Stream ciphers
Stream Ciphers are Pseudorandom Generators made practical!

They are better than PRG’s!

Are Stream Ciphers ciphers? Depends on who you ask.
Stream ciphers

Stream Ciphers are Psuedorandom Generators made practical!

They are better than PRG’s!

Are Stream Ciphers ciphers? Depends on who you ask.

Some people identify the stream cipher with the cipher that results from using it as the pseudo-one-time-pad.
Stream Ciphers are Pseudorandom Generators made practical!

They are better than PRG’s!

Are Stream Ciphers ciphers? Depends on who you ask.

Some people identify the stream cipher with the cipher that results from using it as the pseudo-one-time-pad.

We will not do that.
Stream Ciphers are Psuedorandom Generators made practical!

They are better than PRG’s!

Are Stream Ciphers ciphers? Depends on who you ask.

Some people identify the stream cipher with the cipher that results from using it as the pseudo-one-time-pad.

We will not do that.

However,
Stream Ciphers are Pseudorandom Generators made practical!

They are better than PRG’s!

Are Stream Ciphers ciphers? Depends on who you ask.

Some people identify the stream cipher with the cipher that results from using it as the pseudo-one-time-pad.

We will not do that.

However, we are right, and they are wrong.
Stream ciphers

- As we defined them, PRGs are limited
  - They have fixed-length output
  - They produce output in “one shot”

- In practice, PRGs are based on stream ciphers
  - Can be viewed as producing an “infinite” stream of pseudorandom bits, on demand
  - More flexible, more efficient
Stream ciphers

A **Stream Cipher** is a pair of efficient, deterministic algorithms (Init, GetBits) such that:

1. **Init** does the following:
   1.1 Input: private seed $s$, opt **public** Init Vector (IV) $V$
   1.2 Output: initial state $st_0$

2. **GetBits** does the following:
   2.1 Input: current state $st$
   2.2 Output: a bit $y$ along with updated state $st'$

**Note** In practice, $y$ is a block rather than a bit.
Stream ciphers

- Can use (Init, GetBits) to generate any desired number of output bits from an initial seed
Stream ciphers

- A stream cipher is *secure* (informally) if the output stream generated from a uniform seed is pseudorandom
  - i.e. regardless of how long the output stream is (so long as it is polynomial)
  - See book for formal definition
A one-way function (perm) is function (perm): easy to compute, hard to invert.
A one-way function (perm) with a hard core predicate is a function (perm) that is easy to compute but hard to invert, and (say) the middle bit of \( f^{-1}(x) \) is hard to compute.

**Known**
\[ \exists \text{ One way Perm} \implies \exists \text{ one way perm with a hcp.} \]
\[ \exists \text{ one way perm with hcp} \implies \exists \text{ PRG with expansion 1} \]
\[ \exists \text{ PRG with expa-1} \implies \exists \text{ Stream Ciphers} \]

**Note:** (1-way func \( \implies \exists \) SC’s) known but much harder.
**Note:** Stream Cipher obtained this way too slow to use :-(
**Note:** Proof of concept valuable :-)}
Do Stream Ciphers exist? Practical

Several attempt Stream Ciphers:

1. Linear Feedback Shift Registers. Fast! Used! Not Secure!
2. Trivium. Fast! Used! Empirically Secure?

Note Seems impossible to get Stream Ciphers that are provably (even using Hardness Assumptions) secure and practical.

Note But having the rigor gives the practitioners (1) a target to shoot for, and (2) pitfalls to watch out for.
Example of Linear Feedback Shift Register

- Assume initial content of registers is 0100

- First 4 state transition: 0100 → 1010 → 0101 → 0010 → ...

- First 3 output bits: 001...
Linear Feedback Shift Registers (LFSR): Example

Degree 3 LFSR, 3 constants: \( c_2, c_1, c_0 \in \{0, 1\} \). + is mod 2.

