Symmetric Ciphers:
Stream Ciphers
Content

• Stream and Block Ciphers
• True Random (Stream) Generators, Perfectly Secure Ciphers and the One-Time Pad
• Cryptographically Strong Pseudo Random Generators: Practical and Computationally Secure
• The notion of Linearity
• H/W Stream Ciphers with Linear Feedback Shift Registers (LFSR)
• Trivium – attempting to revive LFSR-based ciphers
• S/W stream ciphers
Stream Ciphers (i)

- Symmetric cryptography
  - Same key used for enc/decryption
  - Stream (bit-based) vs Block Ciphers (block of bits-based)
Stream Ciphers (ii)

- Simple and fast
  - XOR is the sole encryption operator
  - Synchronous vs asynchronous (dotted line present)
  - Former even faster: can pre-compute keystream $s_i$
Stream Ciphers (iii)

- **Definition**
  - Encryption/decryption is the same addition module 2 operation!

\[
\begin{align*}
e_{s_i}(x_i) &= y_i \equiv x_i + s_i \mod 2 \\
d_{s_i}(y_i) &= x_i \equiv y_i + s_i \mod 2
\end{align*}
\]

where \( x_i, y_i, s_i \in 0, 1 \)

<table>
<thead>
<tr>
<th>( x_i )</th>
<th>( s_i )</th>
<th>( y_i \equiv x_i + s_i \mod 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Stream Ciphers (iv)

- Correct decryption
  - \( d_{s_i}(e_{s_i}(x_i)) = x_i \)  

- “Double-XOR” (+ mod 2) in general
  - \( x_i \oplus s_i \oplus s_i = x_i \)

- Efficient to implement & RANDOM!

  Given \( s_i \) is random, probability for \( y=0/1 \) is uniform for any given \( x_i \)
  - Looks like we've got a good deal!
Stream Ciphers (v)

Alice

\[ x_0, \ldots, x_6 = 1000001 = A \]
\[ \oplus \]
\[ s_0, \ldots, s_6 = 0101100 \]
\[ y_0, \ldots, y_6 = 1101101 = m \]

Oscar

Bob

\[ m = 1101101 \rightarrow \]
\[ y_0, \ldots, y_6 = 1101101 \]
\[ \oplus \]
\[ s_0, \ldots, s_6 = 0101100 \]
\[ x_0, \ldots, x_6 = 1000001 = A \]
Stream Ciphers (vi)

• How to design a key-stream generator?
  - Using a True Random Function
  - Using a Cryptographically Strong Pseudo Random Function
  - In this case, the specific function is a stream generator
True Random Generators (i)

- A Random Oracle construction: a perfectly secret scheme

\[ f : \mathbb{Z} \rightarrow \{0, 1\}^l \]

<table>
<thead>
<tr>
<th>x</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1111111000</td>
</tr>
<tr>
<td>1</td>
<td>1010100001</td>
</tr>
<tr>
<td>2</td>
<td>0010011000</td>
</tr>
<tr>
<td>3</td>
<td>1000111001</td>
</tr>
<tr>
<td>4</td>
<td>0001110000</td>
</tr>
<tr>
<td>5</td>
<td>1000101010</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
True Random Generators (ii)

- The One-Time Pad/Vernam Cipher based on a Random Oracle construction actually existed!
  - Information theoretic secure: assumes no limit to attacker resources
  - Entropy/uncertainty of a message (in bits)
    \[
    H(X) = - \sum_{i=1}^{n} p(X = x_i) \log_2(X = x_i)
    \]
  - E.g. For a dice roll $H(X) = 2.585$
  - Perfect Security i.e. access to the ciphertext makes you non-the-wiser
    \[
    H(x|y) = H(x)
    \]
True Random Generators (iii)

