CSL7480 Cryptography **23 Jan 2023**

Lecture 9: Cryptography

Lecturer: Somitra Sanadhya *Scribe: Kulkarni Tanmay Shreevallabh (B20CS029)*

9.1 Recap

9.1.1 Stream Ciphers

A Pseudo Random Generator (PRG) is a function G such that ,

$$G : \{0,1\}^n \rightarrow \{0,1\}^m, \qquad m >> n$$

The encryption scheme (Gen, Enc, Dec) is such that

$$Gen = G(.) \text{ (where G is a PRG)}$$

$$Enc(k,m) : G(k) \oplus \text{m} = \text{c}$$

$$Dec(k,c) : G(k) \oplus \text{c} = \text{m}$$

The problem with the above function is that:

1. It is deterministic
2. Not secure
   - Dr. Hongjun Wu elaborated this in his paper titled "The Misuse of RC4 in Microsoft word and excel" (https://eprint.iacr.org/2005/007.pdf)
   - It can clearly be seen that, $Enc(k, m_1) \oplus Enc(k, m_2) = m_1 \oplus m_2$. This is leak of information.

9.1.2 Using Initial Vectors (IVs)

Requirements :

1. Generator of the IV
2. We need an enhanced PRG. Now, we view a pseudo-random generator as taking two inputs: a seed s and an initial vector IV of length n. The IV will remain public.
3. The requirement is that $G(s, IV)$ is pseudo-random even when IV is public (but s is kept secret).

4. The above can be formalised by requiring that no polynomial-time distinguisher D can tell the difference between: $((IV_1; G(s; IV_1); IV_2; G(s; IV_2)))$ and $(IV1; r1; IV2; r2)$ where $r1$ and $r2$ are independently-chosen uniformly-random strings of appropriate length.

Given a generator as above, the Encryption scheme can be defined as:

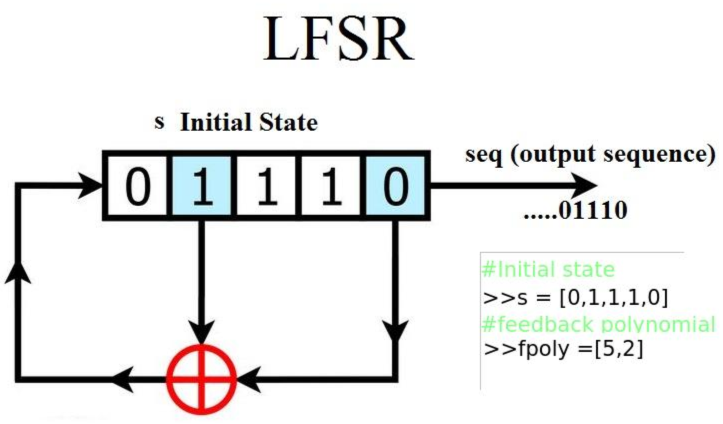
$$Enc(k, m_1) = (IV_1, G(IV_1, k) \oplus m)$$

Stream Ciphers are designed keeping in mind their software or hardware friendliness. One interesting component of hardware friendly stream ciphers is a Linear Feedback Shift Register (LFSR). We shall now take a look at how an LFSR works.

9.2 Linear Feedback Shift Registers (LFSRs)

9.2.1 Introduction

In computing, a linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state.
The most commonly used linear function for the feedback is the exclusive OR (XOR).Thus, an LFSR is most often a shift register whose input bit is driven by the XOR of some bits of the overall shift register value.
The discussion from hereon will be regarding LFSRs which have XOR as its linear function.



```
#Initial state
>>s = [0,1,1,1,0]
#feedback polynomial
>>fpoly =[5,2]
```

LFSRs are capable of generating pseudo-random sequences and are hence used in stream ciphers.
The initial value of the LFSR is called the seed (s mentioned previously). The LFSR generates one bit of the output every clock cycle. The stream of bits so generated happens to be pseudo-random.
It is also worth noting that the first n outputs of the stream generated by the LFSR is the seed itself. The bits which are used as the input to the feedback function are called the tappings bits. The bits are numbered from 1 to n from left to right. In the above diagram, the tapping bits are therefore $\{2, 5\}$.

9.2.2 State Diagram

The state of an $n - bit$ LFSR at any point of time is defined by the $n - bit$ number stored in the LFSR at that point.