Key is \( st_0 \) is 3 bits: \((s_2^0, s_1^0, s_0^0)\). NO IV (for now).

\[
st_1 = (s_2^1, s_1^1, s_0^1) = (c_2 s_2^0 + c_1 s_1^0 + c_0 s_0^0, s_2^0, s_1^0).
\]

\[
st_{t+1} = (s_2^t, s_1^t, s_0^t) = (c_2 s_2^t + c_1 s_1^t + c_0 s_0^t, s_2^t, s_1^t).
\]

In English: Bits shift right, left most bit is \( c \)-combo of prior bits.

\[
y_1 = s_0^0 \quad y_2 = s_1^0 \quad y_3 = s_2^0
\]

\[
y_4 = s_2^1 = c_2 y_3 + c_1 y_2 + c_0 y_1
\]

\[
y_t = s_2^{t-3} = c_2 y_{t-3} + c_1 y_{t-2} + c_0 y_{t-1}
\]

In English: \( y_t \) is (1) left most bit of \( st_{t-3} \) & (2) \( c \)-combo of prior \( y \).
Note the Two Definitions of $y_t$

$$y_t = s_{3t-3} = c_2y_{t-3} + c_1y_{t-2} + c_0y_{t-1}$$
Note the Two Definitions of $y_t$

$$y_t = s_{3}^{t-3} = c_2 y_{t-3} + c_1 y_{t-2} + c_0 y_{t-1}$$

1. $y_t = s_3^{t-3}$ is why LFSRs are so fast to compute. Note that all of the operations we do, shift and $+ \text{ mod } 2$ (also called $\oplus$) are very quick. YEAH!
Note the Two Definitions of $y_t$

$$y_t = s_{t-3}^3 = c_2 y_{t-3} + c_1 y_{t-2} + c_0 y_{t-1}$$

1. $y_t = s_{t-3}^3$ is why LFSRs are so fast to compute. Note that all of the operations we do, shift and $+ \mod 2$ (also called $\oplus$) are very quick. YEAH!

2. $y_t = c_2 y_{t-3} + c_1 y_{t-2} + c_0 y_{t-1}$ is why (later) LFSRs are crackable. BOO!.
Linear Feedback Shift Registers (LFSR)

Degree $n$ LFSR, $n$ constants: $c_{n-1}, \ldots, c_0 \in \{0, 1\}$. + is mod 2.

Key is $st_0$ is $n$ bits: $(s_{n-1}^0, \ldots, s_0^0)$. NO IV (for now).

$$st_1 = (s_{n-1}^1, \ldots, s_0^1) = (c_{n-1}s_{n-1}^0 + \cdots + c_0s_0^0, s_{n-1}^0, s_{n-3}^0, \ldots, s_1^0).$$

$$st_{t+1} = (s_{n-1}^t, \ldots, s_0^t) = (c_{n-1}s_{n-1}^t + \cdots + c_0s_0^t, s_{n-1}^t, s_{n-2}^t, \ldots, s_1^t).$$

In English: Bits shift right, left most bit is $c$-combo of prior bits.

$$y_1 = s_0^0 \quad \cdots \quad y_n = s_{n-1}^0$$

$$y_n = s_n^1 = c_{n-1}y_n + \cdots + c_0y_1$$

$$y_t = s_{n-t}^{t-n} = c_{n-1}y_{t-n} + \cdots + c_0y_{t-n}$$

In English: $y_t$ is (1) left most bit of $st_{t-n}$ & (2) $c$-combo of prior $y$. 
Note the Two Definitions of $y_t$

\[ y_t = s_{n}^{t-n} = c_{n-1} y_{t-n} + \cdots + c_{0} y_{t-n} \]
Note the Two Definitions of $y_t$

$$y_t = s_t^{t-n} = c_{n-1}y_{t-n} + \cdots + c_0y_{t-n}$$

1. $y_t = s_n^{t-3}$ is why LFSR’ are so fast to compute $y_t$. Note that all of the operations we do, shift and $+$ mod 2 (also called $\oplus$) are very quick. YEAH!
Note the Two Definitions of $y_t$

$y_t = s_n^{t-n} = c_{n-1}y_{t-n} + \cdots + c_0y_{t-n}$

1. $y_t = s_n^{t-3}$ is why LFSR’s are so fast to compute $y_t$. Note that all of the operations we do, shift and $+$ mod 2 (also called $\oplus$) are very quick. YEAH!