- A stream cipher with the following conditions
  1. Keystream generation (pad): TRG → s₀, s₁, s₂, ..., sₙ where n = size of the plaintext message
  2. Keystream is kept secret (of course!)
  3. Used only **ONCE**! (one-time)

\[
\begin{align*}
y₀ &\equiv x₀ + s₀ \mod 2 \\
y₁ &\equiv x₁ + s₁ \mod 2 \\
    &\vdots \\
yₙ &\equiv xₙ + sₙ \mod 2
\end{align*}
\]

- System of n equations with 2n unknowns
  - **Unsolvable!** (P.T.O)
  - i.e. **Non-linear**
True Random Generators (iv)

- Linear algebra refreshed: Gaussian Elimination
  - Forward elimination
    - $A \cdot x = b \rightarrow U \cdot x = c$
    - $2x + y - 2z = 2$  
    - $2x + 4y - 2z = 2$
    - $4x + 9y - 3z = 8$  
    - $y + z = 4$
    - $-2x - 3y + 7z = 10$  
    - $4z = 8$
  - Backward substitution
    - $(x, y, z) = (-1, 2, 2)$
    - Take out the last equation and you get no (or infinitely many) solution
True Random Generators (v)

• What if the pad is reused?

\[ X_1 \oplus S = Y_1 \quad X_2 \oplus S = Y_2 \]

\[ \Rightarrow X_1 \oplus Y_1 = S = X_2 \oplus Y_2 \]

\[ \Rightarrow X_1 \oplus Y_1 = X_2 \oplus Y_2 \]

• S (secret) goes out of the equation!

• With a single known \((X_1, Y_1)\) pair we get

\[ Y_2 = X_1 \oplus X_2 \oplus Y_1 \]
True Random Generators (vi)

• Short message paradox
  
  - A = 00001, B = 00010 etc...
  
  - Y1:  A    B    C
         - 00001  00010  00011
  
  - K?:
         - 00110  01101  10000
  
  - X1?: F    O    R
         - 00110  01111  10011
  
  - K?:
         - 00111  01101  10111

  Keys in-depth? <<<

  - X1?: N    O    T etc...
         - 01110  01111  10100
True Random Generators (vii)

- **Practical implications**
  - Out-of-band channel is required to transport all pads
  - In the vernam cipher (cold war) days, it was used by spies to secure teleprinter communication
  - Pad: Punched tape
  - Out-of-band: Physical transportation of key material i.e. not over the line!
True Random Generators (viii)

- ... and Computer Security? e.g. Network Security
  - Sources of true randomness
    - h/w: e.g. capacitors, transistors
    - s/w: e.g. network packet/process statistics
  - yet, 3rd requirement implies that the same amount of bandwidth required to transport network data is required to transport key material out-of-band!
  - How about Tera bytes?!
  - In any case if such a channel was in place it would make sense to dump the plaintext through it straight away
- We require pseudo randomness that is cryptographically strong
Pseudo Random Generators (i)

- Security/practicality compromise
  - Restrict true randomness to a much short key
  - Which is then stretched pseudo randomly
  - Still requires an out-of-band channel, but with significantly smaller bandwidth requirements
Pseudo Random Generators (ii)

- Pseudo Random Generators
  - Compute next substream from prior state
  - Deterministic i.e. once k is securely established, both sides can generate the same keystream
    \[
    s_0 = seed \\
    s_{i+1} = f(s_i), \ i = 0, 1, \ldots
    \]

- in this case seeded by k
Pseudo Random Generators (iii)

- Characteristics of a True Random Generator
  - TRG $\rightarrow s_0, s_1, s_2, \ldots, s_n$
  - Uniformly distributed
  - No correlation, no bias, ..., no patterns!
  - Essentially:
    - TRG $\rightarrow \ldots s_{20}, s_{21}, s_{22}, \ldots$
    - Knowledge of substream is not useful to disclose prior/following keystream bits
    - A *cryptographically strong* PRG is indistinguishable from a TRG with feasible computational resources
Pseudo Random Generators (iv)

