CS 392/681 – Computer Security

Module 1 - Private Key Cryptography
Logistics

- Office hours Thursday 3 to 5 (tentative).
- Lab 0 due today.
- Lab 1 assigned. Due next Thursday!!
- ISIS is still unstable. Will fix things in a week.
- Out of town this week. Will be back just a few hours before class on next Thursday.
- Out of town again on Friday and gone for 9 days. Class will be held as usual.
- My office hours will be taken by Kulesh.
Data Encryption

- Encryption is the process of encoding a message such that its meaning is not obvious.
- Decryption is the reverse process, i.e., transforming an encrypted message to its original form.

We denote plaintext by P and ciphertext by C.

\[ C = E(P), \quad P = D(C) \text{ and } P = D(E(P)), \] where E() is the encryption function (algorithm) and D() the decryption function.
Kerckhoff’s Principle

- How do you prevent and eavesdropper from computing P, given C?
  - Keep the encryption algorithm E() secret.
    - BAD IDEA!!
  - Choose E() (and corresponding D()) from a large collection, based on secret key.
    - GOOD IDEA!! Kerckhoff’s principle.

C = E(K, P) and P = D(K, C)
Symmetric and Asymmetric Cryptosystems

- Just by changing key we have different encryptions of one plaintext.
- If the encryption key and the decryption key are the same then we have a symmetric encryption scheme (also private key, one-key).
- If the encryption key and the decryption key are different then we have an asymmetric encryption scheme (also public key, two-key).
- A cryptosystem then a five-tuple consisting of 1) The set of all plaintexts 2) The set of all ciphertexts 3) The set of all keys 4) A family of encryption functions 5) A family of decryption functions.
Example – Caesar Cipher

- Let messages be all lower case from a through z (no spaces or punctuation).
  itsnotthathardtoread
- Represent letters by numbers from 0 to 25.
- Encryption function
  \[ C_i = E(P_i ) = P_i + K. \]
  where K is secret key and addition done modulo 26.
- Decryption is
  \[ P_i = D(C_i ) = C_i - K. \]
- UNIX ROT13 uses K as 13.
Cryptanalysis

- A cryptosystem had to be secure against the following kinds of attacks:
  - Ciphertext only attack.
  - Known plaintext attack.
  - Chosen plaintext attack.
  - Adaptive chosen plaintext attack.
  - Chosen ciphertext attack.
  - Chosen key attack.

- Of course there is one attack against which no cryptosystem can offer protection – rubber hose attack.
Brute Force Attacks.

- Since the key space is finite, given a ciphertext a cryptanalyst can try and check all possible keys.
- For above to be not feasible, key space should be large!!
  - How large? How about $2^{56}$?
- Large enough to make it impractical for an adversary. But what is impractical today, may not be so tomorrow.
- In practice, for a “good” cryptosystem, the only possible attack should be the brute force attack, which should be impractical into the foreseeable future, as long as message may have value.
Substitution Ciphers

- Basic idea - substitute each block of plaintext by a different block.
- If plaintext is English then
  - Mona-alphabetic substitution.
  - Poly-alphabetic substitution.
- If plaintext is binary string then map one block of bits to another.

Plaintext: 0011010101010001 ... 10100101
Ciphertext: 0100010000011100 ... 00101001

- This is called block encryption. Very common.
Encryption by Mono-alphabetic Substitution.

Key space is large 26! (How do you remember a key? See example).

However, mono-alphabetic substitution is easy to break as it preserves source first order statistics.

Large key space is necessary but not sufficient condition for security!
Encryption by Poly-alphabetic Substitution.

- Encrypt plaintext a pair at a time. Two letters specify a rectangle. Substitute by opposite corner pair. Eg: VX -> SM.
- If they fall in same row or column, then using next pair in circular manner. Eg: LY -> TP.
- Repeated letters are broken by filler letter.
- I/J chosen randomly.

• Playfair cipher.
• Used by British Army in WW1 and WW2.
• Can be broken easily today with only a ciphertext of length about 100.
Poly-alphabetic Cipher – Vigenere.

- Use \( K \) mono-alphabetic ciphers – \( E_1, E_2, \ldots, E_k \).
- In position \( i \) of plaintext, use cipher \( E_i \).
- Example using Caesar ciphers ...

Plaintext:   helloiloveyouwontyoutellmeyourname
Key:          polytechnicpolytechnicpolytechnicpoly
Ciphertext: wswjhmnnv........................................coxc

- A little harder to break but trivial once you know key length!
- Some well known techniques for determining key length – See text.
Vernam - The Perfect Substitution 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?
  - A large book shared by transmitter and receiver.
  - Initial key followed by previous messages themselves!!
  - Random number sequence based on common shared and secret seed.
- Such a cipher is difficult to break but not very practical.

