The Salsa20 stream cipher

D. J. Bernstein

Thanks to:
University of Illinois at Chicago
NSF CCR–9983950
Alfred P. Sloan Foundation

Salsa20: additive stream cipher, expanding key and nonce into long stream of bytes to add to plaintext.

Key k: 16 or 32 bytes. Same speed either way, simplifying hardware.

Nonce n: 8 bytes. Can send 2^{64} messages under one key.

Stream $\text{Salsa20}_k(n)$: 2^{70} bytes for each message.

m cipher

is at Chicago
0
oundation

Salsa20: additive stream cipher, expanding key and nonce into long stream of bytes to add to plaintext.

Key k: 16 or 32 bytes. Same speed either way, simplifying hardware.

Nonce n: 8 bytes. Can send 2^{64} messages under one key.

Stream $\text{Salsa20}_k(n)$: 2^{70} bytes for each message.

For authentication
combine Salsa20 w
http://cr.yp.to

Given message m
Send $(n, c, \text{Poly13}$
$(s, c) = \text{Salsa20}_k($

Very fast; short se
provably secure if
better than encryp

Easily adapt to "A
i.e., allow unencry

Salsa20: additive stream cipher, expanding key and nonce into long stream of bytes to add to plaintext.

Key k: 16 or 32 bytes. Same speed either way, simplifying hardware.

Nonce n: 8 bytes. Can send 2^{64} messages under one key.

Stream $\text{Salsa20}_k(n)$: 2^{70} bytes for each message.

For authentication, combine Salsa20 with Poly1305, `http://cr.yp.to/mac.html`.

Given message m with nonce n: Send $(n, c, \text{Poly1305}_r(c, s))$ where $(s, c) = \text{Salsa20}_k(n) \oplus (0, m)$.

Very fast; short secret key (k, r); provably secure if Salsa20 is secure; better than encrypt-then-MAC.

Easily adapt to "AEAD," i.e., allow unencrypted header.

stream cipher,
l nonce
of bytes
t.

ytes.
way,
re.

sages

n):
message.

For authentication,
combine Salsa20 with Poly1305,
`http://cr.yp.to/mac.html`.

Given message m with nonce n:
Send $(n, c, \text{Poly1305}_r(c, s))$ where
$(s, c) = \text{Salsa20}_k(n) \oplus (0, m)$.

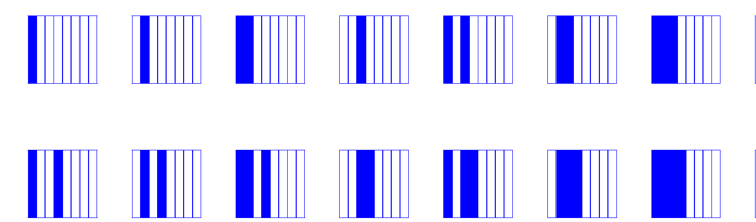
Very fast; short secret key (k, r);
provably secure if Salsa20 is secure;
better than encrypt-then-MAC.

Easily adapt to "AEAD,"
i.e., allow unencrypted header.

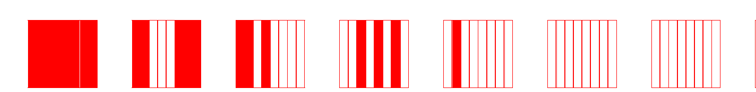
Let's watch how S
generates block of
from key (1, 2, 3, .
nonce (255, 227, 1

Notation: ▮| means
Little-endian every

Key:



Nonce:



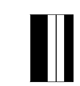
For authentication,
combine Salsa20 with Poly1305,
`http://cr.yp.to/mac.html`.

Given message m with nonce n:
Send $(n, c, \text{Poly1305}_r(c, s))$ where
$(s, c) = \text{Salsa20}_k(n) \oplus (0, m)$.