2. $y_t = c_{n-1}y_{t-n} + \cdots + c_0y_{t-1}$ is why (later) LFSR’s are crackable. BOO!
LFSR of degree $n$ is defined by $c_{n-1}, \ldots, c_0$ all in \{0, 1\}

Key is $st'_0$ is $n$ bits. IV is $n$ bits $IV$. $st_0 = st'_0 \oplus IV$.
All the rest is the same.
In English: XOR the private key with the public IV.
Why do this? Next Slide.
Two Ways to Use Stream Ciphers

Two Ways to Use Stream Ciphers. We illustrate with LFSR.

1. **Syn Mode** Alice gen and send private key. Bob sends message of length $L$, using $y_1 \cdots y_L$. Bob uses key to get $y_1 \cdots y_L$ and decode. Bob responds with message of length $M$ using $y_{L+1} \cdots y_{L+M}$. They both keep in sync.

2. **Unsyn Mode** Alice gen and send private key. Alice sends public IV. Bob sends message of length $L$ AND IV. Alice uses key and IV to get $y_1 \cdots y_L$ and can decode. Alice sends message of length $M$ AND IV. Bob uses Key and IV to get $y_1 \cdots y_M$ and can decode. They do not have to be in sync.

When to use which? **Discuss**
Two Ways to Use Stream Ciphers

Two Ways to Use Stream Ciphers. We illustrate with LFSR.

1. **Syn Mode** Alice gen and send private key. Bob sends message of length $L$, using $y_1 \cdots y_L$. Bob uses key to get $y_1 \cdots y_L$ and decode. Bob responds with message of length $M$ using $y_{L+1} \cdots y_{L+M}$. They both keep in sync.

2. **Unsyn Mode** Alice gen and send private key. Alice sends public IV. Bob sends message of length $L$ AND IV. Alice uses key and IV to get $y_1 \cdots y_L$ and can decode. Alice sends message of length $M$ AND IV. Bob uses Key and IV to get $y_1 \cdots y_M$ and can decode. They do not have to be in sync.

When to use which? **Discuss**

1. Use **Sync** if all communication will be in one session and communication is clear.

2. Use **Unsync** if Alice talks Monday, Bob replies Tuesday perhaps on a diff device or if Comm is noisy.
LFSRs as stream ciphers

- Key + IV used to initialize the state of the LFSR

- Every clock tick:
  1. State Updated
  2. Bit output
LFSRs

1. State (and output) “cycles” if state ever repeated

2. *Maximal-length LFSR* cycles through all $2^n - 1$ nonzero states

3. Known how to set feedback coefficients so as to achieve maximal length

4. Maximal-length LFSRs have good statistical properties . . .

5. Are LFSRs secure? Vote YES, NO, UNKNOWN TO SCIENCE.
LFSRs

1. State (and output) “cycles” if state ever repeated

2. *Maximal-length LFSR* cycles through all \(2^n - 1\) nonzero states

3. Known how to set feedback coefficients so as to achieve maximal length

4. Maximal-length LFSRs have good statistical properties . . .

5. Are LFSRs secure? Vote YES, NO, UNKNOWN TO SCIENCE. NO.
Example of Bad Security

Degree 3. $c_0, c_1, c_2$ unknown. $s_0^0, s_1^0, s_2^0$ unknown.

$y_1 = s_0^0$
$y_2 = s_1^0$
$y_3 = s_2^0$
Example of Bad Security

Degree 3. $c_0, c_1, c_2$ unknown. $s_0^0, s_1^0, s_2^0$ unknown.

$y_1 = s_0^0$
$y_2 = s_1^0$
$y_3 = s_2^0$

$y_4 = c_2 y_3 + c_1 y_2 + c_0 y_1$
$y_5 = c_2 y_4 + c_1 y_3 + c_0 y_2$
$y_6 = c_2 y_5 + c_1 y_4 + c_0 y_3$
Example of Bad Security

Degree 3. $c_0, c_1, c_2$ unknown. $s_0^0, s_1^0, s_2^0$ unknown.

