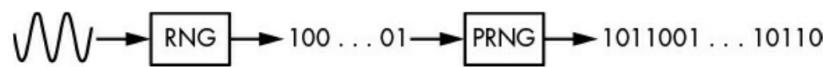


Crypto 2

Note: this material is not intended to replace the live lecture for students.

Contents

2.1	Randomness	2
2.1.1	TRNG	4
2.1.2	PRNG	5
2.1.3	Tests	6
2.2	The Lehmer generator and LCG	7
2.2.1	Park Miller RNG	8
2.3	Stream Ciphers & Operation modes	9
2.3.1	Synchronized	9
2.3.2	Unsynchronized	10
2.4	Stream Ciphers: LFSR	11
2.4.1	Fibonacci LFSRs	12
2.4.2	Galois LFSRs	15
2.4.3	Several LFSRs and the Correlation Attack	17
2.5	Stream Ciphers: Permutation	18
2.5.1	Keccak-Sponge key stream	18
2.5.2	The Keccak- <i>f</i> permutations	19
2.6	Bibliography	22



2.1 Randomness

What is randomness ?

A pseudo-random sequence is a vague notion embodying the idea of a sequence in which each term is unpredictable to the uninitiated and whose digits pass a certain number of tests traditional with statisticians and depending somewhat on the uses to which the sequence is to be put.

[Lehmer51, pag. 143]

Why randomness ?

Randomness is found everywhere in cryptography: in the generation of secret keys, in encryption schemes, and even in the attacks on cryptosystems. Without randomness, cryptography would be impossible because all operations would become predictable, and therefore insecure.

[Aumasson18, Chapter 2]

Figure 2.1.1: Synchronous Stream Cipher

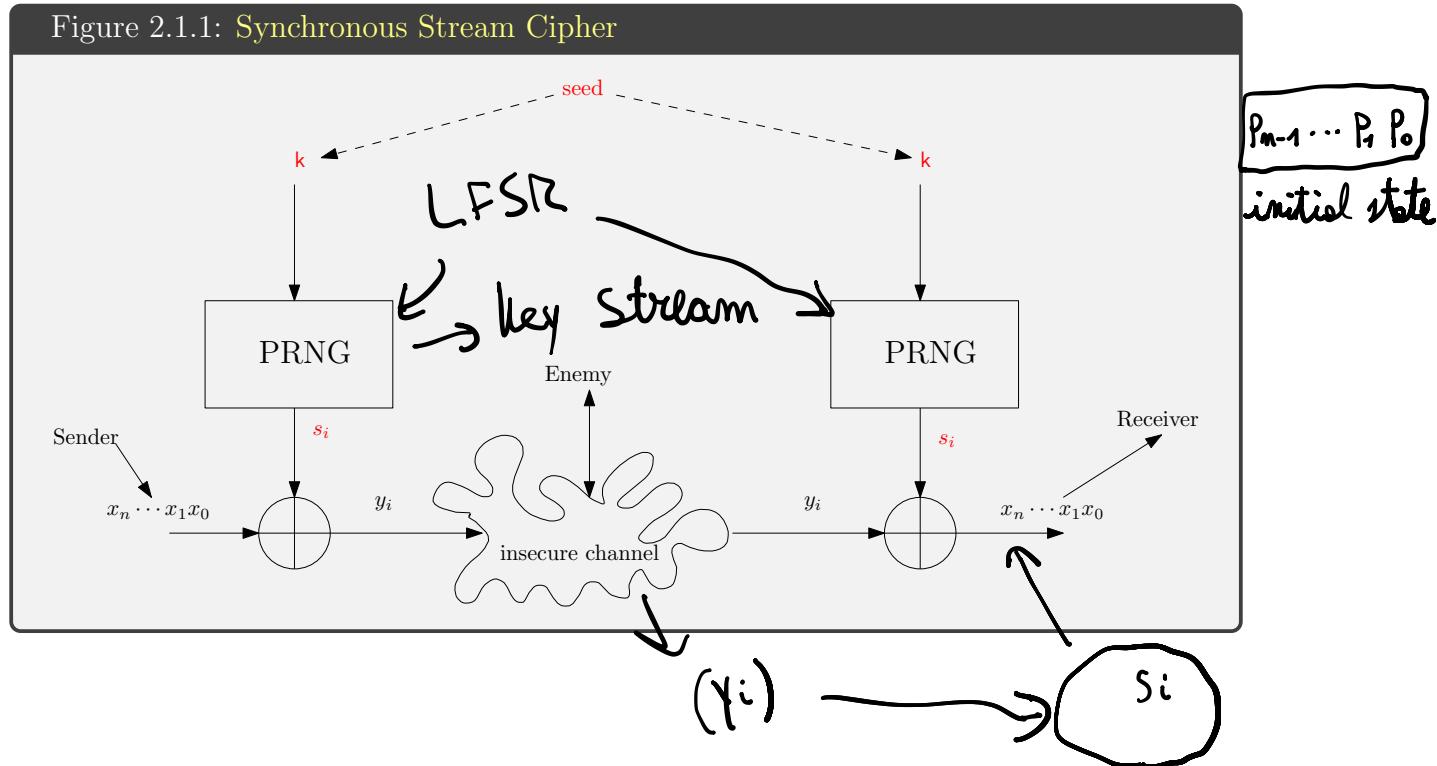
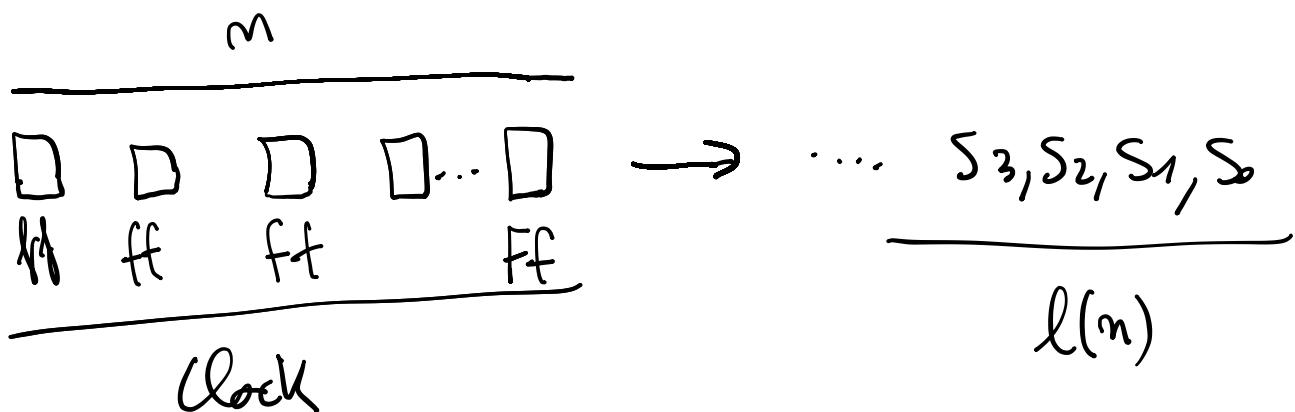


