Cryptography: Asymmetric Cryptography (finish)

Fall 2016

Adam (Ada) 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 ...
Announcements

• Lab 1 is due Monday at 5pm!
Diffie-Hellman: Conceptually

Alice

+ Common paint

= Secret colours

Public transport

(assume that mixture separation is expensive)

+ =

Bob

+ Common paint

= Secret colours

= Common secret

[from Wikipedia]
Do Q1 on your worksheet
Do Q2 on your worksheet
Do Q3 on your worksheet
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 $\mathbb{Z}_p^*$
    - $\mathbb{Z}_p^*$=\{1, 2 ... p-1\}; $\forall a \in \mathbb{Z}_p^*$ $\exists i$ such that $a=g^i \mod p$
    - Modular arithmetic: numbers “wrap around” after they reach p

Pick secret, random X

<table>
<thead>
<tr>
<th>Alice</th>
<th>$g^x \mod p$</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$g^y \mod p$</td>
<td></td>
</tr>
</tbody>
</table>

Pick secret, random Y

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

Compute $k=(g^x)^y = g^{xy} \mod p$
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
    - If you could take the discrete logarithm efficiently, you could break Diffie Hellman by learning $k=g^{xy} \mod p$
    - This is *not* enough for Diffie-Hellman to be secure! Why? (Q5)


**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
Requirements for Public Key Encryption

• **Key generation:** computationally easy to generate a pair (public key $PK$, private key $SK$)

• **Encryption:** given plaintext $M$ and public key $PK$, easy to compute ciphertext $C = E_{PK}(M)$

• **Decryption:** given ciphertext $C = E_{PK}(M)$ and private key $SK$, easy to compute plaintext $M$
  – Infeasible to learn anything about $M$ from $C$ without $SK$
  – Trapdoor function: $\text{Decrypt}(SK, E_{PK}(M)) = M$
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 $\varphi(ab) = \varphi(a) \varphi(b)$
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 \( \varphi(ab) = \varphi(a) \varphi(b) \)

• Euler’s theorem: if \( a \in \mathbb{Z}_n^* \), then \( a^{\varphi(n)} \equiv 1 \pmod{n} \)
  \( \mathbb{Z}_n^* \): integers relatively prime to \(n\)
RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

• Key generation:
  – Generate large primes \( p, q \)
    • Say, 1024 bits each (need primality testing, too)
  – Compute \( n=pq \) and \( \varphi(n)=(p-1)(q-1) \)
  – Choose small \( e \), relatively prime to \( \varphi(n) \)
    • Typically, \( e=3 \) or \( e=2^{16}+1=65537 \)
  – Compute unique \( d \) such that \( ed = 1 \mod \varphi(n) \)
    • Modular inverse: \( d = e^{-1} \mod \varphi(n) \)
  – Public key = \((e,n)\); private key = \((d,n)\)

• Encryption of \( m \): \( c = m^e \mod n \)

• Decryption of \( c \): \( c^d \mod n = (m^e)^d \mod n = m \)
Why RSA Decryption Works

e \cdot d = 1 \mod \varphi(n), \text{ thus } e \cdot d = 1 + k \cdot \varphi(n) \text{ for some } k

Let } m \text{ be any integer in } \mathbb{Z}_n^* \text{ (not all of } \mathbb{Z}_n)\n\begin{align*}
c^d \mod n &= (m^e)^d \mod n = m^{1+k \cdot \varphi(n)} \mod n \\
&= (m \mod n) \times (m^k \cdot \varphi(n) \mod n)
\end{align*}

Recall: Euler’s theorem: if } a \in \mathbb{Z}_n^*, \text{ then } a^{\varphi(n)} = 1 \mod n

\begin{align*}
c^d \mod n &= (m \mod n) \times (1 \mod n) \\
&= m \mod n
\end{align*}

Proof omitted: True for all } m \text{ in } \mathbb{Z}_n, \text{ not just } m \text{ in } \mathbb{Z}_n^*
Why is RSA Secure?

• **RSA problem:** given $c$, $n=pq$, and $e$ such that $\gcd(e, \varphi(n))=1$, find $m$ such that $m^e \equiv c \mod n$
  
  – In other words, recover $m$ from ciphertext $c$ and public key $(n,e)$ by taking $e^{\text{th}}$ root of $c$ modulo $n$
  
  – There is no known efficient algorithm for doing this

• **Factoring problem:** given positive integer $n$, find primes $p_1, \ldots, p_k$ such that $n=p_1^{e_1}p_2^{e_2}\ldots p_k^{e_k}$

• If factoring is easy, then RSA problem is easy (knowing factors means you can compute $d = \text{inverse of } e \mod (p-1)(q-1)$)
  
  – 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

In practice, OAEP is used: 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)

• 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
  – 1024+ 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... )
Problem: How does Alice know that the public key she received is really Bob’s public key?
Threat: Man-In-The-Middle (MITM)
Distribution of Public Keys

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - \( \text{sig}_{CA}(\text{"Bob"}, \text{PK}_B) \)
- 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

Keychain Access

Click to unlock the System Roots keychain.

