

# Stream Ciphers

Dhananjoy Dey

Indian Institute of Information Technology, Lucknow  
[ddey@iiitl.ac.in](mailto:ddey@iiitl.ac.in)

February 10, 2021



# Disclaimers

All the pictures used in this presentation are taken from freely available websites.

If there is a reference on a slide all of the information on that slide is attributable to that source whether quotation marks are used or not.

Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement nor does it imply that the products mentioned are necessarily the best available for the purpose.



# Outline

- 1 Introduction
- 2 Statistical Tests
  - Five Basic Tests
- 3 LFSR
- 4 RC4
- 5 Salsa20/20



# Outline

- 1 Introduction
- 2 Statistical Tests
  - Five Basic Tests
- 3 LFSR
- 4 RC4
- 5 Salsa20/20



# One-Time Pad

## Encryption

e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111

Encryption: Plaintext  $\oplus$  Key = Ciphertext

# One-Time Pad

## Encryption

e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111

Encryption:  $\text{Plaintext} \oplus \text{Key} = \text{Ciphertext}$

	h	e	i	l	h	i	t	l	e	r
Plaintext:	001	000	010	100	001	010	111	100	000	101
Key:	111	101	110	101	111	100	000	101	110	000

# One-Time Pad

## Encryption

e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111

Encryption:  $\text{Plaintext} \oplus \text{Key} = \text{Ciphertext}$

	h	e	i	l	h	i	t	l	e	r
Plaintext:	001	000	010	100	001	010	111	100	000	101
Key:	111	101	110	101	111	100	000	101	110	000
Ciphertext:	110	101	100	001	110	110	111	001	110	101
	s	r	l	h	s	s	t	h	s	r

# One-Time Pad

## Decryption

e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111

Decryption: Ciphertext  $\oplus$  Key = Plaintext



# One-Time Pad

## Decryption

e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111

Decryption: Ciphertext  $\oplus$  Key = Plaintext

	s	r	l	h	s	s	t	h	s	r
Ciphertext:	110	101	100	001	110	110	111	001	110	101
Key:	111	101	110	101	111	100	000	101	110	000
Plaintext:	001	000	010	100	001	010	111	100	000	101
	h	e	i	l	h	i	t	l	e	r

# One-Time Pad

- **Provably secure** ...
  - Ciphertext provides no info about plaintext
  - All plaintexts are equally likely



# One-Time Pad

- **Provably secure** ...
  - Ciphertext provides no info about plaintext
  - All plaintexts are equally likely
- ... but, only when be used correctly
  - Key must be random, used only once
  - Key is known only to sender and receiver



# One-Time Pad

- **Provably secure** ...
  - Ciphertext provides no info about plaintext
  - All plaintexts are equally likely
- ... but, only when be used correctly
  - Key must be random, used only once
  - Key is known only to sender and receiver
- **Note:** Key is same size as message



# One-Time Pad

- **Provably secure** ...
  - Ciphertext provides no info about plaintext
  - All plaintexts are equally likely
- ... but, only when be used correctly
  - Key must be random, used only once
  - Key is known only to sender and receiver
- **Note:** Key is same size as message
- So, why not distribute message instead of pad?



# Real-World One-Time Pad

- Project **VENONA**
  - Encrypted spy messages from U.S. to Moscow in 30's, 40's & 50's
  - Nuclear espionage, etc.
  - Thousands of messages
- Spy carried one-time pad into U.S.
- Spy used key to encrypt secret messages
- Repeats within the "one-time" key made cryptanalysis possible



# VENONA Decrypt (1944)

*[C% Ruth] learned that her husband [v] was called up by the army but he was not sent to the front. He is a mechanical engineer and is now working at the ENORMOUS [ENORMOZ] [vi] plant in SANTA FE, New Mexico. [45 groups unrecoverable] detain VOLOK [vii] who is working in a plant on ENORMOUS. He is a FELLOWCOUNTRYMAN [ZEMLYaK] [viii]. Yesterday he learned that they had dismissed him from his work. His active work in progressive organizations in the past was cause of his dismissal. In the FELLOWCOUNTRYMAN line LIBERAL is in touch with CHESTER [ix]. They meet once a month for the payment of dues. CHESTER is interested in whether we are satisfied with the collaboration and whether there are not any misunderstandings. He does not inquire about specific items of work [KONKRETNAYa RABOTA]. In as much as CHESTER knows about the role of LIBERAL's group we beg consent to ask C. through LIBERAL about leads from among people who are working on ENOURMOUS and in other technical fields.*



# VENONA Decrypt (1944)

*[C% Ruth] learned that her husband [v] was called up by the army but he was not sent to the front. He is a mechanical engineer and is now working at the ENORMOUS [ENORMOZ] [vi] plant in SANTA FE, New Mexico. [45 groups unrecoverable] detain VOLOK [vii] who is working in a plant on ENORMOUS. He is a FELLOWCOUNTRYMAN [ZEMLYaK] [viii]. Yesterday he learned that they had dismissed him from his work. His active work in progressive organizations in the past was cause of his dismissal. In the FELLOWCOUNTRYMAN line LIBERAL is in touch with CHESTER [ix]. They meet once a month for the payment of dues. CHESTER is interested in whether we are satisfied with the collaboration and whether there are not any misunderstandings. He does not inquire about specific items of work [KONKRETNAYa RABOTA]. In as much as CHESTER knows about the role of LIBERAL's group we beg consent to ask C. through LIBERAL about leads from among people who are working on ENOURMOUS and in other technical fields.*