- If 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 plaintexts uniformly distributed and keys are random then such a system offers unconditional security – perfect secrecy! (Under the right mathematical formulation and assumptions).

- 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.
Perfect Secrecy

- Given a ciphertext, cryptanalyst cannot reduce uncertainty.
- Property can be formulated in more mathematically rigorous manner.

Four possible Keys $K_1, K_2, K_3, K_4$
Imperfect Secrecy

- Given $C_1$ we know Message is $M_3$!!
- Given $C_5$, only one bit of uncertainty.
### Encryption by Transposition

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>3</td>
<td>9</td>
<td>11</td>
<td>12</td>
<td>4</td>
<td>5</td>
<td>13</td>
</tr>
</tbody>
</table>

- Harder to break than substitution ciphers
- Preserve first order statistics
- One can arrange plaintext in table and sort rows and columns.
Product Ciphers

- To get improved security one can encrypt the ciphertext again.
- If one uses same algorithm – super encryption. May or may not be useful. For example, super-encryption with Caesar cipher is as good as single encryption!
- If one uses different algorithms – product cipher.
- Product ciphers based on sequence of substitutions and transpositions are very popular.
- You will see one later – DES.
Shannon Characteristics of Good Ciphers

- The amount of secrecy needed should determine the amount of labor appropriate for encryption and decryption.
- The set of keys and enciphering algorithms should be free from complexity.
- The implementation of the process should be as simple as possible.
- Errors in ciphering should not propagate and cause corruption of future information in the message.
- The size of enciphered text should be no longer than the text of the original message.
Confusion and Diffusion

- Confusion: The cryptanalyst should not be able to predict what changing one character in the plaintext will do to the ciphertext.
- Diffusion: Changes in the key should affect many parts in the ciphertext.
DES - Data Encryption Standard

- Private key. Encrypts by series of substitution and transpositions.
- 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) recently completed.
DES - Overview

Module 1 - Private Key Crypto
DES – Each iteration.
DES - Computation of $F(R_{i-1}, K_i)$

- $R$ (32 bits) through $E$ to 48 bits
- 48 bits + $K$ (48 bits)
- Through $S_1$ to $S_8$, each S-box
- Resized to 32 bits

Module 1 - Private Key Crypto
Computation of F:

- Expansion function $E$:
  - maps bit string of length 32 to bit string of length 48.
  - Permutates bits in a fixed way and duplicates certain bits
- Key schedule: each round uses a 48 bit key obtained by performing permutations, shifts, and discarding bits from the original 56 bit key. Fixed algorithm for each round
- resulting 48 bit string broken into 8 6-bit strings
S-boxes: S1

<table>
<thead>
<tr>
<th>14</th>
<th>4</th>
<th>13</th>
<th>1</th>
<th>2</th>
<th>15</th>
<th>11</th>
<th>8</th>
<th>3</th>
<th>10</th>
<th>6</th>
<th>12</th>
<th>5</th>
<th>9</th>
<th>0</th>
<th>7</th>
</tr>
</thead>
<tbody>
<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>

$S_{ij}$ is the table entry from $S(b_1b_2b_3b_4b_5b_6)$

- row: $b_1b_2$
- column: $b_3b_4b_5b_6$

$S(011001) = table[1,9] = 6_d = 0110$
Plain text

Initial permutation (IP)

Round-1 (key $K_1$)

Rounds 2-15

Round-16 (key $K_{16}$)

swap

IP inverse

Cipher text
\[
\begin{array}{c|c}
L_{15} \oplus F(R_{15}, K_{16}) & R_{15} \\
\hline
\end{array}
\]

**IP inverse**

\[
{\text{Cipher text}}
\]

\[
\begin{array}{c|c}
L_{15} \oplus F(R_{15}, K_{16}) & R_{15} \\
\hline
\end{array}
\]

**IP**

\[
\begin{array}{c|c}
L_{15} \oplus F(R_{15}, K_{16}) & R_{15} \\
\hline
\end{array}
\]

**Round-1 (K_{16})**

\[
\begin{array}{c|c}
R_{15} & L_{15} \oplus F(R_{15}, K_{16}) \oplus F(R_{15}, K_{16}) \\
\hline
\end{array}
\]

= 

\[
\begin{array}{c|c}
R_{15} & L_{15} \\
\hline
\end{array}
\]

**Since**  
\[b \oplus b = 0\]

\[b \oplus 0 = b\]
Encryption Modes - ECB.