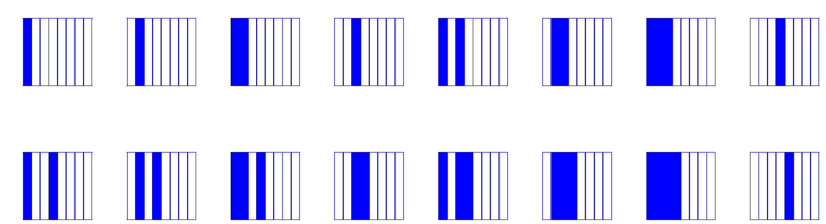
Very fast; short secret key (k, r);
provably secure if Salsa20 is secure;
better than encrypt-then-MAC.

Easily adapt to "AEAD,"
i.e., allow unencrypted header.

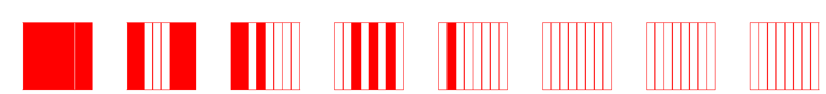
Let's watch how Salsa20
generates block of 64 bytes
from key $(1, 2, 3, \ldots, 16)$,
nonce $(255, 227, 11, 84, 2, 0, 0, 0)$.

Notation:  means $1 + 2 + 16$.
Little-endian everywhere.

Key:



Nonce:



,

with Poly1305,

o/mac.html.

with nonce n:

05$_r(c, s)$) where

$n) \oplus (0, m)$.

cret key (k, r);

Salsa20 is secure;

ot-then-MAC.

AEAD,"

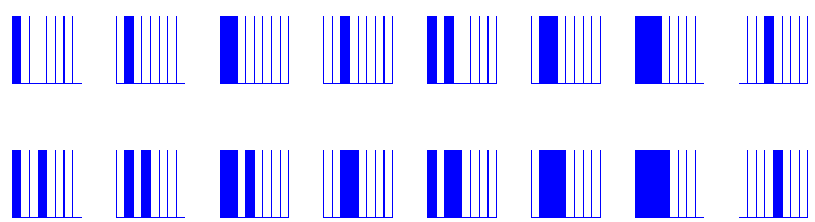
pted header.

Let's watch how Salsa20 generates block of 64 bytes from key $(1, 2, 3, \ldots, 16)$, nonce $(255, 227, 11, 84, 2, 0, 0, 0)$.

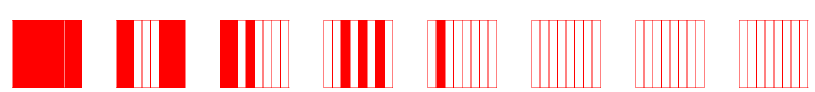
Notation: ▌▏ means $1 + 2 + 16$.

Little-endian everywhere.

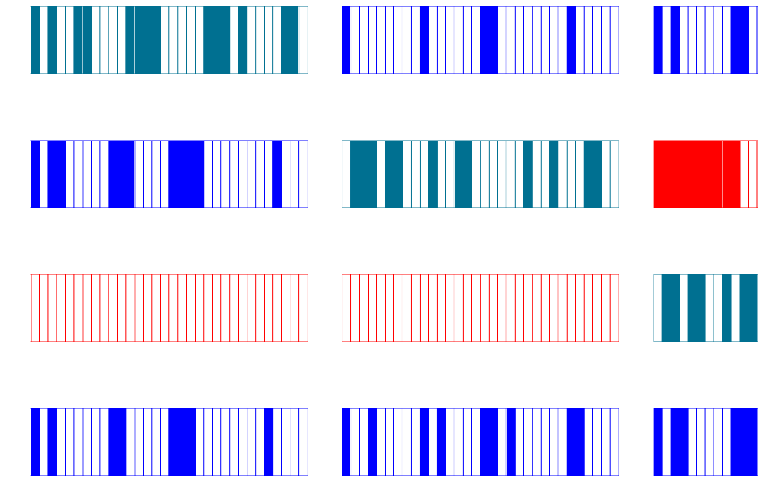
Key:



Nonce:

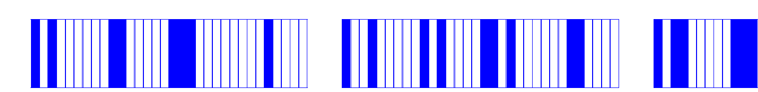


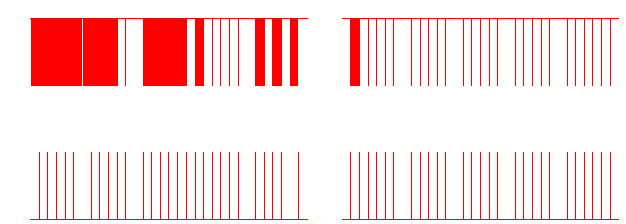
Build 4×4 array o



Diagonal entries a

Other entries are k





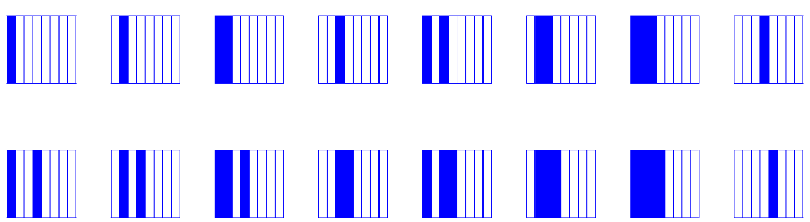
; bl

; ke

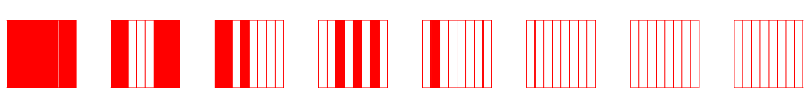
Let's watch how Salsa20 generates block of 64 bytes from key $(1, 2, 3, \ldots, 16)$, nonce $(255, 227, 11, 84, 2, 0, 0, 0)$.

Notation: means $1 + 2 + 16$. Little-endian everywhere.

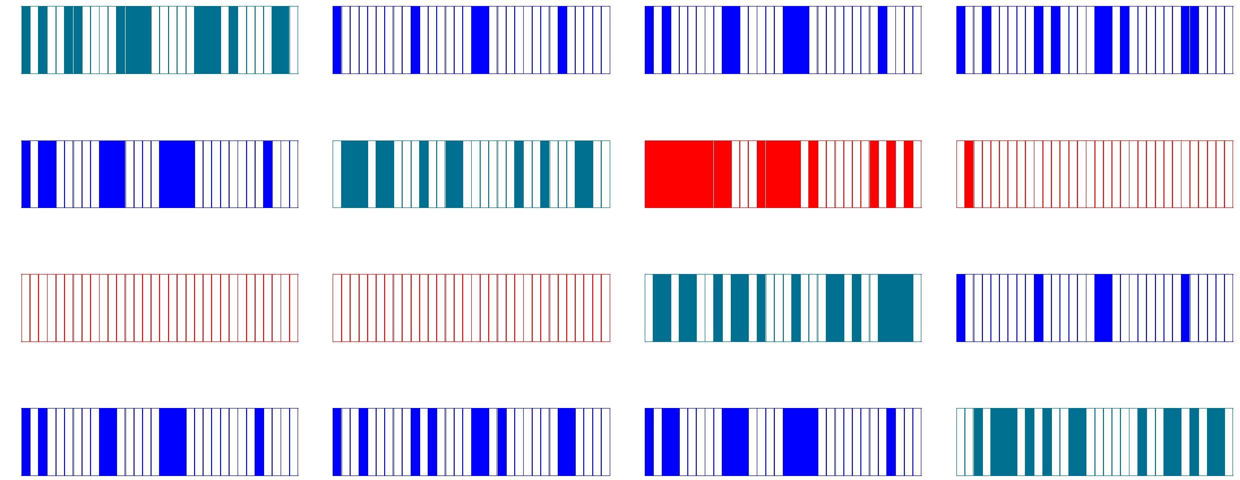
Key:



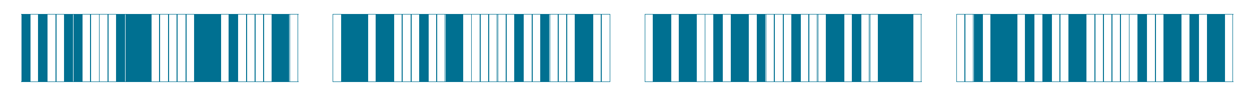
Nonce:

