Ciphers

• Overview

• Naïve Usage

• Types of Ciphers
Ciphers???

• Mechanisms for message security
  – But isn’t everything 😊
• Fast ?
• Secure ?
• Suited for environment ?
Ciphers???

The compromise of individual blocks should not lead to the compromise of past communication!
Problems

• Using a cipher requires knowledge of threats in the environment in which it will be used
  – Is the set of possible messages small?
  – Do the messages exhibit regularities that remain after encryption?
  – Can an active wire-tapper rearrange or change parts of the message?
Collision Attack: Birthday Attacks

• With 23 people in the same room chance of same birthday is over 50% !!!

• For $N$ possible values expect a collision after seeing approx. $\sqrt{N}$ of them

• If $N=2^n$ ($n$-bit key) after $2^{n/2}$ (“birthday bound”) messages a collision is expected!
Birthday Attack Deployed

• For 64-bit key, after seeing $2^{32}$ transactions Eve can find message sent with same key! (how can she know? Using keyed MAC of standard message header?)

• Eve can then substitute old messages for new ones (e.g., reversing money transfers)
Collision Attack: Meet in the Middle

- Cousin of Birthday Attack

- $C = E_{K_2}(E_{K_1}(M))$
- This does not have $2^n$ bit security!
- Why?
- For a known $(C, M)$ pair do this:
  - $T$: Build table $E_K(M)$ for all $K$
  - Compute $D_K(C)$ for all $K$ and lookup in $T$
  - Takes $2^{n+1}$ steps only
Attack: Pre-computation

- If set of possible messages $M$ is small
- Public key cipher $f$ used
- Idea: pre-compute set of possible cipher-texts $f(M)$, build table $(m, f(m))$
- When cipher-text $f(m)$ appears, use table to find $m$
- Also called forward searches
Example

• Cathy knows Alice will send Bob one of two enciphered messages: BUY or SELL
• Using public\(_B\), Cathy pre-computes
  \[ m_1 = E_{public_B}(“BUY”) \]
  \[ m_2 = E_{public_B}(“SELL”) \]
• Cathy sees Alice send Bob \( m_2 \)
• Cathy knows Alice sent SELL
Another example: may not be obvious

- Digitized sound
  - Seems like far too many possible plaintexts
    - Initial calculations suggest $2^{32}$ such plaintexts
  - Analysis of redundancy in human speech reduced this to about $100,000$ ($\approx 2^{17}$)
    - Small enough to worry about pre-computation attacks
Misordered Blocks

- Alice sends Bob message
  - Message is LIVE (11 08 21 04)
  - Enciphered message is 44 57 21 16
- Eve intercepts it, rearranges blocks
  - Now enciphered message is 16 21 57 44
- Bob gets enciphered message, deciphers it
  - He sees EVIL
Notes

• Signing each block won’t stop it!
• Two approaches:
  – Crypto-hash the *entire* message and sign it
  – Place sequence numbers in each block of message, so recipient can tell intended order, then sign each block
Statistical Regularities

- If plaintext repeats, ciphertext may too
- Example using DES:
  - input (in hex):
    \[3231\ 3433\ 3635\ 3837\ \boxed{3231}\ 3433\ 3635\ 3837\]
  - corresponding output (in hex):
    \[\text{ef7c}\ 4bb2\ b4ce\ 6f3b\ \boxed{\text{ef7c}}\ 4bb2\ b4ce\ 6f3b\]
- Fix: cascade blocks together (chaining)
  - More details later
What These Mean

• Use of strong cryptosystems, well-chosen (or random) keys not enough to be secure

• Other factors:
  – Protocols directing use of cryptosystems
  – Ancillary information added by protocols
  – Implementation (not discussed here)
  – Maintenance and operation (not discussed here)
Stream, Block Ciphers

- **$E$** encipherment function
  - $E_k(b)$ encipherment of message $b$ with key $k$
  - In what follows, $m = b_1b_2 \ldots$, each $b_i$ of fixed length

- **Block cipher**
  - $E_k(m) = E_k(b_1)E_k(b_2) \ldots$

- **Stream cipher**
  - $k = k_1k_2 \ldots$
  - $E_k(m) = E_{k_1}(b_1)E_{k_2}(b_2) \ldots$
  - If $k_1k_2 \ldots$ repeats itself, cipher is *periodic* and the length of its period is one cycle of $k_1k_2 \ldots$
Examples

- **Vigenère cipher**
  - $b_i = 1$ character, $k = k_1 k_2 \ldots$ where $k_i = 1$ character
  - Each $b_i$ enciphered using $k_i \mod \text{length}(k)$
  - Stream cipher

- **DES**
  - $b_i = 64$ bits, $k = 56$ bits
  - Each $b_i$ enciphered separately using $k$
  - Block cipher
Stream Ciphers

• Often (try to) implement one-time pad by xor’ing each bit of key with one bit of message
  – Example:

  \[
  m = 00101 \\
  k = 10010 \\
  c = 10111
  \]

• But how to generate a good key?
Synchronous Stream Ciphers

• *n*-stage Linear Feedback Shift Register:
  – *n* bit register \( r = r_0 \ldots r_{n-1} \)
  – *n* bit “tap sequence” \( t = t_0 \ldots t_{n-1} \)
  – Use:
    • Use \( r_{n-1} \) as key bit
    • Compute \( x = r_0 t_0 \oplus \ldots \oplus r_{n-1} t_{n-1} \)
    • Shift \( r \) one bit to right, dropping \( r_{n-1} \), \( x \) becomes \( r_0 \)
Example