- Know Plaintext Attack setting (e.g. network packet headers)
  - X: 10001000010000000000000000100001100010
  - S: 11111..................10001......................
  - Y: 011100010101000100010000100001000
  - i.e.
    - Plaintext patterns dissipate into the ciphertext
    - Partly distinguishable from random!
  - It is important that it is computationally infeasible to
    - PRG → 11111............10001......................
      - else all plaintext is disclosed!
Hardware Stream Ciphers (i)

- Linear Feedback Shift Registers (LFSR)
  - H/w primitive for hardware implementations

\[ s_{i+3} \equiv s_{i+1} + s_i \mod 2 \]
### Hardware Stream Ciphers (ii)

No 000 -
Or would get stuck
In an all-0 stream

$m=3$
LFSR degree
i.e. number of flip-flops

<table>
<thead>
<tr>
<th>clk</th>
<th>$FF_2$</th>
<th>$FF_1$</th>
<th>$FF_0 = s_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Not a good idea to use $k$ as seed

↩ Starts repeating (Deterministic)
Hardware Stream Ciphers (iii)

- Why linear?

\[ P(x) = x^m + p_{m-1}x^{m-1} + \ldots + p_1x + p_0 \]
Hardware Stream Ciphers (iv)

- An LFSR can be represented as a polynomial
- Repeated keystreams are equivalent to keys in-depth of the One-Time Pad
  - Not desirable
- Maximum sequence length
  - $2^m - 1$
  - Guaranteed by a primitive polynomial (cannot be factored)
## Hardware Stream Ciphers

- **Feedback Coefficient Vector notation**

```
(0,1,2)  (0,1,3,4,24)  (0,1,46)  (0,1,5,7,68)  (0,2,3,5,90)  (0,3,4,5,112)
(0,1,3)  (0,3,25)    (0,5,47)  (0,2,5,6,69)  (0,1,5,8,91)  (0,2,3,5,113)
(0,1,4)  (0,1,3,4,26) (0,2,3,5,48) (0,1,3,5,70)  (0,2,5,6,92)  (0,2,3,5,114)
(0,2,5)  (0,1,2,5,27) (0,4,5,6,49) (0,1,3,5,71)  (0,2,93)    (0,5,7,8,115)
(0,1,6)  (0,1,2,8)   (0,2,3,4,50) (0,3,9,10,72) (0,1,5,6,94)  (0,1,2,4,116)
(0,1,7)  (0,2,29)   (0,1,3,6,51) (0,2,3,4,73)  (0,11,95)  (0,1,2,4,117)
(0,1,3,4,8) (0,1,30) (0,3,5,2)   (0,1,2,6,74) (0,6,9,10,96) (0,2,5,6,118)
(0,1,9)  (0,3,31)   (0,1,2,6,53) (0,1,3,6,75)  (0,6,97)    (0,8,119)
(0,3,10) (0,2,3,7,32) (0,3,6,8,54) (0,2,4,5,76)  (0,3,4,7,98) (0,1,3,4,120)
(0,2,11) (0,1,3,6,33) (0,1,2,6,55) (0,2,5,6,77)  (0,1,3,6,99) (0,1,5,8,121)
(0,3,12) (0,1,3,4,34) (0,2,4,7,56) (0,1,2,7,78)  (0,2,5,6,100) (0,1,2,6,122)
(0,1,3,4,13) (0,2,35) (0,4,57)   (0,2,3,4,79) (0,1,6,7,101) (0,2,123)
(0,5,14) (0,2,4,5,36) (0,1,5,6,58) (0,2,4,9,80)  (0,3,5,6,102) (0,37,124)
(0,1,15) (0,1,4,6,37) (0,2,4,7,59) (0,4,81)    (0,9,103)  (0,5,6,7,125)
(0,1,3,5,16) (0,1,5,6,38) (0,1,60)   (0,4,6,9,82) (0,1,3,4,104) (0,2,4,7,126)
(0,3,17) (0,4,39)   (0,1,2,5,61) (0,2,4,7,83)  (0,4,105)  (0,1,127)
(0,3,18) (0,3,4,5,40) (0,3,5,6,62) (0,5,84)    (0,1,5,6,106) (0,1,2,7,128)
(0,1,2,5,19) (0,3,41) (0,1,63)   (0,1,2,8,85) (0,4,7,9,107)
(0,3,20) (0,1,2,5,42) (0,1,3,4,64) (0,2,5,6,86)  (0,1,4,6,108)
(0,2,21) (0,3,4,6,43) (0,1,3,4,65) (0,1,5,7,87)  (0,2,4,5,109)
(0,1,22) (0,5,44)   (0,3,66)   (0,8,9,11,88) (0,1,4,6,110)
(0,5,23) (0,1,3,4,45) (0,1,2,5,67) (0,3,5,6,89)  (0,2,4,7,111)
```
Hardware Stream Ciphers (vi)