- “Ruth” == Ruth Greenglass, “Liberal” == Julius Rosenberg,  
“Enormous” == the atomic bomb





# Stream Cipher

based on one-time pad



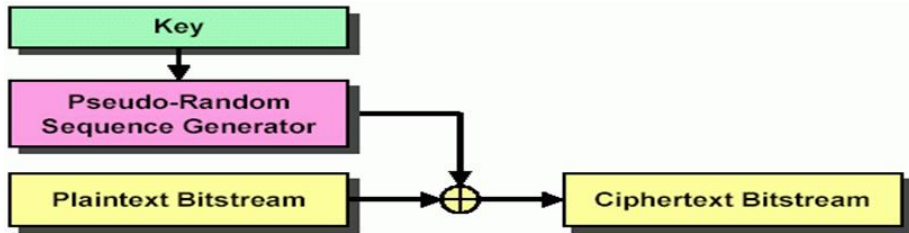
# Stream Cipher

## based on one-time pad

- Except that key is relatively short
- Key is stretched into a long keystream
- Keystream is used just like a one-time pad



# Stream Cipher



<b>Plaintext Stream</b>	1	1	1	1	1	1	1	1	0	0	0	0	0	0	...
<b>Pseudo-Random Stream</b>	1	0	0	1	1	0	1	0	1	1	0	1	0	0	...
<b>Ciphertext Stream</b>	0	1	1	0	0	1	0	1	1	1	0	1	0	0	...



# Stream Cipher

## Main Characteristics

- **Speed:** faster in hardware
- **Hardware implementation cost:** low
- **Error propagation:** limited or no error propagation
- **Synchronization requirement:** to allow for proper decryption, the sender and receiver must be synchronized



# Classification of Stream Ciphers

- **Synchronous Stream Ciphers:**

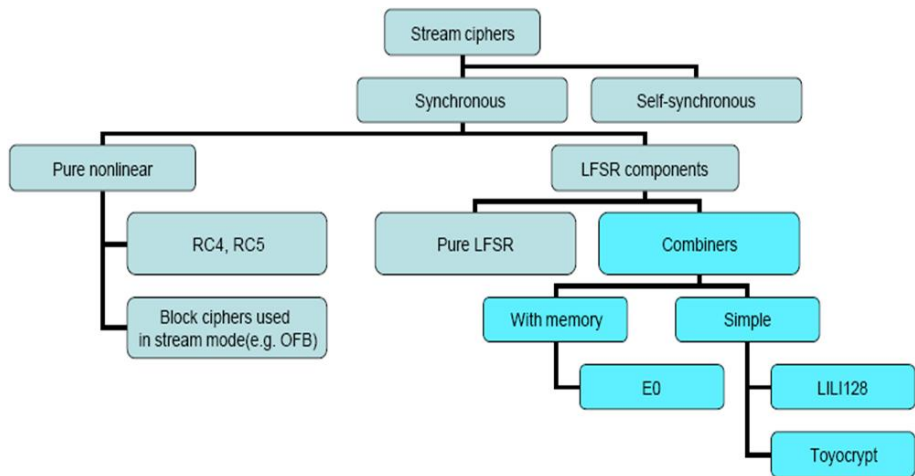
A synchronous stream cipher is one in which the keystream is generated independently of the plaintext message and of the ciphertext.

- **Self-Synchronous Stream Ciphers:**

A self-synchronizing or asynchronous stream cipher is one in which the keystream is generated as a function of the key and a fixed number of previous ciphertext bits.



# Classification of Stream Ciphers



# The eSTREAM Project

## Timeline

- 14-15 Oct 04 : workshop hosted by ECRYPT to discuss SASC (The State of the Art of Stream Ciphers)
- Nov 04 : call for Primitives
- 29 Apr 05 : the deadline of submission to ECRYPT.  
34 (32 + 2) primitives have been submitted to ECRYPT
- 13 Jun 05 : website is launched to promote the public evaluation of the primitives.
- 02-03 Feb 06 : workshop SASC 2006 hosted by ECRYPT
- Feb 06 : The end of the first evaluation phase of eSTREAM.



# The eSTREAM Project

## Timeline

- Jul 06 : The beginning of the second evaluation phase of eSTREAM.
- 31 Jan -  
01 Feb 07 : workshop SASC 2007 hosted by ECRYPT
- Apr 07 : the beginning of the third evaluation phase of eSTREAM
- Feb 08 : workshop SASC 2008
- May 08 : the final report of the eSTREAM
- Jan 12 : the final report of the eSTREAM Portfolio in 2012





# Submission Requirements

- Submissions had to be either fast in software or resource friendly in hardware

	key	IV	tag (optional)
<b>Profile 1</b>	128	64 or 128	32, 64, 96, or 128
<b>Profile 2</b>	80	32 or 64	32 or 64

- Designers required to give an IP statement.