$y_1 = s_0^0$
$y_2 = s_1^0$
$y_3 = s_2^0$

$y_4 = c_2 y_3 + c_1 y_2 + c_0 y_1$
$y_5 = c_2 y_4 + c_1 y_3 + c_0 y_2$
$y_6 = c_2 y_5 + c_1 y_4 + c_0 y_3$

3 linear equations in 3 variables. Can find $c_0, c_1, c_2$. Cracked!

For $n$-degree LFSR can crack after $2n$ iterations.  
**Moral:** Linearity is *bad* cryptography.
LFSR and Linearity

Linearity makes LFSR’s fast

Linearity makes LFSR’s crackable
LFSR and Linearity

Linearity makes LFSR’s fast

Linearity makes LFSR’s crackable

It’s that old saying:

_He who lives by linearity, dies by linearity._
Recall: The Essence of Crypto is to make computation
   1. Easy for Alice and Bob.
   2. Hard for Eve.
LFSR makes computation easy for all three!
Nonlinear Feedback Shift Registers (FSRs)

- Add nonlinearity to prevent attacks
  - Nonlinear feedback
  - Output is a nonlinear function of the state
  - Multiple (coupled) LFSRs
  - ... or any combination of the above

- Still want to preserve statistical properties of the output, and long cycle length
Nonlinear Feedback Shift Registers

Assume \( n \) even. \(+\) is mod 2.

Let \( f(x_1, \ldots, x_n) = x_1x_2 + x_3x_4 + \cdots x_{n-1}x_n. \)

\( st_0 \) is \( n \) bits: \((s_{n-1}^0, \ldots, s_0^0)\).

For \( i = 1 \) to \( \infty \)

\[
st_i = (s_{n-1}^i, \ldots, s_0^i) = (f(s_{n-1}^{i-1}, \ldots, s_0^{i-1}), s_{n-1}^{i-1}, s_{n-2}^{i-1}, \ldots, s_1^{i-1})
\]

\[ y_i = s_0^i \]

In English: Bits shift right, left bit is \( f \) of bits at last stage.

Is this a good stream cipher? Vote Y (with HA), N, UN
Nonlinear Feedback Shift Registers

Assume $n$ even. $+$ is mod 2.
Let $f(x_1, \ldots, x_n) = x_1x_2 + x_3x_4 + \cdots x_{n-1}x_n$.

$s_{t0}$ is $n$ bits: $(s_{n-1}^0, \ldots, s_0^0)$.

For $i = 1$ to $\infty$

$$s_{ti} = (s_{n-1}^i, \ldots, s_0^i) = (f(s_{n-1}^{i-1}, \ldots, s_0^{i-1}), s_{n-1}^{i-1}, s_{n-2}^{i-1}, \ldots, s_1^{i-1})$$

$$y_i = s_0^i$$

In English: Bits shift right, left bit is $f$ of bits at last stage.
Is this a good stream cipher? Vote Y (with HA), N, UN UN:
Nonlinear Feedback Shift Registers

Assume \( n \) even. + is mod 2.
Let \( f(x_1, \ldots, x_n) = x_1x_2 + x_3x_4 + \cdots x_{n-1}x_n \).
\( st_0 \) is \( n \) bits: \((s_{n-1}^0, \ldots, s_0^0)\).

For \( i = 1 \) to \( \infty \)

\[
st_i = (s_{n-1}^i, \ldots, s_0^i) = (f(s_{n-1}^{i-1}, \ldots, s_0^{i-1}), s_{n-1}^{i-1}, s_{n-2}^{i-1}, \ldots, s_1^{i-1})
\]

\[
y_i = s_0^i
\]

In English: Bits shift right, left bit is \( f \) of bits at last stage.
Is this a good stream cipher? Vote Y (with HA), N, UN
UN: I made up this cipher last month for example of nonlinear.
Trivium

- Designed by De Cannièere and Preneel in 2006 as part of eSTREAM competition
- Intended to be simple and efficient (especially in hardware)
- Essentially no attacks better than brute-force search are known
Trivium Hardware Abstractly
Trivium

- Three coupled Feedback Shift Regisers (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 registers of third FSR
  - run for 4 x 288 clock ticks to finish init.
Trivium-Initialization