Figure 2.1.2: Probabilistic Encryption

To construct a scheme secure for encrypting multiple messages, we must design a scheme in which encryption is *randomized* so that when the same message is encrypted multiple times, different ciphertexts can be produced. This may seem impossible since decryption must always be able to recover the message. However, we will soon see how to achieve it.



2.1.1 TRNG

If you toss a coin you become a TRNG.

antico adagio

In practice, the cryptoequipment must contain a true random number generator (e.g., a noisy diode) for generating K ,

[DH76, pag. 648]

Figure 2.1.3: True Random Number Generator

In computing, a hardware random number generator (HRNG) or true random number generator (TRNG) is a device that generates random numbers from a physical process, rather than by means of an algorithm. Such devices are often based on microscopic phenomena that generate low-level, statistically random "noise" signals, such as thermal noise, the photoelectric effect, involving a beam splitter, and other quantum phenomena. These stochastic processes are, in theory, completely unpredictable, and the theory's assertions of unpredictability are subject to experimental test. This is in contrast to the paradigm of pseudo-random number generation commonly implemented in computer programs.

A hardware random number generator typically consists of a transducer to convert some aspect of the physical phenomena to an electrical signal, an amplifier and other electronic circuitry to increase the amplitude of the random fluctuations to a measurable level, and some type of analog to digital converter to convert the output into a digital number, often a simple binary digit 0 or 1. By repeatedly sampling the randomly varying signal, a series of random numbers is attained.

The main application for electronic hardware random number generators is in cryptography, where they are used to generate random cryptographic keys to transmit data securely. They are widely used in Internet encryption protocols such as Transport Layer Security (TLS).



This SSL Accelerator computer card uses a hardware random number generator to generate cryptographic keys to encrypt data sent over computer networks.

2.1.2 PRNG

2.1.4 PRNG

Un PRNG è una funzione $G : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2^{l(n)}$, $l(n) > n$ (*expansion factor*) tale che nessun avversario \mathcal{A} riesce con probabilità $> 1/2$ a distinguere il PRNG da un generatore casuale.

The PRG indistinguishability experiment $\text{PRG}_{\mathcal{A}, G}(n)$:

- A uniform bit $b \in \{0, 1\}$ è scelto. Se $b = 0$ allora scegli un uniforme $r \in \{0, 1\}^{l(n)}$; se $b = 1$ allora scegli un uniforme $s \in \{0, 1\}^n$ e setta $r := G(s)$.
- L'avversario \mathcal{A} è dato r , e produce un bit b' .
- Il risultato dell'esperimento è definito come 1 se $b' = b$, e 0 altrimenti.

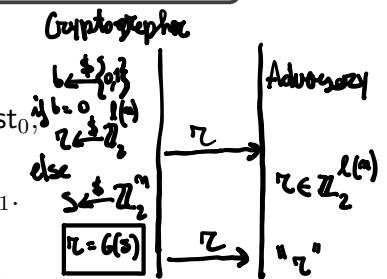
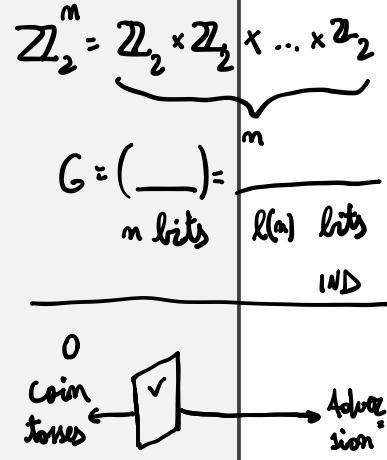
G espanderà n biti $s \in \mathbb{Z}_2^n$ in una lunga sequenza $G(s)$ di $l(n)$ -biti.

È comune costruire un PRNG da due algoritmi (`Init`, `GetBits`):

`Init` prende come input un seme s e un optional *IV* e produce uno stato iniziale st_0 ,

`GetBits` prende un input stato st_i , produce un bit y e aggiorna lo stato a st_{i+1} .

Ilo stati st_i sono gli stati in un grande buffer o memoria chiamata "pool di entropia".

2.1.5 (`Init`, `GetBits`) - PRNG $G(s)$

```

 $\text{st}_0 = \text{Init}(s, IV)$ 
for  $i = 1$  to  $l(n)$ :
     $(y_i, \text{st}_i) = \text{GetBits}(\text{st}_{i-1})$ 
return  $y_1, \dots, y_{l(n)}$ 

```

$$\begin{aligned} m &= s \\ l(m) &= 1000 \end{aligned}$$

$$G(10010) = \underbrace{\quad}_{5 \text{ bits}} \underbrace{\quad}_{1000 \text{ bits}}$$

Attacchatore deve comprendere se i 1000 biti vengono da $b=0$ ($\Rightarrow \{0,1\}^5$ ripetuto 1000 volte) o se vengono dalla funzione $G(s)$.

PRN 6: Lehmer Generator Linear Congruential Generator

2.1 Randomness

Politecnico di Torino.

$G(S_0)$

$S_0 \in \mathbb{Z}_2^{2.1.3}$ Tests

$S_0 \in \mathbb{N}$

$S_0, S_1, S_2, S_3, \dots$
 $\underbrace{\quad\quad\quad}_{l(n)}$

$a, b "m"$

for example

$a = 1103 515245$

$b = 12345$

$m = 2^{31}$

$$S_1 \equiv a \cdot S_0 + b \pmod{m}, S_2 \equiv a \cdot S_1 + b \pmod{m}$$

Cryptographic vs. Non-Cryptographic PRNGs

seed $\rightarrow S_0 = 12345$

rand() ANSI C

There are both cryptographic and non-cryptographic PRNGs. Non-crypto PRNGs are designed to produce uniform distributions for applications such as scientific simulations or video games. However, you should never use non-crypto PRNGs in crypto applications, because they're insecure—they're only concerned with the quality of the bits' probability distribution and not with their predictability. Crypto PRNGs, on the other hand, are unpredictable, because they're also concerned with the strength of the underlying *operations* used to deliver well-distributed bits.

