Lecture 11 – Public key Crypto

Ryan Cunningham
University of Illinois
ECE 422/CS 461 – Summer 2017
Review: Integrity

Problem: Sending a message over an untrusted channel without being changed

Provably-secure solution: Random function

Practical solution:

Pseudorandom function (PRF)

Input: arbitrary-length \( k \)  
Output: fixed-length value

Secure if practically indistinguishable from a random function, unless know \( k \)

Real-world use: Message authentication codes (MACs) built on cryptographic hash functions

Popular example: \( \text{HMAC-SHA256}_k(m) \)

[Cautions?]
Review: Confidentiality

Problem: Sending message in the presence of an eavesdropper without revealing it

Provably-secure solution: One-time pad

Practical solution:

Pseudorandom generator (PRG)

Input: fixed-length \( k \)

Output: arbitrary-length stream

Secure if practically indistinguishable from a random stream, unless know \( k \)

Real-world use: Stream ciphers (can’t reuse \( k \))

Popular example: AES-128 + CTR mode

Block ciphers (need padding/IV) Popular example: AES-128 + CBC mode

[Cautions?!!]
Key Exchange
**Issue:** How do we get a shared key?

Alice ↔ Bob

Eve

No shared secret (yet!)

**Amazing fact:**
Alice and Bob can have a public conversation to derive a shared key!

**Diffie-Hellman (D-H) key exchange**

1976: Whit Diffie, Marty Hellman, improving partial solution from Ralph Merkle (earlier, in secret, by Malcolm Williamson of British intelligence agency)

Relies on a mathematical hardness assumption called *discrete log problem* (a problem believed to be hard)
New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE
A visual analogy:

“Mixing paints”

Mixing in a new color is a little bit like exponentiation.

Hard to invert?

Two different ways at arriving at the same final result.
Group Theory Basics
Schnorr groups

A Schnorr group $G$ is a subset of numbers, under multiplication, modulo a prime $p$. (a “safe prime”)

- We can check if a number $x$ is an element of the group
- If $x$ and $y$ are in the group, then $x \cdot y$ is in the group too
  ($x \cdot y$ means $x$ times $y \mod p$)

- $g$ is a generator of the group if every element of the group can be written as $g^x$ for some exponent $x$.

$g^x$  Exponent,  $0 \leq x < (p - 1)/2$

$g$  Generator, an element of the group
What is a Group?

A class of mathematical objects (it generalizes “numbers mod $p$”)

Definition: A group $(G, \ast)$ is a set of elements $G$, and a binary operation $\ast$

- (Closed): for any $x, y \in G$, we know $x \ast y \in G$

- (Identity): we know the identity $e$ in $G$
  for any $x \in G$, we have $e \ast x = x = x \ast e$

- (Inverses): for any $x$, we can compute $x^{-1} \ast x = e$

- (Associative): For $x, y, z \in G$, $x \ast (y \ast z) = (x \ast y) \ast z$
Schnorr Groups in more detail

To generate a Schnorr group:
1. Pick a random, large, (e.g. 2048 bits) “safe prime” $p$
   - $p$ is a “safe prime” if $(p - 1) / 2$ is also prime
2. Pick a random number $g_0$ in the range 2 to $(p - 1)$
3. Let $g = (g_0)^2 \mod p$. If $g = 1$, loop at step 2
   - This is the “generator” of the group.
   - A number $x$ is in the group if $x^2 \neq 1 \mod p$
   - The order of each element is $(p - 1) / 2$
     - $g^{(p - 1)/2} = 1 \mod p$
   - We can compute inverses $x^{-1}$ s.t. $x^{-1} x = 1 \mod p$
Problems assumed “hard” in Schnorr groups:
- **Discrete logarithm problem**
  Given \( g^x \) for some random \( x \), find \( x \)
- **Diffie Hellman problem (computational)**
  Given \( g^a, g^b \) for random \( a, b \) compute \( g^{ab} \)
- **Diffie Hellman problem (decisional)**
  Flip a bit \( c \), generate random exponents \( a, b, r \)
  Given \( (g^a, g^b, g^{ab}) \) if \( c=0 \), or \( (g^a, g^b, g^r) \) if \( c=1 \),
  Guess \( c \)

*These problems are thought to be hard in other groups too, e.g. some Elliptic Curves*
Diffie-Hellman protocol

1. Alice and Bob agree on public parameters (maybe in standards doc)

2. Alice  
   Generates random secret exponent $a$.

   Bob  
   Generates random secret value $b$.

