3-Des and Hash Functions and Long messages

**Des – 56bit Key**

64-bit input $\rightarrow$ Des $\rightarrow$ 64 Output

Takes $2^{56}$ Attempts to brute force. Major issue, with cpu speeds today we can crack $2^{56}$ items easily so it is not secure enough.

**2Des**

Input $\rightarrow$ Des(key) $\rightarrow$ Des(key) $\rightarrow$ Output

It is easy to see that this can also be cracked by $2^{56}$ attempts and so is not secure enough.

**2Des using 2 different keys**

Input $\rightarrow$ Des (key1) $\rightarrow$ Des (key2) $\rightarrow$ Output

As a warmup let us estimate the number of functions that map 64 bits to 64 bits \( (2^{64}) \rightarrow (2^{64}) \)

This is the same as the number of attempts to discover the function by brute force which is: \( (2^{64})! \approx ((2^{64})/e)^{(2^{64})} \). Taking logs, the number of bits needed to represent such a function is \( \sim 62 \cdot 2^{64} \)

Idea: Encrypt m with k1 and decrypt c with k2 and check if the two results, I1 and I2, match

\[ E_{k1}(m) \rightarrow D_{k2}(c) \]

Algorithm for cracking this

Given(m1,c)

Have to get k1 k2

1. Compute for all k1, I1 = \( E_{k1}(m) \) \( [2^{56}] \)
2. Compute for all k2, I2 = \( D_{k2}(c) \) \( [2^{56}] \)
3. Sort I1 and I2 and match them

This narrows it down to at most $2^{56}$ possibilities. In fact, we will show that in an expected sense it narrows down to at most $2^{48}$ possibilities – this may seem counter-intuitive since it indicates that we have less security with two keys as compared to one.
Theorem: With high probability (>= 1 - (1/2^80)) 2-des can be cracked in O(2^56) steps using no more than 3 plaintext-ciphertext pairs

Proof:
E(# key pairs that survive)= 2^64 * (2^56/2^64) * (2^56/2^64) = 2^48
With one message-ciphertext pair (m1c1) we reduce from 2^112 → 2^48, i.e. a drop by 2^(-64).
With 3 message-ciphertext pairs P(wrong key pair survives) = (1/(2^64))^3 = 2^(-192)
Hence P((# wrong key pairs) > 1) <= 2^112 * 2^(-192) = (1/(2^80))
With probability >= 1-2^80 you are left with the right k1 k2

3-Des

Input → E_{k1} → D_{k2} → E_{k1} → Output

3-Des uses 2 56 bit keys but unlike 2-des achieves security that needs 2^112 attempts to break.

AES

Developed in 2001, AES is patented and therefore not as widely used because of the legal ramifications of using a patented technology.

Hash Functions – One way function

h(512 – bits) → 128 bits
Hard to invert: given a Y it is hard to find an X such that h(X) = Y
Collision-free: It is hard to find collisions, i.e., X1, X2 such that h(X1) = h(X2)

Theorem
Hash functions + key = Secret key cryptography
Applications  
Authentication  
Storage  
Encryption  

Example of authentication with hash function:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hi I’m A</td>
<td></td>
</tr>
<tr>
<td>&lt;-------- Challenge -------</td>
<td></td>
</tr>
<tr>
<td>h(Challenge</td>
<td>Secret) &lt;-- -------</td>
</tr>
</tbody>
</table>

Example of storage/encryption with hash function – same idea as encrypting long messages

\[ M = m_1 | m_2 | m_3 \ldots \]

\[ IV_0 \]

\[ IV_i = h(IV_0 + i|Secret) \]

\[ C= IV_0 | (IV_1 XOR m_1) | (IV_2 XOR m_2)\ldots \]

**What to do with a long message?**

1. Electronic code block (ecb)
   \[ M = m_1 | m_2 | m_3 \ldots \]
   \[ C = E_k(m_1) | E_k(m_2) | E_k(m_3)\ldots \]
   Problems: Frequency analysis
   Also, pieces may be reordered to create new messages

2. Cipher block chaining (cbc)
   \[ M = m_1 | m_2 | m_3 \ldots \]
   \[ R = r_1 | r_2 | r_3 \ldots \]
   \[ C = r_1| E_k(m_1 XOR r_1) | r_2 | E_k(m_2 XOR r_2) | r_3 | E_k(m_3 XOR r_3)\ldots \]
   Problem – Uses twice as many bits to transmit the data and is inefficient.

3. IV based schemes
Output feedback mode  (WEP is IV based)
IV₀ \rightarrow (E_k) \rightarrow IV₁ \rightarrow (E_k) \rightarrow IV₂ \ldots 
C = IV₀ | (m₁ XOR IV₁) | (m₂ XOR IV₂) | …

Cipher Feedback mode
IV₀ \rightarrow E_k \rightarrow IV₁ 
(IV₁ XOR m₁) \rightarrow E_k \rightarrow IV₂ 
(IV₂ XOR m₂) \rightarrow E_k \rightarrow IV₃ 
…

C = IV₀ | (M₁ XOR IV₁) | (M₂ XOR IV₂) | …

Counter Mode
IV₀
IV₁ = E_k(IV₀ + i)

C = IV₀ | (IV₁ XOR M₁) | (IV₂ XOR M₂) | …