Stream Ciphers

Dhananjoy Dey

Indian Institute of Information Technology, Lucknow
ddey@iiitl.ac.in

December 23, 2022

Disclaimers

1

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

2

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.

3

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

Outline

Block vs. Stream Cipher

- **Block Cipher**

[1] Adding a small amount of memory to a block cipher results in a stream cipher with large blocks.

Block vs. Stream Cipher

- **Block Cipher**
  - It processes plaintext in relatively large blocks (e.g., $n \geq 64$ bits).
  - The same function is used to encrypt successive blocks; thus (pure) block ciphers are memoryless[1].

[1] Adding a small amount of memory to a block cipher results in a stream cipher with large blocks.

Block vs. Stream Cipher

- **Block Cipher**
  - It processes plaintext in relatively large blocks (e.g., $n \geq 64$ bits).
  - The same function is used to encrypt successive blocks; thus (pure) block ciphers are memoryless[1].
- **Stream Ciphers**
  - It processes plaintext in blocks as small as a single bit.
  - The encryption function may vary as plaintext is processed.
  - Thus it is said to have memory.
  - It is also called state ciphers since encryption depends on not only the *key* and *plaintext*, but also on the *current state*.

[1] Adding a small amount of memory to a block cipher results in a stream cipher with large blocks.

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: Plaintext $\oplus$ Key = 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: Plaintext $\oplus$ Key = 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** $\cdots$
  - Ciphertext provides no info about plaintext
  - All plaintexts are equally likely
- $\cdots$ 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?

Stream Cipher

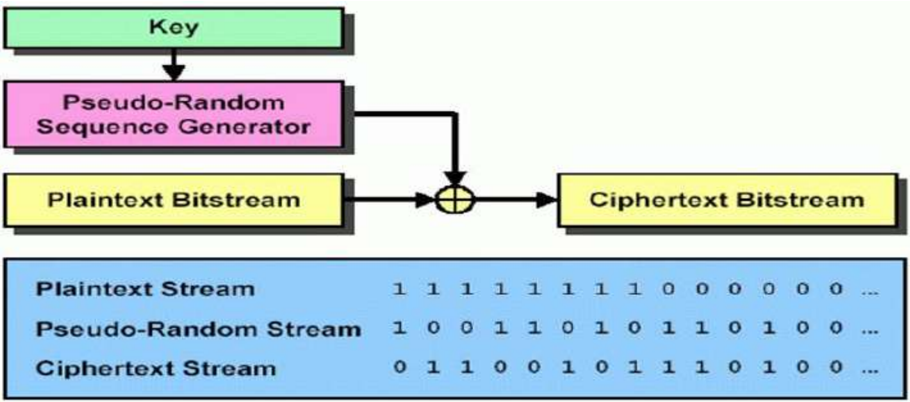
based on one-time pad

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



Stream Cipher

Main Characteristics

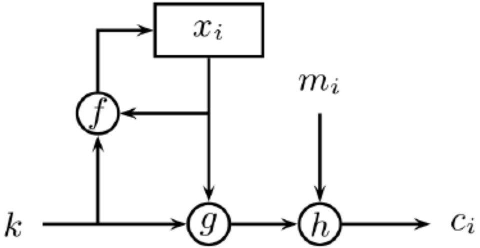
- **Speed:** faster in hardware
- **Hardware implementation cost:** low
- **Error propagation:** limited or no error propagation
- **Synchronization requirement:** to allow for proper decryption, the sender and receiver must be synchronized

Difference Between Stream Cipher and Pseudorandom Generator

- The output length is not fixed and the keystream is computed recursively using an internal state and the key.
- The initial state is derived from a key and an initialization vector.
- Stream cipher is an encryption scheme based on a keystream generator.
- Encryption is defined by XORing the plaintext with the keystream

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.

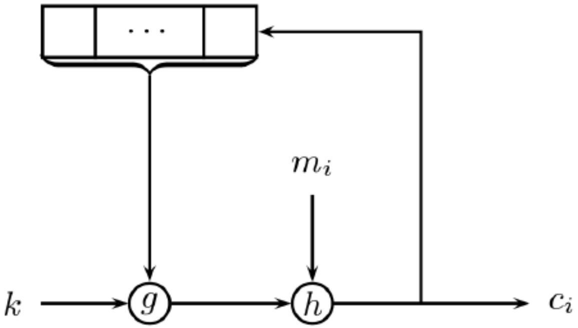


where f is the feedback function of the cipher, g is the key stream extractor and h combines the key stream with the message stream. x_0 is called the initial state and may depend on the key.