3. Computes $x = (g^b)^a = g^{ba}$

   Computes $x' = (g^a)^b = g^{ab}$

   (Notice that $x == x'$)

   Can use $k := \text{hash}(x)$ as a shared key.
Passive eavesdropping attack

Eve knows: $g, g^a, g^b$

Eve wants to compute $x = g^{ab}$

Best known approach:

Find $a$ or $b$, by solving discrete log, then compute $x$

No known efficient algorithm.

[What’s D-H’s big weakness?]
Alice does D-H exchange, *really with Mallory*, ends up with $g^{au}$

Bob does D-H exchange, *really with Mallory*, ends up with $g^{bv}$

Alice and Bob each think they are talking with the other, but really Mallory is between them and knows both secrets

*Bottom line:* D-H gives you secure connection, but you don’t know who’s on the other end!
Defending D-H against MITM attacks:

• Cross your fingers and hope there isn’t an active adversary.

• Rely on out-of-band communication between users. [Examples?]

• Rely on physical contact to make sure there’s no MITM. [Examples?]

• Integrate D-H with user authentication.

  If Alice is using a password to log in to Bob, leverage the password:

    Instead of a fixed $g$, derive $g$ from the password – Mallory can’t participate w/o knowing password.

• Use digital signatures. [More next week.]
Public Key Encryption

Suppose Bob wants to receive data from lots of people, confidentially...

Schemes we’ve discussed would require a separate key shared with each person

Example: a journalist who wishes to receive secret tips
Public Key Encryption

- **Key generation:** Bob generates a keypair
  public key, $k_{pub}$ and private key, $k_{priv}$

- **Encrypt:** Anyone can encrypt the message $M$, resulting in ciphertext $C = Enc(k_{pub}, M)$

- **Decrypt:** Only Bob has the private key needed to decrypt the ciphertext: $M = Dec(k_{priv}, C)$

- **Security:** Infeasible to guess $M$ or $k_{priv}$, even knowing $k_{pub}$ and seeing ciphertexts
Public Key Encryption w/ ephemeral key exchange

Key generation (Alice):
\[ k_{\text{priv}} := b \text{ generated randomly, and } k_{\text{pub}} := g^b \]

Encrypt (Bob):
Generate random \( a \), set \( x := \text{hash}(k_{\text{pub}}^a) \), encrypt \( M \) using AES with key \( x \). Send \((g^a, C)\)

Decrypt (Alice):
Compute \( x = \text{hash}( (g^a)^b ) \), decrypt using AES
Public Key Digital Signatures

Suppose Alice publishes data to lots of people, and they all want to verify integrity...

Can’t share an integrity key with everybody, or else anybody could forge messages

*Example:* administrator of a source code repository
Public Key Digital Signature

- Key generation: Bob generates a keypair
  public key, $k_{pub}$ and private key, $k_{priv}$

- Bob can sign a message $M$, resulting in
  signature $S = \text{Sign}(k_{priv}, M)$

- Anyone who knows $k_{pub}$ can check the
  signature: $\text{Verify}(k_{pub}, M, S) = ? 1$

- “Unforgeable”: Computationally infeasible to
guess $S$ or $k_{priv}$, even knowing $k_{pub}$ and seeing
  signatures on other messages
Best known, most common public-key algorithm: **RSA**
Rivest, Shamir, and Adleman 1978
(earlier by Clifford Cocks of British intelligence, in secret)
How RSA works

Key generation:
1. Pick large (say, 1024 bits) random primes $p$ and $q$
2. Compute $N := pq$ (RSA uses multiplication mod $N$)
3. Pick $e$ to be relatively prime to $(p-1)(q-1)$
4. Find $d$ so that $ed \mod (p-1)(q-1) = 1$
5. Finally:
   - Public key is $(e,N)$
   - Private key is $(d,N)$

