CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Preview Lecture 6
September 18, 2017
Padding

- Bit padding
- Byte padding

Data Encryption Standard (DES)

Multiple Encryption

- Composition
- Group property
- Birthday attack
Padding
Padding

Block ciphers are designed to handle sequences of blocks.

To send a message $m$ of arbitrary length, it must first be *encoded* into a new message $m'$ that consists of a sequence of blocks.

$m'$ is then encrypted, transmitted, and decrypted.

After decrypting, $m'$ must be *decoded* to recover the original message $m$.

A *padding rule* describes the encoding and decoding process.
How to pad

An obvious padding rule is to append 0’s to the end of $m$ until its length is a multiple of the block length $b$.

Unfortunately, this can’t be properly decoded, since the receiver does not know how many 0’s to discard from $m'$. 

**Condition:** A padding rule must describe how much padding was added.

**Suggestions?**
Some easy padding rules

1. Pad with 0’s, then *prepend* a block containing the length of $m$.  
   Example: $b = 8$, $m = 01011$, $m' = 00000101 \text{ 01011000}$. 
   Drawback: Must know the length of $m$ before beginning.

2. Pad with 0’s, then *append* a block containing the length of $m$.  
   Example: $b = 8$, $m = 01011$, $m' = 01011000 \text{ 00000101}$.  
   Drawback: Wasteful of space

3. Pad with a single 1 bit followed by 0’s. 
   Example: $b = 8$, $m = 01011$, $m' = 01011100$.  
   Drawback: Need to count bits.

What happens if the length of $m$ is already a multiple of $b$?
Compact bit padding

Here’s a padding rule that is both space efficient and easy to decode.

- Choose $\ell = \lceil \log_2 b \rceil$. This is the number of bits needed to represent (in binary) any number in the interval $[0 \ldots (b - 1)]$.
- Choose $p$ as small as possible so that $|m| + p + \ell$ is a multiple of $b$.
- Pad $m$ with $p$ 0’s followed by a length $\ell$ binary representation of $p$.

To unpad, interpret the last $\ell$ bits of the message as a binary number $p$; then discard a total of $p + \ell$ bits from the right end of the message.
Bit padding

For arbitrary bit strings, append $p$ 0’s to the message followed by an $\ell$-bit number whose value is $p$.

$\ell$ must be big enough to represent any value for $p$ between 0 and $b - 1$, so choose $\ell = \lceil \log_2 b \rceil$.

Choose $p$ so that $|m| + p + \ell$ is a multiple of $b$.

The padded message is $m \cdot 0^p \cdot \overline{p}$, where $\overline{p}$ is the binary representation of $p$. 
Bit padding examples

1. $b = 8, m = 01011, m' = 010110000$.  
2. $b = 8, m = 1110, m' = 11100001$.  
3. $b = 8, m = 111, m' = 11100010$.  
4. $b = 8, m = 010110, m' = 0101100000000111$. 
Bit padding on 64-bit blocks

- At most 63 0’s ever need to be added, so a 6-bit length field is sufficient.
- A message $m$ is then padded to become $m' = m \cdot 0^p \cdot \bar{p}$, where $\bar{p}$ is the 6-bit binary representation of $p$.
- $p$ is chosen as small as possible so that $|m'| = |m| + p + 6$ is a multiple of 64.
### Block codes on byte strings

Often messages and blocks consist of a sequence of 8-bit bytes.

In that case, padding can be done by adding an integral number of bytes to the message.

At least one byte is always added to avoid ambiguity.
PKCS7 padding

PKCS7 #7 is a message syntax described in internet RFC 2315.

- Fill a partially filled last block having $k$ “holes” with $k$ bytes, each having the value $k$ when regarded as a binary number.
- If $k = 0$, an empty block is added before padding.

Example: Block length = 8 bytes.
$m = “\text{hello}”.$
$m' = 68 \ 65 \ 6C \ 6C \ 6F \ 03 \ 03 \ 03.$

On decoding, if the last block of the message does not have this form, then a decoding error is indicated.

Example: The last block cannot validly end in $\ldots 25 \ 00 \ 03.$

What is the last block if $k = 0$?
Possible information leakage from padding

Suppose Alice uses AES (block length 128) in ECB mode to send 129-bit messages.

