Introduction to Cryptography

Subject 4:
Public Key Cryptography

Prof. Amir Herzberg
Computer Science Department, Bar Ilan University
http://amir.herzberg.name

© Amir Herzberg, 2003-4. Permission is granted for academic use without modification. For other use please contact author.
Outline: Public Key Cryptography

- Motivation and mechanisms
- Modular arithmetic
- Diffie-Hellman Key Agreement
- Public-key cryptosystems
  - Concept, components and syntax
  - El-Gamal
  - RSA
  - Defining secure PKCS
  - Secure usage of RSA (Padding etc.)
- Conclusions
Public Key Cryptography

- Concept [DH76]: some operations are asymmetric, e.g.: everybody can send me mail, only I can read it.
- Idea: use a public key – known to adversary
- Encryption – public key cryptosystem
  - Encrypt with public key, decrypt with private key
- Digital signatures
  - Sign with private key, verify with public key
- Key agreement
  - Use public/private key pair to agree on shared secret key
Public keys are easier…

- **To distribute:**
  - From directory (ensure or trust authentication)
  - From incoming message (if authenticated)
  - Less keys to distribute (same public key to all)

- **To maintain:**
  - Can keep in non-secure storage
  - Validate (e.g. against hash) before using
  - **Less keys:** $O(|parties|)$, not $O(|parties|^2)$
But public key crypto is harder...

- Requires related public, private keys
  - Private key `reverses` public key
  - Public key does not expose private key

- Substantial overhead
  - Successful cryptanalytic shortcuts \(\rightarrow\) need long keys (cf. shared key!)
  - Elliptic Curves (EC) may allow shorter key (almost no shortcuts found)
  - Complex computations
  - RSA: very complex (slow) key generation

- Based on modular arithmetic, EC, ...

<table>
<thead>
<tr>
<th>Year</th>
<th>AES</th>
<th>RSA, DH</th>
<th>EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>72</td>
<td>1028</td>
<td>139</td>
</tr>
<tr>
<td>2010</td>
<td>78</td>
<td>1369</td>
<td>160</td>
</tr>
<tr>
<td>2020</td>
<td>86</td>
<td>1881</td>
<td>188</td>
</tr>
<tr>
<td>2030</td>
<td>93</td>
<td>2493</td>
<td>215</td>
</tr>
<tr>
<td>2040</td>
<td>101</td>
<td>3214</td>
<td>244</td>
</tr>
</tbody>
</table>

Commercial-grade security
Lenstra & Verheul [LV02]
Outline: Public Key Cryptography

- Motivation and mechanisms
- Modular arithmetic
- Diffie-Hellman Key Agreement
- Public-key cryptosystems
  - Concept, components and syntax
  - El-Gamal
  - RSA
  - Defining secure PKCS
  - Secure usage of RSA (Padding etc.)
- Conclusions
Recall: Modular arithmetic

- Basic part of (integer) number theory
- For every integers $x, n$ there are unique $q, y$ s.t. $x = qn + r$; we call $r$ residue mod $n$
- Notation: $x = y \text{ mod } n$
  - Reads: “$x$ is congruent to $y$ modulo $n$”
  - If $x$ and $y$ have the same remainder when divided by $n$, namely $x = y + l \cdot n$ for some integer $l$
- Regular arithmetic laws apply
  - E.g. distributive, commutative, associative,…
  - $(a \cdot b) \text{ mod } n = [(a \text{ mod } n) \cdot (b \text{ mod } n)] \text{ mod } n$
Recall: Modular arithmetic

- Regular arithmetic laws apply to residues
  - E.g. distributive, commutative, associative,…
    \[(a*b) \mod n = [(a \mod n)*(b \mod n)] \mod n\]
  - First reduce, then multiply!
- \(n\) divides \(x\), denoted \(n \mid x\), if \(x=0 \mod n\)
- \(x,n\) are co-primes if there is their only (and greatest) common divisor is 1.
- Property: if \(x,n\) are co-primes then \(x\) has inverse \(\mod n\), i.e. \(x^{-1}\) s.t. \(x*x^{-1}=1 \mod n\)
  - Find using Extended Euclidean (GCD) alg.
Hard Modular Math Problems

Hard problems:
- No efficient solution
- In spite of extensive efforts

Factoring: given the product of two uniformly chosen primes, it is infeasible to find the primes

Discrete logarithm in finite field

Verification of solution is easy
- Factoring: multiply factors
- Discrete log: exponentiation
  - Efficient exponentiation $mod\ n: O((\lg\ n)^3)$
- `One-way` hard problems
- Can be used for candidate one-way functions
Discrete logarithm in finite field