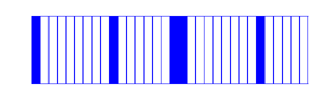


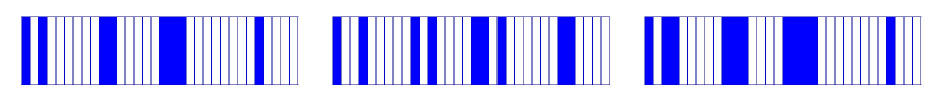
Build 4×4 array of 4-byte words:

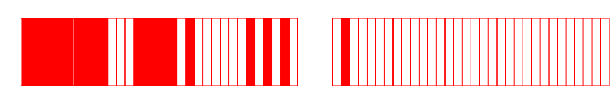


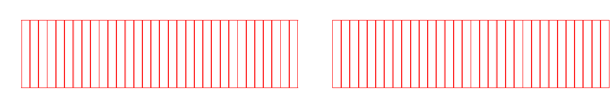
Diagonal entries are constants:



Other entries are key 

; nonce

; block counter

; key again.

alsa20

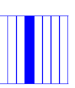
64 bytes

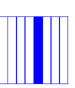
. . . , 16),

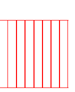
1, 84, 2, 0, 0, 0).

s 1 + 2 + 16.

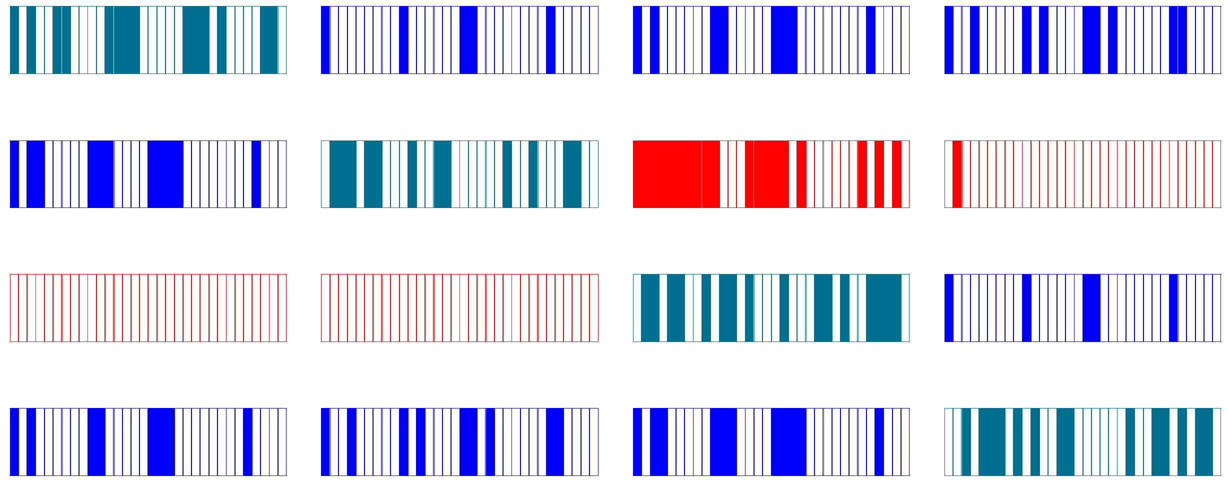
where.



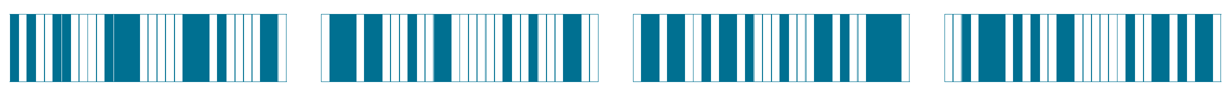
.

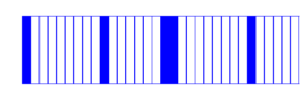
.

