Using Cryptography
CMSC 414

October 16, 2017
Digital Certificates

Recall:

\[ K_{\text{pub}} = (n, e) \iff \text{This is an RSA public key} \]

*How do we know who this is for?*

Need to bind *identity* to a *public key*

We can do this using a **Digital Certificate**

This is a binding that has been *signed* by some third party

- Often a *trusted third party*
- Sometimes *self-signed*
What’s in a Digital Certificate?

$ openssl x509 -in mmarsh.req.cert -noout -text
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 1428829381 (0x552a34c5)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=CA, OU=CA, O=soucis
    Validity
      Not Before: May 5 20:29:24 2017 GMT
      Not After : Jan 30 20:29:24 2020 GMT
    Subject: O=soucis, OU=user, CN=Michael Marsh
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (4096 bit)
        Modulus (4096 bit):
            e0:ff:f7
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Authority Key Identifier:

    X509v3 Subject Key Identifier:
      Signature Algorithm: sha256WithRSAEncryption
          ...
          08:ab:ee:29
How Do We Know if This is Valid?

Certificates have an **Issuer**

The issuer signs the certificate, with an **validity period**

We then need the certificate for the **issuer**

How do we know *that* certificate is valid...?

[Note: Slides marked with an asterisk contain cynicism]
Certification Authorities (*)

A Root Certification Authority (root CA) is a Root of Trust
That means we trust it implicitly

CAs issue certificates to users, servers, or intermediate CAs
Usually, a root CA only certifies intermediate CAs

What we end up with is a Certificate Chain

Your browser is pre-loaded with a lot of root CAs, and you trust them, whether you really do or should
What Does a CA Do? (*)

A CA *should* check that the name in a *certificate request* matches the principal *sending* the request.

If the certificate is for a service, the CA *should* check that this is, in fact, the public key for that service, and the requester *actually represents* the service.

This usually works, but...

*Rogue CAs* might sign bogus certificates.

Some CAs only provide this level of verification for higher-paying customers, and issue *less-secure* certificates for others.
$ openssl req -in ~/Downloads/XXXX.req -noout -text
Certificate Request:
  Data:
    Version: 0 (0x0)
    Subject: O=soucis, OU=user, CN=XXXX
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (4096 bit)
        Modulus (4096 bit):
          00:a8:fe:4a:3e:d0:4e:d1:ad:93:b3:76:fe:c1:78:
          ...
          a1:0d:8d
        Exponent: 65537 (0x10001)
    Attributes: a0:00
    Signature Algorithm: sha1WithRSAEncryption
      ...
      08:e9:1e:1d:83:1c:2b:32

Unlike the certificate, this is signed with the subject’s private key, not the issuer’s
PGP, GPG, and the Web of Trust

*Pretty Good Privacy, GNU Privacy Guard*

Similar idea, but *without a root of trust*

Instead of relying on a third party we don’t know…

1. Alice and Bob meet (possibly at a *key-signing party*)
2. They exchange public keys
3. They sign those keys and hand each other the certs

Anyone with a cert for Alice can validate Bob’s cert from Alice

If Bob has a cert signed by Alice, we sometimes refer to him as “Alice’s Bob” when he uses this cert

Forms a **Web of Trust** that can connect users

Requires ability to form a *trusted path* from verifier to subject
How Do We Use Certificates? (*)

CA-based certificates are commonly used on the web for encrypted connections (HTTPS)

PGP-based certificates are “commonly” used in email for signing messages

Requires the mail client to have a PGP/GPG-capable extension

![Imagery showing how to verify PGP-signed messages]
Public Key Infrastructure (*)

Certificates are only one component

We also need (this is not the precise technical specification)

- Distribution mechanism for CA certificates
- Revocation Lists

A Revocation is a CA-issued statement that a certificate is no longer valid (other than due to expiry)

⇒ This can be due to issues like private key disclosure

⇒ Your browser almost certainly never checks for these

With these, we have a Public Key Infrastructure (PKI)
Group Exercise 1

In your crypto-exercises repository run
git pull upstream master to merge changes to the upstream repository that you initially forked.