- Select $p$ as random $n$-bit prime
- Select $g \in \{2, p-1\}$ such that for every $a \in \{1, p-1\}$ exists
  $b \in \{1, p-1\}$ such that $a = g^b \mod p$.
  - Such $g$, called generator of $GF(p)$, always exists.
  - $b$ is unique

**The Discrete Log Assumption:** Given $a \in_R \{1, p-1\}$, it
  is infeasible to find $b \in \{1, p-1\}$ such that $a = g^b \mod p$.
  - Time complexity of best known algorithm: $O\left(e^{\frac{3}{\sqrt{n\log(n)}}}\right)$

- Candidate One Way Function: $f(b) = g^b \mod p$
Discrete Log Assumption

- Define function $\varepsilon_{DL}(t, n)$ from integers to $[0,1]$
  - Bound on probability of success of adversary
- Such that if algorithm $ADV$ runs up to $t$ steps, then:
  $\text{Prob}(ADV(a,g,p)\in b \text{ s.t. } b\in\{1,p-1\} \text{ and } a=g^b)<\varepsilon_{DL}(t,n)$
- Probability over keys, $a$ and adversary coin tosses
- Discrete Log Assumption (DLA):
  - $\varepsilon_{DL}(t, n)$ is negligible (close to $2^{-n}$) for large $n$
  - Known algorithms require $t \geq e^{3\sqrt{n\log(n)}}$
  - $n=2048$ considered secure till 2020
Much of the research is using asymptotic (polynomial complexity, e.g. poly-time) analysis, definitions

For any poly-time algorithm $ADV$, and every polynomial $Q$, there is some length $N$ such that for every $n>N$ holds: $\text{Prob}(ADV(a,g,p)=b \text{ s.t. } b \in \{1,p-1\} \text{ and } a=g^b)<1/Q(n)$ for random $p$ (prime $<2^n$), generator $g$, and $b \in_R \{1,p-1\}$.

We use the more applied `concrete security` assumption (in previous foil)
Outline: Public Key Cryptography

- Motivation and mechanisms
- Modular arithmetic
- Diffie-Hellman Key Agreement
- Public-key cryptosystems
  - Concept, components and syntax
  - El-Gamal
  - RSA
  - Defining secure PKCS
  - Secure usage of RSA (Padding etc.)
- Conclusions
The Key Agreement Problem

- Motivation for simple public key problem…
- Alice and Bob want to agree on some secret
  - Trivial if they have shared secret key…
  - Assume no prior shared secrets (e.g. key)
  - Afterwards, may use agreed-on secret as key
- Physical setting
  - Assume Alice and Bob can exchange locked box
  - Origin of box is authentic (e.g. visually)
  - Problem: Alice and Bob have no shared key…
  - Solution ???
Key Agreement Using Two-Lock Box

1. Put $k$ in box
2. Lock and send
3. Add lock
4. Send back
5. Remove lock
6. Send…
7. Remove lock
8. Get $k$ from box
Can we use One Time Pad as lock?

Send \( k' = k \oplus k_B \)

Receive \( k' \)

Send \( k'' = k' \oplus k_A \)

Receive \( k'' \)

Remove lock: Send \( k''' = k'' \oplus k_B \)

Receive \( k''' \)

Retrieval: \( k''' \oplus k_A \)

No! Adversary can find \( k = k' \oplus k'' \oplus k''' = (k \oplus k_B) \oplus (k \oplus k_B \oplus k_A) \oplus (k \oplus k_A) \)
Can we use Exponentiation as lock?

Agree, publish prime $p$ and generator $g$

Send $k' = g^{kb}$

Receive $k'$
Send $k'' = (k')^a$

Receive $k''$
Remove lock: Send $k''' = (k'')^{1/b}$

This seems Ok... but we can simplify.

Receive $k'''$
Retrieve: $x = (k''')^{1/a}$
If all Ok, $x = g^k$
Public Key Agreement [DH]

- Based on Discrete Log problem
- Agree, publish random prime $p$ and generator $g$
- Alice: secret key $a$, public key $P_A = g^a \mod p$
- Bob: secret key $b$, public key $P_B = g^b \mod p$
- To set up a shared key $k$:
  - Alice computes: $(P_B)^a = (g^b \mod p)^a = g^{ba} \mod p$
  - Bob computes: $(P_A)^b = (g^a \mod p)^b = g^{ab} \mod p$
  - $k = g^{ba} \mod p = g^{ab} \mod p$
Caution: Authenticate Public Keys!

