Lecture 2: Private Key Cryptography

CS 392/6813: Computer Security
Fall 2009

Nitesh Saxena

*Adopted from Previous Lectures by Nasir Memon

Course Administration

- HW#1 due by midnight today
  - Solution will be posted soon
- HW#2 will be posted by the weekend
  - Would be due in a week
- TA for the course: Yasemin Avcular
  - Email: yavcul01@students.poly.edu
  - Office hours: 2-3pm Tuesdays, most probably in LC 257
Outline of today’s lecture

- Cryptography Overview
- Private Key Cryptography: Encryption

Cryptography

- Etymology: Secret (Crypt) Writing (Graphy)
- Study of mathematical techniques to achieve various goals in information security, such as confidentiality, authentication, integrity, non-repudiation, etc.
- Not the only means of providing information security, rather a subset of techniques.
- Quite an old field!
Private Key/Public Key Cryptography

- **Private Key**: Sender and receiver share a common (private) key
  - Encryption and Decryption is done using the private key
  - Also called conventional/shared-key/single-key/symmetric-key cryptography

- **Public Key**: Every user has a private key and a public key
  - Encryption is done using the public key and
    Decryption using private key
  - Also called two-key/asymmetric-key cryptography

Cryptography: Cast of Characters

- Alice (A) and Bob (B): communicating parties
- Eve (E): Eavesdropping (or passive) adversary
- Mallory (M): Man-in-the-Middle (or active adversary)
- Trent (T): a trusted third party (TTP)
Common Terminologies

- Plaintext
- Key
- Encrypt (encipher)
- Ciphertext
- Decrypt (decipher)
- Cipher
- Cryptosystem
- Cryptanalysis (codebreaking)
- Cryptology: Cryptography + Cryptanalysis

Private key model
Open vs Closed Design

- Closed Design (as was followed in military communication during the World War I/II)
  - Keep the cipher secret
  - Also sometimes referred to as the “proprietary design”
  - Bad practice! (why?)

- Open Design (*Kerckhoffs’ principle*)
  - Keep everything public, except the key
  - Good practice – this is what we focus upon!

Private Key Encryption: main functions

1. **KeyGen**: \( K = \text{KeyGen}(l) \) (\( l \) is a security parameter)
2. **Enc**: \( C = \text{Enc}(K,M) \)
3. **Dec**: \( M = \text{Dec}(K,M) \)
Security Model and Adversaries

1. Ciphertext only
   - Adversary knows only the ciphertext(s)

2. Known plaintext
   - Adversary knows a set of plaintext-ciphertext pairs

3. Chosen (and adaptively chosen) plaintext (CPA attack)
   - Adversary chooses a number of plaintexts and obtains the corresponding ciphertexts

4. Chosen (and adaptively chosen) ciphertext attack (CCA attack)
   - Adversary chooses a number of ciphertexts and obtains the corresponding plaintexts

1<2<3<4

<table>
<thead>
<tr>
<th>Hardest attack</th>
<th>easiest attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

1 is the hardest and 4 is the easiest attack to perform.
A cryptosystem secure against 4 is the strongest, and secure against 1 is the weakest.

9/17/09
Lecture 2 - Private Key
Cryptography
Brute Force Attacks: Key Recovery

- Since the key space is finite, given a pair of plaintext and ciphertext, a cryptanalyst can try and check all possible keys.
- For above to be not feasible, key space should be large!!
  - How large?
  - Large enough to make it impractical for an adversary. But what is impractical today, may not be so tomorrow. At least $2^{80}$ – see this paper on “selecting cryptographic key sizes”

Block Ciphers and Stream Ciphers

- Block ciphers partition plaintext into blocks and encrypt each block independently (with the same key) to produce ciphertext blocks.
- A stream cipher generates a *keystream* and encrypts by combining the keystream with the plaintext, usually with the bitwise XOR operation.
- We will focus mostly on Block Ciphers
Ciphers we’ll study today

- Classical ones
  - Substitution Ciphers
    - Caesar’s Cipher
    - Monoalphabetic
    - Polyalphabetic
  - Transposition Ciphers

- Modern ones
  - DES/AES
  - Others...