## eSTREAM Portfolio

in 2008

Profile 1	Profile 2
HC-128 Rabbit Salsa20/12 Sosemanuk	F-FCSR-H v2 Grain v1 MICKEY v2 Trivium

in 2012

Profile 1	Profile 2
HC-128 Rabbit Salsa20/12 Sosemanuk	Grain v1 MICKEY 2.0 Trivium



# Recommended Stream Ciphers

Primitive	Recommendation	
	Legacy	Future
HC-128	✓	✓
Salsa20/20	✓	✓
ChaCha	✓	✓
SNOW 2.0	✓	✓
SNOW 3G	✓	✓
SOSEMANUK	✓	✓



# Recommended Stream Ciphers

Primitive	Recommendation	
	Legacy	Future
HC-128	✓	✓
Salsa20/20	✓	✓
ChaCha	✓	✓
SNOW 2.0	✓	✓
SNOW 3G	✓	✓
SOSEMANUK	✓	✓
Grain	✓	✗
Mickey 2.0	✓	✗
Trivium	✓	✗
Rabbit	✓	✗



# Recommended Stream Ciphers

Primitive	Recommendation	
	Legacy	Future
HC-128	✓	✓
Salsa20/20	✓	✓
ChaCha	✓	✓
SNOW 2.0	✓	✓
SNOW 3G	✓	✓
SOSEMANUK	✓	✓
Grain	✓	✗
Mickey 2.0	✓	✗
Trivium	✓	✗
Rabbit	✓	✗
A5/1	✗	✗
A5/2	✗	✗
E0	✗	✗
RC4	✗	✗



# Recommended Stream Ciphers

**Legacy** × Attack exists or security considered not sufficient.  
Mechanism should be replaced in Fielded products  
as a matter of urgency.



# Recommended Stream Ciphers

- Legacy ×** Attack exists or security considered not sufficient.  
Mechanism should be replaced in Fielded products as a matter of urgency.
- Legacy ✓** No known weaknesses at present.  
Better alternatives exist.  
Lack of security proof or limited key size.



# Recommended Stream Ciphers

- Legacy** × Attack exists or security considered not sufficient.  
Mechanism should be replaced in Fielded products as a matter of urgency.
- Legacy** ✓ No known weaknesses at present.  
Better alternatives exist.  
Lack of security proof or limited key size.
- Future** ✓ Mechanism is well studied (often with security proof).  
Expected to remain secure in 10-50 year lifetime.





# Stream Ciphers

- Once upon a time, not so very long ago, stream ciphers were the king of crypto
- Today, not as popular as block ciphers
- *RC4*
  - Based on a changing lookup table
  - Used many places (WEP ...)
- **RFC 7465**: “Prohibiting RC4 Cipher Suites” published in Feb 2015.



# RBG & PRBG

## Definition

A **random bit generator** is a device or algorithm which outputs a sequence of statistically independent and unbiased binary digits.



# RBG & PRBG

## Definition

A **random bit generator** is a device or algorithm which outputs a sequence of statistically independent and unbiased binary digits.

## Definition

A **pseudo-random bit generator (PRBG)** is a deterministic algorithm which, given a truly random binary sequence of length  $k$ , outputs a binary sequence of length  $\ell$  much larger than  $k$  which “appears” to be random. The input to the PRBG is called **seed**, while the output of the PRBG is called a **pseudo-random bit sequence**.



# PRBG & CSPRBG

## Definition

We say that a **PRBG passes all poly-time statistical tests** if no poly-time algorithm can correctly distinguish between an output sequence of the generator and a TRBG of the same length with prob significantly  $> \frac{1}{2}$ .



# PRBG & CSPRBG

## Definition

We say that a **PRBG passes all poly-time statistical tests** if no poly-time algorithm can correctly distinguish between an output sequence of the generator and a TRBG of the same length with prob significantly  $> \frac{1}{2}$ .

## Definition

We say that a **PRBG passes the next-bit test** if there is no poly-time algo which, on input of the first  $\ell$  bits of an output sequence  $s$ , can predict the  $(\ell + 1)^{\text{th}}$  bit of  $s$  with prob significantly  $> \frac{1}{2}$ .



# PRBG & CSPRBG

## Definition

We say that a **PRBG passes all poly-time statistical tests** if no poly-time algorithm can correctly distinguish between an output sequence of the generator and a TRBG of the same length with prob significantly  $> \frac{1}{2}$ .

## Definition

We say that a **PRBG passes the next-bit test** if there is no poly-time algo which, on input of the first  $\ell$  bits of an output sequence  $s$ , can predict the  $(\ell + 1)^{\text{th}}$  bit of  $s$  with prob significantly  $> \frac{1}{2}$ .

## Definition

A PRBG that passes the next-bit test is called a **cryptographically secure PRBG**.



# Linear Congruential Generator

- Designed by D. H. Lehmer in 1949
- $x_n \equiv a.x_{n-1} + b \pmod{m}$ , where  $n \geq 1$ .
- Output depends on the **initial seed**  $x_0$  and  $a, b$ , &  $m$ .



# Linear Congruential Generator

- Designed by D. H. Lehmer in 1949
- $x_n \equiv a.x_{n-1} + b \pmod{m}$ , where  $n \geq 1$ .
- Output depends on the **initial seed**  $x_0$  and  $a, b$ , &  $m$ .

## Theorem

If  $b \neq 0$ , LCG generates a sequence of length  $m$  iff

- (i)  $\gcd(b, m) = 1$ ,
- (ii) if  $p \mid m$ , then  $p \mid (a - 1)$  for all prime factor  $p$  of  $m$ ,
- (iii) if  $4 \mid m$ , then  $4 \mid (a - 1)$ .





# Linear Congruential Generator

- Designed by D. H. Lehmer in 1949
- $x_n \equiv a.x_{n-1} + b \pmod{m}$ , where  $n \geq 1$ .
- Output depends on the **initial seed**  $x_0$  and  $a, b$ , &  $m$ .