Unfortunately, most PRNGs exposed by programming languages, such as libc's `rand` and `drand48`, PHP's `rand` and `mt_rand`, Python's `random` module, Ruby's `Random` class, and so on, are non-cryptographic. Defaulting to a non-crypto PRNG is a recipe for disaster because it often ends up being used in crypto applications, so be sure to use only crypto PRNGs in crypto applications.

[Aumasson18, Chapter 2]

Will your PRNG do the job?

TestU01

2.2 The Lehmer generator and LCG

METHODS IN LARGE-SCALE UNITS

of this session—the Monte Carlo method. In this method it is necessary to produce random variables. The problem here is not one of producing a table of random digits to be published and used by others. On the contrary, one can think ideally of a perfect stream of these random numbers produced at high speed by the machine and passing by a “gate.” Whenever the computer needs a number it opens the gate and takes one. More explicitly, we might list the following desiderata:

- (1) An unlimited sequence of randomly arranged eight-digit numbers;
- (2) A simple process by which the machine may produce the sequence:

$$u_n = f(u_{n-1}, u_{n-2}, \dots, u_{n-k});$$

(3) Immediate access by the machine to the current number of the sequence whenever necessary; of course, the whole sequence need not be retained in the machine.

If we examine these desiderata we see at once that they are inconsistent. In the first place, the number of eight-digit numbers is not unlimited. There are in fact only 100 million of them. Secondly, condition (2) forces the sequence to be ultimately periodic and therefore not random. We therefore scale down our demands and modify (1) and (2) to read:

- (1) Millions of pseudo-random numbers u_n ;
- (2) $u_n = f(u_{n-1})$, f a simple function.

A pseudo-random sequence is a vague notion embodying the idea of a sequence in which each term is unpredictable to the uninitiated and whose digits pass a certain number of tests traditional with statisticians and depending somewhat on the uses to which the sequence is to be put. The worst possible departure from randomness is to have the period of the sequence small or equal to one. In constructing the function f , therefore, it is of the utmost importance to obtain one that produces a guaranteed proper period of immense length.

[Lehmer51]

$m=4$

$$\{0, 1, 2, 3\}$$

$$0, 3, 1, 2$$

Shift
↑

LFSR:

Period

L.C.G.

Here is the general form a of LCG:

$$\begin{aligned}s_0 &= \text{seed} \\ s_{i+1} &\equiv a \cdot s_i + b \pmod{m}, \quad i = 0, 1, \dots\end{aligned}$$

A well-known example is `rand()` of ANSI C:

$$\begin{aligned}s_0 &= 12345 \\ s_{i+1} &\equiv 1103515245 \cdot s_i + 12345 \pmod{2^{31}}, \quad i = 0, 1, \dots\end{aligned}$$

X Exercise 2.2.1

Setting $b = 0$, $s_0 = 47594118$, $a = 23$ e $m = 10^8 + 1$ show that each s_n is divisible by 17. It is possible to show that *The number 23 is the best choice in the sense that no other number produces a longer period, and no smaller number produces a period more than half as long*



2.2.1 Park Miller RNG

2.2.2 Park Miller RNG

```
# Park - Miller RNG
#
# https://www.youtube.com/watch?v=EXsLyfqwopg
#
# http://www.firstpr.com.au/dsp/rand31/p1192-park.pdf

s = 1
a = 7**5
tp=2**31
m=tp-1
i=1
while i < 10002:
    print(i,s)
    s = (a*s)%m
    i+=1
```

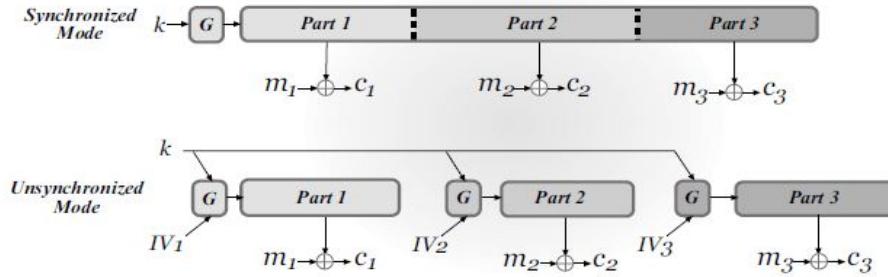
* $s_0 = 47594118$

s_0 div. for 17? Yes, 2723772 with remainder 0

$s_{i+1} \equiv a \cdot s_i \pmod{m}$

$s_{i+1} \equiv 23 \cdot s_i \pmod{10^8 + 1}$

2.3 Stream Ciphers & Operation modes



2.3.1 Synchronized

Using $(\text{Init}, \text{GetBits})$ the parties construct $G(\mathbf{k}, 1^l)$ running l -times using a shared \mathbf{k} . Concretely both begin by computing $\mathbf{st}_0 = \text{Init}(\mathbf{k})$.

To encrypt the first message m_1 of length l_1 the sender runs GetBits l_1 -times beginning at \mathbf{st}_0 getting bits $y_1 \dots y_{l_1} = \mathbf{pad}_1$ along with an updated state \mathbf{st}_{l_1} . It sends $c_1 = \text{Enc}_{\mathbf{k}}(m_1) = \mathbf{pad}_1 \oplus m_1 = \mathbf{pad}_1 \oplus m_1$.

Once the other party receive c_1 it runs GetBits l_1 -times beginning at \mathbf{st}_0 getting bits $y_1 \dots y_{l_1} = \mathbf{pad}_1$ along with an updated state \mathbf{st}_{l_1} . It uses \mathbf{pad}_1 to recover $m_1 = c_1 \oplus \mathbf{pad}_1$.

Later the sender to encrypt a second message m_2 will run GetBits at the state \mathbf{st}_{l_1} to obtain a second $\mathbf{pad}_2 = y_{l_1+1} \dots y_{l_1+l_2}$ and an updated state $\mathbf{st}_{l_1+l_2}$ and so on.