To sign: $S = \text{Sign}(x) = x^d \mod N$

To verify: $\text{Verif}(S) = S^e \mod N$ Check $\text{Verif}(S) =? M$
Why RSA works

“Completeness” theorem:

For all $0 < x < N$, we can show that $\text{Verif}(\text{Sign}(x)) = x$

Proof:

$$\text{Verif}(\text{Sign}(x)) = (x^d \mod pq)^e \mod pq$$

$$= x^{ed} \mod pq$$

$$= x^{a(p-1)(q-1)+1} \mod pq \text{ for some } a$$

(because $ed \mod (p-1)(q-1) = 1$)

$$= (x^{(p-1)(q-1)})^a \mod pq$$

$$= (x^{(p-1)(q-1)} \mod pq)^a \mod pq$$

$$= 1^a \mod pq$$

(because of the fact that if $p, q$ are prime, then for all $0 < x < N$,

$$x^{(p-1)(q-1)} \mod pq = 1$$)

$$= x$$

Fermat's little theorem
Is RSA secure?

**Best known** way to compute \(d\) from \(e\) is factoring \(N\) into \(p\) and \(q\).

**Best known** factoring algorithm:

**General number field sieve**

Takes more than polynomial time but less than exponential time to factor \(n\)-bit number.

(Still takes way too long if \(p,q\) are large enough and random.)

Fingers crossed...

but can’t rule out a breakthrough!
To generate an RSA keypair:

```
$ openssl genrsa -out private.pem 1024
$ openssl rsa -pubout -in private.pem > public.pem
```

To sign a message with RSA:

```
$ openssl rsautl -sign -inkey private.pem -in a.txt > sig
```

To verify a signed message with RSA:

```
$ openssl rsautl -verify -pubin -inkey public.pem -in sig
```
Public key digital signatures on hashes of code releases
“Pretty Good Privacy” tool

- alternate command line

HOW TO USE PGP TO VERIFY THAT AN EMAIL IS AUTHENTIC:

LOOK FOR THIS TEXT AT THE TOP.

-----BEGIN PGP SIGNED MESSAGE-----
HASH: SHA256

HEY,

FIRST OF ALL, THANKS FOR TAKING CARE OF
IF IT'S THERE, THE EMAIL IS PROBABLY FINE.

If you want to be extra safe, check that there's a big block of jumbled characters at the bottom.

https://xkcd.com/1181/
Subtle fact: RSA can be used for either confidentiality or integrity

RSA for confidentiality:

Encrypt with public key, Decrypt with private key

Public key is \((e,N)\)

Private key is \((d,N)\)

To encrypt: \(E(x) = x^e \mod N\)

To decrypt: \(D(x) = x^d \mod N\)

RSA for integrity:

Encrypt (“sign”) with private key

Decrypt (“verify”) with public key
RSA drawback: Performance

Factor of 1000 or more slower than AES.
Dominated by exponentiation – cost goes up (roughly) as cube of key size.
Message must be shorter than $N$.

Use in practice:

Hybrid Encryption (similar to key exchange):
Use RSA to encrypt a random key $k < N$, then use AES

Signing:
Compute $v := \text{hash}(m)$, use RSA to sign the hash

Should always use crypto libraries to get details right
What can go wrong with RSA?

Many have a common theme: tweaking the protocol for efficiency (e.g., small exponents) leads to a compromise.

Twenty Years of Attacks on the RSA Cryptosystem

Dan Boneh
dabo@cs.stanford.edu

Hundreds of things!!
One example of a failure: Common P’s and Q’s
Individually, $N = pq$ is very hard to factor. Turns out, due to poor entropy, many pairs of RSA keys are generated with same $p$

$$N_1 = p q_1$$
$$N_2 = p q_2$$

Given two products with a common factor, easy to compute $\text{GCD}(N_1, N_2)$ with Euclid’s algorithm.
The hard part of crypto: **Key-management**

**Principles:**

0. Always remember, key management is the hard part!

1. Each key should have only one purpose
   (in general, no guarantees when keys reused elsewhere)

2. Vulnerability of a key increases:
   a. The more you use it.
   b. The more places you store it.
   c. The longer you have it.

3. Keep your keys far from the attacker.

4. Protect yourself against compromise of old keys.
   Goal: **forward secrecy** — learning old key shouldn’t help adversary learn new key.