Classification of Stream Ciphers

- **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.



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

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.

https://www.ecrypt.eu.org/stream/

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)
Profile 1 (SW)	128	64 or 128	32, 64, 96, or 128
Profile 2 (HW)	80	32 or 64	32 or 64

- Designers required to give an IP statement.

eSTREAM Portfolio

in 2008

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

in 2012

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

Recommended Stream Ciphers (ENISA - Nov 2014)

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

Recommended Stream Ciphers (ENISA - Nov 2014)

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

Recommended Stream Ciphers (ENISA - Nov 2014)

Primitive	Recommendation: Legacy	Recommendation: 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 (ENISA - Nov 2014)

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

Recommended Stream Ciphers (ENISA - Nov 2014)

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 (ENISA - Nov 2014)

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.

https://www.enisa.europa.eu/publications/algorithms-key-size-and-parameters-report-2014

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 $\cdots$)
- **RFC 7465:** “Prohibiting RC4 Cipher Suites" published in Feb 2015.

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 $\cdots$)
- **RFC 7465:** “Prohibiting RC4 Cipher Suites" published in Feb 2015.
- **ChaCha20** is a modern stream cipher with good performance in s/w.
  - It has been adopted as a replacement for RC4 in several internet standards.

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 ℓ 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* ℓ *bits of an output sequence* s, *can predict the* $(\ell + 1)^{th}$ *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* ℓ *bits of an output sequence* s, *can predict the* $(\ell+1)^{th}$ *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
- $x_n \equiv a.x_{n-1} + b \mod m$, where $n \geq 1$.
- Ouput depends on the **initial seed** x_0 and $a, b, \ \& \ m$.

Linear Congruential Generator

- Designed by D. H. Lehmer in 1949
- $x_n \equiv a.x_{n-1} + b \mod m$, where $n \geq 1$.
- Ouput 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
- $x_n \equiv a.x_{n-1} + b \mod m$, where $n \geq 1$.
- Ouput 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 CSPRBG

- Choose 2 large primes p & q.
- Set $n = p.q$
- 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 $1 \leq x_0 \leq n-1$

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

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

BBS (Blum-Blum-Shub) CSPRBG

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

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

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

Outline

Golomb's Postulates

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

Golomb's Postulates

- Let $s = s_0, s_1, s_2, \ldots$ be an infinite sequence. The subsequence consisting of the first n terms of s is denoted by $s^n = s_0, s_1, \ldots, s_{n-1}$.
- 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 $s = s_0, s_1, s_2, \ldots$ be an infinite sequence. The subsequence consisting of the first n terms of s is denoted by $s^n = s_0, s_1, \ldots, s_{n-1}$.
- 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 $s = s_0, s_1, s_2, \ldots$ be a periodic sequence of period N. The autocorrelation function of s is the integer-valued function $C(t)$ defined as

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

Golomb's Postulates

- Let $s = s_0, s_1, s_2, \ldots$ be an infinite sequence. The subsequence consisting of the first n terms of s is denoted by $s^n = s_0, s_1, \ldots, s_{n-1}$.
- 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 $s = s_0, s_1, s_2, \ldots$ be a periodic sequence of period N. The autocorrelation function of s is the integer-valued function $C(t)$ defined as

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

$C(t)$ 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:

(i) 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,

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

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,

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

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 $s = s_0, s_1, s_2, \ldots, s_{n-1}$ 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 χ^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[2].
- 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 χ^2 distribution with 2 degrees of freedom if $n \geq 21$.

[2] $n_{00} + n_{01} + n_{10} + n_{11} = (n-1)$ 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^{th} type of sequence of length m, $1 \leq i \leq 2^m$.
- The poker test[3] 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 χ^2 distribution with $2^m - 1$ degrees of freedom.

[3] 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 χ^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. Trivium
6. 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 $c_1, \ldots, c_L \in GF(2)$.