- Diffie-Hellman key agreement works… if the public keys are authentic
- If Bob simply receives Alice’s public key, this is subject to `man in the middle` attack
- Suppose authenticated communication… is DH secure?

```
Hi, I’m Alice, g^a mod p
```

```
Hi, I’m Alice, g^e mod p
```

Alice → Eve → Bob
Security of [DH] Key Agreement

- Assuming authenticated communication
- Based on Discrete Log assumption:
  - Given $a \in R\{1,p-1\}$, can't find $b \in \{1,p-1\}$ s.t. $a = g^b \mod p$.
  - If given $g^b \mod p$ it is easy to compute $b$, then adversary exposes $k = g^{ba} \mod p$.
- But DH requires stronger assumption than Disc-Log:
  - Maybe from $g^b \mod p$ and $g^a \mod p$, Adversary can compute $k = g^{ba} \mod p$ (without knowing $a,b$)?

\[
P_A = g^a \mod p \quad P_B = g^b \mod p
\]
Can adversary compute partial information about $k = g^{ba} \mod p$?

Actually, finding at least one bit about $k$ is easy:

- Legendre Symbol: $J_p(x) = \{ 1 \text{ if } x = g^{2y} \text{ (a square)}, \ 0 \text{ if } x \mid p, \ -1 \text{ if } x = g^{2y+1} \}$
- For prime $p$, compute by Euler’s criterion: $J_p(x) = x^{(p-1)/2} \mod p$
- This gives the least significant bit (parity) of $b$ from $g^b \mod p$
- From the parity (LSb) of $b, a$, adversary deduces parity of $ba$
- The Legendre symbol of $k = g^{ba} \mod p$

Finding the MSb (in fact, $\log(\log(n))$ MS-bits) is hard

Common heuristic: use $k' = h(k)$ where $h$ is crypto-hash
**DH- \( h \) Assumption**

- **DH- \( h \) Assumption**: Assume \( p, g, a \) and \( b \) are chosen (pseudo) randomly. Given \( g, p, g^b \mod p \) and \( g^a \mod p \), the value \( k' = h(g^{ba} \mod p) \) is pseudorandom.
  - \( h \) is a specific hash function, e.g. SHA-1, MD5, RIPE-MD
  - Pseudorandom: indistinguishable from random
  - Asymptotic/polynomial or concrete definition

![Diagram]

- Alice sends \( P_A = g^a \mod p \)
- Bob sends \( P_B = g^b \mod p \)
Key Agreement - Summary

- Great to establish shared secret key
  - If using authenticated/known public keys $g^a$, $g^b$
  - Or using authenticated channel
  - `Man in the middle` attack must be done during key agreement
  - Even unauthenticated DH protects against attacks before/after key agreement

- But… often we can just encrypt directly using the recipient’s public key!
Outline: Public Key Cryptography

- Motivation and mechanisms
- Modular arithmetic
- Diffie-Hellman Key Agreement
- Public-key cryptosystems
  - Concept, components and syntax
  - El-Gamal
  - RSA
  - Defining secure PKCS
  - Secure usage of RSA (Padding etc.)
- Conclusions
Public Key Cryptosystems

- Everybody knows the encryption key (public)
- Only recipient knows the decryption key (called secret or private key)
- Public key available from directory, 3rd party
- Easier to find and establish keys

\[ \text{plaintext} \xrightarrow{\text{Encrypt}} \text{ciphertext} = \text{Encrypt}_{\text{Pub}_B}(\text{plaintext}) \xrightarrow{\text{Decrypt}} \text{plaintext} \]

\( \text{Pub}_B \) (Bob’s Public key) \n\( \text{Priv}_B \) (Bob’s Private key)
Encryption with Public Code Book

Public key (codebook): English → Code

Sell: ecdrq
Sent: wetrg

Private key (codebook): Code → English

But: trghf
Buy: ecrth

Eve

ecdrq

Alice
Bob

ecdrq

Sell

ecdrq

ecdrq

Sell

12/31/2003
http://Amir.Herzberg.name
26
PKC component algorithms

Key Generate

\[ n \quad \$ \text{(random)} \]

\[ \text{Encrypt}_{\text{PUB}}(\text{Plaintext}) \]

\[ \text{Plaintext} \]

\[ \text{Ciphertext} = \text{Encrypt}_{\text{PUB}}(\text{Plaintext}) \]

\[ \text{Decrypt}_{\text{PRIV}}(\text{Ciphertext}) \]

\[ \text{Plaintext}' \]

\[ \text{Decrypt}_{\text{PRIV}}(\text{Ciphertext}) \]
A PK cryptosystem is defined...

...by three efficient algorithms:

1. **Key-Generate**
   - Inputs: random string and the required key length,
   - Output: pair \(<Pub, Priv>\) of a public key \(Pub\) and a corresponding private key \(Priv\).

2. **Encrypt**
   - Inputs: plaintext \(p\), public key \(Pub\) and random bits
   - Output: corresponding ciphertext \(c = Encrypt_{Pub}(p)\).