NOTE 2.3.1

We remark that in this mode, the stream cipher does not need to use the initialization IV .

2.3.2 Unsynchronized

In this case ciphertexts are pairs :

$$\text{Enc}_{\mathbf{k}}(m_j) = \langle IV_j, G(\mathbf{k}, IV_j, 1^{|m_j|}) \oplus m_j \rangle$$

where $G(\mathbf{k}, IV_j, 1^{|m_j|})$ is the **pad** obtained by running $\text{Init}(\mathbf{k}, IV_j)$. Decryption is performed in the natural way.

NOTE 2.3.2

Typical mistakes are related to the optional IV e.g. a fixed constant in soft or hardware. If IV is random then $\text{Enc}_{\mathbf{k}}(m) = \langle IV, G(\mathbf{k}, IV, 1^{|m|}) \oplus m \rangle$ is CPA-secure. But not CCA-secure.

Exercise 2.3.3

Let $G(\mathbf{k}, IV, 1^{|m|})$ as before and consider the following symmetric encryption scheme:

\mathbf{k} is random generated by **Gen** and shared by the parties.

Enc: on input a key \mathbf{k} and a message m , choose a random IV and output:

$$\text{Enc}_{\mathbf{k}}(m) = \langle IV, G(\mathbf{k}, IV, 1^{|m|}) \oplus m \rangle$$

Dec: on input a key \mathbf{k} and a ciphertext $\langle IV, c \rangle$ output the plaintext message:

$$\text{Dec}_{\mathbf{k}}(\langle IV, c \rangle) = G(\mathbf{k}, IV, 1^{|c|}) \oplus c = m$$

Show that the above is not CCA-secure. Hint: an adversary can guess the bit b using $m_0 = \text{all zeros}$ and $m_1 = \text{all ones}$ and a CCA-oracle query.

2.4 Stream Ciphers: LFSR

A [linear-feedback shift register \(LFSR\)](#) is a [shift register](#) whose state transition is linear.

The state \mathbf{s} is a row vector of m bits, i.e. $\mathbf{s} \in \{0, 1\}^m$.

The sequence of bits in the rightmost position of the state is **the output stream** or key stream.

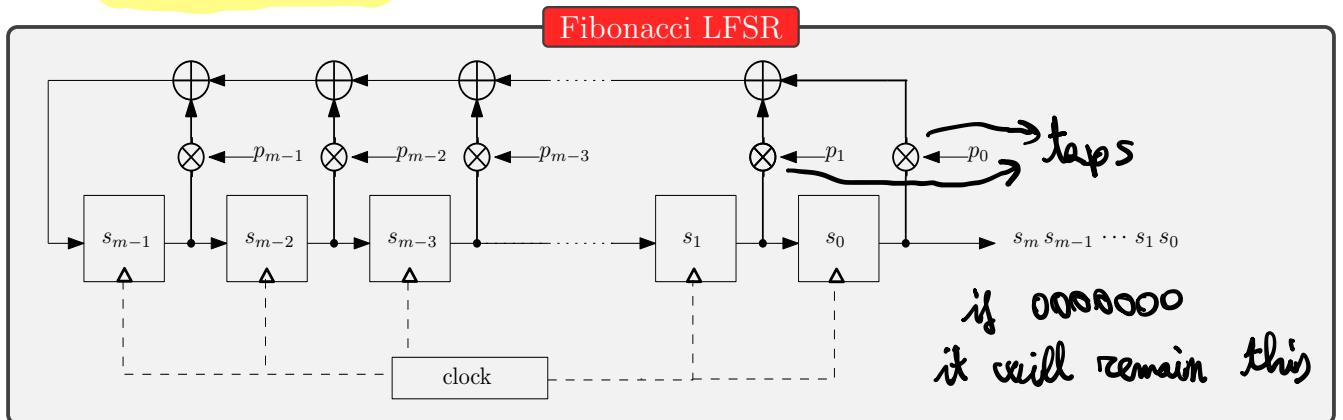
The bits in the LFSR state that influence its behaviour are called **taps**.

A **maximum-length** LFSR produces an m-sequence (i.e., it cycles through all possible $2^m - 1$ states within the shift register except the state where all bits are zero), unless it contains all zeros, in which case it will never change.

Here I discuss: Fibonacci and Galois LFSRs. In both cases the LFSR is specified by either the characteristic polynomial $\chi_L(x)$ of the linear transition map or by the so called **feedback polynomial** $P(x)$ which is the reciprocal of $\chi_L(x)$.

Two given LFSRs are **mirror** LFSRs if they characteristic polynomial are reciprocal between them. The output stream of a LFSR is reversible and given by the output stream of its reciprocal LFSR. So if a LFSR has maximum-length also its mirror has maximal-length.

2.4.1 Fibonacci LFSRs



The output bit s_m is computed as a feedback

$$s_m = f(s) = \sum_{j=0}^{m-1} s_j \cdot p_j$$

The feedback polynomial is : $P(x) = 1 + p_{m-1}x + \dots + p_1x^{m-1} + p_0x^m$ and the characteristical polynomial is $\chi_L(x) = x^m P\left(\frac{1}{x}\right) = p_0 + p_1x + \dots + p_{m-1}x^{m-1} + x^m$

$$\chi_L(x) = x^m P\left(\frac{1}{x}\right) = p_0 + p_1x + \dots + p_{m-1}x^{m-1} + x^m$$

$p_0 \rightarrow p_{m-1}$

Here the involved linear map L is:

$$L = \begin{bmatrix} p_{m-1} & 1 & 0 & 0 & \cdots & 0 \\ p_{m-2} & 0 & 1 & 0 & \cdots & 0 \\ p_{m-3} & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ p_1 & 0 & 0 & 0 & \cdots & 1 \\ p_0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

(mod 2)

polynomial

XOR — +

— · ⊗

+

Being the state $s = [s_{m-1} s_{m-2} s_{m-3} \dots s_1 s_0]$ the transition to the state s' is given by the multiplication: $s \cdot L = s'$, $s \cdot L = s_0 \cdot P_{m-1} + s_{m-1} \cdot P_{m-2} + \dots + s_1 \cdot P_1 + s_0 \cdot P_0$

or at a bits level:

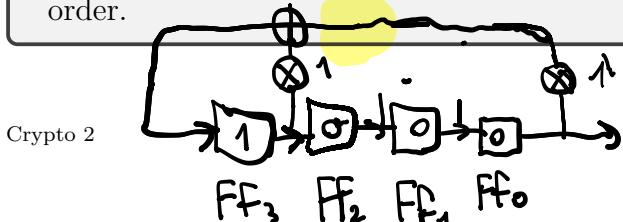
$$s' = [f(s) s_{m-1} s_{m-2} s_{m-3} \dots s_2 s_1].$$

$$P_M \rightarrow x^4 + x^3 + x^2 + x + 1$$

$$P_{M-1} \rightarrow 0 \quad 0 \quad 1 \quad 0 \quad 0$$

2.4.1 Example

Run the Fibonacci LFSR specified by $\chi_L(x) = x^4 + x^3 + 1$ from the state [1000]. How long is its period? Then run its mirror and check that it cycle through the output stream in reverse order.



$$\chi_L(x) = P_0 + P_1x + P_2x^2 + P_3x^3 + x^4$$

Cryptography 2024

derived = 1 0 0 1

2.4 Stream Ciphers: LFSR

initial state $[1\ 0\ 0\ 0]$

1	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

1st clock
2nd
3rd
4th

1	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

1	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

Politecnico di Torino

The generatrix power series

The key stream output s_0, s_1, \dots determines a formal power series^a:

$$\sum_{j=0}^{\infty} s_j x^j = s_0 + s_1 x + s_2 x^2 + \dots$$

It is possible to show that

$$\sum_{j=0}^{\infty} s_j x^j = \frac{Q(x)}{P(x)}$$

where $P(x)$ is the feedback polynomial of the LFSR and

$$Q(x) = \sum_{\mu=0}^{\mu=m-1} \left(\sum_{j=m-\mu}^m s_{\mu+j-m} p_j \right) x^{\mu}$$

depends on the initial state of the register.

One interesting consequence of the above equation is that it can be used to improve the implementation of key stream by reducing the taps of the LFSR. Here an example: since

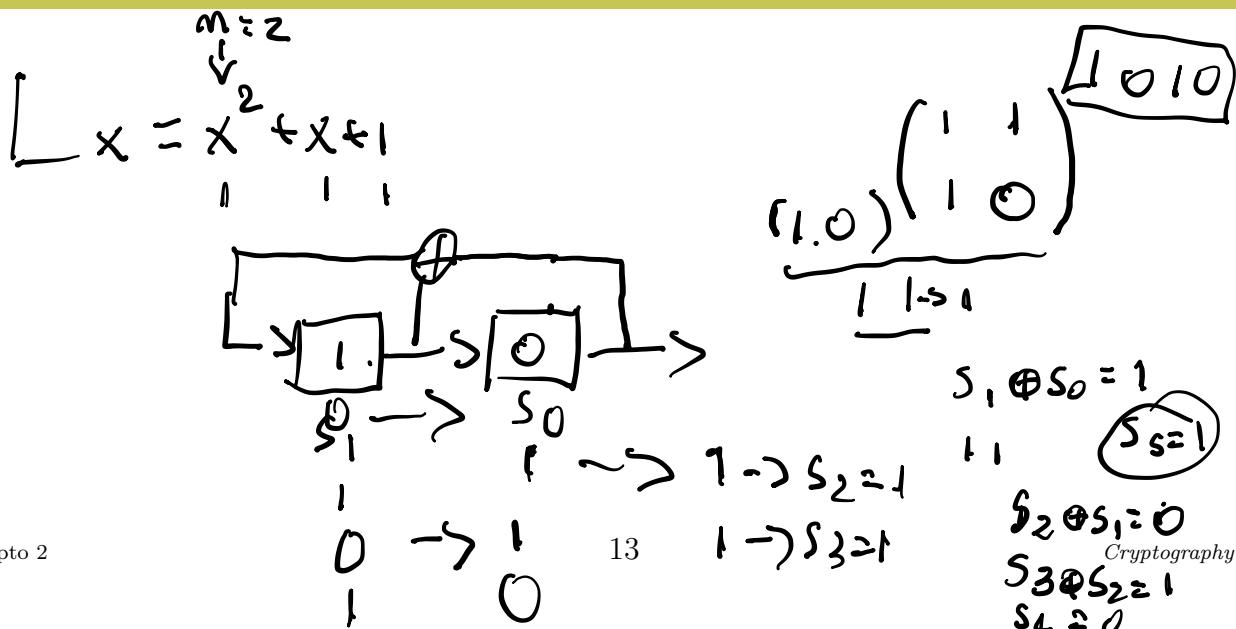
$$1 + x + x^8 = (1 + x^2 + x^3 + x^5 + x^6)(1 + x + x^2)$$

the key stream produced by a LFSR with feedback polynomial $1 + x^2 + x^3 + x^5 + x^6$ can be obtained using a LFSR with feedback polynomial $1 + x + x^8$.

^acalled generatrix function

Exercise 2.4.2

If a key stream satisfies $s_{t+6} = s_t + s_{t+1} + s_{t+3} + s_{t+4}$ then it also satisfies $s_{t+8} = s_t + s_{t+7}$.



LFSR are not crypto secure. They are broken in the KPA (Known Plaintext Attack).

2.4.3 A handmade attack to a 3-bits LFSR

2.4.3 A handmade attack to a 3-bits LFSR

Handwritten notes:

- State sequence: $s_5 s_4 s_3 s_2 s_1 s_0$
- Feedback polynomial: $s_m = s_2 p_2 + s_1 p_1 + p_0 s_0 \pmod{2}$
- Equations derived from known plaintext (101, 110, 111):

