Introduction to Modern Cryptography
Lecture 3

November 15, 2016

Instructor: Benny Chor
Teaching Assistant: Orit Moskovich

School of Computer Science
Tel-Aviv University

Fall Semester, 2016–17
Tuesday 12:00–15:00
Venue: Meron Hall, Trubowicz 102 (faculty of Law)

Course site: http://tau-crypto-f16.wikidot.com/

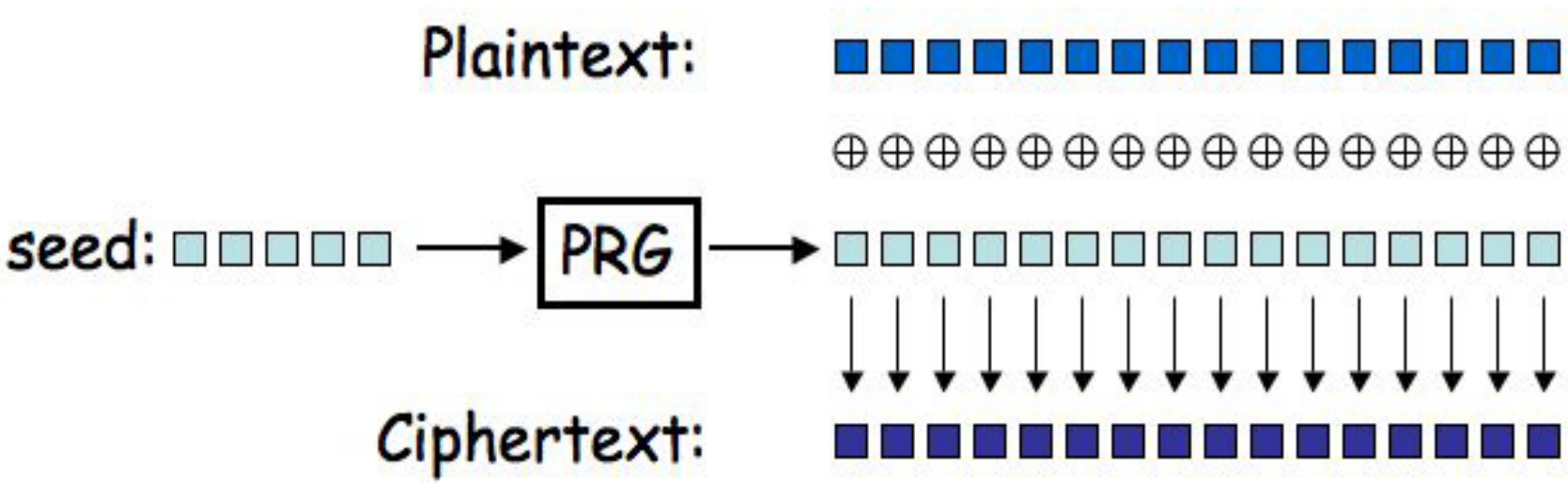
Lecture 3: Plan

- Groups, subgroups, Lagrange theorem.
- The multiplicative group $(Z_m^*, \cdot \bmod m)$.
- Euler's totient (or phi) function, $\phi(n)$.
- Integer gcd: Run time analysis.

- Stream ciphers (synchronous mode).
- LFSR and the shrinking generator.
- Concrete example: RC4.
- PRGs and bit commitment.
- Stream ciphers (asynchronous mode).
- Pseudo random functions and pseudo random permutations.

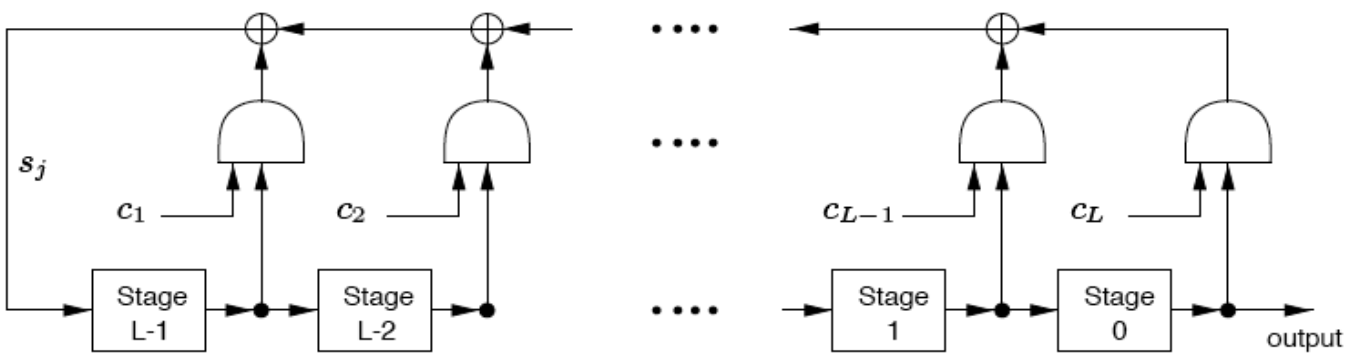
Synchronous Stream Ciphers ("imitating" one-time pad)

- Start with a secret, random key ("seed"). Generate (online) a keying stream by applying the PRG, G, to the seed. The i-th bit of the keying stream is the i-th bit of G's output.
- Combine the keying stream by bitwise XORing with the plaintext, to produce the ciphertext.
- This type of stream cipher is called synchronous (why?).
- Decryption is done in the same manner (XORing ciphertext with keying stream).