Eve has a plaintext-ciphertext pair \((m', c')\) and intercepts a new cipher text \(c\) for an unknown message \(m\).

Because of padding, both \(c\) and \(c'\) are two blocks long. Let \(c_2\) and \(c'_2\) be the second blocks of each, respectively.

Then the last bit of \(m\) is the same as the last bit of \(m'\) iff \(c_2 = c'_2\), so Eve learns the last bit of \(m\).
Data Encryption Standard (DES)
Data encryption standard (DES)

The Data Encryption Standard is a block cipher that operates on 64-bit blocks and uses a 56-bit key.

It became an official Federal Information Processing Standard (FIPS) in 1976. It was officially withdrawn as a standard in 2005 after it became widely acknowledged that the key length was too short and it was subject to brute force attack.

Nevertheless, triple DES (with a 112-bit key) is approved through the year 2030 for sensitive government information.

Feistel networks

DES is based on a *Feistel network*.

This is a general method for building an invertible function from any function $f$ that scrambles bits.

It consists of some number of stages.

- Each stage $i$ maps a pair of $n$-bit words $(L_i, R_i)$ to a new pair $(L_{i+1}, R_{i+1})$. ($n = 32$ in case of DES.)
- By applying the stages in sequence, a $t$-stage network maps $(L_0, R_0)$ to $(L_t, R_t)$.
- $(L_0, R_0)$ is the plaintext, and $(L_t, R_t)$ is the corresponding ciphertext.
A Feistel network

One stage

Each stage works as follows:

\[ L_{i+1} = R_i \]  \hspace{1cm} (1)

\[ R_{i+1} = L_i \oplus f(R_i, K_i) \]  \hspace{1cm} (2)

Here, \( K_i \) is a subkey, which is generally derived in some systematic way from the master key \( k \), and \( f \) is the scrambling function (shown as \( F \) in the diagram).

The inversion problem is to find \((L_i, R_i)\) given \((L_{i+1}, R_{i+1})\). Equation 1 gives us \( R_i \). Knowing \( R_i \) and \( K_i \), we can compute \( f(R_i, K_i) \). We can then solve equation 2 to get

\[ L_i = R_{i+1} \oplus f(R_i, K_i) \]
Properties of Feistel networks

The security of a Feistel-based code lies in the construction of the scrambling function $f$ and in the method for producing the subkeys $K_i$.

The invertibility follows just from properties of $\oplus$ (exclusive-or).
DES Feistel network

DES uses a 16 stage Feistel network.

The pair $L_0R_0$ is constructed from a 64-bit message by a fixed initial permutation IP.

The ciphertext output is obtained by applying $IP^{-1}$ to $R_{16}L_{16}$.

The scrambling function $f(R_i, K_i)$ operates on a 32-bit data block and a 48-bit key block. Thus, $48 \times 16 = 768$ key bits are used.

The key bits are all derived in a systematic way from the 56-bit primary key and are far from independent of each other.


S-boxes

The scrambling function $f(R_i, K_i)$ is the heart of DES.

At the heart of the scrambling function are eight “S-boxes” that compute Boolean functions with 6 binary inputs

$$c_0, x_1, x_2, x_3, x_4, c_1$$

and 4 binary outputs $y_1, y_2, y_3, y_4$.

Each computes some fixed function in $\{0, 1\}^6 \rightarrow \{0, 1\}^4$.

The eight S-boxes are all different and are specified by tables.
Special properties of S-boxes

For fixed values of \((c_0, c_1)\), the resulting function on inputs \(x_1, \ldots, x_4\) is a permutation from \(\{0, 1\}^4 \rightarrow \{0, 1\}^4\).

Hence, can regard an S-box as performing a substitution on four-bit “characters”, where the substitution performed depends both on the structure of the particular S-box and on the values of its “control inputs” \(c_0\) and \(c_1\).

Thus, DES’s 8 S-boxes are capable of performing 32 different substitutions on 4-bit fields.
DES scrambling network

Figure 4.5: The DES Function $f(R_{i-1}, K_1)$. 
Connecting the boxes

The S-boxes together have a total of 48 input lines.