The state diagram of an LFSR is a Directed graph $G(V, E)$, where V contains all possible $n - bit$ states of the LFSR can achieve and E contains edge $\{n_1, n_2\}$ iff the LFSR transitions from state n_1 to state n_2 after one clock cycle. A few points to note here are:

1. The state diagram of an LFSR depends on the bits that we have tapped. an $n - bit$ LFSR loaded with the same seed will have different state diagrams when the tapping bits are changed.

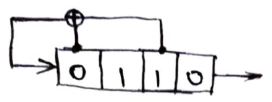
2. The state diagram of a LFSR will always contain directed cycles.

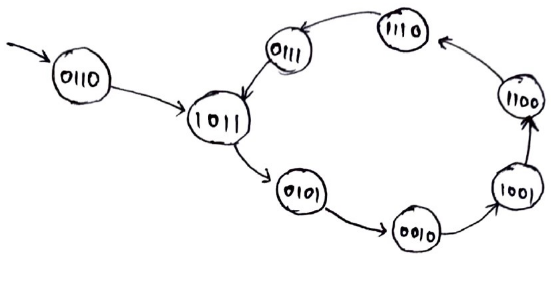
3. The state 0 ($000...n - bits$) is a persistent state and the LFSR, if loaded with 0 as its initial seed, will come back to the same state after every clock cycle. Thus there will be a self loop in the state diagram of every LFSR from the state 0 to itself.

4. Combining the above two points, it can be claimed that the maximal length of a cycle present in the state diagram of an $n - bit$ LFSR is 2^{n-1}.

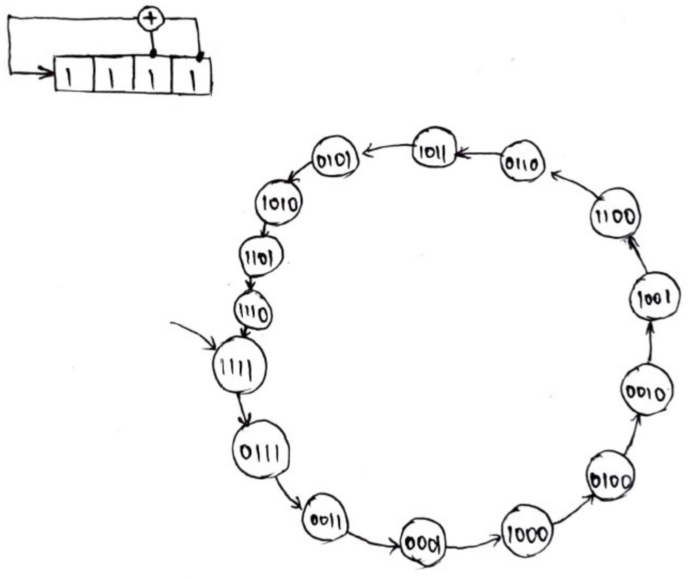
Some examples of a state diagram of a 4-bit LFSR are shown below.

1. Initial Seed = 0110, Tapping Positions = 1,3.





2. Initial Seed = 1111, Tapping Positions = 3,4.



9.2.3 Feedback Polynomial

A way of representing a LFSR is using a feedback polynomial $f(x)$.
For an $n - bit$ shift register the degree of the polynomial is n.

While representing an $n - bit$ LFSR $f(x)$ is always of the form :

$$f(x) = 1 + C_1x^1 + C_2x^2 + C_3x^3...C_nx^n$$

where $C_i = 1$ if the i^{th} bit is tapped, and $C_i = 0$ otherwise, $\forall \ i \in \{1, n\}$
The feedback polynomial for the previous two examples, would therefore be

$$f_1(x) = 1 + x + x^3, \text{ and}$$

$$f_2(x) = 1 + x^3 + x^4$$

Only the LFSRs with *primitive* polynomials as their feedback polynomials produce cycles of maximal length. To read more about *primitive* polynomials, refer https://en.wikipedia.org/wiki/Primitive_polynomial_(field_theory)#Pseudo-random_bit_generation.
Necessary but possibly insufficient conditions for $f(x)$ to be a *primitive* polynomial are:

1. Number of terms of the form x^p $s.t$ $p > 0$, are even.
2. The sequence $\{i_1, i_2...i_k, 0\}$, $s.t$ $C_i = 1 \, \forall \, i \in \{i_1, i_2...i_k\}$ is called the *tapping sequence* of the polynomial. Note that the tapping sequence for a polynomial always contains 0, because the coefficient of

x^0 is always 1 for LFSRs. For a primitive polynomial, all the elements in the tapping sequence are pairwise co-prime.