Linear Feedback Shift Registers

- An LFSR is a function that produces a binary output stream.
- The device in the picture (from Menezes, Oorschot and Vanstone's book) has L stages (or delay elements).
- The $c_i \in \{0, 1\}$ in the picture are the hardwiring of the device. They are secret, and $c_L = 1$.
- The initial state is $[s_{L-1}, \ldots, s_1, s_0]$.
- The output sequence (stream) $s_0, s_1, s_2, \ldots$. For $j \leq \ell - 1$, $s_j = c_j$, while for larger indices, the outputs are determined by the recursion $s_j = \sum_{i=1}^{L} c_i s_{j-i} \mod 2$.

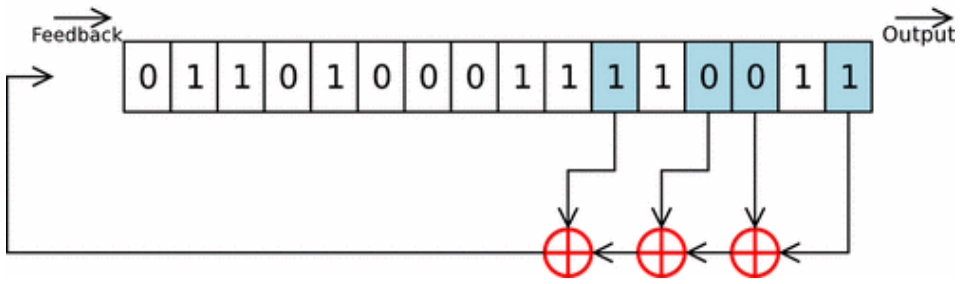


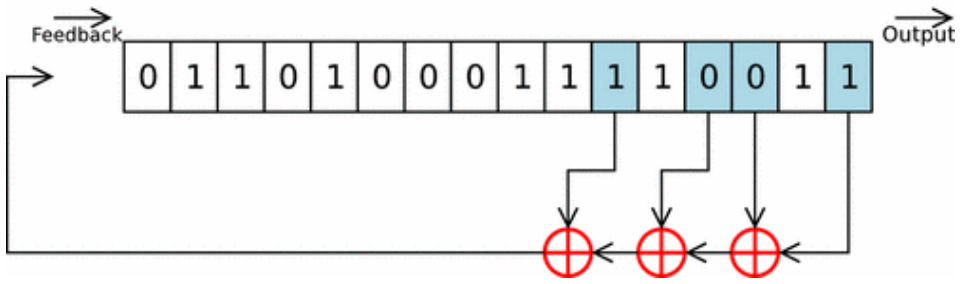
Linear Feedback Shift Registers and Polynomials over Z_2

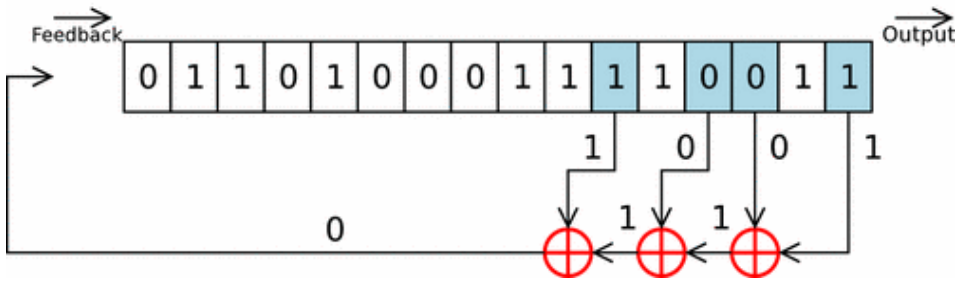
The coefficients c_i define a polynomial over Z_2, with terms corresponding to non zero coefficients.

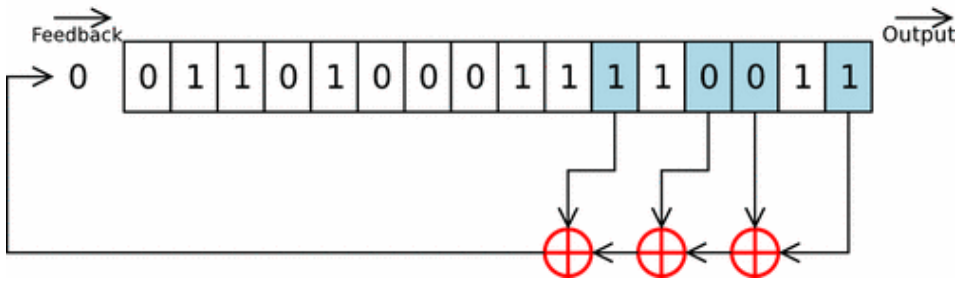
The following animation of a LFSR is taken from https://commons.wikimedia.org/wiki/File:Lfsr.gif.