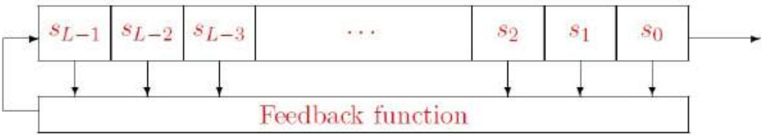
The initial state is an L-bit word $S = (s_{L-1}, \ldots, s_1, s_0)$ and new bits are generated by the recursion

$$s_j = (c_1.s_{j-1} \oplus c_2 s_{j-2} \oplus \ldots \oplus c_L.s_{j-L}) \mod 2, \; for \; j \geq L$$

At each iteration step (clock tick), the state S is updated from $(s_{j-1}, \ldots, s_{j-L})$ to $(s_j, s_{j-1}, \ldots, s_{j-L+1})$, 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)

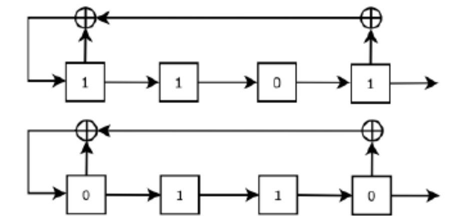


- Let the length of the register be L.
- One defines a set of bits $(c_1, \ldots, c_L)$ where $c_i = 1$ if that cell is tapped and $c_i = 0$ otherwise.
- The initial internal state of the register is given by the bit sequence $(s_{L-1}, \ldots, s_1, s_0)$.
- The output sequence is then defined to be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

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

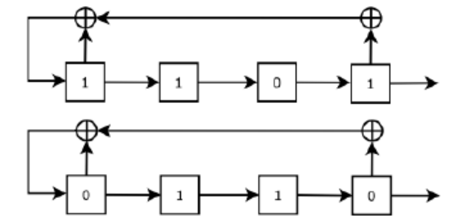
Linear Feedback Shift Registers (LFSR)

Example (LFSR)



Linear Feedback Shift Registers (LFSR)

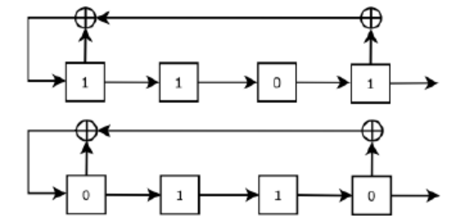
Example (LFSR)



- Connection polynomial:

Linear Feedback Shift Registers (LFSR)

Example (LFSR)



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

Linear Feedback Shift Registers (LFSR)

Example (LFSR)

$$
\begin{array}{ccc}
1101 & \rightarrow & 1 \\
0110 & \rightarrow & 0 \\
0011 & \rightarrow & 1 \\
1001 & \rightarrow & 1 \\
0100 & \rightarrow & 0 \\
0010 & \rightarrow & 0 \\
0001 & \rightarrow & 1 \\
1000 & \rightarrow & 0 \\
1100 & \rightarrow & 0 \\
1110 & \rightarrow & 0 \\
1111 & \rightarrow & 1 \\
0111 & \rightarrow & 1 \\
1011 & \rightarrow & 1 \\
0101 & \rightarrow & 1 \\
1010 & \rightarrow & 1
\end{array}
$$

Linear Feedback Shift Registers (LFSR)

Example (LFSR)

$$
\begin{array}{ccc}
1101 & \to & 1 \\
0110 & \to & 0 \\
0011 & \to & 1 \\
1001 & \to & 1 \\
0100 & \to & 0 \\
0010 & \to & 0 \\
0001 & \to & 1 \\
1000 & \to & 0 \\
1100 & \to & 0 \\
1110 & \to & 0 \\
1111 & \to & 1 \\
0111 & \to & 1 \\
1011 & \to & 1 \\
0101 & \to & 1 \\
1010 & \to & 1 \\
1101 & &
\end{array}
$$

Linear Feedback Shift Registers (LFSR)

Definition

Let $s_0, s_1, s_2, \ldots$ *be a linear recurring sequence. The 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 $s_0, s_1, s_2, \ldots$ be a linear recurring sequence. The 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 $2^n - 1$.