\( K_1, \ldots, K_{80} \) Random
\( IV_1, \ldots, IV_{80} \) Random
\((a_1, \ldots, a_{93}) \leftarrow (K_1, \ldots, K_{80}, 0, \ldots, 0)\)
\((b_1, \ldots, s_{84}) \leftarrow (IV_1, \ldots, IV_{80}, 0, \ldots, 0)\)
\((c_1, \ldots, s_{111}) \leftarrow (0, \ldots, 0, 1, 1, 1)\)
For \( i = 1 \) to \( 4 \times 288 \) do

1. \( t_1 \leftarrow a_{86} + a_{91}a_{92} + b_{79}\)
2. \( t_2 \leftarrow b_{70} + b_{83}b_{84} + c_1 + c_{87}\)
3. \( t_3 \leftarrow c_{66} + c_{100}c_{110} + c_{111} + a_{69}\)
4. \((a_1, \ldots, a_{93}) \leftarrow (t_3, a_1, \ldots, a_{91})\)
5. \((b_1, \ldots, b_{83}) \leftarrow (t_1, b_1, \ldots, b_{82})\)
6. \((c_1, \ldots, s_{111}) \leftarrow (t_2, c_1, \ldots, c_{110})\)

Note no random bits output. This is just initialization.
Trivium-Iteration

We omit superscripts for readability.
For $i = 1$ to $N$ do

1. $y_i = a_{66} + a_{93} + b_{70} + b_{75} + c_{66} + c_{111}$ (ith random bit).
2. $t_1 \leftarrow a_{86} + a_{91}a_{92} + b_{79}$
3. $t_2 \leftarrow b_{70} + b_{83}b_{84} + c_1 + c_{87}$
4. $t_3 \leftarrow c_{66} + c_{100}c_{110} + c_{111} + a_{69}$
5. $(a_1, \ldots, a_{93}) \leftarrow (t_3, a_1, \ldots, a_{92})$
6. $(b_1, \ldots, b_{83}) \leftarrow (t_1, b_1, \ldots, s_{83})$
7. $(c_1, \ldots, c_{111}) \leftarrow (t_2, c_1, \ldots, c_{110})$

Note the three diff parts of $s$ are three coupled nonlinear FSR.
Trivium based on LFSR though not LFSR

Note:

1. \( t_1, t_2, t_3 \) are nonlinear combos of prior bits.
2. \((a_1, \ldots, a_{93}) \leftarrow (t_3, a_1, \ldots, a_{92})\)
3. \((b_1, \ldots, b_{83}) \leftarrow (t_1, s_1 \ldots, s_{82})\)
4. \((c_1, \ldots, s_{111}) \leftarrow (t_2, s_1, \ldots, s_{110})\)

Since \( t_1, t_2, t_3 \) nonlinear, Trivium is NOT LFSR

But

Shift to the right and left most bit is BLAH

is very much like LFSR.

**Benefit:** Shifting is Fast!
Facts About Trivium

1) Has been build in hardware with 3488 logic gates. Small! Fast!
2) So far has not been broken. That we know of!
3) Naive method is $2^{80}$ steps. Guess all keys.
4) If only do $\sim 700$ init steps then Cube Attack is $2^{68}$ steps.
5) Seems to have long period but hard to know:
   1. Nonlin makes it hard to predict. Good for practical A and B.
   2. Nonlin makes it hard to analyze. Bad for theorists A and B.
6) Trivium is also the name of a rock band!
7) Two Papers on Trivium on course website
Why the name Trivium?

We quote the paper

*The word trivium is Latin for “the three-fold way”, and refers to the three-fold symmetry of TRIVIUM. The adjective trivial which was derived from it, has a connotation of simplicity, which is also one of the characteristics of TRIVIUM.*

(Quote continued on next slide)
Why the name Trivium?

Moreover, with some imagination, one might recognize the shape of a Trivial Pursuit board in Fig. 1

(Quote continued on next slide)
Why the name Trivium?

While we admit this respect “Mercedes” would have been a more appropriate name.
Why the name Trivium?