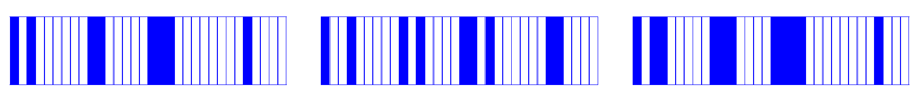
Build 4 × 4 array of 4-byte words:

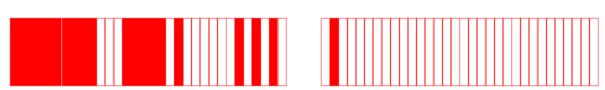


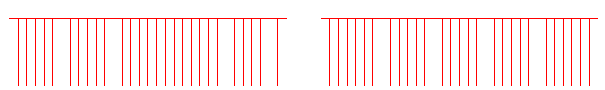
Diagonal entries are constants:



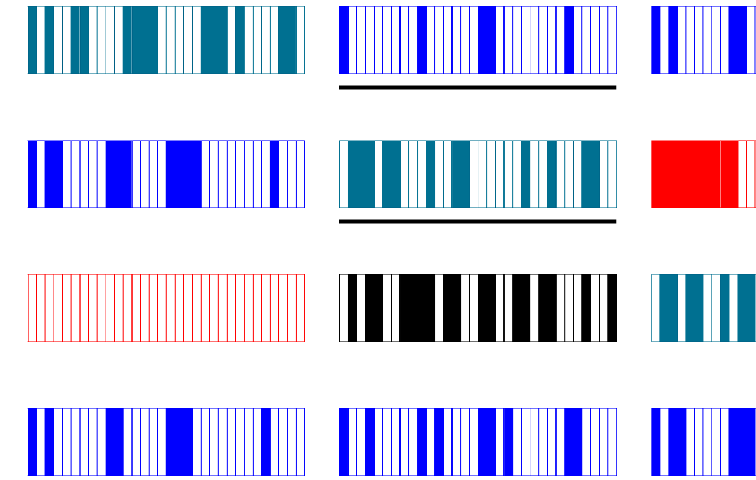
Other entries are key 

; nonce

; block counter

; key again.