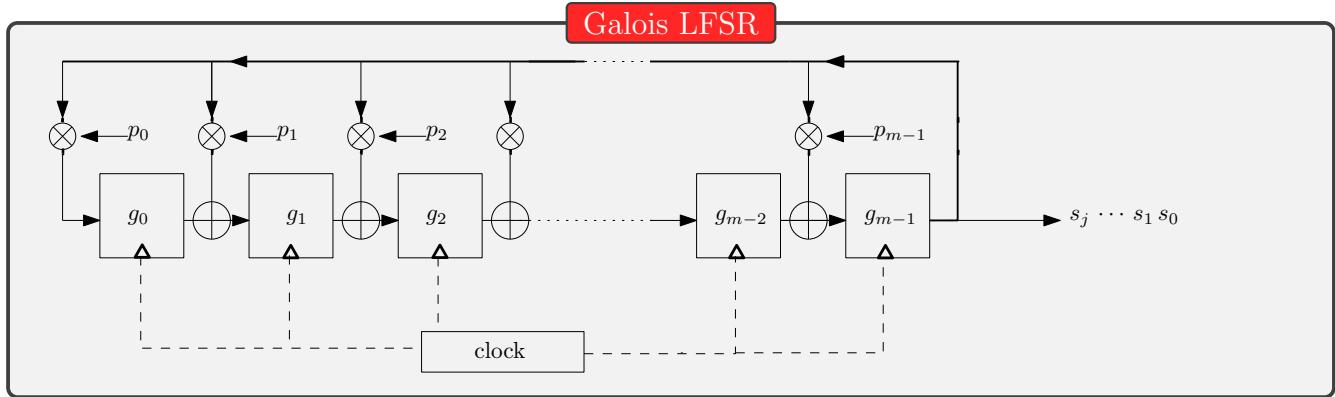
$$\begin{aligned} s_3 &= s_2 p_2 + s_1 p_1 + p_0 s_0 \\ s_4 &= s_3 p_2 + s_2 p_1 + p_0 s_1 \\ s_5 &= s_4 p_2 + s_3 p_1 + p_0 s_2 \end{aligned}$$
- Tables of binary additions:

1	0	1	= 1
1	1	0	= 1
1	1	1	= 0

1	0	1	= 1
0	1	1	= 0
0	1	0	= 1

1	0	1	= 1
0	1	1	= 0
0	0	1	= 0
1	1	1	= 1

2.4.2 Galois LFSRs



In the Galois configuration the state $s = [g_0 g_1 \dots g_{m-2} g_{m-1}]$ is regarded as a polynomial

$$s(x) = g_0 + g_1 x + \dots + g_{m-2} x^{m-2} + g_{m-1} x^{m-1}$$

← state

$q_0 q_1 q_2 q_3$

When the system is clocked the new state is

$$s'(x) = x \otimes s(x) \quad x \cdot s(x) = g_0 x + g_1 x^2 + \dots + g_{m-1} x^m$$

where \otimes is as in the Galois cipher with $G(x) = \chi_L(x) = x^m + p_{m-1}x^{m-1} + \dots + p_1x + p_0$. So

$$x \cdot s(x) = g_0 x + g_1 x^2 + \dots + g_{m-2} x^{m-1} + g_{m-1} x^m \quad G(x)$$

and to get the remainder of the division by $\chi_L(x)$ we replace x^m by $p_{m-1}x^{m-1} + \dots + p_1x + p_0$ and add bits (xor).

$$\begin{array}{cccccc} & g_0 x & & g_1 x^2 & \dots & g_{m-2} x^{m-1} \\ \overline{g_{m-1} p_0} & g_{m-1} p_1 x & \overline{g_{m-1} p_2 x^2} & \dots & g_{m-1} p_{m-1} x^{m-1} \\ \hline g_{m-1} p_0 & (g_0 + g_{m-1} p_1)x & (g_1 + g_{m-1} p_2)x^2 & \dots & (g_{m-2} + g_{m-1} p_{m-1})x^{m-1} \end{array}$$

Here the linear map L :

$$L = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \\ p_0 & p_1 & p_2 & p_3 & \dots & p_{m-1} \end{bmatrix} \quad (\text{mod } 2)$$

the passage to a new state s' is given by the multiplication:

$$s \cdot L = s'$$

Exercise 2.4.4

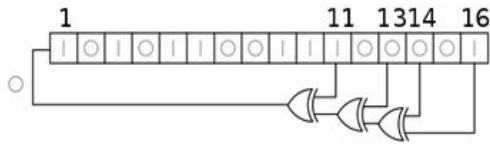
Show that the calculation of $\mathbf{s} \cdot \mathbf{L}$ of the Galois LFSR can be done using only bit operations. Namely, show that

$$\mathbf{s} \cdot \mathbf{L} = \begin{cases} \text{shiftright}(\mathbf{s}) & \text{if } g_{m-1} = 0 , \\ \text{shiftright}(\mathbf{s}) \oplus \mathbf{p} & \text{if } g_{m-1} = 1 \end{cases}$$

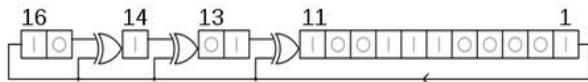
where $\mathbf{p} = [p_0 p_1 p_2 p_3 \cdots p_{m-1}]$.

Exercise 2.4.5

Write the matrices \mathbf{L} and their $\chi_{\mathbf{L}}(x)$ for the two examples of wikipedia. Namely:



and



Galois & Fibonacci

The output stream $\cdots s_{m-1} \cdots s_2 s_1 s_0$ of a Galois LFSR with $G(x) = \chi_{\mathbf{L}}(x)$ is the same as the output stream of a Fibonacci LFSR with $\chi_{\mathbf{L}}(x)$ and initial state $s_{m-1} \cdots s_2 s_1 s_0$.

Exercise 2.4.6

Show the claim in the above note. Hint: use that $s_k = (\mathbf{s} \cdot \mathbf{L}^k) \cdot \mathbf{e}$ where \mathbf{s} is the initial state of the register and \mathbf{e} is the column with all zeros with one in the last place.

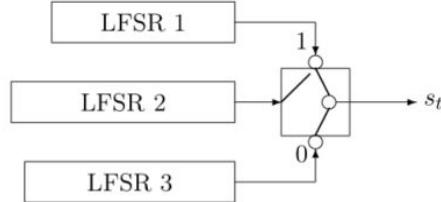
2.4.3 Several LFSRs and the Correlation Attack

Geffe Generator

This PRNG is composed of three LFSRs of distinct lengths combined by the boolean function

$$f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_3 .$$

It is worth noticing that this function corresponds to an IF: if $x_2 = 0$, then $f(x_1, x_2, x_3) = x_3$ and if $x_2 = 1$, $f(x_1, x_2, x_3) = x_1$.