3. If the tapping sequence for a primitve polynomial is $\{i_1, i_2, i_3...0\}$, such that $i_1 > i_2 > i_3... > 0$ then the sequence $\{i_1 - i_1, i_1 - i_2, i_1 - i_3...i_1 - 0\} = \{0, i_1 - i_2, i_1 - i_3...i_1\}$ will also be a tapping sequence of a primitive polynomial.

Some primitive polynomials of various degrees can be found here: https://en.wikipedia.org/wiki/Linear-feedback_shift_register#Example_polynomials_for_maximal_LFSRs
Richard Brent has been tabulating primitve trinomials, such as $x^{74207281} + x^{30684570} + 1$. This can be used to create a pseudo-random number generator of the huge period, $2^{74207281} - 1 \approx 3 * 10^{22338617}$.

9.3 Chosen Plain Text Attack

9.3.1 Attack Model

The security model is defined through the Adversarial game. The game is as follows:

1. The challenger chooses an $n - bit$ secret key k uniformly from at random from the set of all $n - bit$ strings and sets up the Encryption function.

$$k \xleftarrow{\$} \{0,1\}^n$$

$$Enc_k(.)$$

2. The attacker is then allowed to ask for the Encryption of as many messages as he/she wants.

$$\begin{array}{ccc} Challenger & & Attacker \\ & \xleftarrow{\quad \mathrm{m}_1 \quad} & \\ & \xrightarrow[Enc_k(m_1)]{} & \\ & \xleftarrow{\quad \mathrm{m}_2 \quad} & \\ & \xrightarrow[Enc_k(m_2)]{} & \\ & . & \\ & . & \\ & . & \end{array}$$

3. The attacker finally chooses two messages $\{m_o^*, m_1^*\}$ s.t $|m_o^*| = |m_1^*|$ and sends them to the challenger

$$\begin{array}{ccc} Challenger & & Attacker \\ & \xleftarrow{\quad \{\mathrm{m}_o^*, m_1^*\} \quad} & \end{array}$$

4. The challenger then decides which one of $\{m_o^*, m_1^*\}$ does he/she want to encrypt, and sends the encrypted text to the attacker.

5. The attacker then produces b'.
Attacker wins if $b' == b$

6. If the attacker loses, i.e $b' \neq b$, the attacker can continue asking queries, even after the challenge.

$$\begin{array}{ccc} \mathit{Challenger} & & \mathit{Attacker} \\ b \xleftarrow{\$} \{0,1\}^* & & \\ Enc_k(m_b) = c & & \\ & \xrightarrow{\text{c}} & \\ & \mathit{Challenge\ unsuccessful} & \\ & \xleftarrow{\text{m}_1} & \\ & \xrightarrow[Enc_k(m_1)]{} & \end{array}$$

7. If $Pr[Attacker\ wins] \leq \frac{1}{2} + \in(n)$, where $\in(n)$ is a negligible function over the security parameter n, then the scheme is said to be Chosen Plaintext Attack (CPA) Secure.

9.3.2 Difference between the above and Single message Eavesdropping Security and Multimessage Eavesdropping security

1. Single Message Eavesdropping Security (SMES)
   (a) No queries are allowed before the challenge
   (b) $\{m_o^*, m_1^*\}$ were given to the challenger directly.
2. Multimessage Eavesdropping Security (MMES)
   (a) No queries are allowed before the challenge
   (b) Only $M_o = (m_o^1, m_o^2, m_o^3 ... m_o^t)$
   and $M_1 = (m_1^1, m_1^2, m_1^3 ... m_1^t)$ were given to the challenger by the attacker.
   (c) It can be seen that no encryption scheme (Gen, Enc, Dec) can be MME secure if Enc is a deterministic function.
   Proof:
   Let (Gen, Enc, Dec) be an encryption scheme, such that Enc is deterministic.
   Choose $M_o = (m_o, m_o)$
   and $M_1 = (m_o, m_1)$
   Clearly, the attacker can win with probability = 1, as if the returned set of cipher texts contains the same text twice, M_o was chosen, and M_1 was chosen otherwise.

In CPA, Queries are allowed before, and even after the first challenge. That is, the attacker can ask queries, then challenge, then continue to ask queries, then challenge again, and so on.

9.3.3 An Example of Chosen Plaintext Attack form WWII

In World War II a famous example for chosen-plaintext attacks took place. US cryptoanalysts intercepted an encrypted message from the japanese, which they were able to partially decode. It stated that the Japanese were planning an attack on AF, where AF was a ciphertext, the US was unable to decode.