3. **Decrypt**
   - Inputs: ciphertext \(c\) and private key \(Priv\)
   - Output: the (decrypted) plaintext block, i.e.
     \(p = Decrypt_{Priv}(c) = Decrypt_{Priv}(Encrypt_{Pub}(p))\).
Assume Bob knows Alice’s public key \( P_A = g^a \mod p \)

Bob chooses *ephemeral* keys \( r \) and \( v = g^r \mod p \)

Bob computes \( (P_A)^r = g^{ar} \mod p \)

Bob encrypts message \( m \) using \( (P_A)^r \) as shared key, e.g.: \( c = m \oplus (P_A)^r \oplus (g^{ar} \mod p) \)

Bob sends \( c, v \)

Alice uses \( v^a = g^{ar} \mod p \) to decrypt, e.g. \( m = c \oplus g^{ar} \mod p \)
El-Gamal PKC

- Minor variant of [DH] PKC
- Shared key encryption: by multiplication
  - Multiplication with non-zero is permutation mod $g$
  - Good encryption if used once…
- Bob chooses $r$ and $v = g^r \mod p$
- Bob encrypts message $m$ using $(P_A)^r$ as shared key:
  $c = m \cdot (P_A)^r = m \cdot g^{ar} \mod p$; sends $c, v$
- Alice uses $v^a = g^{ar} \mod p$ to decrypt: $m = c / g^{ar} \mod p$

Alice $[P_A = g^a \mod p]$  $\rightarrow$  Bob

$c = m \cdot (g^a)^r, v = g^r \mod p$
Drawbacks of [DH], El-Gamal PKC

- Ciphertext is longer than plaintext
- Randomization required for encryption
  - Both are not real drawbacks – required for good encryption…
- Some information about $g^a \text{ mod } p$ leaks
  - At least the Legendre symbol…
  - Used for attacking (e.g. when using small $g$), see [BJN00]
  - Standard solution: $key = h(g^a \text{ mod } p)$ (like for key agreement)
- Encryption, decryption: $O(lg(|p|))$ multiplications
- Not widely used → Marketing, engineering drawback… also security (although DH established)
Practical Public Key Cryptosystems

- Computationally intensive
  - RSA (and some others): especially key generation
- Most proposals require long block and key sizes (see table)
- We’ll look at RSA
  - Most well known and widely used
  - Number-theory deterministic functions $E^{RSA}, D^{RSA}$
  - Input randomized and padded
  - Private key $d$, public key $(e, n)$
- RSA is based on Euler’s Theorem…

<table>
<thead>
<tr>
<th>Year</th>
<th>AES</th>
<th>RSA, DH</th>
<th>EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>78</td>
<td>1369</td>
<td>160</td>
</tr>
<tr>
<td>2020</td>
<td>86</td>
<td>1881</td>
<td>188</td>
</tr>
<tr>
<td>2030</td>
<td>93</td>
<td>2493</td>
<td>215</td>
</tr>
<tr>
<td>2040</td>
<td>101</td>
<td>3214</td>
<td>244</td>
</tr>
</tbody>
</table>

Lenstra & Verheul [LV02]
Outline: Public Key Cryptography

- Motivation and mechanisms
- Modular arithmetic
- Diffie-Hellman Key Agreement
- Public-key cryptosystems
  - Concept, components and syntax
  - El-Gamal
  - RSA
  - Defining secure PKCS
  - Secure usage of RSA (Padding etc.)
- Conclusions
Recall: Euler Theorem & Function $\Phi(n)$

- The Euler function of $n$, denoted $\Phi(n)$, is the number of positive integers less than $n$ and co-prime to $n$.

<table>
<thead>
<tr>
<th>$n$</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>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Phi(n)$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>4</td>
<td>12</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

- For primes $p, q$ holds $\Phi(pq) = (p-1)(q-1)$

- Euler’s Theorem: if $a, n$ are co-primes then $a^{\Phi(n)} \equiv 1 \mod n$
  - Fermat’s Theorem: if $p$ is primes then $a^{p-1} \equiv 1 \mod p$
  - Also: $a^x \mod \Phi(n) = a^x \mod n$; $a^1 \mod \Phi(n) = a \mod n$

- Also: $a^{-1} = a^{\Phi(n)-1} \mod n \rightarrow$ easy to compute inverse $\mod n$
RSA Public Key Cryptosystem

