Information Security
CS 526

Topic 6: Public Key Encryption and Digital Signatures
Readings for This Lecture

• Required: On Wikipedia
  – Public key cryptography
  – RSA
  – Diffie–Hellman key exchange
  – ElGamal encryption

• Required:
Review of Secret Key (Symmetric) Cryptography

• Confidentiality
  – stream ciphers (uses PRNG)
  – block ciphers with encryption modes

• Integrity
  – Cryptographic hash functions
  – Message authentication code (keyed hash functions)

• Limitation: sender and receiver must share the same key
  – Needs secure channel for key distribution
  – Impossible for two parties having no prior relationship
  – Needs many keys for n parties to communicate
Concept of Public Key Encryption

- Each party has a pair (K, K\(^{-1}\)) of keys:
  - K is the **public** key, and used for encryption
  - K\(^{-1}\) is the **private** key, and used for decryption
  - Satisfies \(D_{K^{-1}}[E_K[M]] = M\)

- Knowing the public-key K, it is computationally infeasible to compute the private key K\(^{-1}\)
  - How to check (K,K\(^{-1}\)) is a pair?
  - Offers only computational security. Secure PK Encryption impossible when P=NP, as deriving K\(^{-1}\) from K is in NP.

- The public-key K may be made publicly available, e.g., in a publicly available directory
  - Many can encrypt, only one can decrypt

- Public-key systems aka **asymmetric** crypto systems
Public Key Cryptography Early History

• Proposed by Diffie and Hellman, documented in “New Directions in Cryptography” (1976)
  1. Public-key encryption schemes
  2. Key distribution systems
     • Diffie-Hellman key agreement protocol
  3. Digital signature

• Public-key encryption was proposed in 1970 in a classified paper by James Ellis
  – paper made public in 1997 by the British Governmental Communications Headquarters

• Concept of digital signature is still originally due to Diffie & Hellman
Public Key Encryption Algorithms

- Almost all public-key encryption algorithms use either number theory and modular arithmetic, or elliptic curves.
  - RSA
    - Based on the hardness of factoring large numbers
  - El Gamal
    - Based on the hardness of solving discrete logarithm
    - Use the same idea as Diffie-Hellman key agreement
Diffie-Hellman Key Agreement Protocol

Not a Public Key Encryption system, but can allow A and B to agree on a shared secret in a public channel (against passive, i.e., eavesdropping only adversaries)
Setup: \( p \) prime and \( g \) generator of \( \mathbb{Z}_p^* \), \( p \) and \( g \) public.

\[
K = (g^b \mod p)^a = g^{ab} \mod p
\]

Pick random, secret \( a \)
Compute and send \( g^a \mod p \)
\( K = (g^a \mod p)^b = g^{ab} \mod p \)

Pick random, secret \( b \)
Compute and send \( g^b \mod p \)
Example: Let p=11, g=2, then

\[
\text{A chooses 4, B chooses 3, then shared secret is}
\]

\[
(2^3)^4 = (2^4)^3 = 2^{12} = 4 \pmod{11}
\]

Adversaries sees \(2^3 = 8\) and \(2^4 = 5\), needs to solve one of \(2^x = 8\) and \(2^y = 5\) to figure out the shared secret.

<table>
<thead>
<tr>
<th>a</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>(g^a)</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1024</td>
<td>2048</td>
</tr>
<tr>
<td>(g^a \pmod{p})</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>5</td>
<td>10</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Three Problems Believed to be Hard to Solve

- **Discrete Log (DLG) Problem**: Given $<g, h, p>$, computes $a$ such that $g^a = h \mod p$.

- **Computational Diffie Hellman (CDH) Problem**: Given $<g, g^a \mod p, g^b \mod p>$ (without $a, b$) compute $g^{ab} \mod p$.

- **Decision Diffie Hellman (DDH) Problem**: distinguish $(g^a, g^b, g^{ab})$ from $(g^a, g^b, g^c)$, where $a, b, c$ are randomly and independently chosen.

- If one can solve the DL problem, one can solve the CDH problem. If one can solve CDH, one can solve DDH.
Assumptions

- DDH Assumption: DDH is hard to solve.
- CDH Assumption: CDH is hard to solve.
- DLG Assumption: DLG is hard to solve.