Linear Complexity

Definition

*The **linear complexity** of an infinite binary sequence s, denoted $L(s)$, is defined as follows:*

1. (i) *if s is the zero sequence $s = 0, 0, 0, \ldots,$ then $L(s) = 0$;*
2. (ii) *if no LFSR generates s, then $L(s) = \infty$;*
3. (iii) *otherwise, $L(s)$ is the length of the shortest LFSR that generates s.*

Linear Complexity

Definition

*The **linear complexity** of an infinite binary sequence s, denoted $L(s)$, is defined as follows:*

1. *if s is the zero sequence $s = 0, 0, 0, \ldots,$ then $L(s) = 0$;*
2. *if no LFSR generates s, then $L(s) = \infty$;*
3. *otherwise, $L(s)$ is the length of the shortest LFSR that generates s.*

Definition

The linear complexity of a finite binary sequence s^n, denoted $L(s^n)$, is the length of the shortest LFSR that generates a sequence having s^n as its first n terms.

Properties of Linear Complexity

(i) For any $n \geq 1$, the linear complexity of the subsequence s^n satisfies $0 \leq L(s^n) \leq n$.

(ii) $L(s^n) = 0$ iff s^n is the zero sequence of length n.

(iii) $L(s^n) = n$ iff $s^n = 0, 0, 0, \ldots, 0, 1$.

(iv) If s is periodic with period N, then $L(s) \leq N$.

(v) $L(s \oplus t) \leq L(s) + L(t)$, where $s \oplus t$ denotes the bitwise XOR of s and t.

Non-linear FSR (NLFSR)

Example

- Consider a 4-stage NFSR with a feedback function

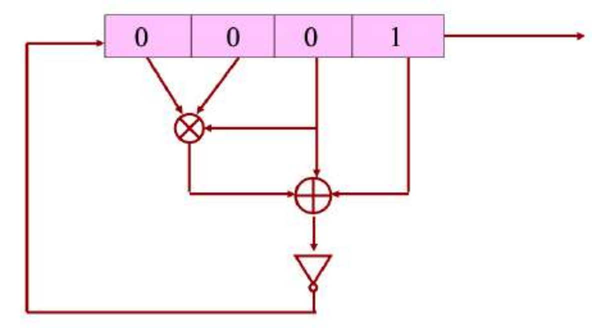
$$f(x_0, x_1, x_2, x_3) = 1 + x_0 + x_1 + x_1x_2x_3$$

Non-linear FSR (NLFSR)

Example

- Consider a 4-stage NFSR with a feedback function

$$f(x_0, x_1, x_2, x_3) = 1 + x_0 + x_1 + x_1 x_2 x_3$$



Non-linear FSR (NLFSR)

Example

$f(x_0, x_1, x_2, x_3) = 1 + x_0 + x_1 + x_1 x_2 x_3$ – de Bruijn FSR

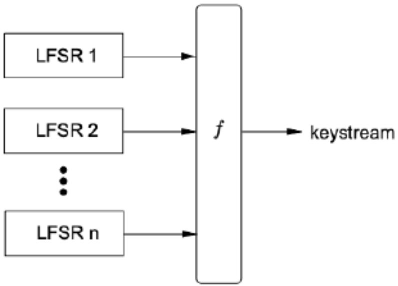
$$
\begin{array}{ccc}
0001 & \rightarrow & 1 \\
0000 & \rightarrow & 0 \\
1000 & \rightarrow & 0 \\
1100 & \rightarrow & 0 \\
1110 & \rightarrow & 0 \\
1111 & \rightarrow & 1 \\
0111 & \rightarrow & 1 \\
1011 & \rightarrow & 1 \\
1101 & \rightarrow & 1 \\
0110 & \rightarrow & 0 \\
0011 & \rightarrow & 1 \\
1001 & \rightarrow & 1 \\
0100 & \rightarrow & 0 \\
1010 & \rightarrow & 0 \\
0101 & \rightarrow & 1 \\
0010 & \rightarrow & 1
\end{array}
$$

Stream Ciphers Based on LFSRs

- **combination generator**
- **filter generator**
- **shrinking generator**

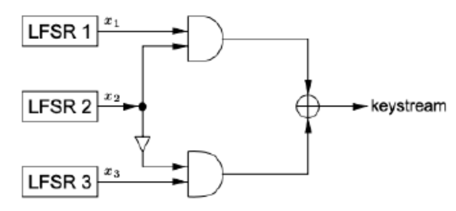
Non-linear Combination Generator

- One general technique for destroying the linearity inherent in LFSRs is to use several LFSRs in parallel.
- The key stream is generated as a non-linear function f of the outputs of the component LFSRs.
- Such key stream generators are called non-linear combination generators, and f is called the combining function.