## Theorem

If  $b \neq 0$ , LCG generates a sequence of length  $m$  iff

- (i)  $\gcd(b, m) = 1$ ,
- (ii) if  $p \mid m$ , then  $p \mid (a - 1)$  for all prime factor  $p$  of  $m$ ,
- (iii) if  $4 \mid m$ , then  $4 \mid (a - 1)$ .

LCGs are not very useful for cryptographic purpose.



# RSA CSPRNG

- Choose 2 large primes  $p$  &  $q$ .
- Set  $n = p \cdot q$
- Choose a random  $e$  s/t  $0 < e < \phi(n)$  s/t  $\gcd(e, \phi(n)) = 1$ .
- Choose a random seed  $x_0$  s/t  $1 \leq x_0 \leq n - 1$

$$x_i \equiv x_{i-1}^e \pmod{n}.$$

- Let  $b_i$  be the least significant bit of  $x_i$ .
- $\ell$  random bits are  $b_1, b_2, \dots, b_\ell$ .



# BBS (Blum-Blum-Shub) CSPRBG

- Generate 2 large primes  $p$  &  $q$  s/t both  $\equiv 3 \pmod{4}$
- Set  $n = p \cdot q$
- Select a random integer  $x$  s/t  $\gcd(x, n) = 1$
- Set initial seed  $x_0 \equiv x^2 \pmod{n}$

$$x_i \equiv x_{i-1}^2 \pmod{n}$$

- Let  $b_i$  be the least significant bit of  $x_i$ .
- $\ell$  random bits are  $b_1, b_2, \dots, b_\ell$ .



# Outline

- 1 Introduction
- 2 Statistical Tests**
  - Five Basic Tests
- 3 LFSR
- 4 RC4
- 5 Salsa20/20



# Golomb's Postulates

- Let  $s = s_0, s_1, s_2, \dots$  be an infinite sequence. The subsequence consisting of the first  $n$  terms of  $s$  is denoted by  $s^n = s_0, s_1, \dots, s_{n-1}$ .



# Golomb's Postulates

- Let  $s = s_0, s_1, s_2, \dots$  be an infinite sequence. The subsequence consisting of the first  $n$  terms of  $s$  is denoted by  $s^n = s_0, s_1, \dots, s_{n-1}$ .
- A **run** of  $s$  is a subsequence of  $s$  consisting of consecutive 0's or consecutive 1's which is neither preceded nor succeeded by the same symbol. A run of 0's is called a **gap**, while a run of 1's is called a **block**.



# Golomb's Postulates

- Let  $s = s_0, s_1, s_2, \dots$  be an infinite sequence. The subsequence consisting of the first  $n$  terms of  $s$  is denoted by  $s^n = s_0, s_1, \dots, s_{n-1}$ .
- A **run** of  $s$  is a subsequence of  $s$  consisting of consecutive 0's or consecutive 1's which is neither preceded nor succeeded by the same symbol. A run of 0's is called a **gap**, while a run of 1's is called a **block**.

## Definition

Let  $s = s_0, s_1, s_2, \dots$  be a periodic sequence of period  $N$ . The *autocorrelation function* of  $s$  is the *integer-valued function*  $C(t)$  defined as

$$C(t) = \frac{1}{N} \sum_{i=0}^{N-1} (2 \cdot s_i - 1) \cdot (2 \cdot s_{i+t} - 1), \quad \text{for } 0 \leq t \leq N-1.$$



# Golomb's Postulates

- Let  $s = s_0, s_1, s_2, \dots$  be an infinite sequence. The subsequence consisting of the first  $n$  terms of  $s$  is denoted by  $s^n = s_0, s_1, \dots, s_{n-1}$ .
- A **run** of  $s$  is a subsequence of  $s$  consisting of consecutive 0's or consecutive 1's which is neither preceded nor succeeded by the same symbol. A run of 0's is called a **gap**, while a run of 1's is called a **block**.

## Definition

Let  $s = s_0, s_1, s_2, \dots$  be a periodic sequence of period  $N$ . The *autocorrelation function* of  $s$  is the *integer-valued function*  $C(t)$  defined as

$$C(t) = \frac{1}{N} \sum_{i=0}^{N-1} (2s_i - 1)(2s_{i+t} - 1), \quad \text{for } 0 \leq t \leq N-1.$$

$C(t)$  measures the amount of similarity between the sequence  $s$  and a shift of  $s$  by  $t$  positions. If  $s$  is a random periodic sequence of period  $N$ , then  $|N.C(t)|$  can be expected to be quite small for all values of  $t$ ,  $0 < t < N$ .





# Golomb's Postulates

Let  $s$  be a periodic sequence of period  $N$ . **Golomb's randomness postulates** are the following:

- ① In the cycle  $s^N$  of  $s$ , the number of 1's differs from the number of 0's by at most 1.



# Golomb's Postulates

Let  $s$  be a periodic sequence of period  $N$ . **Golomb's randomness postulates** are the following:

- (i) In the cycle  $s^N$  of  $s$ , the number of 1's differs from the number of 0's by at most 1.
- (ii) In the cycle  $s^N$ , at least half the runs have length 1, at least one-fourth have length 2, at least one-eighth have length 3, etc., as long as the number of runs so indicated exceeds 1. Moreover, for each of these lengths, there are (almost) equally many gaps and blocks.