- Select two large primes $p, q$ and let $n = pq$
- Let $\Phi(n) = (p-1)(q-1)$ [Euler function]
- Select random public exponent $e$ which is prime
  - Or co-prime to $\Phi(n)$, i.e. no common divisor
- Let $d = e^{-1} \mod \Phi(n)$, i.e. $ed = 1 \mod \Phi(n)$.
  - Find $d$ from $\Phi(n)$ with extended Euclidean (GCD) algorithm (simple… see e.g. in Wikipedia)
- Public key is $<n, e>$, private key is $d$.
  - Notice: given $p, q$ it is easy to find $\Phi(n) = (p-1)(q-1)$ and from it $d$
  - Assumption:
    - it is hard to find $d$ (or $\Phi(n)$, or $p, q$) given only $<n, e>$
- $E_{RSA}(m) = m^e \mod n$
- $D_{RSA}(c) = c^d \mod n = (m^e)^d = m^{ed} \mod n$
  - Euler: $m^{ed} = m^{(1 \mod \Phi(n))} = m \mod n$
Efficient RSA computations

- RSA Key Generation: find random primes $p, q$
  - $p, q$ should be about same length ($\ln l/2$) and not related
  - Computationally intensive; randomized tests, e.g. Miller-Rabin

- RSA Encryption and Decryption are both exponentiations

- Exponentiation $mod\ n$ takes $O((\log n)^3)$ operations
  - By `square and multiply` algorithm

- For security, $n$ should be $\geq 2048$ bits $\rightarrow$ slow!
  - 512 bits is definitely feasible to factor
  - 768, 1024 bits sometimes used... insecure now or `soon``
  - Choice depends on application (sensitivity level and lifetime)

- But several speed-ups are known...
Improving RSA Encryption Efficiency

- Idea: Use short public key $e$ for efficiency
  - Often using $e=3$ or $e=2^{16}+1=65537$
- This makes encryption ($m^e$) very efficient
  - E.g. $m^{65537}=m*(m^4)^4$, i.e. only 7 multiplications
- Of course, use different $n$ (and $d$!) for each user
- Some attacks on tiny $e$ (e.g. 3)
  - If same or related messages are encrypted for $e$ recipients
  - When $m<n^{1/e}$, solve over integers $m=c^{1/e}$ …
- Prevent attacks:
  - By appropriate (random) padding: details later
  - By using larger $e$ (e.g. 65537, not 3…)
- What about Decryption?
Improving Decryption Efficiency

- Can we use short private key $d$ for efficiency?
  - No; $d$ should be roughly as long as $n$
  - Otherwise adversary can guess $d$
  - Also, to have short $d$, we must chose $d$ and compute $e$
    (i.e. $e$ would not be short…)
  - In some special cases shorter $d$ is used (with care…)

- Some savings still possible…

- In some cases we can use batch decryption
  - Decryption of several messages together
  - More efficiently than decryption of each message
  - We will not cover this

- Standard technique: compute $m^d$ using Chinese Remainder Theorem…
Using Chinese Remainder Theorem (CRT)

- CRT (simplified): let \( n=pq \) where \( p<q \) are primes. Then:
  - Given \( m=a \mod p, m=b \mod q \), for any \( a,b \), the unique solution \( m \in \mathbb{Z}_n \) is \( m=(((a-b)*u) \mod p)*q + b \) where \( u*q=1 \mod p \).

- To decrypt, \( m=c^d \mod n \)
  - Note: \( m=a=c^d \mod p, m=b=c^d \mod q \)

- Let \( \alpha=d \mod (p-1) \) i.e. \( d=k*(p-1)+\alpha \)

- Compute: \( a=c^d \mod p= c^{k*(p-1)+\alpha} \mod p \)
  - By Fermat’s theorem, \( c^{p-1}=1 \mod p \)
  - \( \Rightarrow a=c^\alpha \mod p \)

- Similarly let \( \beta=d \mod (q-1) \rightarrow b=c^\beta \mod q \)

- \( \Rightarrow m=(((a-b)*u) \mod p)*q + b \) where \( u*q=1 \mod p \)

- Find \( u, \alpha, \beta \) only once (for all \( m,c \)) and quite efficiently

- Find \( a=c^\alpha \mod p, b=c^\beta \mod q \rightarrow m: \) takes \( O((\log \sqrt{n})^3)=O((\log n)^3)/8 \)
RSA Security

- If adversary can factor $n$ and find $p, q$, then:
  - Adversary can find $\Phi(n)=(p-1)(q-1)$
  - $d=e^{-1} \mod \Phi(n)$
  - Factoring `completely` breaks RSA.
  - But factoring is considered hard problem.