Modify one word u



The modification i

add two underlinec

rotate left by 7 bit

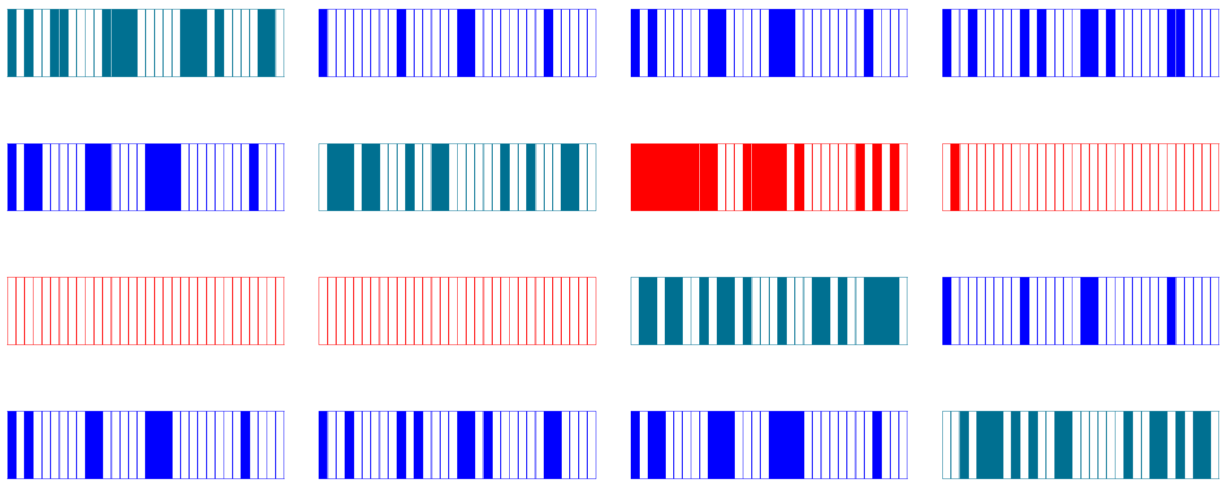
xor into next word

`x[9] ^= (x[1]+x`

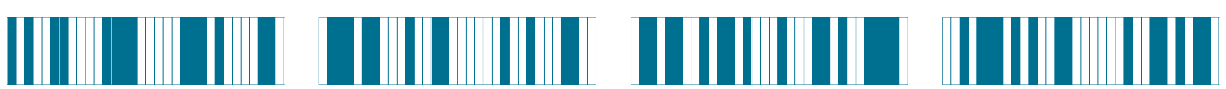
Will do long series

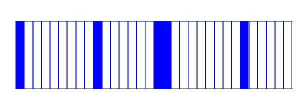
simple modificatio

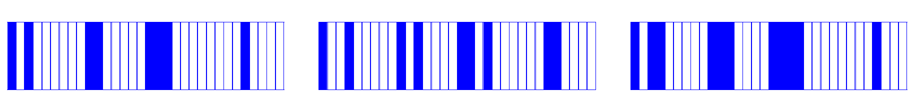
Build 4×4 array of 4-byte words:

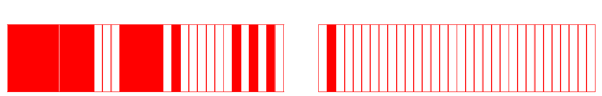


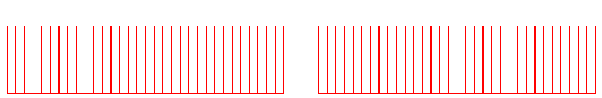
Diagonal entries are constants:



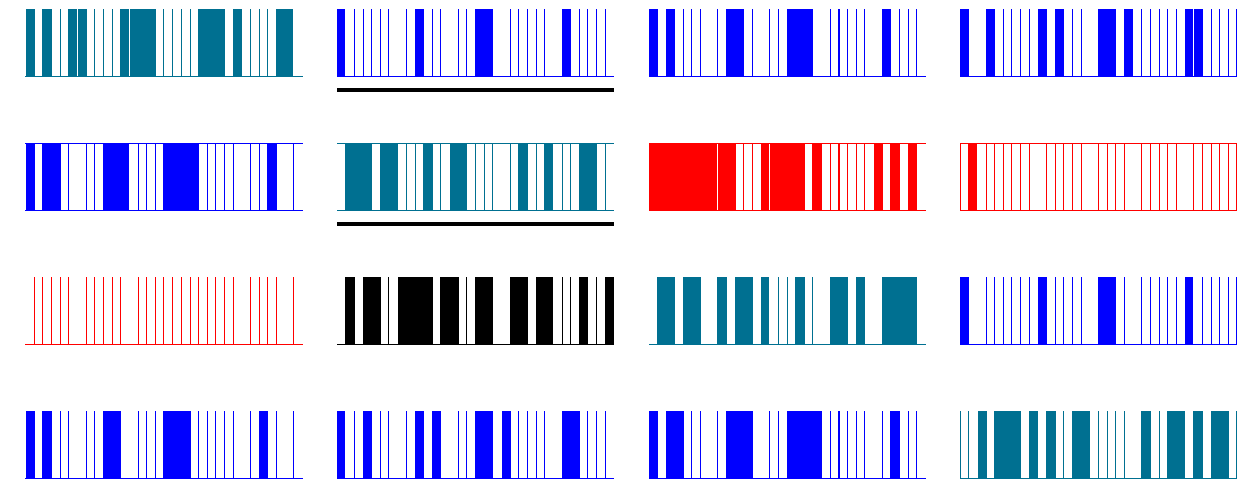
Other entries are key 

; nonce

; block counter

; key again.