Caesar Cipher (or Shift Cipher)

- Substitution cipher
- Let messages be all lower case from a through z (no spaces or punctuation).
- Represent letters by numbers from 0 to 25.
- Encryption function
  \[ C_i = E(P_i) = P_i + K \pmod{26} \]
  where K is secret key
- Decryption is
  \[ P_i = D(C_i) = C_i - K \pmod{26} \]
Caesar Cipher (or Shift Cipher)

- Easy to brute force: key space is 26
- Not secure against even ciphertext-only attack (the one where adversary had the least capability)

Monoalphabetic Substitution

```
POKEMONMASTER
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
POLYTECHNIUVRSBKWADEFJMQXZ
```

```
KBUTRBSPDPFTA
```
Monoalphabetic Substitution

- Key space is large $26! = 4 \times 10^{26}$
  - Quite large, however,
  - Can be broken (not secure against ciphertext-only) using language characteristics!

Polyalphabetic Cipher – Vigenere.

- Use $K$ mono-alphabetic ciphers – $E_1$, $E_2$, … $E_k$.
- In position $i$, of plaintext, use cipher $E_i$.
- Example using Caesar ciphers …

  Plaintext: helloiloveyouwontyoutellmeyourname
  Key: polytechnicpolytechnicpolytechnicpoly
  Ciphertext: wswjhmnv………………………………coxc

- A little harder to break but frequency analysis is possible
- Some well known techniques for determining key length – See text.
One time Pad or Vernam Cipher: Best Possible Cipher

- If we use Vigenere with key length as long as plaintext, then cryptanalysis will be difficult!
- If we change key every time we encrypt then cryptanalyst’s job becomes even more difficult. One-time pad or Vernam Cipher.
- How do we get such long keys?
- Such a cipher is difficult to break but not very practical.


- plaintext is binary string and key is binary string of equal length then encryption can be done by a simple xor operation.
  - Plaintext: 01010000010001010011
  - Key: 11010101001001100111
  - Ciphertext: 10000101011000110100
- If the key is random and is not re-used, then such a system offers unconditional security – perfect secrecy!
- Intuitively perfect secrecy can be seen from the fact that given any plaintext and ciphertext, there is a key which maps the selected plaintext to the selected ciphertext. So given a ciphertext, we get no information whatsoever on what key or plaintext could have been used.
- How do we obtain “random” bit-strings for shared secret keys?
- Again system is not practical.
Transposition

Harder to break than substitution ciphers
Still susceptible to frequency analysis

Product Ciphers

Substitution and transposition ciphers are not secure due to language characteristics
What about using two or more of these ciphers
- Two or more substitutions
- Two or more Transpositions
- A few substitutions and a few transposition
  ➔ Transition from classical to modern ciphers
DES – Data Encryption Standard

- Encrypts by series of substitution and transpositions.
- Based on Feistel Structure
- Worldwide standard for more than 20 years.
- Has a history of controversy.
- Designed by IBM (Lucifer) with later help (interference?) from NSA.
- No longer considered secure for highly sensitive applications.
- Replacement standard AES (advanced encryption standard) recently completed.

DES – Overview (Block Operation)
DES – Each Round

DES – Function F
Key Schedule -- KS

- Operation Tables of DES (Key Schedule, PC-1, PC-2)

<table>
<thead>
<tr>
<th>Key schedule of shifts</th>
<th>Key permutation PC-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rot. (s)</td>
<td>No. of shifts</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key permutation PC-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
Operation Tables (IP, IP$^{-1}$, E and P)

Initial Permutation (IP)

<table>
<thead>
<tr>
<th>b1</th>
<th>b2</th>
<th>b3</th>
<th>b4</th>
<th>b5</th>
<th>b6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>10</td>
<td>16</td>
<td>26</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>22</td>
<td>14</td>
<td>18</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>18</td>
<td>14</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>10</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Reverse Initial Permutation (IP$^{-1}$)

<table>
<thead>
<tr>
<th>b1</th>
<th>b2</th>
<th>b3</th>
<th>b4</th>
<th>b5</th>
<th>b6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12</td>
<td>14</td>
<td>22</td>
<td>10</td>
<td>26</td>
</tr>
<tr>
<td>0</td>
<td>22</td>
<td>10</td>
<td>14</td>
<td>12</td>
<td>26</td>
</tr>
<tr>
<td>0</td>
<td>26</td>
<td>22</td>
<td>14</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