Video by Anne Canteaut about Boolean Functions

https://en.wikipedia.org/wiki/Correlation_attack

NOTE 2.4.7

There are several PRNG used in crypto constructed by using shift registers e.g. Mersenne Twister, Trivium, E0, A5/1.

2.5 Stream Ciphers: Permutation

The oracle model of a Permutation.

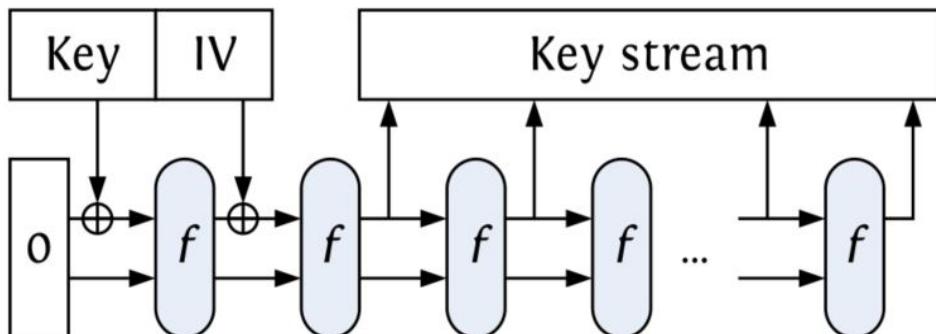
ORACLES. For convenience, a random oracle R is a map from $\{0,1\}^*$ to $\{0,1\}^\infty$ chosen by selecting each bit of $R(x)$ uniformly and independently, for every x . Of course no actual protocol uses an infinitely long output, this just saves us from having to say how long “sufficiently long” is. We denote by 2^∞ the set of all random oracles.

[BeRo95, pag. 6]

My function behaves as a random oracle, or it cannot be distinguished from a random permutation

2.5.1 Keccak-Sponge key stream

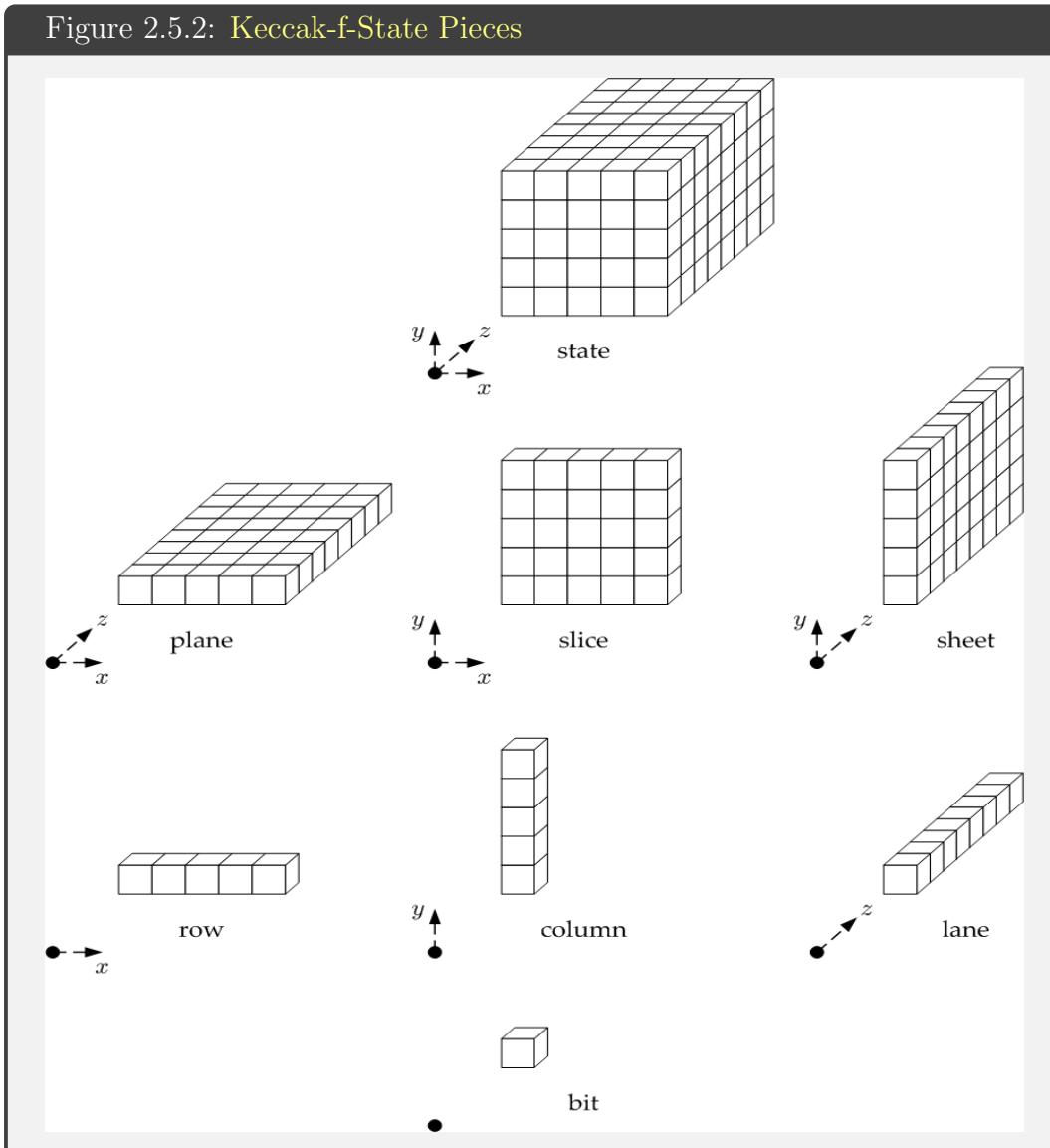
2.5.1 Keccak-sponge stream cipher



2.5.2 The Keccak-f permutations

There are 7 Keccak-f permutations of $\{0,1\}^b$, indicated Keccak-f[b] , where $b = 25 \times 2^l$ and l ranges from 0 to 6.