- DDH assumed difficult to solve for large $p$ (e.g., at least 1024 bits).
- Warning:
  - New progress by Joux means solving discrete log for $p$ values with some property can be done quite fast.
  - Look out when you need to use/implement public key crypto
  - May want to consider Elliptic Curve-based algorithms
ElGamal Encryption

- Public key \(<g, p, h=g^a \mod p>\)
- Private key is \(a\)
- To encrypt: chooses random \(b\), computes \(C=[g^b \mod p, g^{ab} * M \mod p]\).
  - Idea: for each \(M\), sender and receiver establish a shared secret \(g^{ab}\) via the DH protocol. The value \(g^{ab}\) hides the message \(M\) by multiplying it.
- To decrypt \(C=[c_1,c_2]\), computes \(M\) where
  - \(((c_1^a \mod p) * M) \mod p = c_2\).
    - To find \(M\) for \(x * M \mod p = c_2\), compute \(z\) s.t. \(x*z \mod p = 1\), and then \(M = C_2*z \mod p\)
- CDH assumption ensures \(M\) cannot be fully recovered.
- IND-CPA security requires DDH.
RSA Algorithm

• Invented in **1978** by Ron Rivest, Adi Shamir and Leonard Adleman

• Security relies on the difficulty of factoring large composite numbers

• Essentially the same algorithm was discovered in 1973 by Clifford Cocks, who works for the British intelligence
RSA Public Key Crypto System

**Key generation:**

1. Select 2 large prime numbers of about the same size, p and q
   - Typically each p, q has between 512 and 2048 bits
2. Compute $n = pq$, and $\Phi(n) = (q-1)(p-1)$
3. Select $e$, $1 < e < \Phi(n)$, s.t. $\gcd(e, \Phi(n)) = 1$
   - Typically $e=3$ or $e=65537$
4. Compute $d$, $1 < d < \Phi(n)$ s.t. $ed \equiv 1 \mod \Phi(n)$
   - Knowing $\Phi(n)$, d easy to compute.

**Public key:** $(e, n)$

**Private key:** $d$
RSA Description (cont.)

**Encryption**

Given a message $M$, $0 < M < n$ \( M \in \mathbb{Z}_n - \{0\} \)

use public key \((e, n)\)

compute $C = M^e \mod n$ \( C \in \mathbb{Z}_n - \{0\} \)

**Decryption**

Given a ciphertext $C$, use private key \((d)\)

Compute $C^d \mod n = (M^e \mod n)^d \mod n = M^{ed} \mod n = M$
RSA Example

- $p = 11$, $q = 7$, $n = 77$, $\Phi(n) = 60$
- $d = 13$, $e = 37$  ($ed = 481$;  $ed \mod 60 = 1$)
- Let $M = 15$. Then $C \equiv M^e \mod n$
  - $C \equiv 15^{37} \pmod{77} = 71$
- $M \equiv C^d \mod n$
  - $M \equiv 71^{13} \pmod{77} = 15$
RSA Example 2

- Parameters:
  - \( p = 3, q = 5, n = pq = 15 \)
  - \( \Phi(n) = ? \)
- Let \( e = 3 \), what is \( d \)?
- Given \( M = 2 \), what is \( C \)?
- How to decrypt?
Hard Problems RSA Security
Depends on

Ciphertext: $C$ 

Plaintext: $M$

\[ C = M^e \mod (n=pq) \]
\[ C^d \mod n \]

1. **Factoring Problem**: Given $n=pq$, compute $p,q$
2. **Finding RSA Private Key**: Given $(n,e)$, compute $d$ s.t. $ed = 1 \pmod{\Phi(n)}$.
   - Known to be equivalent to Factoring problem.
   - Implication: cannot share $n$ among multiple users
3. **RSA Problem**: From $(n,e)$ and $C$, compute $M$ s.t. $C = M^e$
   - Aka computing the $e$’th root of $C$.
   - Can be solved if $n$ can be factored
RSA Security and Factoring