The US now believed, Midway Island was the target, but couldn't convince the authorities of this assumption, since the general belief was that Midway Island could not possibly be the aim. The US then carried out a chosen-plaintext attack by encrypting the fake message that Midway island was low on water supply. The Japanese intercepted this message and immeditately reported to their superiors "AF is low on water."

Of course, this was the proof the US cryptoanalysts needed and the US immediately sent several aircraft carriers, resulting in the rescue of Midway Island.

9.4 Oracle

9.4.1 Introduction

An oracle, by definition is a medium through which god speaks. In a mathematical context, an oracle can be any mathematical device/function which is used by an algorithm, without the knowledge of how the oracle works internally.
Putting it simply, in a mathematical context, an oracle is a mathematical device about which we only know *what* is does, and not *how* it does whatever it is doing.

9.4.2 Oracle in the context of Cryptography

In the context of cryptography, an oracle is very similar to what it is in the mathematical context. The idea is that the adversary, while trying to break an encryption scheme (Gen, Enc, Dec) uses an algorithm A, which uses the Encryption algorithm $Enc_k(.)$ as its oracle, **without** the knowledge of the secret key k.

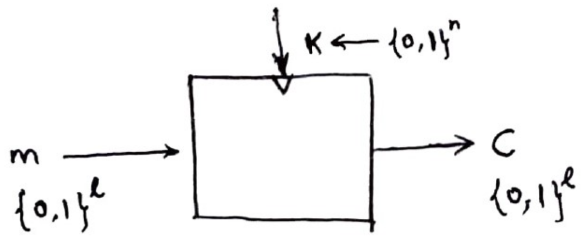
The idea of the ecryption scheme being CPA secure from the adversary, is then notationally represented as:

$$Pr[A^{Enc_k(.)}(win)] \quad \leq \quad \tfrac{1}{2} + \in (n)$$

Where A is the algorithm used by the adversary, which uses $Enc(.)$ as its *oracle* , and $\in (n)$ is a negligible function over the security parameter n.

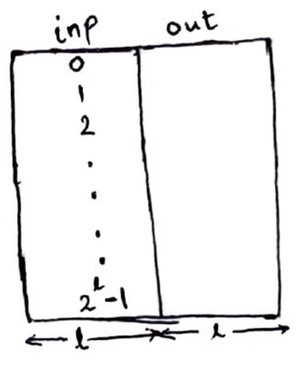
9.5 Block Cipher ≡ Pseudo Random Permutation (PRP)

9.5.1 Introduction



The block cipher can be viewed as a function (black box) that maps every $l-bit$ input, to an $l-bit$ output. The reason it is also called a Pseudo Random Permutation, is because the function used by a block cipher is one-to-one.
Thus, the function can be seen as a device that generates and arrangement (permutation) of all $l-bit$ strings. Since the mapping is one-to-one, it is clear that the decryption is going to be deterministic.

9.5.2 Underlying structure



1. The underlying structure of the PRP is essentially a table, that stores the output of the function for every input.

2. The number of bits required to store such a table, at first glance seem to be $2 \cdot 2^l \cdot l$. However, we do not actually need to store the input, the same job can be done by the index. Thus the actual number of bits required to store such a table is $2^l \cdot l$

3. The total number of pseudorandom functions possible while mapping a $l-bit$ input to an $l-bit$ output are 2^{2^l}.

4. The total number of pseudorandom permutations possible while mapping an $l - bit$ input to an $l - bit$ output are $(2^l)!$.

5. Although the above mentioned table cannot be implemented practically due to the exponential amount of bits required to store the table, one must not forget that the above is just a thought experiment. In reality, we only allow polynomial time queries, thus we need not store the entire table in memory, instead we only need to store the mapping of the queries which the attacker has previously asked. This has to be done to ensure that the cipher does not give a different output for the same input. If that happens, it becomes trivially easy to break the CPA game. The attacker will just query for the encryption of the same message twice, and if the outputs he/she receives both the times are same, then the encryption algorithm was used, and if the answers were generated randomly, the outputs in both the cases will be different, with a very high probability. Thus allowing a trivial break of the CPA scheme.

- https://www.youtube.com/watch?v=sKUhFpVxNWc
- https://en.wikipedia.org/wiki/Primitive_polynomial_(field_theory)
- https://en.wikipedia.org/wiki/Linear-feedback_shift_register#
- https://www.grin.com/document/351992#:~:text=In%20World%20War%20II%20a,US%20was%20unable%20to%20decode.