[How can we get this?]
Building a **secure channel**

What if you want confidentiality and integrity at the same time?

**Encrypt, then MAC**

not the other way around

**Use separate keys** for confidentiality and integrity.

Need two shared keys,  
but only have one?  
That’s what PRGs are for!

If there’s a reverse (Bob to Alice) channel, use separate keys for that too
Issue: How big should keys be?

Want prob. of guessing to be infinitesimal... but watch out for Moore’s law – safe size gets 1 bit larger every 18 months

128 bits usually safe for ciphers/PRGs

Need larger values for MACs/PRFs
due to **birthday attack**

Often trouble if adversary can find any two messages with same MAC

Attack: Generate random values, look for coincidence.
Requires $O(2^{\frac{|k|}{2}})$ time, $O(2^{\frac{|k|}{2}})$ space.
For 128-bit output, takes $2^{64}$ steps: doable!

Upshot: Want output of MACs/PRFs to be twice as big as cipher keys e.g. use HMAC-SHA256 alongside AES-128
<table>
<thead>
<tr>
<th>Key Type</th>
<th>Originator Usage Period (OUP)</th>
<th>Cryptoperiod</th>
<th>Recipient Usage Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Signature Key</td>
<td>1-3 years</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public Signature Key</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetric Authentication Key</td>
<td>&lt;= 2 years</td>
<td></td>
<td>&lt;= OUP + 3 years</td>
</tr>
<tr>
<td>Private Authentication Key</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public Authentication Key</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetric Data Encryption Key</td>
<td>&lt;= 2 years</td>
<td></td>
<td>&lt;= OUP + 3 years</td>
</tr>
<tr>
<td>Symmetric Key Wrapping Key</td>
<td>&lt;= 2 years</td>
<td></td>
<td>&lt;= OUP + 3 years</td>
</tr>
<tr>
<td>Symmetric RBG keys</td>
<td>Determined by design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetric Master Key</td>
<td>About 1 year</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Private Key Transport Key</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public Key Transport Key</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetric Key Agreement Key</td>
<td>1-2 years (1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Private Static Key Agreement Key</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public Static Key Agreement Key</td>
<td>1-2 years (2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Private Ephemeral Key Agreement Key</td>
<td>One key agreement transaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public Ephemeral Key Agreement Key</td>
<td>One key agreement transaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetric Authorization Key</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Private Authorization Key</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public Authorization Key</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Minimum of Strength</td>
<td>Symmetric Algorithms</td>
<td>Factoring Modulus</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------</td>
<td>----------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>(Legacy)</td>
<td>80</td>
<td>2TDEA*</td>
<td>1024</td>
</tr>
<tr>
<td>2016 - 2030</td>
<td>112</td>
<td>3TDEA</td>
<td>2048</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016 - 2030 &amp; beyond</td>
<td>128</td>
<td>AES-128</td>
<td>3072</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016 - 2030 &amp; beyond</td>
<td>192</td>
<td>AES-192</td>
<td>7680</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016 - 2030 &amp; beyond</td>
<td>256</td>
<td>AES-256</td>
<td>15360</td>
</tr>
</tbody>
</table>
Attacks against Crypto

1. Brute force: trying all possible private keys

2. Mathematical attacks: factoring

3. Timing attacks: using the running time of decryption

4. Hardware-based fault attack: induce faults in hardware to generate digital signatures

5. Chosen ciphertext attack

6. Architectural Changes
So Far:
Message Integrity
Confidentiality
Key Exchange
Public Key Crypto

Next Week:
HTTPS and TLS: Secure channels for the web
Bitcoin and Cryptocurrencies
Post Quantum:

When will a quantum computer be built?

15 years, $1 billion USD, nuclear power plant (PQCrypto 2014, Matteo Mariantoni)

What will be impacted?

Public key crypto:

- RSA
- Elliptic Curve Cryptography (ECDSA)
- Finite Field Cryptography (DSA)
- Diffie–Hellman key exchange

Symmetric key crypto:

Aes, Triple Des

Need Larger Keys

Hash functions:

- SHA-1, SHA-2 and SHA-3

Use longer output