- Security depends on the difficulty of factoring $n$
  - Factor $n \Rightarrow$ compute $\Phi(n) \Rightarrow$ compute $d$ from $(e, n)$
  - Knowing $e, d$ such that $ed = 1 \pmod{\Phi(n)} \Rightarrow$ factor $n$
- The length of $n=pq$ reflects the strength
  - 700-bit $n$ factored in 2007
  - 768 bit factored in 2009
- RSA encryption/decryption speed is quadratic in key length
- 1024 bit for minimal level of security today
  - likely to be breakable in near future
- Minimal 2048 bits recommended for current usage
- NIST suggests 15360-bit RSA keys are equivalent in strength to 256-bit
- Factoring is easy to break with quantum computers
- Recent progress on Discrete Logarithm may make factoring much faster
RSA Encryption & IND-CPA Security

• The RSA assumption, which assumes that the RSA problem is hard to solve, ensures that the plaintext cannot be fully recovered.

• Plain RSA does not provide IND-CPA security.
  – For Public Key systems, the adversary has the public key, hence the initial training phase is unnecessary, as the adversary can encrypt any message he wants to.

  – How to break IND-CPA security?
Real World Usage of Public Key Encryption

- Often used to encrypt a symmetric key
  - To encrypt a message \( M \) under an RSA public key \( (n,e) \),
    generate a new AES key \( K \), compute
    \[ [K^e \mod n, \text{AES-CBC}_K(M)] \]

- One often needs random padding.
  - Given \( M \), chooses random \( r \), and generates \( F(M,r) \), and then encrypts as
    \[ F(M,r)^e \mod n \]
  - From \( F(M,r) \), one should be able to recover \( M \)
  - This provides randomized encryption
Digital Signatures: The Problem

- Consider the real-life example where a person pays by credit card and signs a bill; the seller verifies that the signature on the bill is the same with the signature on the card.
- Contracts are valid if they are signed.
- Signatures provide non-repudiation.
  - ensuring that a party in a dispute cannot repudiate, or refute the validity of a statement or contract.
- Can we have a similar service in the electronic world?
  - Does Message Authentication Code provide non-repudiation? Why?
Digital Signatures

- **MAC**: One party generates MAC, one party verifies integrity.
- **Digital signatures**: One party generates signature, many parties can verify.
- **Digital Signature**: a data string which associates a message with some originating entity.
- **Digital Signature Scheme**:
  - a signing algorithm: takes a message and a (private) signing key, outputs a signature
  - a verification algorithm: takes a (public) verification key, a message, and a signature
- **Provides**:
  - Authentication, Data integrity, Non-Repudiation
Digital Signatures and Hash

- Very often digital signatures are used with hash functions, hash of a message is signed, instead of the message.
- Hash function must be:
  - Pre-image resistant
  - Weak collision resistant
  - Strong collision resistant
RSA Signatures

Key generation (as in RSA encryption):

- Select 2 large prime numbers of about the same size, p and q
- Compute n = pq, and \( \Phi = (q - 1)(p - 1) \)
- Select a random integer e, \( 1 < e < \Phi \), s.t. \( \gcd(e, \Phi) = 1 \)
- Compute d, \( 1 < d < \Phi \) s.t. \( ed \equiv 1 \pmod{\Phi} \)

Public key: \((e, n)\) used for verification

Private key: \(d\) used for generation
RSA Signatures with Hash (cont.)

Signing message $M$
- Verify $0 < M < n$
- Compute $S = h(M)^d \mod n$

Verifying signature $S$
- Use public key $(e, n)$
- Compute $S^e \mod n = (h(M)^d \mod n)^e \mod n = h(M)$
Non-repudiation

• Nonrepudiation is the assurance that someone cannot deny something. Typically, nonrepudiation refers to the ability to ensure that a party to a contract or a communication cannot deny the authenticity of their signature on a document or the sending of a message that they originated.

• Can one deny a signature one has made?

• Does email provide non-repudiation?
# The Big Picture

<table>
<thead>
<tr>
<th>Secrecy / Confidentiality</th>
<th>Secret Key Setting</th>
<th>Public Key Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream ciphers</td>
<td>RSA, El Gamal, etc.</td>
<td></td>
</tr>
<tr>
<td>Block ciphers + encryption modes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authenticity / Integrity</th>
<th>Secret Key Setting</th>
<th>Public Key Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Authentication Code</td>
<td></td>
<td>Digital Signatures: RSA, DSA, etc.</td>
</tr>
</tbody>
</table>
Coming Attractions …

• User authentication