# Golomb's Postulates

Let  $s$  be a periodic sequence of period  $N$ . **Golomb's randomness postulates** are the following:

- (i) In the cycle  $s^N$  of  $s$ , the number of 1's differs from the number of 0's by at most 1.
- (ii) In the cycle  $s^N$ , at least half the runs have length 1, at least one-fourth have length 2, at least one-eighth have length 3, etc., as long as the number of runs so indicated exceeds 1. Moreover, for each of these lengths, there are (almost) equally many gaps and blocks.
- (iii) The autocorrelation function  $C(t)$  is two-valued. That is for some integer  $K$ ,

$$C(t) = \frac{1}{N} \sum_{i=0}^{N-1} (2s_i - 1) \cdot (2s_{i+t} - 1) = \begin{cases} N, & \text{if } t = 0, \\ K, & \text{if } 1 \leq t \leq N - 1. \end{cases}$$



# Golomb's Postulates

Let  $s$  be a periodic sequence of period  $N$ . **Golomb's randomness postulates** are the following:

- (i) In the cycle  $s^N$  of  $s$ , the number of 1's differs from the number of 0's by at most 1.
- (ii) In the cycle  $s^N$ , at least half the runs have length 1, at least one-fourth have length 2, at least one-eighth have length 3, etc., as long as the number of runs so indicated exceeds 1. Moreover, for each of these lengths, there are (almost) equally many gaps and blocks.
- (iii) The autocorrelation function  $C(t)$  is two-valued. That is for some integer  $K$ ,

$$C(t) = \frac{1}{N} \sum_{i=0}^{N-1} (2s_i - 1) \cdot (2s_{i+t} - 1) = \begin{cases} N, & \text{if } t = 0, \\ K, & \text{if } 1 \leq t \leq N - 1. \end{cases}$$

A binary sequence which satisfies Golomb's randomness postulates is called a **pseudo-noise sequence or a pn-sequence**.



## Frequency Test (Monobit Test)

- The purpose of this test is to determine whether the number of 0's and 1's in  $s$  are approximately the same, as would be expected for a random sequence.
- Let  $s = s_0, s_1, s_2, \dots, s_{n-1}$  be a binary sequence of length  $n$ .
- Let  $n_0, n_1$  denote the number of 0's and 1's in  $s$ , respectively.
- The statistic used is

$$X_1 = \frac{(n_0 - n_1)^2}{n}$$

which approximately follows a  $\chi^2$  distribution with 1 degree of freedom if  $n \geq 10$ .



## Serial Test (2-bit Test)

- The purpose of this test is to determine whether the number of occurrences of 00, 01, 10, and 11 as subsequences of  $s$  are approximately the same, as would be expected for a random sequence.
- Let  $n_0, n_1$  denote the number of 0's and 1's in  $s$ , respectively, and let  $n_{00}, n_{01}, n_{10}, n_{11}$  denote the number of occurrences of 00, 01, 10, 11 in  $s$ , respectively<sup>1</sup>.
- The statistic used is

$$X_2 = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1$$

which approximately follows a  $\chi^2$  distribution with 2 degrees of freedom if  $n \geq 21$ .



<sup>1</sup> $n_{00} + n_{01} + n_{10} + n_{11} = (n-1)$  since the subsequences are allowed to overlap. ≡ ↻ 🔍

# Poker test

- Let  $m$  be a positive integer such that  $\lfloor \frac{n}{m} \rfloor \geq 5.2^m$ , and let  $k = \lfloor \frac{n}{m} \rfloor$ .
- Divide the sequence  $s$  into  $k$  non-overlapping parts each of length  $m$
- Let  $n_i$  be the number of occurrences of the  $i^{\text{th}}$  type of sequence of length  $m$ ,  $1 \leq i \leq 2^m$ .
- The poker test<sup>2</sup> determines whether the sequences of length  $m$  each appear approximately the same number of times in  $s$ , as would be expected for a random sequence.
- The statistic used is

$$X_3 = \frac{2^m}{k} \left( \sum_{i=1}^{2^m} n_i^2 \right) - k$$

which approximately follows a  $\chi^2$  distribution with  $2^m - 1$  degrees of freedom.

---

<sup>2</sup>Note that the poker test is a generalization of the frequency test: setting  $m = 1$  in the poker test yields the frequency test.



# Runs test

- The purpose of the runs test is to determine whether the number of runs of various lengths in the sequence  $s$  is as expected for a random sequence.
- The expected number of gaps (or blocks) of length  $i$  in a random sequence of length  $n$  is  $e_i = (n - i + 3)/2^{i+2}$ .
- Let  $k$  be equal to the largest integer  $i$  for which  $e_i \geq 5$ .
- Let  $B_i, G_i$  be the number of blocks and gaps, respectively, of length  $i$  in  $s$  for each  $i, 1 \leq i \leq k$ .
- The statistic used is

$$X_4 = \sum_i^k \frac{(B_i - e_i)^2}{e_i} + \sum_i^k \frac{(G_i - e_i)^2}{e_i}$$

which approximately follows a  $\chi^2$  distribution with  $2k - 2$  degrees of freedom.