- Fact: exists efficient algorithm that factors $n$, given $<n,e,d>$ s.t. $ed=1 \mod \Phi(n)$.
  - Can’t use the same $n$ for multiple users with different $d$
  - Notice: we can easily find $d$ from $\Phi(n)$ so $\Phi(n) \to d \to p, q$

- What is the security that we can assume for RSA?
  - Keeping $d$ secret – not very useful security property
  - One-way permutation (hard to invert on random input)?
  - But RSA has a trapdoor – easy to compute knowing $d$…
RSA Assumption: Trapdoor OWP

- RSA is assumed to be Trapdoor One-Way Permutation:
  - Hard to invert on random input
  - Except if given trapdoor…
- Define function $\varepsilon_{RSA}(t, N)$ from integers to $[0,1]$
- Such that if algorithm $ADV$ runs up to $t$ steps, then:
  \[ \text{Prob}(ADV(n,e,x^e \mod n)=x)<\varepsilon_{RSA}(t, N) \]
- Probability over keys, $x$ and adversary coin tosses
- RSA Assumption:
  - $\varepsilon_{RSA}(t, n)$ is negligible (close to $2^{-n}$) for large $n$
  - Known algorithms require $t \geq e^{3\sqrt{n\log(n)}}$
  - $n=2048$ considered secure till 2020
  - Surprisingly, almost identical to DLA…
RSA Security – Cautions…

- The RSA assumption is quite well established:
  - RSA is a Trapdoor One-Way Permutation
  - Hard to invert on random input – without secret key
- But is it a secure cryptosystem?
- Does it hide *everything* about the input?
  - Theorem [G]: RSA hides $\log(\log(n))$ least and most significant bits of uniformly-distributed random input
  - But some (other) information about pre-image may leak
- And… adversary can detect a repeating message
Outline: Public Key Cryptography

- Motivation and mechanisms
- Modular arithmetic
- Diffie-Hellman Key Agreement
- Public-key cryptosystems
  - Concept, components and syntax
  - El-Gamal
  - RSA
  - Defining secure PKCS
  - Secure usage of RSA (Padding etc.)
- Conclusions
Defining Secure Public Key Encryption

- Recall: Encryption is like disguise…
- With a good disguise you…
  - Can’t even tell the most pretty from the most ugly
  - Can’t even identify the same disguised person
    - Talk to disguised Rachel
    - She is out of sight for a moment
    - Is it Rachel again or maybe its Leah?
  - As long as they are roughly same size
    - A giant can never disguise as a dwarf
  - All this… while you know the (public) encryption key!
- Even the `Public code-book system` is not good enough…
- Let’s review and suggest a fix…
Recall: Public Code Book System

Public key (codebook): English → Code

Sell: ecdrq
Sent: wetrg

Buy: ecrth
But: trghf

Private key (codebook): Code → English

Sell: ecdrq
Buy: ecrth

Eve

Alice

Bob

Sell

ecdrq

ecdrq

ecdrq

Sell

12/31/2003
http://Amir.Herzberg.name
Public Numeric Code Book System

Public key (codebook): English ➔ Code

- Sell: 1563
- Sent: 7654

Private key (codebook): Code ➔ English

- 1563: Sell
- 1564: But

Alice ➔ Bob

- 1563 ➔ Eve
- But: 1564
- Buy: 9832

Eve ➔ Bob

Bob ➔ Alice

12/31/2003

http://Amir.Herzberg.name
Randomized Public Numeric Code Book System

Public codebook

Sell: 1235. 1266. Sell: ecdrq
Sent: weta

Sell+31

$ r \in \mathbb{R}\{0..99\} $

Sell

Alice

Private codebook

weta: 1266 987
weta

Bob

weta, $ r = 31 $

1235 = 1266 - 31

Sell
Plaintext-Only Indistinguishability

Key Generate

$ (\text{random})$

$\langle \text{Pub}, \text{Priv}\rangle$

Attacker selects plaintexts $x[0], x[1]$

$x[0], x[1]$

$b \in R\{0, 1\}$

$\text{Encrypt}_{\text{Pub}}(x[b])$

Adversary tries to find $b$

Adversary: $b' (= b \ ???)$

Exercise: Define! (we’ll define CCA instead)
Chosen Ciphertext Attack (CCA) Indistinguishability

- Given algorithm $A$ with oracle to $D[sk]$, i.e. $A^{D[sk]}$

- $\text{CCA-IND}^A,<KG,E,D>,k,q,a$:
  - $(pk, sk) \leftarrow KG(1^k)$; /* $k$ is security parameter */
  - $(p[0], p[1], \text{state}) \leftarrow A^{D[sk]}(X) \text{ s.t. } |X| \leq q$ ("choose", $pk$, $1^k$);
    /* Adversary $A$, allowed up to $q$ calls to oracle $D$ */
  - $b \in_R \{0,1\}$
  - $c = E_k(p[b])$
  - $b' \leftarrow A^{D[sk]}(X) \text{ s.t. } |X| \leq a \text{ and } c \notin X(\text{"find"}, c, \text{state});$
  - If $b' = b$ and $|p[1]| = |p[0]|$, return win, else return loss;

\[
\text{ADV}_{A,<KG,E,D>,k,q,a}^{\text{CCA-IND}} = \Pr(\text{CCA - IND}^A,<KG,E,D>,k,q,a = \text{"win"}) - \frac{1}{2}
\]
IND-CCA{1/2} Secure PKCS