“Using Crypto Exercise 1” contains instructions for setting up and using your own OpenSSL-based Certification Authority.
Symmetric and Asymmetric Crypto

Notation

To keep things sane, let’s define

- \( k_{AB} \) a *secret key* shared by \( A \) and \( B \)
- \( K_A \) the *public key* of \( A \)
- \( k_A \) the *private key* of \( A \)

- \( \{P\}_{k_{AB}} \) symmetric-key encryption of plaintext \( P \) with \( k_{AB} \)
- \( E_A(P) \) asymmetric-key encryption of plaintext \( P \) with \( K_A \)
- \( D_A(C) \) asymmetric-key decryption of ciphertext \( C \) with \( k_A \)
- \( S_A(M) \) message \( M \) signed with \( k_A \)

The short version:

- \( \{P\}_k \) is symmetric-key encryption
- \( E(), D(), \) and \( S() \) are asymmetric-key operations
# Symmetric and Asymmetric Crypto

## Key Sizes and Security

From NIST publication SP 800-57 Part 1 Rev. 4 (January 2016), by Elaine Barker

<table>
<thead>
<tr>
<th>Security Strength</th>
<th>Symmetric Cryptosystem</th>
<th>RSA Key Length</th>
<th>ECC Key Length</th>
<th>Hash Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>2-key 3DES</td>
<td>1024</td>
<td>160–223</td>
<td>SHA-1</td>
</tr>
<tr>
<td>112</td>
<td>3-key 3DES</td>
<td>2048</td>
<td>224–255</td>
<td>SHA-224</td>
</tr>
<tr>
<td>128</td>
<td>AES-128</td>
<td>3072</td>
<td>256–383</td>
<td>SHA-256</td>
</tr>
<tr>
<td>192</td>
<td>AES-192</td>
<td>7680</td>
<td>384–511</td>
<td>SHA-384</td>
</tr>
<tr>
<td>256</td>
<td>AES-256</td>
<td>15360</td>
<td>512+</td>
<td>SHA-512</td>
</tr>
</tbody>
</table>
Symmetric and Asymmetric Crypto

Performance

3DES and AES can be implemented in \textit{hardware} $\Rightarrow$ very fast

software implementations slower, but still fairly fast using lookup tables

RSA, ElGamal, ECC must be implemented in software, and private key operations are fairly slow
Symmetric and Asymmetric Crypto
Getting the Best of Both Worlds

@A:
1. select random nonce $n_A$
2. $M_1 = E_B(n_A)$
3. $s_1 = S_A(M_1)$

$A \rightarrow B: \langle M_1, s_1 \rangle$

@B:
1. $V_A(M_1, s_1)$
2. compute $n_A = D_B(M_1)$
3. select symmetric key $k_{AB}$
4. compute $n_B = n_A \oplus k_{AB}$
5. $M_2 = E_A(n_B)$
6. $s_2 = S_B(M_2)$

$B \rightarrow A: \langle M_2, s_2 \rangle$

@A:
1. $V_B(M_2, s_2)$
2. compute $n_B = D_A(M_2)$
3. compute $k_{AB} = n_B \oplus n_A$
Group Encryption

Sometimes we want to send encrypted data to a large number of principals

▶ Videoconferencing

▶ Premium cable channels

▶ Subscription services

Public keys ⇒ too many, takes too long to compute them all

Secret keys ⇒ too many, expensive key exchange (though doesn’t have to be done often)
Group Encryption

Both result in one of

- lots of individual messages (can't take advantage of efficient delivery mechanisms)

- extremely long messages that contain lots of copies of encryption

- extremely long messages that contain one encrypted message and lots of encryptions of the message key

Must be a better way...
Group Keys

We would like to have a \textit{single} symmetric key for a stream, that can be shared with everyone

This could be distributed by a central authority individually to every user, or agreed upon/determined by the set of users.

A common way to do either of these is with \textbf{Key Trees}.

There are many ways to do this, all of which share common issues:

- Key distribution/agreement needs to be fairly efficient
- Only legitimate group members should be able to learn the key
- Keys must be changed when members \textit{join} or \textit{leave} the group
Group Exercise 2

Now you’re going to combine public-key and secret-key cryptography for yourself. See “Using Crypto Exercise 2” in the README. The instructions assume that you’re using python, but you may use any language you like.