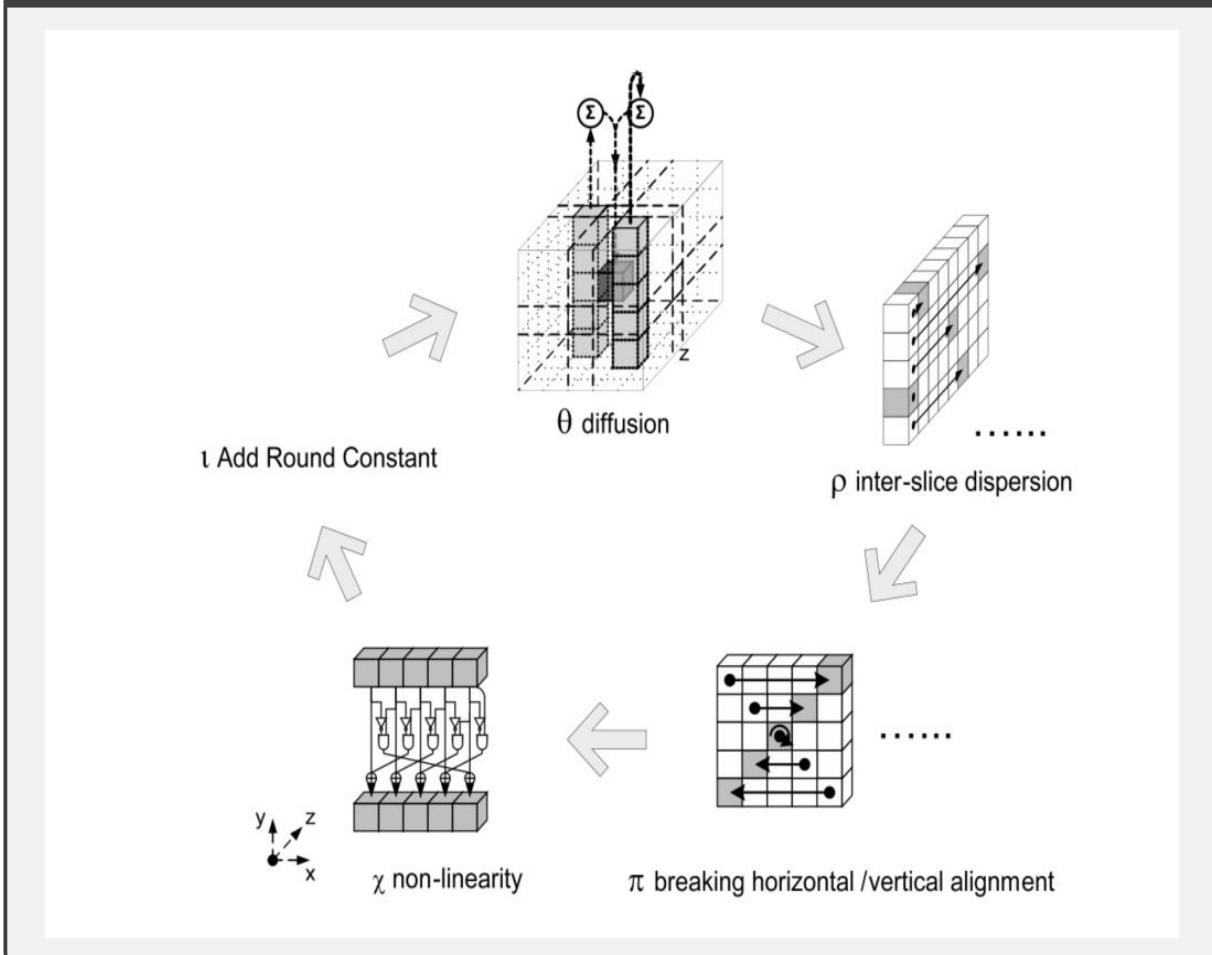
The elements of $\{0,1\}^b$ are regarded as a 3-dimensional array:



To compute Keccak-f[b](x) there are $12 + 2l$ rounds, e.g. Keccak-f[25] has 12 rounds, Keccak-f[1600] has 24 rounds.

A round consists of 5 invertible steps mappings:

Figure 2.5.3: Keccak-f Round



<https://keccak.team/index.html>

Figure 2.5.4: Keccak-f[1600](0)

2.6 Bibliography

Here is the list of books I used for the preparation of this note.

- [Aumasson18] Jean-Philippe Aumasson, *Serious Cryptography: A Practical Introduction to Modern Encryption*, No Starch Press, 2018.
- [KatLin15] Jonathan Katz; Yehuda Lindell, *Introduction to Modern Cryptography* Second Edition, Chapman & Hall/CRC, Taylor & Francis Group, 2015.
- [Paar10] Paar, Christof, Pelzl, Jan, *Understanding Cryptography, A Textbook for Students and Practitioners*, Springer-Verlag, 2010.
- [PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling , Brian P. Flannery *Numerical Recipes in C The Art of Scientific Computing*, Second Edition, CAMBRIDGE UNIVERSITY PRESS, 1992.

Here is the list of papers:

- [BeRo95] Bellare, Mihir and Rogaway, Phillip; *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, Proceedings of the First Annual Conference on Computer and Communications Security <https://cseweb.ucsd.edu/~mihir/papers/ro.pdf>
- [BDPA11] Bertoni,G.;Daemen,J.;Peeters,M. and Van Assche, G.: *The Keccak reference*, <https://keccak.team/files/Keccak-reference-3.0.pdf>
- [BDPA11b] Bertoni,G.;Daemen,J.;Peeters,M. and Van Assche, G.: *Cryptographic sponge functions*, <https://keccak.team/files/CSF-0.1.pdf>
- [DH76] Diffie, W.; Hellman, M. *New directions in cryptography*, (1976). IEEE Transactions on Information Theory. 22 (6): 644654.
- [GM82] Goldwasser, S. and Micali, S.; *Probabilistic Encryption & How To Play Mental Poker Keeping Secret All Partial Information*, Proc. 14th Symposium on Theory of Computing: 365–377. (1982)
- [Lehmer51] Lehmer, D.H.; *Mathematical methods in large-scale computing units.*, Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery, 1949, pp. 141–146. Harvard University Press, Cambridge, Mass., 1951.
- [Muk13] Pratyay Mukherjee; *An Overview of eSTREAM Ciphers*, Centre of Excellence in Cryptology, Indian Statistical Institute, Kolkata, India, (2013). <http://www.cs.au.dk/~pratyay/eSTREAM.pdf>

and some interesting links:

NIST sp800-22 r1a

https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch37.html

<https://nullprogram.com/blog/2019/04/30/>