Each of these lines is the output of a corresponding $\oplus$-gate.

- One input of each of these $\oplus$-gates is connected to a corresponding bit of the 48-bit subkey $K_i$. (This is the only place that the key enters into DES.)
- The other input of each $\oplus$-gate is connected to one of the 32 bits of the first argument of $f$.

Since there are 48 $\oplus$-gates and only 32 bits in the first argument to $f$, some of those bits get used more than once.

The mapping of input bits to $\oplus$-gates is called the *expansion permutation* $E$. 
Expansion permutation

The expansion permutation connects input bits to $\oplus$ gates. We identify the $\oplus$ gates by the S-box inputs to which they connect.

- Input bits 32, 1, 2, 3, 4, 5 connect to the six $\oplus$ gates that go input wires $c_0, x_1, x_2, x_3, x_4, c_1$ on S-box 1.
- Bits 4, 5, 6, 7, 8, 9 are connect to the six $\oplus$ gates that go input wires $c_0, x_1, x_2, x_3, x_4, c_1$ on S-box 2.
- The same pattern continues for the remaining S-boxes.

Thus, input bits 1, 4, 5, 8, 9, ... 28, 29, 32 are each used twice, and the remaining input bits are each used once.
Connecting the outputs

The 32 bits of output from the S-boxes are passed through a fixed permutation $P$ (transposition) that spreads out the output bits.

The outputs of a single S-box at one stage of DES become inputs to several different S-boxes at the next stage.

This helps provide the desirable “avalanche” effect, where a change in one input bit spreads out through the network and causes many output bits to change.
Obtaining the subkey

The scrambling function operates on a 32-bit data block and a 48-bit key block (called a subkey).

The 56-bit master key $k$ is split into two 28-bit pieces $C$ and $D$. At each stage, $C$ and $D$ are rotated by one or two bit positions. Subkey $K_i$ is then obtained by applying a fixed permutation (transposition) to $CD$. 
Security considerations

DES is vulnerable to a brute force attack because of its small key size.

However, it has turned out to be remarkably resistant to recently-discovered (in the open world) sophisticated attacks.

**Differential cryptanalysis:** Can break DES using “only” $2^{47}$ chosen ciphertext pairs.

**Linear cryptanalysis:** Can break DES using $2^{43}$ chosen plaintext pairs.

Neither attack is feasible in practice.
Multiple Encryption
Composition of cryptosystems

We formally defined the composition of two ciphers in Lecture 5. Encrypting a message multiple times with the same or different ciphers and keys seems to make the cipher stronger, but that's not always the case. The security of the composition can be difficult to analyze.

For example, with the one-time pad, the encryption and decryption functions $E_k$ and $D_k$ are the same. The composition $E_k \circ E_k$ is the identity function!
Composition within practical cryptosystems

Practical symmetric cryptosystems such as DES and AES are built as a composition of simpler systems.

Each component offers little security by itself, but when composed, the layers obscure the message to the point that it is difficult for an adversary to recover.

The trick is to find ciphers that successfully hide useful information from a would-be attacker when used in concert.
Double Encryption

*Double encryption* is when a cryptosystem is composed with itself. Each message is encrypted twice using two different keys $k'$ and $k''$, so $E^2_{(k'', k')} = E_{k''} \circ E_{k'}$ and $D^2_{(k'', k')} = D_{k'} \circ D_{k''}$.

$(E, D)$ is the *underlying* or *base* cryptosystem and $(E^2, D^2)$ is the *doubled* cryptosystem. $\mathcal{M}$ and $\mathcal{C}$ are unchanged, but $\mathcal{K}^2 = \mathcal{K} \times \mathcal{K}$.

The size of the keyspace is squared, resulting in an apparent doubling of the effective key length and making a brute force attack much more costly.

However, *it does not always increase the security* of a cryptosystem as much as one might na"îvely think, for other attacks may become possible.
Example: Double Caesar

Consider Double Caesar, the Caesar cipher composed with itself.

It has $26^2 = 676$ possible key pairs $(k'', k')$. One might hope that double Caesar is more resistant to a brute force attack.

Unfortunately, still only 26 possible distinct encryption functions and only 26 possible decryptions of each ciphertext.