Non-linear Combination Generator

Example (Geffe Generator)

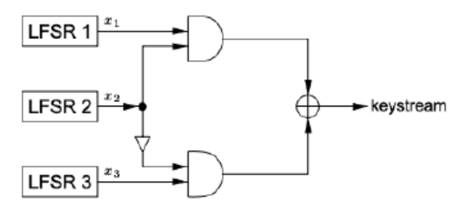


- Consider 3 maximum-length LFSRs whose lengths L_1, L_2, L_3 are pairwise relatively prime, with nonlinear combining function

$$f(x_1, x_2, x_3) = x_1x_2 \oplus (1 + x_2)x_3 = x_1x_2 \oplus x_2x_3 \oplus x_3.$$

Non-linear Combination Generator

Example (Geffe Generator)



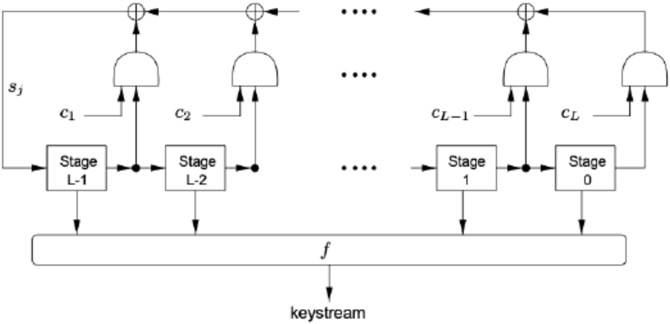
- Consider 3 maximum-length LFSRs whose lengths L_1, L_2, L_3 are pairwise relatively prime, with nonlinear combining function

$$f(x_1, x_2, x_3) = x_1x_2 \oplus (1 + x_2)x_3 = x_1x_2 \oplus x_2x_3 \oplus x_3.$$

- The keystream generated has period $(2^{L_1} - 1)(2^{L_2} - 1)(2^{L_3} - 1)$ and linear complexity $L = L_1L_2 + L_2L_3 + L_3$.

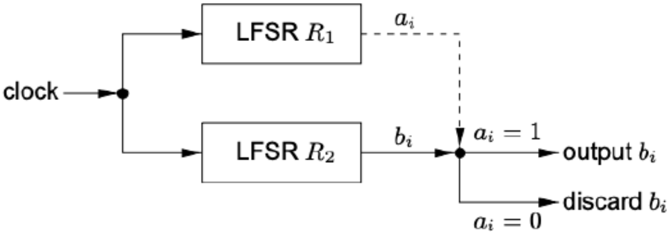
Filter Generator

- A filter generator is a running-key generator for stream cipher applications.
- It consists of a single LFSR which is filtered by a non-linear function f.



Shrinking Generator

- A control LFSR R_1 is used to select a portion of the output sequence of a second LFSR R_2
- Due to its simplicity, it was a promising candidate for high-speed encryption applications.



Outline

RC4

- A self-modifying lookup table (or Synchronous stream cipher) designed by Ron Rivest in 1987.
- Table always contains a permutation of the byte values $0, 1, \ldots, 255$
- Initialize the permutation using key
- At each step, RC4 does the following

RC4

- A self-modifying lookup table (or Synchronous stream cipher) designed by Ron Rivest in 1987.
- Table always contains a permutation of the byte values $0, 1, \ldots, 255$
- 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) designed by Ron Rivest in 1987.
- Table always contains a permutation of the byte values $0, 1, \ldots, 255$
- 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) designed by Ron Rivest in 1987.
- Table always contains a permutation of the byte values $0, 1, \ldots, 255$
- 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 $K[0], K[1], \ldots, K[n-1]$ of n bytes, $1 \leq n \leq 255$
Output: State array $S[0], S[1], \ldots, S[255]$

for $i = 0$ to 255 **do**
    $S[i] = i$
end for
$j = 0$
for $i = 0$ to 255 **do**
    $j = (j + S[i] + K[i \bmod n]) \bmod 256$
    Swap the values of $S[i]$ and $S[j]$
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