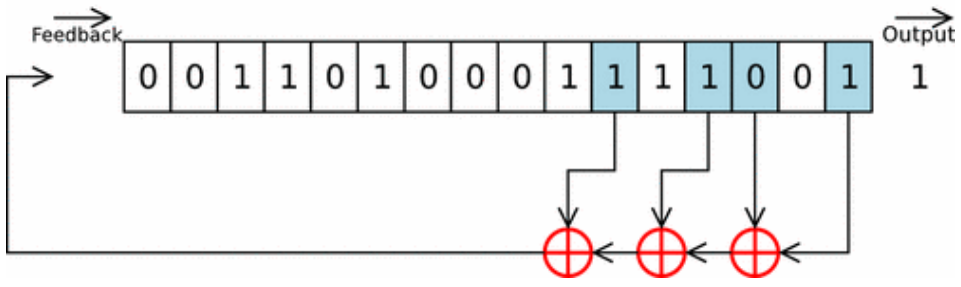
This specific LFSR corresponds to the feedback polynomial $f(x) = x^{16} + x^5 + x^3 + x^2 + 1$ (over Z_2).

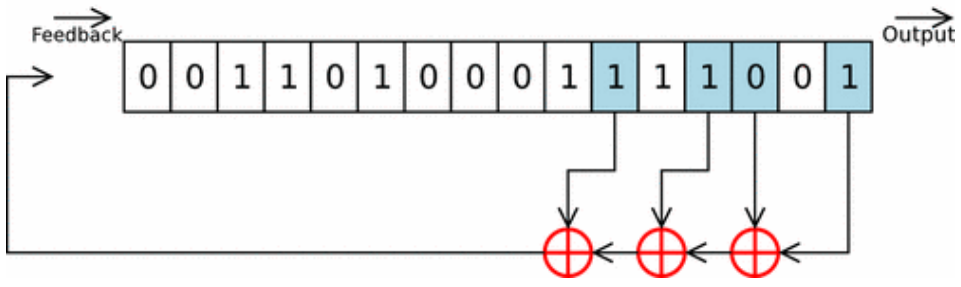

Feedback
Output
0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1

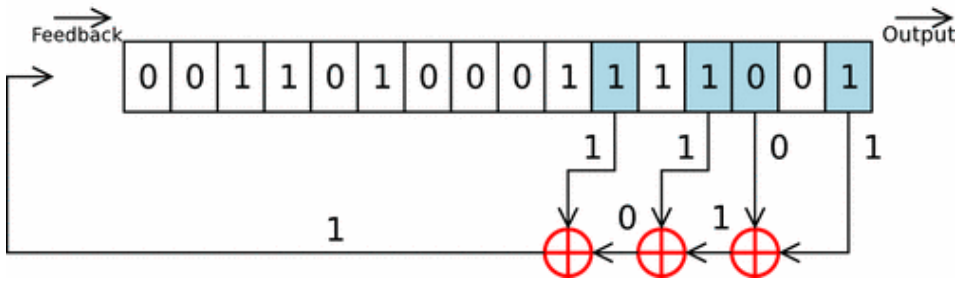

Feedback
Output
0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1

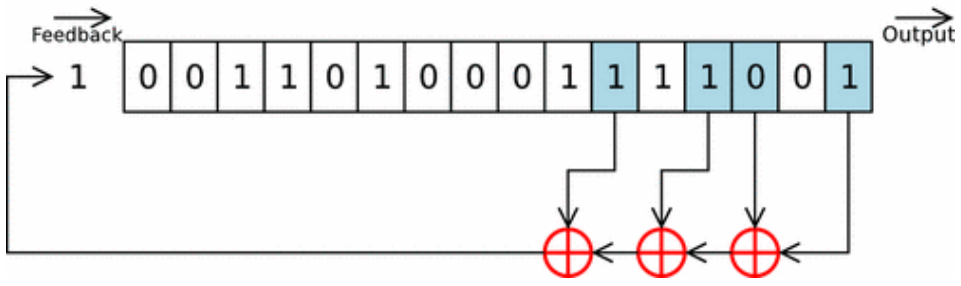

Feedback
Output
0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1
1 0 0 1
1 1
0