S-boxes: S1 (as an example)

<table>
<thead>
<tr>
<th>0000</th>
<th>0001</th>
<th>0010</th>
<th>0011</th>
<th>0100</th>
<th>0101</th>
<th>0110</th>
<th>0111</th>
<th>1000</th>
<th>1001</th>
<th>1010</th>
<th>1011</th>
<th>1100</th>
<th>1101</th>
<th>1110</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>2</td>
<td>15</td>
<td>11</td>
<td>8</td>
<td>3</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>9</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>14</td>
<td>2</td>
<td>13</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>11</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>14</td>
<td>8</td>
<td>13</td>
<td>6</td>
<td>2</td>
<td>11</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>10</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>3</td>
<td>14</td>
<td>10</td>
<td>0</td>
<td>6</td>
<td>13</td>
</tr>
</tbody>
</table>

Is the table entry from \( S(b_1b_2b_3b_4b_5b_6) \)

row : \( b_1b_2 \)

column : \( b_3b_4b_5b_6 \)

\[ S(011001) = 6d = 0110 \]
DES Decryption

- Same as the encryption algorithm with the “reversed” key schedule – NEXT!
DES Security

- S-Box design not well understood (secret).
- Has survived some recent sophisticated attacks (differential cryptanalysis)
- Key is too short. Hence is vulnerable to brute force attack.
- 1998 distributed attack took 3 months.
- $1,000,000$ machine will crack DES in 35 minutes – 1997 estimate. $10,000 – 2.5$ days.
Super-encryption.

- If key length is a concern, then instead of encrypting once, encrypt twice!!
  \[ C = E_{K2}(E_{K1}(P)) \]
  \[ P = D_{K2}(D_{K1}(C)) \]
- Does this result in a larger key space?
- Encrypting with multiple keys is known as super-encryption.
- May not always be a good idea
Double DES

Double DES is almost as easy to break as single DES (Needs more memory though!)

Double DES – Meet-in-the-middle Attack (due to Diffie-Hellman)

- Based on the observation that, if
  
  \[ C = E_{K2}(E_{K1}(P)) \]

  Then
  
  \[ X = E_{K1}(P) = D_{K2}(C). \]

- Given a known \((P, C)\) pair, encrypt \(P\) with all possible values of \(K\) and store result in table \(T\).

- Next, decrypt \(C\) with all possible keys \(K\) and check result. If match occurs then check key pair with new known \((P, C)\) pair. If match occurs, you have found the keys. Else continue as before.

- Process will terminate successfully.
Meet-in-the-middle Explanation.

- The first match does not say anything as we have $2^{64}$ ciphertexts and $2^{112}$ keys.
- On the average $2^{112} / 2^{64} = 2^{48}$ keys will produce same ciphertext.
- So there could be $2^{48}$ possible candidates
- We can use a second pair $(P', C')$
- So, probability that false alarm will survive two known $(P, C)$ pairs is $2^{48} / 2^{64} = 2^{-16}$.
- One can always check a third pair to further reduce the chance of a false alarm.

Triple DES

- Triple DES (2 keys) requires $2^{112}$ search. Is reasonably secure.
- Triple DES (3 keys) requires $2^{112}$ as well
- Which one is better?
Electronic Code Book (ECB) Mode

Although DES encrypts 64 bits (a block) at a time, it can encrypt a long message (file) in Electronic Code Book (ECB) mode.

- If same key is used then identical plaintext blocks map to identical ciphertext.
Cipher Block Chain (CBC) Mode.

CBC – Pros and Cons.

- If IV is different than different instances of same message (or block) will get encrypted differently.
  - IV does not need to be secret
  - IV is sent along with the ciphertext blocks
- What happens if \( k \)th cipher block \( C_k \) gets corrupted in transmission.
  - With ECB – Only decrypted \( P_k \) is affected.
  - With CBC?
    - Only blocks \( P_k \) and \( P_{k+1} \) are affected!!
- What if one plaintext block \( P_k \) is changed?
  - With ECB only \( C_k \) affected.
  - With CBC all subsequent ciphertext blocks will be affected.
    - "Avalanche effect"
    - This leads to an effective integrity protection mechanism (or message authentication code (MAC))
Advanced Encryption Standard (AES)