# Autocorrelation test

- The purpose of this test is to check for correlations between the sequence  $s$  and (non-cyclic) shifted versions of it.
- Let  $d$  be a fixed integer,  $1 \leq d \leq \lfloor n/2 \rfloor$ .
- The number of bits in  $s$  not equal to their  $d$ -shifts is  $A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$ .
- The statistic used is

$$X_5 = \frac{2(A(d) - \frac{n-d}{2})}{\sqrt{n-d}}$$

which approximately follows an  $N(0, 1)$  distribution if  $n - d \geq 10$ . Since small values of  $A(d)$  are as unexpected as large values of  $A(d)$ , a two-sided test should be used.



# Outline

- 1 Introduction
- 2 Statistical Tests
  - Five Basic Tests
- 3 **LFSR**
- 4 RC4
- 5 Salsa20/20



# Linear Feedback Shift Registers (LFSR)

- A standard way of producing a binary stream of data is to use a feedback shift register.
- These are small circuits containing a number of memory cells, each of which holds one bit of information.
- The set of such cells **forms a register**.
- In each cycle a certain predefined set of cells are 'tapped' and their value is passed through a function, called **the feedback function**.
- The register is then shifted down by one bit, with the output bit of the feedback shift register being the bit that is shifted out of the register.
- The combination of the tapped bits is then fed into the empty cell at the top of the register.



# Linear Feedback Shift Registers (LFSR)

## Definition

A LFSR of degree  $L$  (or length  $L$ ) is defined by feedback coefficients  $c_1, \dots, c_L \in GF(2)$ .

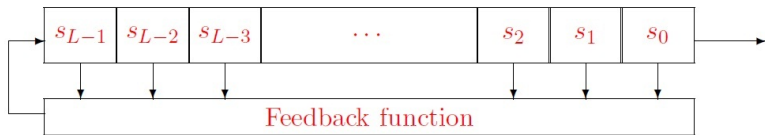
The initial state is an  $L$ -bit word  $S = (s_{L-1}, \dots, s_1, s_0)$  and new bits are generated by the recursion

$$s_j = (c_1 \cdot s_{j-1} \oplus c_2 s_{j-2} \oplus \dots \oplus c_L \cdot s_{j-L}) \pmod 2, \text{ for } j \geq L$$

At each iteration step (clock tick), the state  $S$  is updated from  $(s_{j-1}, \dots, s_{j-L})$  to  $(s_j, s_{j-1}, \dots, s_{j-L+1})$ , i.e., by shifting the register to the right. The rightmost bit  $s_{j-L}$  is output.

**The output of an LFSR is called a *linear recurring sequence*.**

# Linear Feedback Shift Registers (LFSR)



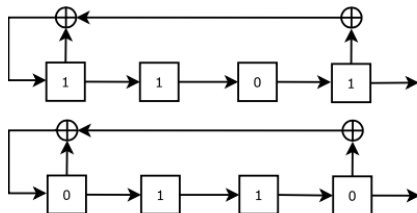
- Let the length of the register be  $L$ .
- One defines a set of bits  $(c_1, \dots, c_L)$  where  $c_i = 1$  if that cell is tapped and  $c_i = 0$  otherwise.
- The initial internal state of the register is given by the bit sequence  $(s_{L-1}, \dots, s_1, s_0)$ .
- The output sequence is then defined to be  $s_0, s_1, s_2, \dots, s_{L-1}, s_L, s_{L+1}, \dots$  where for  $j \geq L$  we have

$$s_j = c_1 \cdot s_{j-1} \oplus c_2 s_{j-2} \oplus \dots \oplus c_L \cdot s_{j-L}.$$



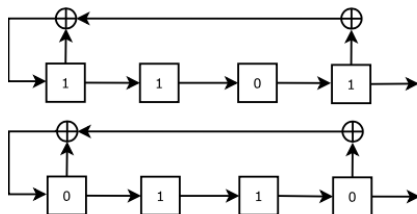
# Linear Feedback Shift Registers (LFSR)

## Example



# Linear Feedback Shift Registers (LFSR)

## Example

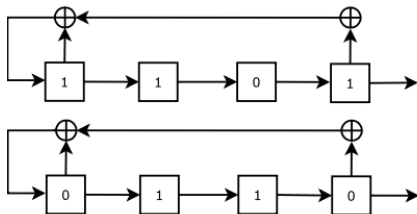


- Connection polynomial :



# Linear Feedback Shift Registers (LFSR)

## Example



- Connection polynomial :  $c(x) = x^4 + x + 1$
- Initial state is  $(1, 1, 0, 1)$





# Linear Feedback Shift Registers (LFSR)

## Example

1101	→	1
0110	→	0
0011	→	1
1001	→	1
0100	→	0
0010	→	0
0001	→	1
1000	→	0
1100	→	0
1110	→	0
1111	→	1
0111	→	1
1011	→	1
0101	→	1
1010	→	1

# Linear Feedback Shift Registers (LFSR)

## Example

1101 → 1  
0110 → 0  
0011 → 1  
1001 → 1  
0100 → 0  
0010 → 0  
0001 → 1  
1000 → 0  
1100 → 0  
1110 → 0  
1111 → 1  
0111 → 1  
1011 → 1  
0101 → 1  
1010 → 1  
1101

# Linear Feedback Shift Registers (LFSR)

## Definition

Let  $s_0, s_1, \dots$  be a linear recurring sequence. The (least) period of the sequence is the smallest integer  $N \geq 1$  s/t

$$s_{j+N} = s_j$$

for all sufficiently large values of  $j$ .