• However

  – A single LFSR would not make a CSPRG

\[
 s_{i+m} \equiv \sum_{j=0}^{m-1} p_j \cdot s_{i+j} \mod 2; \quad s_i, p_j \in \{0, 1\}; \quad i = 0, 1, 2, \ldots
\]

With 2m known stream bits,
the entire feedback function can be disclosed
Hardware Stream Ciphers (vii)

- Solution
  - Non-linear combination of LFSRs e.g. Trivium
Hardware Stream Ciphers (viii)

• Phases
  - Initialization (seed):
    • 80-bit Initializing Vector and Key populate the leftmost positions of LFSRs A and B respectively; $C_{109-111}$ are set to 1; rest all zeroed out
    • IV is public, whose purpose is to re-seed the PRG (to avoid repeating streams) without requiring an out-of-band channel
  - Warm-up:
    • Discard the first 288x4 keystream bits
    • Easier part of stream that would enable key disclosure
  - Encryption:
    • XOR
Hardware Stream Ciphers (ix)

• Trivium is an attempt to revive LFSR-based stream ciphers, having the desirable property of efficient implementation i.e. chip area (gates + flip-flops)
  – eSTREAM project: targets IoT security characterized by resource-constrained devices
• A5/1 most widely deployed LFSR-based stream cipher which was a linear combination of LFSRs and used in GSM
  – Plagued by known plaintext attacks and other protocol deficiencies
  – In 3G replaced by KASUMI – a block cipher-based stream cipher
• Overall, stream cipher design is much less understood than block cipher design, and the latter has caught up even in terms of efficient implementation in h/w and s/w
  – Cube attack came very close to break it (broke a simpler version)
  – Yet, the lessons learned from LFSR-based ciphers proved very useful for block cipher design, especially the linearity concept
Software Stream Ciphers (i)

- Efficient implementation
  - In terms of CPU instructions
- C's `srand(s_0) / rand()`
  - *Linear* congruential generator

\[ s_{i+1} \equiv 1103515245 \cdot s_i + 12345 \mod 2^{31}, \quad i = 0, 1, \ldots \]

- \( S_0 \) is 32-bit
- Is it secure? <<<
Software Stream Ciphers (ii)

- How about using seed as an IV and coefficients as key?
  \[ S_0 = IV \]
  \[ S_{i+1} = AS_i + B \mod m, \ i = 0, 1, \ldots \]

- Consider an 80-bit key e.g. split between A and B; \( m = 2^{40} - 1 \)
  - \( S_2 = AS_1 + B \mod m \),
  - \( S_3 = AS_2 + B \mod m \),
  - Is the key secure? <<<
Software Stream Ciphers (iii)

• RC4 most-widely deployed s/w stream cipher not based on block ciphers
  – Utilized in WEP (obsolete)
  – Still supported by some TLS implementations (against standard recommendations) but only for legacy purposes; dumped by mainstream web browsers as of early 2016
  – A byte stream generator, at each round permuting the values from 0...255 and then outputs the next keystream byte as a function of this state
  – Multitude of attacks

• Solution
  – Block ciphers!
CPS2323 Reading List

Textbook:


Supplementary reading:


