Practical Constructions of Symmetric-Key Primitives
Plan

- **Stream ciphers**
  LFSR, Trivium, RC4

- **Block ciphers**
  Substitution-permutation networks, Feistel Networks, DES, 3DES, AES, Differential and Linear cryptanalysis

- **Hash functions**
  From block ciphers, MD5, SHA-\{0,1,2,3\}
Stream ciphers
Stream cipher

Formally, we view a stream cipher\(^2\) as a pair of deterministic algorithms (Init, GetBits) where:

- **Init** takes as input a seed \( s \) and an optional *initialization vector* \( IV \), and outputs an initial state \( st_0 \).

- **GetBits** takes as input state information \( st_i \), and outputs a bit \( y \) and updated state \( st_{i+1} \). (In practice, \( y \) is a *block* of several bits; we treat \( y \) as a single bit here for generality and simplicity.)
LFSR: Linear Feedback Shift Register

- **NB! Not cryptographically secure**

- Registers \( s_{n-1}, ..., s_0 \)

- Feedback coefficients \( c_{n-1}, ..., c_0 \)

- At tick
  (a) output value of left-most register
  (b) compute new value of left-most register
  (c) shift remaining registers to the right
**FIGURE 6.1:** A linear-feedback shift register.

\[
\begin{align*}
  s_i^{(t+1)} &:= s_{i+1}^{(t)}, & i &= 0, \ldots, n - 2 \\
  s_{n-1}^{(t+1)} &:= \bigoplus_{i=0}^{n-1} c_i s_i^{(t)}.
\end{align*}
\]

Figure 6.1 shows a degree-4 LFSR with \( c_0 = c_2 = 1 \) and \( c_1 = c_3 = 0 \).
FIGURE 6.1: A linear-feedback shift register.

\[ y_i = s_{i-1}^{(0)}, \quad i = 1, \ldots, n \]

\[ y_i = \bigoplus_{j=0}^{n-1} c_j y_{i-n+j-1} \quad i > n. \]
- n-bit state $\rightarrow 2^n$ distinct states at best
- All-zero state stuck
- *maximum-length* LFSR cycles through all non-zero states
- “Well-understood” how to obtain such.
- Good statistical properties, e.g., every n-bit string has roughly equal probability
DEFINITION 3.14  Let \( \ell \) be a polynomial and let \( G \) be a deterministic polynomial-time algorithm such that for any \( n \) and any input \( s \in \{0,1\}^n \), the result \( G(s) \) is a string of length \( \ell(n) \). We say that \( G \) is a pseudorandom generator if the following conditions hold:

1. \textbf{(Expansion:)} For every \( n \) it holds that \( \ell(n) > n \).

2. \textbf{(Pseudorandomness:)} For any \textsf{PPT} algorithm \( D \), there is a negligible function \( \text{negl} \) such that

\[
| \Pr[D(G(s)) = 1] - \Pr[D(r) = 1]| \leq \text{negl}(n),
\]

where the first probability is taken over uniform choice of \( s \in \{0,1\}^n \) and the randomness of \( D \), and the second probability is taken over uniform choice of \( r \in \{0,1\}^{\ell(n)} \) and the randomness of \( D \).

We call \( \ell \) the expansion factor of \( G \).
State-reconstruction attack

- Adversary may reconstruct state of degree-n LFSR after observing $2n$ bits.

- ... so not a pseudo-random generator

- Assume initial state, feedback coefficients unknown.

- First $n$ outputs reveal initial state.
Next $n$ outputs yield uniquely solvable system of equations:

\[
y_{n+1} = c_{n-1} y_n \oplus \cdots \oplus c_0 y_1 \\
\vdots \\
y_{2n} = c_{n-1} y_{2n-1} \oplus \cdots \oplus c_0 y_n.
\]
FSR: (Non-linear)
Feedback shift register

\[ s_i^{(t+1)} := s_{i+1}^{(t)}, \quad i = 0, \ldots, n - 2 \]
\[ s_{n-1}^{(t+1)} := g(s_0^{(t)}, \ldots, s_{n-1}^{(t)}) \]

(g non-linear)
Trivium

- eSTREAM 2008 portfolio stream cipher

- To date, no cryptanalytic attacks better than exhaustive key search are known against the full Trivium cipher.
  [No formal statement?]

- Comprises non-linear FSRs A, B, C of degree 93, 84, 111.

- (Compound state 288 bits).

- initialised by 80-bit key, 80-bit IV

- Remaining 168 bits fixed

- GetBits run 4*288 times with output discarded
FIGURE 6.2: A schematic illustration of Trivium with (from top to bottom) three coupled, nonlinear FSRs $A$, $B$, and $C$. 
RC4

- Stream cipher
- 128 bit key
- Used many places, e.g., 802.11 WEP
- Considered insecure.
- Recall synchronised vs unsynchronised modes of stream ciphers
FIGURE 3.4: Synchronized mode and unsynchronized mode.
**ALGORITHM 6.1**
Init algorithm for RC4

**Input:** 16-byte key $k$

**Output:** Initial state $(S, i, j)$
(Note: All addition is done modulo 256)

```
for $i = 0$ to 255:
    $S[i] := i$
    $k[i] := k[i \mod 16]$

$j := 0$

for $i = 0$ to 255:
    $j := j + S[i] + k[i]$
    Swap $S[i]$ and $S[j]$

$i := 0$, $j := 0$

return $(S, i, j)$
```

**ALGORITHM 6.2**
GetBits algorithm for RC4

**Input:** Current state $(S, i, j)$

**Output:** Output byte $y$; updated state $(S, i, j)$
(Note: All addition is done modulo 256)

```
i := i + 1
j := j + S[i]
Swap S[i] and S[j]
t := S[i] + S[j]
y := S[t]
return (S, i, j), y
```
Statistical attack (no IV)

- Recall $i,j = 0$ initially

- Say $S_0$ is uniform permutation of $\{0,\ldots, 255\}$

- $\Pr(S_0[2] = 0 \land S_0[1] \neq 2) = \frac{1}{256} \times (1 - \frac{1}{255}) \approx \frac{1}{256}$

- Assume so.
  Define $X = S_0[1]$  
  (Note $X \neq 2$)

**Algorithm 6.2**
GetBits algorithm for RC4

**Input:** Current state $(S, i, j)$
**Output:** Output byte $y$; updated state $(S, i, j)$
(Note: All addition is done modulo 256)

1. $i := i + 1$
2. $j := j + S[i]$
3. Swap $S[i]$ and $S[j]$
4. $t := S[i] + S[j]$
5. $y := S[t]$
6. return $(S, i, j), y$
Statistical attack (no IV)

- Pr(S₀[2] = 0 && S₀[1] != 2) = 1/256 * (1-(1/255)) ≈ 1/256

- X = S₀[1] != 2

- At the end of iter 1: S₁[X] = S₀[1] = X

**Algorithm 6.2**

GetBits algorithm for RC4

**Input:** Current state (S, i, j)

**Output:** Output byte y; updated state (S, i, j)

(Note: All addition is done modulo 256)

```
S₀ := S₀[1] = X
S₁ := S₀[X]; S₁[X] = S₀[1] = X
```

[Iteration 1]
Statistical attack (no IV)

- \( \Pr(S_0[2] = 0) \approx 1/256 \)

- \( X = S_0[1] \neq 2 \)

- At the end of iter 1:
  \( S_1[X] = S_0[1] = X \)

- At the end of iter 2:
  \( \Pr[S_0[2] = 0 \text{ and } S_0[1] \neq 2] + \frac{1}{256} \cdot \left(1 - \Pr[S_0[2] = 0 \text{ and } S_0[1] \neq 2]\right) \)
  \( \approx \frac{2}{256} \)

**ALGORITHM 6.2**
GetBits algorithm for RC4

*Input:* Current state \((S, i, j)\)

*Output:* Output byte \(y\); updated state \((S, i, j)\)

(Note: All addition is done modulo 256)

- \(i := i + 1\)
- \(j := j + S[i]\)
- Swap \(S[i]\) and \(S[j]\)
- \(t := S[i] + S[j]\)
- \(i = 2\)
- \(j = X + S_1[2] = X + S_0[2] = X\)
- \(S_2[2] = S_1[X] = X; S_2[X] = S_1[2]\)
- \(t = S_2[2] + S_2[X] = X + 0 = X\)
- \(y = S_2[X] = 0\)
Statistical attack (prepended IV)

- No IV, but practice: concatenate IV to key k.

- You should be concerned.

- Attack idea: “Key” is (IV + k), no IV = first n bytes of key. Use that to guess byte n+1.

- Assume 3-byte IV (WEP case)

- Consider IV = (3, 255, X)
Statistical attack (prepended IV)

- Consider IV = (3, 255, X)

- Easy to check that after 4 iterations:
  \[ S[0] = 3, \quad S[1] = 0, \quad S[3] = X + 6 + k[3]. \]  \hspace{1cm} (6.1)

- \( i \) subsequently greater than 3 in init

- \( S[0], S[1], S[3] \) touched in init subsequently iff \( j \) in \( \{0, 1, 3\} \)

- Suppose \( j \) uniform. Then odds of not touching is \((253/256)^{252} \approx 0.05\).