# Linear Feedback Shift Registers (LFSR)

## Definition

Let  $s_0, s_1, \dots$  be a linear recurring sequence. The (least) period of the sequence is the smallest integer  $N \geq 1$  s/t

$$s_{j+N} = s_j$$

for all sufficiently large values of  $j$ .

## Proposition

The period of a sequence generated by an LFSR of degree  $n$  is at most  $2^n - 1$ .



# Outline

- 1 Introduction
- 2 Statistical Tests
  - Five Basic Tests
- 3 LFSR
- 4 RC4
- 5 Salsa20/20



# RC4

- A self-modifying lookup table (or Synchronous stream cipher)
- Table always contains a permutation of the byte values  $0, 1, \dots, 255$
- Initialize the permutation using key
- At each step, RC4 does the following



# RC4

- A self-modifying lookup table (or Synchronous stream cipher)
- Table always contains a permutation of the byte values  $0, 1, \dots, 255$
- Initialize the permutation using key
- At each step, RC4 does the following
  - Swaps elements in current lookup table
  - Selects a keystream byte from table
- Each step of RC4 produces a byte



# RC4

- A self-modifying lookup table (or Synchronous stream cipher)
- Table always contains a permutation of the byte values  $0, 1, \dots, 255$
- Initialize the permutation using key
- At each step, RC4 does the following
  - Swaps elements in current lookup table
  - Selects a keystream byte from table
- Each step of RC4 produces a byte
  - Efficient in software
- Each step of A5/1 produces only a bit





# RC4

- A self-modifying lookup table (or Synchronous stream cipher)
- Table always contains a permutation of the byte values  $0, 1, \dots, 255$
- Initialize the permutation using key
- At each step, RC4 does the following
  - Swaps elements in current lookup table
  - Selects a keystream byte from table
- Each step of RC4 produces a byte
  - Efficient in software
- Each step of A5/1 produces only a bit
  - Efficient in hardware



# RC4 Key Scheduling Algorithm (KSA)

**Input:** Key array  $K[0], K[1], \dots, K[n - 1]$  of  $n$  bytes,  $1 \leq n \leq 255$

**Output:** State array  $S[0], S[1], \dots, S[255]$

- 1: **for**  $i = 0$  to 255 **do**
- 2:      $S[i] = i$
- 3: **end for**
- 4:  $j = 0$
- 5: **for**  $i = 0$  to 255 **do**
- 6:      $j = (j + S[i] + K[i \bmod n]) \bmod 256$
- 7:     Swap the values of  $S[i]$  and  $S[j]$
- 8: **end for**



# RC4 Pseudorandom Generation Algorithm (PRGA)

- For each keystream byte, swap elements in table and select byte

**Input:** State array  $S[0], S[1], \dots, S[255]$

**Output:** Output bytes  $B$

- 1:  $i = 0$
- 2:  $j = 0$
- 3: **while** Keystream is generated **do**
- 4:    $i = i + 1$
- 5:    $j = (j + S[i]) \bmod 256$
- 6:   Swap the values of  $S[i]$  and  $S[j]$
- 7:    $B = S[(S[i] + S[j]) \bmod 256]$
- 8:   **Output**  $B$
- 9: **end while**

- Use keystream bytes like a one-time pad
- Note:** first 256 bytes should be discarded
  - Otherwise, related key attack exists



# Outline

- 1 Introduction
- 2 Statistical Tests
  - Five Basic Tests
- 3 LFSR
- 4 RC4
- 5 Salsa20/20



# Salsa20/20

- Salsa20 is based on three simple operations:
  - modular addition of 32-bit words  $a$  and  $b \bmod 2^{32}$ , denoted by  $a \boxplus b$ ,
  - XOR-sum of 32-bit words  $a$  and  $b$ , denoted by  $a \oplus b$ ,
  - circular left shift of a 32-bit word  $a$  by  $t$  positions, denoted by  $a \lll t$ .

---

<sup>3</sup>Strings are interpreted in little-endian notation, i.e., the least significant bit of each word is stored first.



# Salsa20/20

- Salsa20 is based on three simple operations:
  - modular addition of 32-bit words  $a$  and  $b \pmod{2^{32}}$ , denoted by  $a \boxplus b$ ,
  - XOR-sum of 32-bit words  $a$  and  $b$ , denoted by  $a \oplus b$ ,
  - circular left shift of a 32-bit word  $a$  by  $t$  positions, denoted by  $a \lll t$ .
- The Salsa20/20 cipher takes a **256-bit key**, a **64-bit nonce** and a **64-bit counter**.
- The state array  $S$  of Salsa20 is a  $4 \times 4$  matrix of sixteen 32-bit words<sup>3</sup>

---

<sup>3</sup>Strings are interpreted in little-endian notation, i.e., the least significant bit of each word is stored first.



# Salsa20/20

The state array  $S$ :

$$S = \begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 & y_7 \\ y_8 & y_9 & y_{10} & y_{11} \\ y_{12} & y_{13} & y_{14} & y_{15} \end{pmatrix}$$



# Salsa20/20

The state array  $S$ :

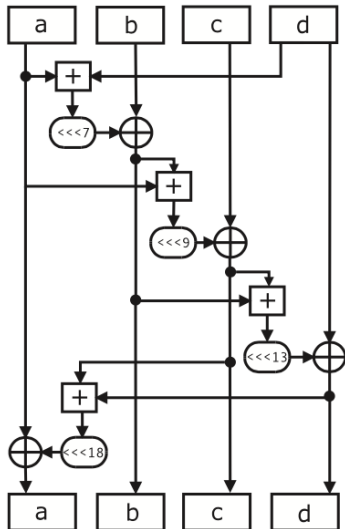
$$S = \begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 & y_7 \\ y_8 & y_9 & y_{10} & y_{11} \\ y_{12} & y_{13} & y_{14} & y_{15} \end{pmatrix}$$

- Salsa20 is based on **quarter-rounds**, **row-rounds** and **column-rounds**.
- The **quarter-rounds** operate on four words, the **row-rounds** transform the four rows and the **column-rounds** transform the four columns of the state matrix.





## Salsa20/20: Quarter-round



# Salsa20/20: Row-round

$$\text{row-round}(S) = \begin{pmatrix} z_0 & z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 & z_7 \\ z_8 & z_9 & z_{10} & z_{11} \\ z_{12} & z_{13} & z_{14} & z_{15} \end{pmatrix},$$

where

$$(z_0, z_1, z_2, z_3) = \text{quarter-round}(y_0, y_1, y_2, y_3),$$

$$(z_5, z_6, z_7, z_4) = \text{quarter-round}(y_5, y_6, y_7, y_4),$$

$$(z_{10}, z_{11}, z_8, z_9) = \text{quarter-round}(y_{10}, y_{11}, y_8, y_9),$$

$$(z_{15}, z_{12}, z_{13}, z_{14}) = \text{quarter-round}(y_{15}, y_{12}, y_{13}, y_{14}).$$



# Salsa20/20: Column-round

- The *column-round* function is the transpose of the row-round function: the words in the columns are permuted, the quarter-round map is applied to each of the columns and the permutation is reversed.
- Let  $S$  be a state matrix as above; then

$$\text{column-round}(S) = (\text{row-round}(S^T))^T.$$



# Salsa20/20: Double-round

- A *double-round* is the composition of a column-round and a row-round.
- Let  $S$  be a state matrix as above; then

$$\text{double-round}(S) = \text{row-round}(\text{column-round}(S)).$$



# Salsa20/20: Double-round

- A *double-round* is the composition of a column-round and a row-round.
- Let  $S$  be a state matrix as above; then

$$\text{double-round}(S) = \text{row-round}(\text{column-round}(S)).$$

Salsa20 runs 10 successive double-rounds, i.e., 20 quarter-rounds, in order to generate 64 bytes of output.

The *initial state* depends on the *key*, a *nonce* and a *counter*.



# Salsa20/20

- The Salsa20/20 stream cipher takes a 256-bit key  $k = (k_1, \dots, k_8)$  and a unique 64-bit message number  $n = (n_1, n_2)$  (nonce) as input.
- A 64-bit block counter  $b = (b_1, b_2)$  is initially set to zero.
- The initialization algorithm copies  $k, n, b$  and the four 32-bit constants

$$y_0 = 61707865, y_5 = 3320646E, y_{10} = 79622D32, \& y_{15} = 6B206574$$

into the sixteen 32-bit words of the Salsa20 state matrix:



# Salsa20/20

The state array  $S$ :

$$S = \begin{pmatrix} y_0 & k_1 & k_2 & k_3 \\ k_4 & y_5 & n_1 & n_2 \\ b_1 & b_2 & y_{10} & k_5 \\ k_6 & k_7 & k_8 & y_{15} \end{pmatrix}$$

- The key stream generator computes the output state by *ten double-round* iterations and a final addition  $\text{mod } 2^{32}$  of the initial state matrix:

$$Salsa20_k(n, b) = S + \text{double-round}^{10}(S).$$



# Salsa20/20

The state array  $S$ :

$$S = \begin{pmatrix} y_0 & k_1 & k_2 & k_3 \\ k_4 & y_5 & n_1 & n_2 \\ b_1 & b_2 & y_{10} & k_5 \\ k_6 & k_7 & k_8 & y_{15} \end{pmatrix}$$

- The key stream generator computes the output state by *ten double-round* iterations and a final addition  $\text{mod } 2^{32}$  of the initial state matrix:

$$Salsa20_k(n, b) = S + \text{double-round}^{10}(S).$$

ChaCha20 is a modification of Salsa20





# Stream Ciphers

- Stream ciphers were popular in the past



# Stream Ciphers

- Stream ciphers were popular in the past
  - Efficient in hardware
  - Speed was needed to keep up with voice, etc.
  - Today, processors are fast, so software-based crypto is usually more than fast enough



# Stream Ciphers

- Stream ciphers were popular in the past
  - Efficient in hardware
  - Speed was needed to keep up with voice, etc.
  - Today, processors are fast, so software-based crypto is usually more than fast enough
- Future of stream ciphers?







# Stream Ciphers

- Stream ciphers were popular in the past
  - Efficient in hardware
  - Speed was needed to keep up with voice, etc.
  - Today, processors are fast, so software-based crypto is usually more than fast enough
- Future of stream ciphers?
  - Shamir declared “the death of stream ciphers”
  - May be greatly exaggerated . . .



# References

-  S. W. Golomb,  
*Shift Register Sequences*, Aegean Park Press, 1982.
-  Andreas Klein,  
*Stream Ciphers*, Springer, 2013.
-  R. A. Rueppel,  
*Analysis and Design of Stream Ciphers*, Springer, 1986.
-  Mark Stamp  
*Information Security - Principles and Practice*, John Wiley & Sons, Inc., 2011.



# The End

**Thank you very much for your attention!**