Keychains

- login
- Local Items
- System
- System Roots

Category

- All Items
- Passwords
- Secure Notes
- My Certificates
- Keys
- Certificates

Apple Root CA

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

This certificate is valid

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminCA-CD-T01</td>
<td>certificate</td>
<td>Jan 25, 2016, 4:36:19 AM</td>
</tr>
<tr>
<td>AffirmTrust Commercial</td>
<td>certificate</td>
<td>Dec 31, 2030, 6:06:06 AM</td>
</tr>
<tr>
<td>AffirmTrust Networking</td>
<td>certificate</td>
<td>Dec 31, 2030, 6:08:24 AM</td>
</tr>
<tr>
<td>AffirmTrust Premium</td>
<td>certificate</td>
<td>Dec 31, 2040, 6:10:36 AM</td>
</tr>
<tr>
<td>AffirmTrust Premium ECC</td>
<td>certificate</td>
<td>Dec 31, 2040, 6:20:24 AM</td>
</tr>
<tr>
<td>America Online...cations Authority 1</td>
<td>certificate</td>
<td>Nov 19, 2037, 12:43:00 PM</td>
</tr>
<tr>
<td>America Online...cations Authority 2</td>
<td>certificate</td>
<td>Sep 29, 2037, 7:08:00 AM</td>
</tr>
<tr>
<td>Apple Root CA</td>
<td>certificate</td>
<td>Feb 9, 2035, 1:40:36 PM</td>
</tr>
<tr>
<td>Apple Root CA - G2</td>
<td>certificate</td>
<td>Apr 30, 2039, 11:10:09 AM</td>
</tr>
<tr>
<td>Apple Root CA - G3</td>
<td>certificate</td>
<td>Apr 30, 2039, 11:19:06 AM</td>
</tr>
<tr>
<td>Apple Root Certificate Authority</td>
<td>certificate</td>
<td>Feb 9, 2025, 4:18:14 PM</td>
</tr>
<tr>
<td>Application CA G2</td>
<td>certificate</td>
<td>Mar 31, 2016, 7:59:59 AM</td>
</tr>
<tr>
<td>ApplicationCA</td>
<td>certificate</td>
<td>Dec 12, 2017, 7:00:00 AM</td>
</tr>
</tbody>
</table>
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
    • \(\text{sig}_{\text{Verisign}}(\text{“AnotherCA”}, \text{PK}_{\text{AnotherCA}}), \text{sig}_{\text{AnotherCA}}(\text{“Alice”}, \text{PK}_A)\)
  – What happens if root authority is ever compromised?
You encounter this every day...

SSL/TLS: Encryption & authentication for connections

(More on this later!)
Example of a Certificate

<table>
<thead>
<tr>
<th><strong>Subject Name</strong></th>
<th>US</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Country</strong></td>
<td>US</td>
</tr>
<tr>
<td><strong>State/Province</strong></td>
<td>California</td>
</tr>
<tr>
<td><strong>Locality</strong></td>
<td>Mountain View</td>
</tr>
<tr>
<td><strong>Organization</strong></td>
<td>Google Inc</td>
</tr>
<tr>
<td><strong>Common Name</strong></td>
<td>*.google.com</td>
</tr>
</tbody>
</table>

**Issuer Name**

<table>
<thead>
<tr>
<th><strong>Country</strong></th>
<th>US</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Organization</strong></td>
<td>Google Inc</td>
</tr>
<tr>
<td><strong>Common Name</strong></td>
<td>Google Internet Authority G2</td>
</tr>
</tbody>
</table>

**Serial Number**

| 608271139101222858 |

**Version**

| 3 |

**Certificate Details**

- **Issued by:** Google Internet Authority G2
- **Expires:** Monday, July 6, 2015 at 5:00:00 PM Pacific Daylight Time
- **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

Diagram showing the structure of an X.509 certificate, with sections for Version, Certificate Serial Number, Signature algorithm identifier, Period of validity, Subject's public key info, Subject Name, Issuer Unique Identifier, Subject Unique Identifier, Extensions, and Signature with algorithms parameters encrypted.
Many Challenges...

- 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...
# Colliding Certificates

<table>
<thead>
<tr>
<th>serial number</th>
<th>chosen prefix (difference)</th>
<th>serial number</th>
</tr>
</thead>
<tbody>
<tr>
<td>validity period</td>
<td></td>
<td>validity period</td>
</tr>
<tr>
<td>real cert domain name</td>
<td>Hash to the same MD5 value!</td>
<td>rogue cert domain name</td>
</tr>
<tr>
<td>real cert RSA key</td>
<td>collision bits (computed)</td>
<td></td>
</tr>
<tr>
<td>X.509 extensions</td>
<td>identical bytes (copied from real cert)</td>
<td>X.509 extensions</td>
</tr>
<tr>
<td>signature</td>
<td></td>
<td>signature</td>
</tr>
</tbody>
</table>

Valid for both certificates!

[Sotirov et al. “Rogue Certificates”]
Attacking CAs

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)

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

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.
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

https://keybase.io/

Franzi Roesner
@franziroesner

Verifying myself: I am franziroesner on Keybase.io. 5YGG83pd-i4zvxl2dDUHDMrOouRG386Q_tZ / keybase.io/franziroesner/…

11:14 PM - 19 Nov 2014
Cryptography Summary

• **Goal:** Privacy
  – **Symmetric keys:**
    • One-time pad, Stream ciphers
    • Block ciphers (e.g., DES, AES) \(\rightarrow\) modes: EBC, CBC, CTR
  – Public key crypto (e.g., Diffie-Hellman, RSA)

• **Goal:** Integrity
  – MACs, often using hash functions (e.g., MD5, SHA-256)

• **Goal:** Privacy and Integrity
  – Encrypt-then-MAC

• **Goal:** Authenticity (and Integrity)
  – Digital signatures (e.g., RSA, DSS)