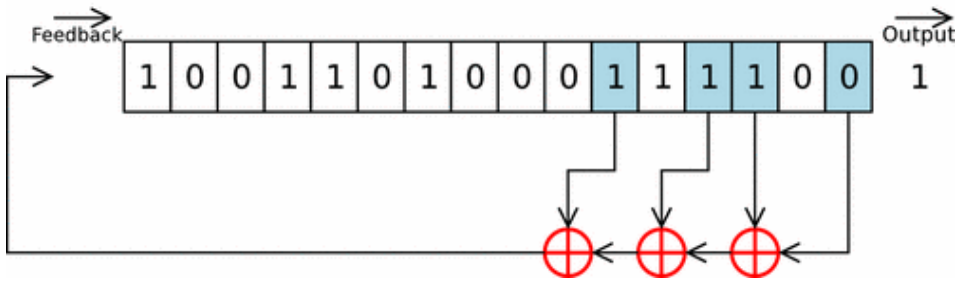

Feedback
Output
0
0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1

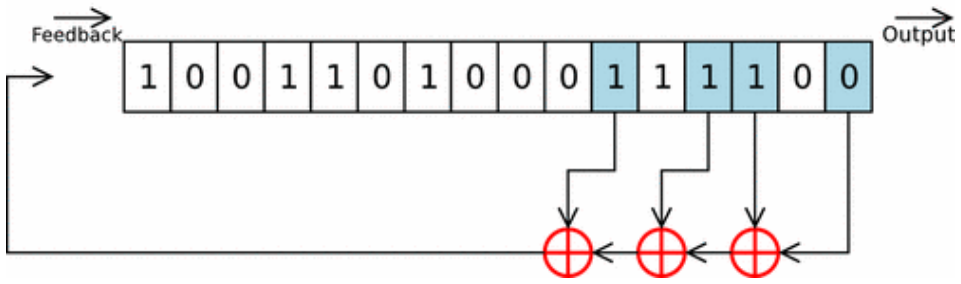

Feedback
Output
0 0 1 1 0 1 0 0 0 1 1 1 1 0 0 1
1

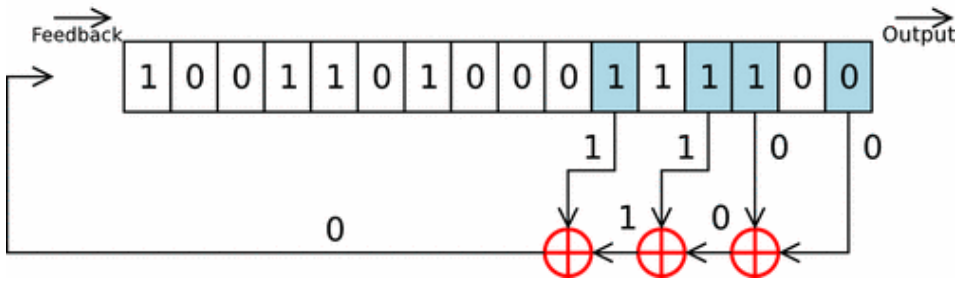

Feedback
Output
0 0 1 1 0 1 0 0 0 1 1 1 1 0 0 1

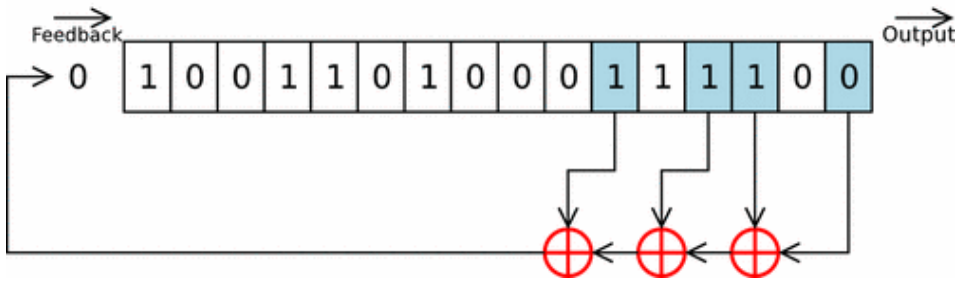

Feedback
Output
0 0 1 1 0 1 0 0 0 1 1 1 1 0 0 1
1 1 0 1
0 1
1

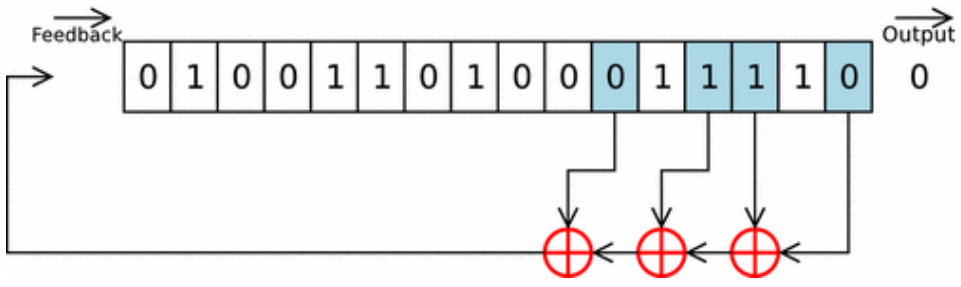

Feedback
Output
1
0 0 1 1 0 1 0 0 0 1 1 1 1 0 0 1

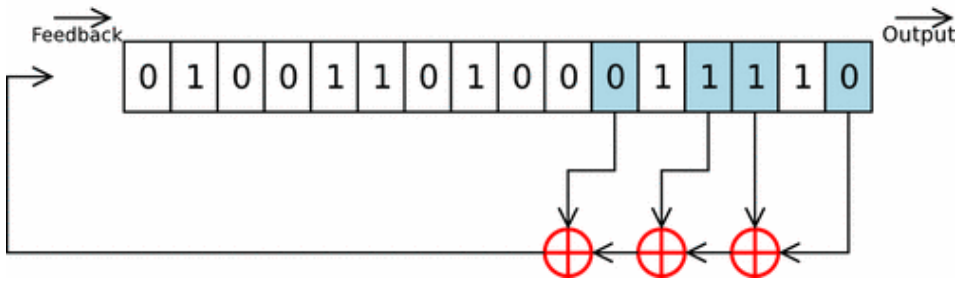

Feedback
Output
1 0 0 1 1 0 1 0 0 0 1 1 1 1 0 0
1

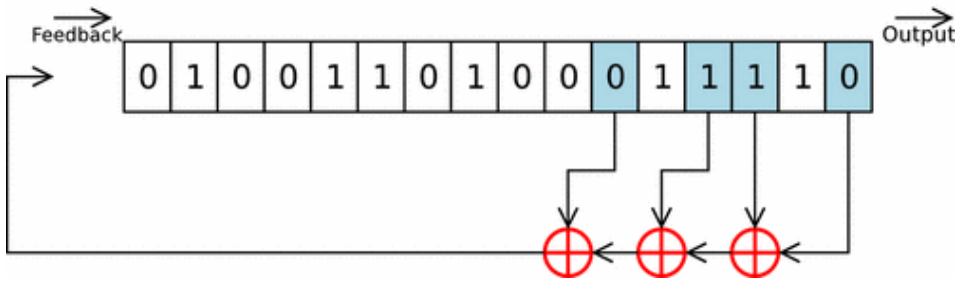

Feedback
Output
1 0 0 1 1 0 1 0 0 0 1 1 1 1 0 0


Feedback
Output
1 0 0 1 1 0 1 0 0 0 1 1 1 1 0 0
1
1
0
0
1
0
0


Feedback
Output
0
1 0 0 1 1 0 1 0 0 0 1 1 1 1 0 0


Feedback
Output
0 1 0 0 1 1 0 1 0 0 0 1 1 1 1 0
0


Feedback
Output
0 1 0 0 1 1 0 1 0 0 0 1 1 1 1 0


Feedback
Output
0 1 0 0 1 1 0 1 0 0 0 1 1 1 1 0

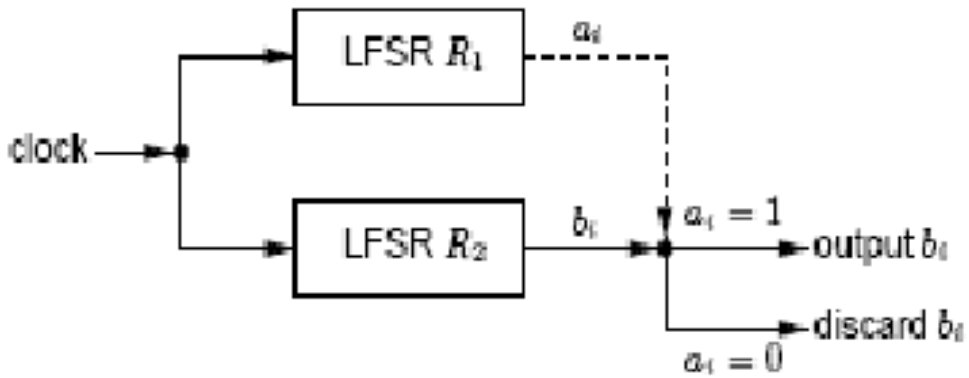
Linear Feedback Shift Registers and Stream Ciphers

- LFSRs have been investigated extensively.
- They have extremely fast implementations in hardware and even in software.
- Closely related to irreducible polynomials over Z_2.
- With correct choice of wiring and initialization, output stream has a very long period.
- However, they are way too weak for cryptographic use – a relatively short output stretch allows to determine secret coefficients efficiently.
- Multiplexing or combining several LFSRs, and adding non-linear components, do produce good stream ciphers.

The Shrinking Generator Stream Ciphers, Based on LFSR (Coppersmith, Krawczyk, Mansour, 1998)

The shrinking generator employs two LFSR, R_1 and R_2. Denote the output bits steam of R_1 by $a_0, a_1, \ldots, a_i, \ldots$ and of of R_2 by $b_0, b_1, \ldots, b_i, \ldots$

The output of the shrinking generator is defined as follows:
If $a_i = 1$, the bit b_i is added (concatenated) to the output sequence.
If $a_i = 0$, the bit b_i is discarded.



(figure from M. Sikandar Hayat Khiyal, Aihab Khan, and Saria Safdar.)

The Shrinking Generator, cont.