Finally, the name provides a nice title for a subsequence cryptanalysis paper: “Three Trivial Attacks on Trivium”.
This Fall I am teaching the senior course in Crypto at UMCP. It's a nice change of pace for me since REAL people REALLY use this stuff!

There is one topic that looks really practical but I could not find on the web if it is or not. A Secure Stream Cipher is (informally) a way to, given a seed and optionally an Init Vector (IV), generate bits that look random. Trivium seems to be one such. According to the Trivium wiki

THEN I HAD STUFF ABOUT TRIVIUM

Is Trivium used?
If so then by whom and for what (for the pseudo 1-time pad?)?
If not then why not?
First Comment on Blog

Great post on Trivial! Hardware Cube Attack. Metal Education. Click HERE for great deal on Tuxedos!

My Response

No response. However, I blocked the comment as it was clearly spam, and not very good spam at that.

Too bad. They called it a Great post. Oh well.
Great post on Trivial! Hardware Cube Attack. Metal Education. Click HERE for great deal on Tuxedos!

My Response
First Comment on Blog

Great post on Trivial! Hardware Cube Attack. Metal Education. Click HERE for great deal on Tuxedos!

My Response
No response. However,

I blocked the comment as it was clearly spam, and not very good spam at that.

Too bad. They called it a Great post. Oh well.
Second Comment on Blog

meh squared
Second Comment on Blog

meh squared

My response
Second Comment on Blog

meh squared

My response

I take it you are not impressed with Trivium. Can you say why so you can enlighten by readers and I can enlighten my class?
An 80-bit key/IV is not secure enough for many modern uses (like encryption on the Internet), though I am not sure what exactly Trivium and other "lightweight ciphers" consider a threat. Their primary intended deployment scenarios are IoT and hardware tokens like auto door locks.

If you are interested in teaching useful (and used) stream ciphers, you could start with RC4, which was widely used in TLS (i.e. encrypting a lot Internet traffic) until it was very badly broken. RC4 exhibits all sorts of interesting weaknesses for teaching, and it is very simple.

My understanding is that the most widely used stream cipher will soon likely be Chacha20 (again for TLS). The authentication mechanism (Poly1305) and other Wegman-Carter-type MACs involve some algebra and probability that are interesting for teaching crypto as well.
Salsa20 Stream Cipher (not to be confused with Nelson’s Salsa Class)

Notation: $\oplus$ is the usual bit-wise XOR. $+$ is mod $2^{32}$ addition. $<<<$ will mean you circular shift bits to the left.

Basic unit: word which is 32 bits.

Basic Operation: On input four words $(a, b, c, d)$, $QR(a, b, c, d)$ is

\[
\begin{align*}
b & := (b \oplus (b + d)) <<< 7 \\
c & := (c \oplus (a + b)) <<< 9 \\
d & := (d \oplus (b + c)) <<< 13 \\
a & := (a \oplus (c + d)) <<< 18
\end{align*}
\]

Note: $\oplus$ and $+$ and $<<<$ are fast! So $QR(a, b, c, d)$ is fast!.

Note: Scrambles up $a, b, c, d$ a lot!
Initially have a $4 \times 4$ array of bytes (8 bits).

<table>
<thead>
<tr>
<th>Const</th>
<th>Key</th>
<th>Key</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Const</td>
<td>nonce</td>
<td>nonce</td>
</tr>
<tr>
<td>Pos</td>
<td>Pos</td>
<td>Const</td>
<td>Key</td>
</tr>
<tr>
<td>Key</td>
<td>Key</td>
<td>Key</td>
<td>Const</td>
</tr>
</tbody>
</table>

View as 8 words by reading up-down, left-right

**Const**: Constants that are standardized. Public

**Key**: Known only to Alice and Bob, used for long time. Private.

**Nonce**: This IV but can only use a string once. Public

**Pos**: These will start at 0 and increment every time used. Public.

**Note**: Nonce: Number-used-once. Public here but not necc
Initialize for $R$ Rounds:
Even round do $QR(a, b, c, d)$ on the rows,
Every odd round do $QR(a, b, c, d)$ on the columns.

How Many Rounds: Salsa20 sets it to 20. Duh.