- National Institute of Science and Technology
  - DES is an aging standard that no longer addresses today's needs for strong encryption
  - Triple-DES: Endorsed by NIST as today's de facto standard
- AES: The Advanced Encryption Standard
  - Finalized in 2001
  - Goal – To define Federal Information Processing Standard (FIPS) by selecting a new powerful encryption algorithm suitable for encrypting government documents
- AES candidate algorithms were required to be:
  - Symmetric-key, supporting 128, 192, and 256 bit keys
  - Royalty-Free
  - Unclassified (i.e. public domain)
  - Available for worldwide export

AES Round-3 Finalist Algorithms:
- MARS
  - Candidate offering from IBM
- RC6
  - Developed by Ron Rivest of RSA Labs, creator of the widely used RC4 algorithm
- Twofish
  - From Counterpane Internet Security, Inc.
- Serpent
  - Designed by Ross Anderson, Eli Biham and Lars Knudsen
- Rijndael: the winner!
  - Designed by Joan Daemen and Vincent Rijmen
Other Symmetric Ciphers and their applications

- IDEA (used in PGP)
- Blowfish (password hashing in OpenBSD)
- RC4 (used in WEP), RC5
- SAFER (used in Bluetooth)

Some Questions

- Enigma is an example of -------- design?
- Encryption can provide confidentiality, but not integrity: true or false?
- World’s best cipher is ---?
- I give you a ciphertext, and ask you to give me the corresponding plaintext – what attack is this? How does it compare to the known plaintext attack?
- All classical ciphers are based on either ---- or ----? Why are they all broken?
- What’s the problem in choosing a long long key? It should give you a lot of security.
Some Questions

- Double encryption in DES increases the key space size from $2^{56}$ to $2^{112}$ – true or false?
- Is known-plaintext an active or passive attack?
- Is chosen-ciphertext attack an active or passive attack?
- Reverse Engineering is applied to what design of systems – open or closed?
- $C=\text{DES}(K,P)$; where (P, C are 64-bit long blocks). What would be $\text{DES}(K,\text{"PPPP"})$ in ECB mode? What it would be in CBC mode?
- Alice needs to send a long encrypted love letter to Bob. Which of the ciphers that we studied today can she use?

AES: Rinjdael

At home reading assignment!
Rijndael

- Joan Daemen (of Proton World International) and Vincent Rijmen (of Katholieke Universiteit Leuven).
- (pronounced “Rhine-doll”)
- Allows only 128, 192, and 256-bit key sizes (unlike the other candidates)
- Variable block length of 128, 192, or 256 bits. All nine combinations of key/block length possible.
- A block is the smallest data size the algorithm will encrypt
- Vast speed improvement over DES in both hardware and software implementations
  - 8416 bytes/sec on a 20MHz 8051 (@ 12 CPI)
  - 8.8 Mbytes/sec on a 200MHz Pentium Pro

Rijndael Structure

- Rijndael consists of
  - an initial Round Key addition;
  - $N_r$-1 Rounds;
  - a final round.
- In pseudo C code, this gives:

```c
Rijndael(State,CipherKey) {
    KeyExpansion(CipherKey,ExpandedKey) ;
    AddRoundKey(State,ExpandedKey);
    For( i=1 ; i<Nr ; i++ )
        Round(State,ExpandedKey + Nb*i) ;
    FinalRound(State,ExpandedKey + Nb*Nr);
}
```
Rijndael

- Key is expanded to a set of n round keys
- Input block X undergoes n rounds of operations (each operation is based on value of the nth round key), until it reaches a final round.
- Strength relies on the fact that it's difficult to obtain the intermediate result (or state) of round n from round n+1 without the round key.

Number of Rounds

- Number of rounds (Nr) as a function of the block (Nb) and key length (Nk) in 32 bit words.