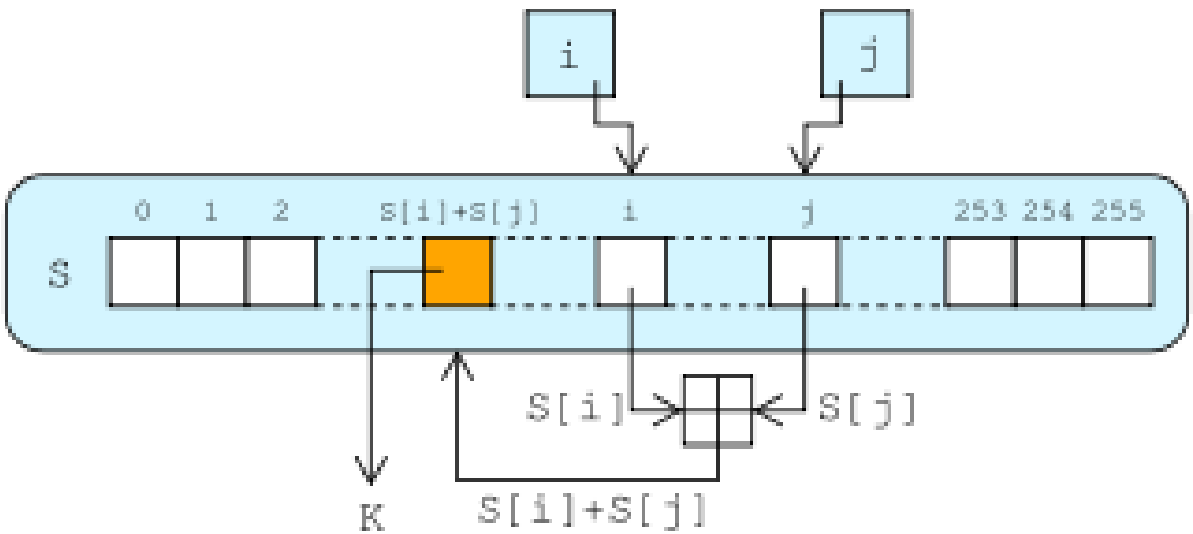
Coppersmith, Krawczyk, and Mansour have shown that the shrinking generator shares the positive aspects of LFSRs (long period, close to uniform distribution of substrings) but is not susceptible to the attacks that are successful on LFSRs.

Example of a Synchronous Stream Cipher: RC4

- Part of the RC family.
- Claimed by RSA as their IP.
- Between 1987 and 1994 its internal was not revealed – little analytic scrutiny.
- Had preferred export status.
- Code was released (anonymously) on the Internet.
- Used in many systems: Lotus Notes, SSL, WEP (wireless networks) etc.
- Various weaknesses were discovered later, so now RC4 is not recommended for use.

RC4: Initialization

1. $j = 0$
2. $S_0 = 0, S_1 = 1, \ldots, S_{255} = 255$ (S_i are 8 bits each).
3. Let the secret, binary seed be $k_0, \ldots, k_{255}$ (repeating bits if key has fewer bits)
4. For $i = 0$ to 255 (permute blocks)
   - $j = (j + S_i + k_i) \mod 256$
   - Swap S_i and S_j

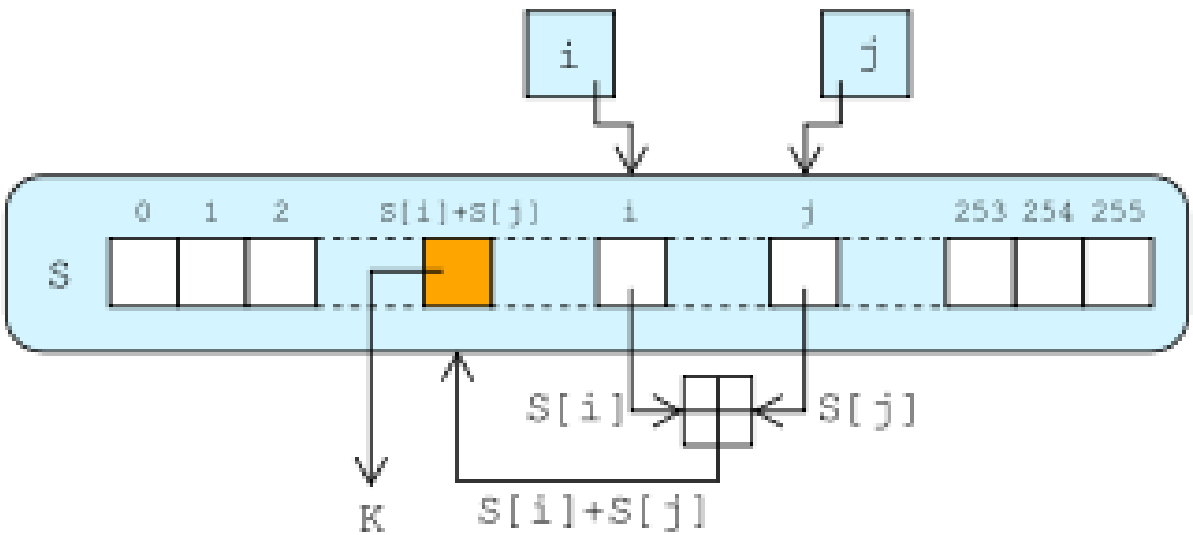


RC4: Keying Stream Creation

Initially $i = 0$. An output byte K is generated as follows:

- $i = i + 1 \mod 256$
- $j = j + S_i \mod 256$
- Swap S_i and S_j
- $r = S_i + S_j \mod 256$
- $K = S_r$

K is the next keying stream byte, and is XORed with the next plaintext byte to produce ciphertext byte.