Let $C = \langle KG, E, D \rangle$ be a PKCS

$ADV^{CCA-IND}_C(k, q, a, t) = \max\{ADV^{CCA-IND}_{A,C,k,q,q}\}$ for $A$ limited to time $t$ (and $q$ pre-choice queries, $a$ adaptive queries)

- Should be negligible for feasible $t, q, a$

Asymptotically: $C$ is IND-CCA2 if for every positive polynomials $p, T$ for `sufficiently long` security parm $k$, $ADV^{CCA-IND}_C(k, q, a, t) < 1/p(k)$ for every $t < T(k)$.

Or: for every Probabilistic Poly Time (PPT) adversary $A$,

$ADV^{CCA-IND}_{A,\langle KG, E, D \rangle, k, q, a} - \frac{1}{2} \in negl$

Where

$negl = \{f : \mathbb{N} \rightarrow \mathbb{R} \mid (\forall poly P(\bullet) > 0)\left(\exists \tilde{k}\left(\forall k > \tilde{k}\right)f(k) < P(k)\right)\}$
Non-Malleability of PKCS [DDN91]

- Bidding application:
  - Government solicit bids by e-mail
  - Bids are sent encrypted (and timestamped)
  - Alice bid is $E_{pub}(b_A)$, Bob’s bid is $E_{pub}(b_B)$, …
  - But Eve wants to bid $E_{pub}(\max(b_A, b_B)+\varepsilon)$

- Can we use any IND-CCA2 PKCS??

- No!
  - Counter example: exercise…
    - Hint: easy for RSA (w/o padding), OTP, hybrids (later).

- Definition, construction: see [DDN91]

- One more motivation for padding RSA…
Outline: Public Key Cryptography

- Motivation and mechanisms
- Modular arithmetic
- Diffie-Hellman Key Agreement
- Public-key cryptosystems
  - Concept, components and syntax
  - El-Gamal
  - RSA
  - Defining secure PKCS
  - Secure usage of RSA (Padding etc.)
- Conclusions
Padding RSA

- Required to...
  - Prevent detection of repeating plaintext
  - Pass distinguishability test, prevent CCA, other attacks
  - Use short $e$ for efficiency (preventing `related message` attack)

- Padding: pre-processing of message and random bits
  - Easily reversible upon decryption

- Common formats: PKCS #1; now at version 2
  - PKCS: Public Key Crypto Standards defined by RSA Inc.
  - Version 1.0 not secure
  - Version 1.5 used in SSL, SSH...

\[
m = \text{Unpad}( D_{d,n}^{RSA} ( E_{e,n}^{RSA} ( \text{Pad}( m, r ) )))
\]
Let $k = |n|$ in bytes

$\text{ValidPad}_{PKCS\#1v1.5}(c) = True$ (padding is valid) if:
- 1st two bytes of $c^d \mod n$ are 0002
- next 8 bytes of $c^d \mod n$ are different from 00

most significant byte

least significant byte

00 02 padding string 00 message

at least 8 bytes

$k$ bytes

Subject to Feedback-only Chosen-Ciphertext Attack (FoCCA) [Bleichenbacher'98]
CCA2 (Adaptive) Distinguishability Test

Key Generate

$ (random)$

$<Pub, Priv>$

$Adversary: x[0], x[1]$  

$y = Encrypt_{Pub}(x[b])$

$Decrypt_{Priv}(c)$  

$Adversary: b' (=b ???)$  

$Decrypt_{Priv}(c)$  

$Adversary: b' (=b ???)$
Feedback-only CCA (FoCCA) Attack

Quite realistic attack!
→ Feedback: only pad Ok / Not
→ Decrypt random message

Key Generate

\[ b \in_R \{0,1\}, \ m \in_R \{0,1\}^n \]

\[ y = \text{Encrypt}_{\text{Pub}}(m) \]

\[ \text{Decrypt}_{\text{Priv}} \]

if `invalid`
Ok otherwise

Adversary tries to find \( m \)

Adversary: \( m' (=m \ ???) \)
Feedback-only Chosen-Ciphertext Attack (FoCCA) [Bleichenbacher’98]

Alice

\[ PK: (n=pq, e) \]