$i = 0$
$j = 0$
while Keystream is generated **do**
    $i = i + 1$
    $j = (j + S[i]) \bmod 256$
    Swap the values of $S[i]$ and $S[j]$
    $B = S[(S[i] + S[j]) \bmod 256]$
    Output B
end while

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

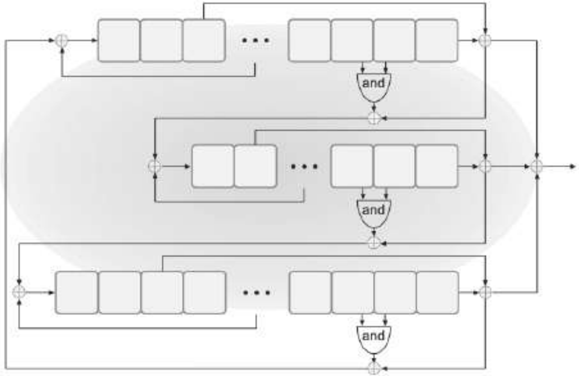
Outline

Trivium

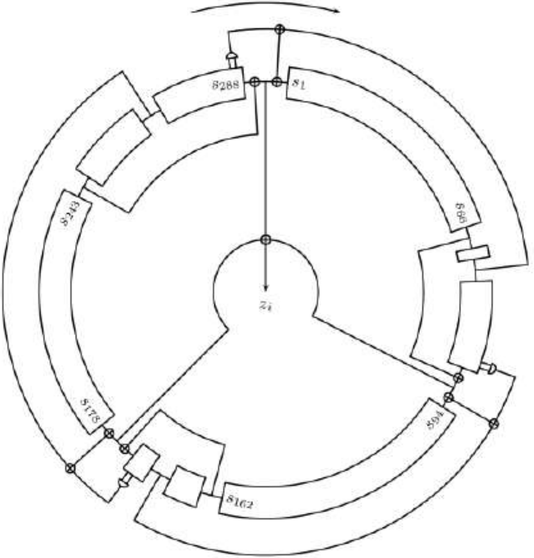
- Designed by De Canniére and Preneel in 2006 as part of eSTREAM competition
- Intended to be simple and efficient (especially in hardware)

Trivium

- Designed by De Canniére and Preneel in 2006 as part of eSTREAM competition
- Intended to be simple and efficient (especially in hardware)



Trivium Hardware



Trivium

- **Parameters:**
  Key size: 80 bit, IV size: 80 bit, Internal state: 288 bit

Trivium

- **Parameters:**
  Key size: 80 bit, IV size: 80 bit, Internal state: 288 bit
- Three coupled FSR of degree 93, 84, and 111.
- **Initialization:**
  - 80-bit key in left-most registers of first FSR
  - 80-bit IV in left-most registers of second FSR
  - Remaining registers set to 0, except for three right-most (all 1s) registers of third FSR
  - run for 4×288 clock ticks to finish initialization

https://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf

Trivium-Initialization

For $i = 1$ to 4×288 do

1. $t_1 \leftarrow s_{66} + s_{91}s_{92} + s_{93} + s_{171}$
2. $t_2 \leftarrow s_{162} + s_{175}s_{176} + s_{177} + s_{264}$
3. $t_3 \leftarrow s_{243} + s_{286}s_{287} + s_{288} + s_{69}$
4. $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
5. $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, ..., s_{176})$
6. $(s_{178}, s_{279}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$

Trivium-Initialization

For $i = 1$ to 4×288 do

1. $t_1 \leftarrow s_{66} + s_{91}s_{92} + s_{93} + s_{171}$
2. $t_2 \leftarrow s_{162} + s_{175}s_{176} + s_{177} + s_{264}$
3. $t_3 \leftarrow s_{243} + s_{286}s_{287} + s_{288} + s_{69}$
4. $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
5. $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, ..., s_{176})$
6. $(s_{178}, s_{279}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$

Note: no random bits output. This is just initialization.