RC4 Properties

- Synchronous stream cipher, with byte oriented operations.
- Based on using a randomly looking permutation of the internal S_i.
- 8–16 machine operations per output byte.
- Very long cipher period (over 10^{100}).
- Used for encryption in SSL web protocol.
- Certain non careful ways of using key known to be vulnerable (2005). This includes WEP!

And Now for Something Completely Different

PRGs and OW Functions (reminder)

- Both PRGs and OW functions are central primitives in the theory of cryptography.
- If P=NP then neither PRGs nor OW functions exist (why?).
- So the existence of both can only be conjectured until P vs. NP is resolved.
- However there are good reasons to believe that both PRGs and OW functions do exist.
- Furthermore, it was shown that PRGs exist iff OW functions exist.
- We will see portions of the easier direction: PRGs imply OW functions.
- We just saw the relation of PRGs to stream ciphers.

Bit Commitment

A two party protocol between computationally bound Alice and Bob.

Goals:

- Alice commits to a bit b (which she is chooses).
- Bob cannot tell what b is after the commitment phase.
- At a decommit phase, Alice reveals b, and Bob is convinced this is indeed the bit Alice committed to.
- Alice cannot convince Bob she committed to $\overline{b}$

In the physical world, if Alice and Bob are in the same room, they can implement bit commitment by using opaque envelops.

But how can we implement commitment if they are half the globe apart?

PRGs and Bit Commitment (Mony Naor, 1991)

To commit to the bit $b \in \{0,1\}$ (known to Alice, but not to Bob):

- Let $G : \{0,1\}^n \mapsto \{0,1\}^{3n}$ be a pseudo random generator.
- Bob chooses $r \in \{0,1\}^{3n}$ at random and sends it to Alice.
- Alice chooses $s \in \{0,1\}^n$ at random, and computes $G(s) \in \{0,1\}^{3n}$.
- If $b = 0$, Alice sends $t = G(s)$ to Bob. Otherwise, she sends $t = G(s) \oplus r$ to Bob (bitwise XOR).
- To decommit, Alice sends s. Bob computes $G(s)$ and checks whether $t = G(s)$ or $t = G(s) \oplus r$.
- Since G is a PRG, it is that easy to see that indeed Bob cannot predict b any better than guessing at random.
- But why can't Alice cheat by picking s, s' such that $t = G(s) \oplus r = G(s')$? (hint: a counting argument – how many pairs s, s' can "hit" r?)

Properties of Naor's Bit Commitment

- Hiding is computational (secrecy does not hold if Bob is computationally unbounded).
- Binding is unconditional (works vs. any adversary).

Can it be done the other way, namely unconditional hiding and computational binding?

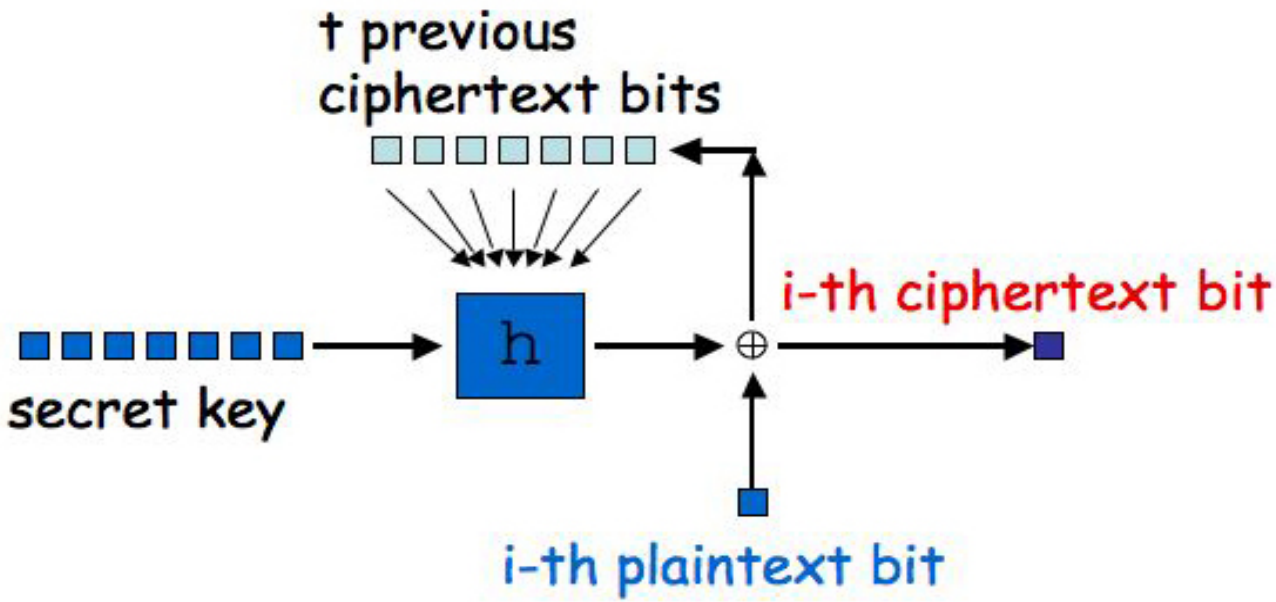
Can unconditional security with respect to both hiding and binding be achieved?