Nonce: How to gen rand-looking strings w/o repeats? Discuss

**Salsa20 Stream Cipher-Init and other Issues**
Salsa20 Stream Cipher-Init and other Issues

Initialize for $R$ Rounds:
Even round do $QR(a, b, c, d)$ on the rows,
Every odd round do $QR(a, b, c, d)$ on the columns.

How Many Rounds: Salsa20 sets it to 20. Duh.

Nonce: How to gen rand-looking strings w/o repeats? Discuss
1) Store all prior strings. Too much space.
2) Concat year-month-day-time and a random string. Works!
3) There are other ways.
We now have a well mixed $4 \times 4$ array of bytes (8 bits). Could that just be our random bits? Discuss
We now have a well mixed $4 \times 4$ array of bytes (8 bits). Could that just be our random bits? Discuss

No! All steps are reversible. From that array one can work backwards and find the Key!
Salsa20 Stream Cipher - GetBits

We now have a well mixed $4 \times 4$ array of bytes (8 bits). Could that just be our random bits? Discuss

No! All steps are reversible. From that array one can work backwards and find the Key!

Just one more step:

Let the $4 \times 4$ array be $x[0], \ldots, x[15]$.

Let the $4 \times 4$ initial array be $in[0], \ldots, in[15]$.

For $i = 0$ to 15 output $x[i] + in[i]$.

Security: Salsa20 was introduced in 2005 and has not been broken. See Wikipedia page for partial attacks (e.g., Salsa8).
How to Design a Good Stream Cipher?

SC’s are designed, used, and not broken
How to Design a Good Stream Cipher?

SC’s are designed, used, and not broken until they are.
How to Design a Good Stream Cipher?

SC’s are designed, used, and not broken until they are.

**Frustrating:** Can prove a Stream Cipher is BAD but not GOOD.
How to Design a Good Stream Cipher?

SC’s are designed, used, and not broken until they are.

Frustrating: Can prove a Stream Cipher is BAD but not GOOD.

Jon Katz:

Absent proofs, the only ways to claim that a stream cipher is good are to (1) follow known design principles and (2) make sure known attacks do not work. It helps lend credibility if they are designed by people who know what they are doing, not just throwing random stuff together, but I realize that’s not very scientific.

Trivium, in particular, always struck me as so simple that it cannot possibly be secure. And yet, there are no attacks. But I don’t think it has been subject to the same scrutiny as AES, or even RC4. ChaCha is actually used, so people care about its security. Hence its security seems solid. For now.
Good Science and Bad Science

Karl Popper (1930’s): A Scientific Theory should be falsifiable. Propose experiments that could show it is not true. The longer the theory survives scrutiny the more likely it is to be true.

1) Classical Mechanics: Good Science. Many experiments proposed and carried out. Confirmed it until had problems with fast speeds and small particles.

2) Quantum Mechanics: Good Science. Many experiments proposed and carried out. So far has not been falsified. Yet.

3) Libertarianism Theory: Bad Science: Everything bad is the governments fault w/o looking at data. Global warming require government action, hence its false.

4) Communism: Bad Science:
Wages go down – Capitalists exploiting the worker.
Wages to up – Capitalists placating the worker to avoid revolution.
Good Crypto and Bad Crypto

A Scientific Theory should be falsifiable. Propose experiments that could show it is not true. The longer the theory survives scrutiny the more likely it is to be true. For now.

An encryption system should be falsifiable. Propose ways to break it. The longer it stays unbroken the more likely it is to be unbreakable. For now. Caveat: let many people try! Kerchoffs’s law very useful here!

Speculation: Does the NSA let outsiders try to break their systems? If not then might not be Good Crypto. I really do not know.
Good Crypto and Bad Crypto

A Scientific Theory should be falsifiable. Propose experiments that could show it is not true. The longer the theory survives scrutiny the more likely it is to be true. For now.

An encryption system should be falsifiable. Propose ways to break it. The longer it stays unbroken the more likely it is to be unbreakable. For now. Caveat: let many people try! Kerchoffs’s law very useful here!

Speculation: Does the NSA let outsiders try to break their systems? If not then might not be Good Crypto. I really do not know. I tried asking them but they wouldn’t tell me!