<table>
<thead>
<tr>
<th>Nr</th>
<th>Nb = 4</th>
<th>Nb = 6</th>
<th>Nb = 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nk = 4</td>
<td>10</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Nk = 6</td>
<td>12</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Nk = 8</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>
Each round performs the following operations:
- Non-linear Layer: No linear relationship between the input and output of a round
- Linear Mixing Layer: Guarantees high diffusion over multiple rounds
- Key Addition Layer: Bytes of the input are simply XOR’ed with the expanded round key

Three layers provide strength against known types of cryptographic attacks: Rijndael provides “full diffusion” after only two rounds
- Linear and differential cryptanalysis
- Known-key and related-key attacks
- Square attack
- Interpolation attacks
- Weak-keys

Rijndael has been shown to be K-secure:
- No key-recovery attacks faster than exhaustive search exist
- No known symmetry properties in the round mapping
- No weak keys
- No related-key attacks: No two keys have a high number of expanded round keys in common
Rijndael: ByteSub

- Each byte at the input of a round undergoes a non-linear byte substitution according to the following:
  1. First, taking the multiplicative inverse in GF(2^8). '00' is mapped onto itself.
  2. Then, applying an affine (over GF(2)) transformation.

Substitution ("S")-box

Affine Transform

Rijndael: ShiftRow

Depending on the block length, each "row" of the block is cyclically shifted according to the above table.
Rijndael: MixColumn

Each column is multiplied by a fixed polynomial
\[ C(x) = '03' \cdot X^3 + '01' \cdot X^2 + '01' \cdot X + '02' \]

This corresponds to matrix multiplication \( b(x) = c(x) \otimes a(x) \):

\[
\begin{bmatrix}
  b_0 \\
  b_1 \\
  b_2 \\
  b_3 \\
\end{bmatrix} =
\begin{bmatrix}
  02 & 03 & 01 & 01 \\
  01 & 02 & 03 & 01 \\
  01 & 01 & 02 & 03 \\
  03 & 01 & 01 & 02 \\
\end{bmatrix}
\begin{bmatrix}
  a_0 \\
  a_1 \\
  a_2 \\
  a_3 \\
\end{bmatrix}
\]

Rijndael: Key Expansion and Addition

Each word is simply XOR'ed with the expanded round key

Key Expansion algorithm:

```c
KeyExpansion(int* Key[4*Nk], int* EKey[Nb*(Nr+1)])
{
  for(i = 0; i < Nk; i++)
    EKey[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
  for(i = Nk; i < Nb * (Nr + 1); i++)
  {
    temp = EKey[i - 1];
    if (i % Nk == 0)
      temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
    EKey[i] = EKey[i - Nk] ^ temp;
  }
}
```
Rijndael: Implementations

- Rijndael is well suited for software implementations on 8-bit processors (important for “Smart Cards”)
  - Operations focus on bytes and nibbles, not 32 or 64 bit integers
  - Layers such as ByteSub can be efficiently implemented using small tables in ROM (e.g. < 256 bytes).
  - No special instructions are required to speed up operation
- For 32-bit implementations:
  - An entire round can be implemented via a fast table lookup routine on machines with 32-bit or higher word lengths
  - Considerable parallelism exists in the algorithm
    - Each layer operates in a parallel manner on bytes of the round state, all four component transforms act on individual parts of the block
    - Although the Key expansion is complicated and cannot be parallelised, it only needs to be performed once until the two parties switch keys.

Rijndael: Implementations

- Hardware Implementations
  - Performs very well in software, but in some cases more performance is required (e.g. server and VPN applications).
  - Multiple S-Box engines, round-key EXORs, and byte shifts can all be implemented efficiently in hardware when absolute speed is required
  - Small amount of hardware can vastly speed up 8-bit implementations
- Inverse Cipher
  - Except for the non-linear ByteSub step, each part of Rijndael has a straightforward inverse and the operations simply need to be undone in the reverse order.
  - Same code that encrypts a block can also decrypt the same block simply by changing certain tables and polynomials for each layer. The rest of the operation remains identical.
Rijndael Future

- Rijndael is an extremely fast, state-of-the-art, highly secure algorithm
- Has efficient implementations in both hardware and software; it requires no special instructions to obtain good performance on any computing platform
- Despite being the chosen by NIST as the AES candidate winner, Rijndael is not yet automatically the new encryption standard
  - Triple-DES, still highly secure and supported by NIST, is expected to be common for the foreseeable future.

Further Reading

- Read text 8.1, 8.2, 10.1, and 10.2.
- North American Crypto archive http://cryptography.org/
- Ron Rivest’s crypto page http://theory.lcs.mit.edu/~rivest/crypto-security.html
- Cryptography archive: http://www.austinlinks.com/Crypto/
- AES home page http://csrc.nist.gov/encryption/aes/