This is because $E_{(k'', k')}^2 = E_k$ for $k = (k' + k'') \mod 26$.

Any attack on the Caesar cipher will work equally well on the Double Caesar cipher. To the attacker, there is no difference between the two systems. Eve neither knows nor cares how Alice actually computed the ciphertext; all that matters are the probabilistic relationships between plaintexts and ciphertexts.
Group property

Let $(E, D)$ be a cryptosystem for which $\mathcal{M} = \mathcal{C}$. Each $E_k$ is then a permutation on $\mathcal{M}$.\(^1\)

The set of all permutations on $\mathcal{M}$ forms a group.\(^2\)

**Definition**

$(E, D)$ is said to have the group property if the set of possible encryption functions $\mathcal{E} = \{E_k \mid k \in \mathcal{K}\}$ is closed under functional composition $\circ$.

That is, if $k', k'' \in \mathcal{K}$, then there exists $k \in \mathcal{K}$ such that

$$E_k = E_{k''} \circ E_{k'}.$$  

\(^1\)A permutation is one-to-one and onto function.

\(^2\)A group has an associative binary operation with an identity element, and each element has an inverse.
Cryptosystems with group property

We’ve seen that the Caesar cipher has the group property.

When $\mathcal{E}$ is closed under composition, then $(\mathcal{E}, \circ)$ is a subgroup of all permutations on $\mathcal{M}$. In this case, double encryption adds no security against a brute force attack.

Even though the key length has doubled, the number of distinct encryption functions has not increased, and the double encryption system will fall to a brute force attack on the original cryptosystem.
Birthday Problem

The *birthday problem* is to find the probability that two people in a set of randomly chosen people have the same birthday.

This probability is greater than 50% in any set of at least 23 randomly chosen people.

23 is far less than the 253 people that are needed for the probability to exceed 50% that at least one of them was born on a specific day, say January 1.
Birthday attack on a cryptosystem

A *birthday attack* is a known plaintext attack on a cryptosystem that reduces the number of keys that must be tried to roughly the square root of what a brute force attack needs.

For example, if the original key length was 56 (as is the case with DES), then only about $\sqrt{2^{56}} = 2^{28}$ keys need to be tried.

Any cryptosystem with the group property is subject to a birthday attack.
How a birthday attack works

Assume \((m, c)\) is a known plaintext-ciphertext pair, so \(E_{k_0}(m) = c\) for Alice’s secret key \(k_0\).

- Choose \(2^{28}\) random keys \(k_1\) and encrypt \(m\) using each.
- Choose another \(2^{28}\) random keys \(k_2\) and decrypt \(c\) using each.
- Look for a common element \(u\) in these two sets.
- Suppose one is found for \(k_1\) and \(k_2\), so \(E_{k_1}(m) = u = D_{k_2}(c)\).

It follows that \(E_{k_2}(E_{k_1}(m)) = c\), so we have succeeded in finding a key pair \((k_2, k_1)\) that works for the pair \((m, c)\).

There is a key \(k\) such that \(E_k = E_{k_2} \circ E_{k_1}\) since we are assuming the group property, so \(E_k(m) = c\).
How a birthday attack works (cont.)

Alice’s key $k_0$ also has $E_{k_0}(m) = c$. If it happens that $E_k = E_{k_0}$, then we have broken the cryptosystem.

We do not need to find $k$ itself since we can compute $E_k$ from $E_{k_1}$ and $E_{k_2}$ and $D_k$ from $D_{k_1}$ and $D_{k_2}$.

There are unlikely to be many distinct keys $k$ such that $E_k(m) = c$, so with significant probability we have cracked the system. (For Caesar, there is only one such $k$.)

Using additional plaintext-ciphertext pairs, we can increase our confidence that we have found the correct key pair. Repeat this process if we have not yet succeeded.

I’ve glossed over many assumptions and details, but that’s the basic idea.
Weaknesses of the birthday attack

The drawback to the birthday attack (from the attacker’s perspective) is that it requires a lot of storage in order to find a matching element.

Nevertheless, if DES were a group, this attack could be carried out in about a gigabyte of storage, easily within the storage capacity of modern workstations.

(We will see later that DES is not a group.)