- 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.
Block Ciphers and Stream Ciphers

- Block ciphers partition plaintext into blocks and encrypt each block independently (with the same key) to produce ciphertext blocks.
  - Caesar, Vigenere, DES etc.
- A stream cipher generates a keystream and encrypts by combining the keystream with the plaintext, usually with the bitwise XOR operation.
- Generation of the keystream can be independent of the plaintext and ciphertext (a synchronous stream cipher) or it can depend on the data and its encryption (self-synchronizing stream cipher).
In their simplest form, a stream ciphers convert plaintext to ciphertext 1 bit at a time.

- A keystream generator outputs a stream of bits: $k_1$, $k_2$, ..., $k_n$. 
Stream Ciphers - Security Issues

- The security of this approach depends totally on the keystream generator.
  - If it outputs an endless stream of zeros, the ciphertext will equal the plaintext and the operation will be meaningless.
  - If it outputs an endless stream of random bits you have a one-time pad and perfect security.
- The reality lies between the simple XOR and the one-time pad.
- So how do we generate random bitstreams?
  - One way is Linear Feedback Shift Register (LFSR)
  - Another way is using a block cipher technique in a “feedback” mode.
Generating Random Key Streams

- Linear feedback shift registers (LFSR)

- $b_i$ are bits and $f$ is a function depending of $b_1, b_2, ..., b_n$. The result $f(b_1, b_2, ..., b_n)$ enters the $n$th position of the register and all other bits $b_n, ..., b_r$ are shifted right one position, $b_1$ being the output bit.
Example LFSR

- Let $n = 4$ and $f(b_1, b_2, b_3, b_4) = b_1 \oplus b_4$. Let 0110 be the starting position (the seed). Then this LFSR will generate the sequence 011001000111101...

- Stream ciphers based on LFSR’s are well studied
Cipher Block Chain (CBC) Mode.

Module 1 - Private Key Crypto
CBC – Pros and Cons.

- If IV is different then different instances of same message (or block) will get encrypted differently.
  - How does receiver know IV?
    - Choose at random and send encrypted as first block.
- 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!!
    - This can also allow some message tampering!
- What if one plaintext block $P_K$ is changed?
  - With ECB only $C_K$ affected.
  - With CBC all subsequent ciphertext blocks will be affected.
  - This leads to an effective MAC based on DES CBC.
Cipher Feedback Mode (CFB).
CFB Properties

- \( J \) is normally 8.
- Change in one plaintext bit is going to affect all subsequent ciphertext bits. So can be used for MAC.
- Change in ciphertext bit results in???
Output Feedback Mode (OFB).

Figure 3.14 J-Bit Output Feedback (OFB) Mode
OFB Properties.

- Bit errors in transmission do not propagate.
- One can selectively flip ciphertext bits to flip corresponding decrypted plaintext bits. Bad!!
DES Security

- S-Box design not well understood (secret).
- Has survived some recent sophisticated attacks (differential cryptanalysis).
- Key is too short (thanks to NSA!). 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.
- In 1999 EFF achieved 245 billion keys per second rate to crack in 22 hours.
DES Cracking machine
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? That is a new mapping that could not have been obtained by a single key?
  - With Caesar cipher - NO!
  - With DES - yes!
- 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.

- Based on the observation that, if
  \[ C = E_{K_2}(E_{K_1}(P)) \]
  Then
  \[ X = E_{K_1}(P) = D_{K_2}(C). \]
- 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 $2^{48}$ false alarms.
- However, with second known (P, C) pair, probability that $E_{K1}(P) = D_{K2}(C)$ is $2^{-64}$.
- 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 (2 keys) requires $2^{112}$ search. Is reasonably secure.

3 keys requires $2^{168}$.
AES History

- 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 defacto 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
**History (cont.)**

- **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**
    - Designed by Joan Daemen and Vincent Rijmen
Rijndael

• The Winner: 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;
  - **Nr**-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>
Rijndael

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
  - Very small correlation between bytes of the round input and the bytes of the output
- **Key Addition Layer**: Bytes of the input are simply XOR’ed with the expanded round key
Rijndael

- 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 $\text{GF}(2^8)$. ‘00’ is mapped onto itself.
  2. Then, applying an affine (over $\text{GF}(2)$) transformation.
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
\[ C(x) = '03'X^3 + '01'X^2 + '01'X + '02' \]

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

```
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.
Other Private Key Cryptosystems

- IDEA
- Twofish
- Blowfish
- RC4, RC5, RC6
- Serpent
- MARS
- Feal