach: Secure Shell (SSH)

- No trusted authorities
  - Trust on First Use

- Basically certificate pinning
Summary

- Public key crypto is a powerful tool
  - Underlies https, ssh, virtually all software updates, etc...
  - But doesn’t solve the key distribution problem

- Certificate authorities (CA) occupy key (and trusted) role
  - Third-party attestation of identity or access
  - Have become hacking targets
    - 2011: Comodo & Diginotar issued fraudulent certs for Hotmail, Gmail, Skype, Yahoo Mail, Firefox...
    - 2013: TurkTrust issued cert for gmail
    - 2014: Indian Nic issued certs for Google and Yahoo!
    - 2016: WoSign issued cert for GitHub

- Ongoing effort to police CAs – Certificate Transparency
  - Make a public, searchable log of every new certificate minted... google can go check if anyone else got a cert that covers google.com or gmail.com
Next time

- Starting Web Security