Back to Stream Ciphers

Can the current plaintext bits influence future ciphertext bits?

Asynchronous (or, Self Synchronizing) Stream Ciphers

- Start with a secret, random key ("seed"), k.
- Generate (online) a keying stream. The i-th bit of the keying stream is a function of the seed and (possibly) the most recent t (constant) bits of the ciphertext, $c_{i-t}, \ldots, c_{i-1}$ (initially all 0).
- Specifically, encryption is $c_i = m_i \bigoplus h(k, c_{i-t}, \ldots, c_{i-1})$
- While decryption is $m_i = c_i \bigoplus h(k, c_{i-t}, \ldots, c_{i-1})$



Synchronous vs. Asynchronous

- In synchronous stream ciphers the receiver must know the location in the ciphertext. Otherwise decryption is not possible, and states should be resynchronized.
- Insertion or deletion of any (non-zero) number of bits is fatal.
- Encrypting of different messages should be done using different "locations" on the keyed stream.
- Synchronous stream ciphers tend to be faster.
- In asynchronous stream ciphers, it is possible to recover from lost or inserted bits after t correct ciphertext bits are received.
- So cipher does not require shared state, and is in fact self synchronizing.

Random Functions

- Suppose $F : \{0,1\}^n \to \{0,1\}^n$ is chosen among all functions from $\{0,1\}^n$ to $\{0,1\}^n$ at random.
- Such F is called a random function.
- Given any set of pairs x_i, y_i with $F(x_i) = y_i$, this gives absolutely no information about the values $F(x)$ for new x.
- Furthermore, for any set of new x, the values $F(x)$ are distributed uniformly and independently.

Keyed Functions

- For each $n > 0$, let $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be an efficiently computable function.
- For each length n, F has two arguments. The first one is called the key.
- We will use the notation $F_k(x) = y$, where k is the key.
- The number of such functions, $F_k : \{0,1\}^n \to \{0,1\}^n$ is 2^n.
- Note that 2^n is way smaller than the number of all functions $F : \{0,1\}^n \to \{0,1\}^n$, which equals $2^{n \cdot 2^n}$.
- We now want to formalize the notion that such collection of functions F_k is pseudo random.

Random and Pseudo Random Functions

- Let $\mathcal{A}_n$ be the uniform distribution over all functions $F : \{0,1\}^n \to \{0,1\}^n$.
- Let $\mathcal{B}_n$ be the uniform distribution over all functions $F_k\{0,1\}^n \to \{0,1\}^n$.
- Let D be a polynomial time machine that has oracle[1] access to a function $G : \{0,1\}^n \to \{0,1\}^n$.
- On input 1^n, D adaptively chooses $x_i \in \{0,1\}^n$ and receives $G(x_i)$ (for polynomially in n many i's).

[1] we will draw such machine D with an oracle access on the board

Random and Pseudo Random Functions, cont.

We say that $F_k\{0,1\}^n \rightarrow \{0,1\}^n$ is a collection of pseudo random functions if any such D (with adaptive queries) cannot distinguish between the two distributions.

Formally, let

$p_{D,A}(n) = \text{Prob}(D(G) \text{ outputs } 1 : G \stackrel{R}{\leftarrow} \mathcal{A}_n)$

$p_{D,B}(n) = \text{Prob}(D(G) \text{ outputs } 1 : G \stackrel{R}{\leftarrow} \mathcal{B}_n)$

then we require that for every $\varepsilon > 0$ and every polynomial distinguisher D there is an n_0 such that for all $n \geq n_0$,

$$\mid p_{D,A}(n) - p_{D,B}(n) \mid < \varepsilon \ .$$

Pseudo Random Functions, Reflections

A collection of pseudo random functions (PRFs) extends the notion of pseudo random generator.
Pseudo random generators expands a seed to a sequence.
Pseudo random functions allow a "random access" to an exponential collection of sequences.
However, the collection F_k cannot be truly random (why?).

We will show (on the board) a beautiful construction (by Goldreich, Goldwasser, Micali 1986) of PRF from PRG. However, we will not present a proof of pseudo randomness.

Pseudo random Permutationss

We say that $F_k\{0,1\}^n \to \{0,1\}^n$ is a collection of pseudo random permutations if

- $F_k\{0,1\}^n \to \{0,1\}^n$ is a collection of pseudo random functions.
- For each k, $F_k\{0,1\}^n \to \{0,1\}^n$ is a permutation.

Important Theoretical Result: It is possible to construct a collection of pseudo-random permutations based on any collection of pseudo-random functions (Luby and Rackoff, 1988).

Block ciphers, which we will now introduce, are concrete constructions (of fixed lengths n) that attempt to provide a collection of pseudo random permutations. We will require that for every key k, both $F_k(\cdot)$ and $F_k^{-1}(\cdot)$ are efficiently computable.

Block Ciphers

- Encrypt a block of input to a block of output.
- Almost always the two blocks are of the same length.
- A block cipher is a concrete implementation of pseudo random permutations, with specific block sizes.
- Typically $n = 64$ (DES) or $n = 128$ (AES).
- Actual lengths of key and blocks may sometimes differ (slightly).