- **4-stage LFSR; $t = 1001$**

<table>
<thead>
<tr>
<th>$r$</th>
<th>$k_i$</th>
<th>new bit computation</th>
<th>new $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>0</td>
<td>$01 \oplus 00 \oplus 10 \oplus 01 = 0$</td>
<td>0001</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>$01 \oplus 00 \oplus 00 \oplus 11 = 1$</td>
<td>1000</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>$11 \oplus 00 \oplus 00 \oplus 01 = 1$</td>
<td>1100</td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
<td>$11 \oplus 10 \oplus 00 \oplus 01 = 1$</td>
<td>1110</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td>$11 \oplus 10 \oplus 10 \oplus 01 = 1$</td>
<td>1111</td>
</tr>
<tr>
<td>1111</td>
<td>1</td>
<td>$11 \oplus 10 \oplus 10 \oplus 11 = 0$</td>
<td>0111</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td>$11 \oplus 10 \oplus 10 \oplus 11 = 1$</td>
<td>1011</td>
</tr>
</tbody>
</table>

- Key sequence has period of 15 ($010001011101110$)
• n-stage Non-Linear Feedback Shift Register: consists of
  – $n$ bit register $r = r_0 \ldots r_{n-1}$
  – Use:
    • Use $r_{n-1}$ as key bit
    • Compute $x = f(r_0, \ldots, r_{n-1})$; $f$ is any function
    • Shift $r$ one bit to right, dropping $r_{n-1}$, $x$ becomes $r_0$

Note same operation as LFSR but more general bit replacement function
Example

- 4-stage NLFSR: $f(r_0, r_1, r_2, r_3) = (r_0 \& r_2) | r_3$

<table>
<thead>
<tr>
<th>$r$</th>
<th>$k_i$</th>
<th>new bit computation</th>
<th>new $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>0</td>
<td>(1 &amp; 0)</td>
<td>0 = 0</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>(0 &amp; 1)</td>
<td>0 = 0</td>
</tr>
<tr>
<td>0011</td>
<td>1</td>
<td>(0 &amp; 1)</td>
<td>1 = 1</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
<td>(1 &amp; 0)</td>
<td>1 = 1</td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
<td>(1 &amp; 0)</td>
<td>0 = 0</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>(0 &amp; 1)</td>
<td>0 = 0</td>
</tr>
<tr>
<td>0011</td>
<td>1</td>
<td>(0 &amp; 1)</td>
<td>1 = 1</td>
</tr>
</tbody>
</table>

- Key sequence has period of 4 (0011)
Eliminating Linearity

- NLFSRs not common
  - We don’t know how to design them to have long period
- Alternate approach: output feedback mode
  - For $E$ encipherment function, $k$ key, $r$ register:
    - Compute $r' = E_k(r)$; key bit is rightmost bit of $r'$
    - Set $r$ to $r'$ and iterate, repeatedly enciphering register and extracting key bits, until message enciphered
  - Variant: use a counter that is incremented for each encipherment rather than a register
    - Take rightmost bit of $E_k(i)$, where $i$ is number of encipherment
Self-Synchronous Stream Cipher

- Take key from message itself (*autokey*)
- Example: Vigenère, key drawn from plaintext
  - *key*  XTHEBOYHASTHEBA
  - *plaintext*  THEBOYHASTHEBAG
  - *ciphertext*  QALFPNFHSLALFCT

- Problem:
  - Statistical regularities in plaintext show in key
  - Once you get any part of the message, you can decipher more
Another Example

- Take key from ciphertext (*autokey*)
- Example: Vigenère, key drawn from ciphertext
  - *key*  
    - XQXBCQOVVNGNRTT
  - *plaintext*  
    - THEBOYHASTHEBAG
  - *ciphertext*  
    - QXBCQOVVNGNRTT

- Problem:
  - Attacker gets key along with ciphertext, so deciphering is trivial
**Variant**

- Cipher feedback mode: 1 bit of ciphertext fed into \( n \) bit register
  - Self-healing property: if ciphertext bit received incorrectly, it and next \( n \) bits decipher incorrectly; but after that, the ciphertext bits decipher correctly
  - Need to know \( k, E \) to decipher ciphertext

\[
k \rightarrow E_k(r) \rightarrow m_i \rightarrow c_i
\]
Block Ciphers

- Encipher, decipher multiple bits at once
- Each block enciphered independently
- Problem: identical plaintext blocks produce identical ciphertext blocks
  - Example: two database records
    - MEMBER: HOLLY INCOME $100,000
    - MEMBER: HEIDI INCOME $100,000
  - Encipherment:
    - ABCQZRME GHQMRSIG CTXUVYSS RMGRPFQN
    - ABCQZRME ORMPABRZ CTXUVYSS RMGRPFQN
Solution: CBC

- Insert information about block’s position into the plaintext block, then encipher.
- **Cipher block chaining mode (CBC):**
  - Exclusive-or current plaintext block with previous ciphertext block:
    - $c_0 = E_k(m_0 \oplus I)$
    - $c_i = E_k(m_i \oplus c_{i-1})$ for $i > 0$
  where $I$ is the initialization vector
Solution: CTR

- **Counter mode (CTR):**
  - Key constructed by encrypting block counter
    - $k_i = E_k(\text{unique\_nonce}||i)$
    - $c_i = m_i \oplus k_i$
  
  *e.g. unique\_nonce = (message number)*
  - Question: why do we need the *nonce* ?
  - Careful: **never** use same $(k, nonce)$ pair !!!