Bob

\[ SK: (p, q, d: ed=1 \mod \varphi(n)) \]

Q:
1. Is this CCA1 or CCA2?
2. Should we require \( c \neq c' \) ?
Feedback-only CCA on Weak Padding

- Weak (PKCS#1v1.5-like) padding:
  - 1st two bytes are 0002
  - The message is next
  - One byte of 00 after the message
  - Followed by at least 8 bytes which are not 00

- $\text{ValidPad}_{\text{WeakPad}}(c) = \text{True}$ (padding is valid) if:
  - 1st two bytes of $c^d \mod n$ are 0002
  - Last 8 bytes of $c^d \mod n$ are different from 00

- Weakness: easy to find from $c = m^e \mod n$ if MSB of $m$ is 02

- How? Let $c' = c \cdot 256^e \mod n$
  - $(c')^d \mod n = (m^e \cdot 256^e)^d \mod n = m \cdot 256$
  - $\text{ValidPad}_{\text{WeakPad}}(c') = \text{True} \Rightarrow \text{MSB}(m) = 02$

<table>
<thead>
<tr>
<th>00</th>
<th>02</th>
<th>message</th>
<th>00</th>
<th>padding string</th>
</tr>
</thead>
</table>

at least 8 bytes
Feedback-only CCA on PKCS1 v1.5

- PKCS 1.5: Padding is correct if:
  - 1st two bytes are 0002
  - next 8 bytes different from 00
- Prob. of correct pad on *random* ciphertext > $0.18 \times 2^{-16}$
- With probability half, a correct pad among 360,000 trials
  - How can we exploit this???
  - Idea: test pad for $c \cdot s_i$ for certain $s_i$
  - Requires: $10^6$ chosen plaintexts for RSA, $2^{40}$ for SSL
  - Details: use RSA’s multiplicative property; see [BI98]

```plaintext
00 02 padding string 00 message
```

at least 8 bytes
RSA Multiplicative Property

- $E_{RSA}(m_1) = m_1^e \mod n$
- $E_{RSA}(m_2) = m_2^e \mod n$
- $E_{RSA}(m_1 m_2) = (m_1 m_2)^e \mod n = (m_1^e \mod n)(m_2^e \mod n) \mod n$
- $E_{RSA}(m_1 m_2) = E_{RSA}(m_1) E_{RSA}(m_2)$

- Used for anonymous cash, voting, other applications
- Does not hold after (PKCS#1) padding
- Allows Chosen Ciphertext Attack:
  - Without padding: Eve selects $r$; `decrypt $(m^e)^r^e$`; receives $m^*r$; finds $m!!$
  - With PKCS#1 v1.5 padding: Feedback-only CCA attack
OAEP & PKCS 1 Version 2.0

- OAEP = Optimal Asymmetric Encryption Padding [BR94]
- Use two cryptographic functions $g, h$
- Security proof in the random oracle model…
Based on Random Oracle model

However, problem found in original argument

Fixed, but… current argument is `wasteful` in security: 1048b RSA → only 40b `effective key length`

Subject of current research for alternatives
  - Preferably using an assumption, not `in random oracle model`

Still, no significant attack against OAEP published so far

→ PKCS #1 ver. 2.0 (OAEP) better than ver. 1.5…
Hybrid Encryption ("enveloping")

- Public key cryptosystems are slow
- Also: most (e.g. RSA) have fixed block size (FIL)
  - And using a long block size is veeeery slooow
- Solution: encrypt (once) a shared key and use it to encrypt (possibly long) plaintext
- Standard technique for Variable-Input Length PKC

\[ \text{Generate key and Encrypt} \]

\[ \text{Encrypt}_{PUB}(key) \]

\[ \text{Decrypt}_{PRIV}(CipherKey) \]

\[ \text{Decrypt}_{key'}(SKCiphertext) \]

- Q: have we seen this?
- A: DH, El-Gamal PKCs
- Exercise: prove security of construction!
Conclusion

- Key agreement – efficient setup of shared key
  - Authenticated channel against man-in-the-middle
- Public key encryption – send encrypted message using only public key
  - Use randomization and padding for security
  - Hybrid encryption (encrypt shared key) for efficiency
- Next: Digital signatures – allow non-repudiation of origin of document
  - Again RSA is the most common mechanism…
Reminder: Random Oracle Method

- Analyze as if \( g(), h() \) are random functions
  - Of course an invalid assumption as \( g(), h() \) are fixed!
  - Whenever \( g() \) or \( h() \) is used, we call the corresponding random-function oracle (black box computing random function)

- Good for screening insecure solutions

- Security under random oracle implies security to many (not all!!) attacks

- Not a complete proof of security, but a good argument/evidence of security.