Trivium-Iteration

For $i = 1$ to $N(\leq 2^{64})$ do

1. $t_1 \leftarrow s_{66} + s_{93}$
2. $t_2 \leftarrow s_{162} + s_{177}$
3. $t_3 \leftarrow s_{243} + s_{288}$
4. $z_i \leftarrow t_1 + t_2 + t_3$

Trivium-Iteration

For $i = 1$ to $N(\leq 2^{64})$ do

1. $t_1 \leftarrow s_{66} + s_{93}$
2. $t_2 \leftarrow s_{162} + s_{177}$
3. $t_3 \leftarrow s_{243} + s_{288}$
4. $z_i \leftarrow t_1 + t_2 + t_3$ **1 bit of key stream**
5. $t_1 \leftarrow t_1 + s_{91} s_{92} + s_{171}$
6. $t_2 \leftarrow t_2 + s_{175} s_{176} + s_{264}$
7. $t_3 \leftarrow t_3 + s_{286} s_{287} + s_{69}$
8. $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
9. $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \leftarrow, s_{176})$
10. $(s_{178}, s_{279}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$

Outline

Salsa20/20

- Salsa20 is based on three simple operations:
  - modular addition of 32-bit words a and $b \mod 2^{32}$, 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 $a \lll t$.

[4] 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 $b \mod 2^{32}$, 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 4×4 matrix of sixteen 32-bit words[4]

[4] 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:

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

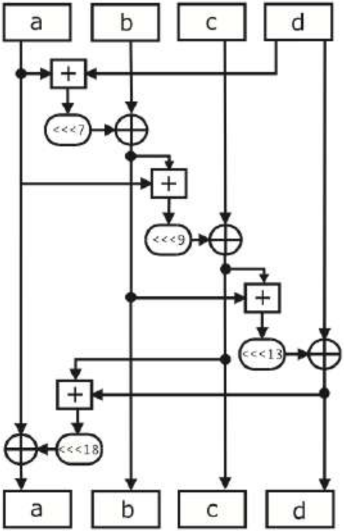
Salsa20/20

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

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

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 $k = (k_1, \ldots, k_8)$ 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 *10 double-round* iterations and a final addition $\mod 2^{32}$ 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 *10 double-round* iterations and a final addition $\mod 2^{32}$ of the initial state matrix:

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

ChaCha20 is a modification of Salsa20

ChaCha20

- ChaCha20 is a stream cipher intended to be extremely efficient in s/w, introduced in 2008.
- It is available as a replacement for RC4 in many systems.
- It is combined with the Poly1305 message authentication code to construct an authenticated encryption (AE) scheme widely used in the TLS protocol.

ChaCha20 Quarter-round

- Let $y = (a, b, c, d)$ be a sequence of four 32-bit words.
- Then a ChaCha quarter-round updates (a, b, c, d) as follows:

  (i) $a = a + b; \quad d = d \oplus a; \quad d \lll 16;$

  (ii) $c = c + d; \quad b = b \oplus c; \quad b \lll 12;$

  (iii) $a = a + b; \quad d = d \oplus a; \quad d \lll 8;$

  (iv) $c = c + d; \quad b = b \oplus c; \quad b \lll 7;$

CChaCha20 Double-round

- ChaCha20 also runs 10 double-rounds.
- However, a ChaCha double-round consists of a column-round and a diagonal-round
- A ChaCha double-round is defined by the 8 ChaCha quarter-rounds

column-round	quarter-round(y_0, y_4, y_8, y_{12}) quarter-round(y_1, y_5, y_9, y_{13}) quarter-round$(y_2, y_6, y_{10}, y_{14})$) quarter-round$(y_3, y_7, y_{11}, y_{15})$
diagonal-round	quarter-round$(y_0, y_5, y_{10}, y_{15})$ quarter-round$(y_1, y_6, y_{11}, y_{12})$ quarter-round(y_2, y_7, y_8, y_{13}) quarter-round(y_3, y_4, y_9, y_{14})

ChaCha20

The state array S:

$$S = \begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ k_1 & k_2 & k_3 & k_4 \\ k_5 & k_6 & k_7 & k_8 \\ b & n_1 & n_2 & n_3 \end{pmatrix}$$

- The ChaCha20 stream cipher takes a 256-bit key $k = (k_1, \ldots, k_8)$ and a unique 96-bit message number $n = (n_1, n_2, n_3)$ (nonce) as input.
- A 32-bit block counter b is initially set to zero and the four 32-bit constants

$$y_0 = 61707865, y_1 = 3320646E, y_2 = 79622D32, y_3 = 6B206574$$

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

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.
- Alfred J. Menezes, Paul C. van Oorschot & Scott A. Vanstone,
  Handbook of Applied Cryptography, CRC Press, 1996.
- 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!