Block ciphers
Recall from Section 3.5.1 that a block cipher is an efficient, keyed permutation $F : \{0, 1\}^n \times \{0, 1\}^\ell \to \{0, 1\}^\ell$. This means the function $F_k$ defined by $F_k(x) \overset{\text{def}}{=} F(k, x)$ is a bijection (i.e., a permutation), and moreover $F_k$ and its inverse $F_k^{-1}$ are efficiently computable given $k$. We refer to $n$ as the key length and $\ell$ as the block length of $F$, and here we explicitly allow them to differ. The

- Key length, block length now constant.
- Recall: Should behave like pseudo-random function
- Concrete security.
- “Good” block cipher: Best known attack no better than brute force. ($2^{128}$ approach for 256 bit key not good.)
Terminology: Attacks

- In a *known-plaintext attack*, the attacker is given pairs of inputs/outputs \( \{(x_i, F_k(x_i))\} \) (for an unknown key \( k \)), with the \( \{x_i\} \) outside the attacker’s control.

- In a *chosen-plaintext attack*, the attacker is given \( \{F_k(x_i)\} \) (again, for an unknown key \( k \)) for a series of inputs \( \{x_i\} \) chosen by the attacker.

- In a *chosen-ciphertext attack*, the attacker is given \( \{F_k(x_i)\} \) for \( \{x_i\} \) chosen by the attacker, as well as \( \{F_k^{-1}(y_i)\} \) for chosen \( \{y_i\} \).
Challenge: Space

- How do we represent a pseudorandom permutation?
- \(2^n!\) permutations of n-bit strings
- Need at least \(\log(2^n!) \approx n \cdot 2^n\) bits
- Not practical
Challenge: Security

- 1-bit input change => (almost) independent outputs
- 1-bit input change => every bit changes with P .5
Confusion

- Due to Shannon [Reference]

- 128 bit = 16 byte block length

- Derive from key 16 one-byte permutations

- Compute round:

\[ F_k(x) = f_1(x_1) \| \cdots \| f_{16}(x_{16}). \]  

(6.2)

- Obviously not pseudo-random
Diffusion

- Add final mixing permutation, shuffling the output of each 1-byte block.
Substitution-permutation network

- Fix public substitution function $S$
- Have key $k$
- Define “S-box” by $f(x) = S(x \oplus k)$

1. Key mixing: Set $x := x \oplus k$, where $k$ is the current-round sub-key;
2. Substitution: Set $x := S_1(x_1) \| \cdots \| S_8(x_8)$, where $x_i$ is the $i$th byte of $x$;
3. Permutation: Permute the bits of $x$ to obtain the output of the round.
FIGURE 6.3: A single round of a substitution-permutation network.
FIGURE 6.4: A substitution-permutation network.
Correctness

**PROPOSITION 6.3** Let $F$ be a keyed function defined by an SPN in which the $S$-boxes are all permutations. Then regardless of the key schedule and the number of rounds, $F_k$ is a permutation for any $k$. 
Avalanche effect

- Recall: small change in input must affect every bit of output.

1. The $S$-boxes are designed so that changing a single bit of the input to an $S$-box changes at least \textit{two bits} in the output of the $S$-box.
2. The mixing permutations are designed so that the output bits of any given $S$-box are used as input to multiple $S$-boxes in the next round.
Attacks on reduced-round SPN

- Trivial case: One round, no final key mixing.
- Suppose adversary has single known plaintext \((x, F_k(x))\).
- Invert mixing permutation, S-boxes. Obtain \(x \oplus k\). Have \(x\), so can obtain \(k\).
Attacks on reduced-round SPN

- Easy case: One round, final key mixing.

- Suppose adversary has single known plaintext \((x, F_k(x))\).

- Invert mixing permutation, S-boxes. Obtain \(x \oplus k\). Have \(x\), so can obtain \(k\).
Feistel Networks

- Use something non-invertible instead of S-boxes.
- Round function takes $l/2$ input, produce $l/2$ output.
- Subkeys derived from master keys.
- Output of round:

\[
L_i := R_{i-1} \quad \text{and} \quad R_i := L_{i-1} \oplus f_i(R_{i-1}).
\]  

(6.3)
FIGURE 6.5: A three-round Feistel network.
DES

- Considered secure, but key is too small
- 16-round Feistel network
- 64-bit blocks
- 56-bit key
- Key-schedule derives subways $k_1 \ldots k_{16}$
- Round function constructed as SPN
FIGURE 6.6: The DES mangler function.

[pp. 214]
Summary
Summary

- **Stream ciphers**
  LFSR, Trivium, RC4

- **Block ciphers**
  Substitution-permutation networks, Feistel Networks, DES, 3DES, AES, Differential and Linear cryptanalysis

- **Hash functions**
  From block ciphers, MD5, SHA-\{0,1,2,3\}