Modify one word using two others:

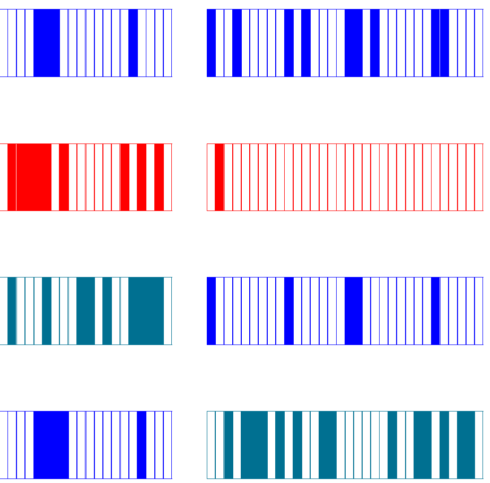


The modification is very simple:
add two underlined words;
rotate left by 7 bits;
xor into next word down.

```
x[9] ^= (x[1]+x[5]) <<< 7
```

Will do long series of these simple modifications, as in TEA.

of 4-byte words:



re constants:

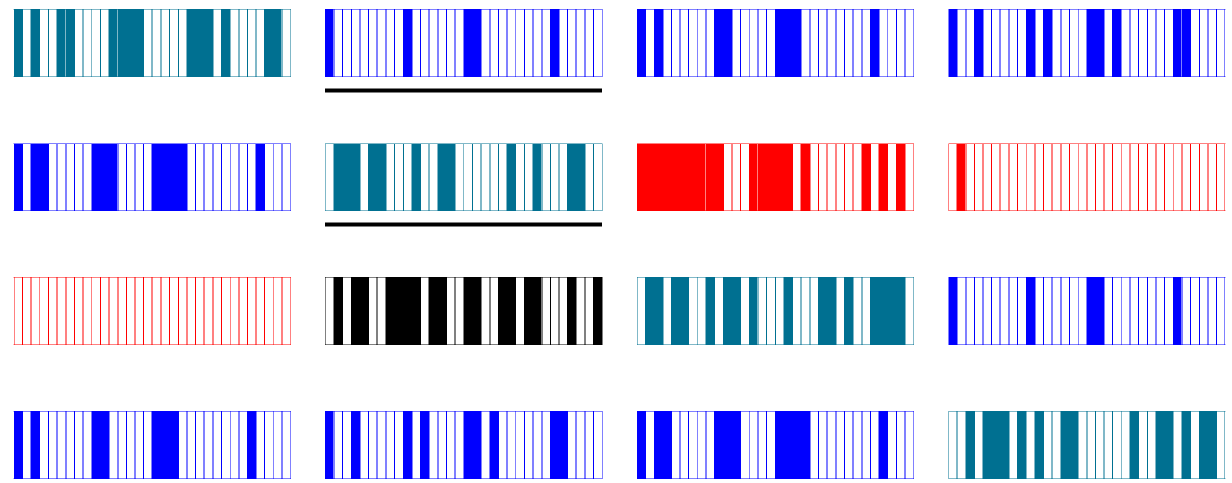
key

; nonce

ock counter

y again.

Modify one word using two others:

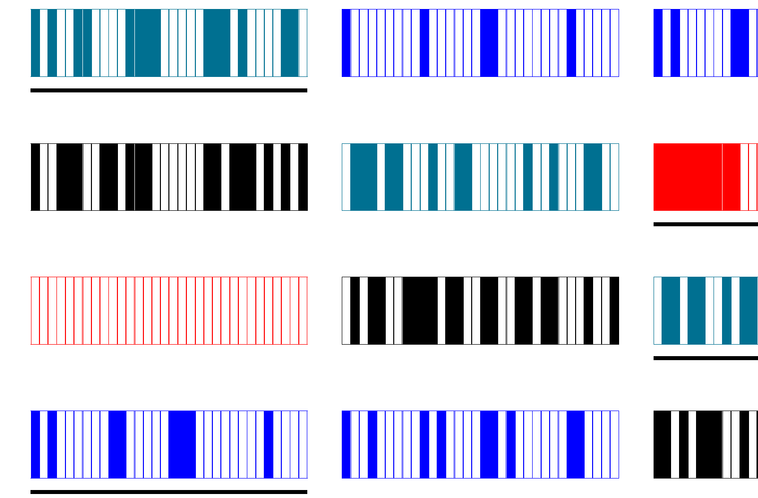


The modification is very simple:
add two underlined words;
rotate left by 7 bits;
xor into next word down.

```
x[9] ^= (x[1]+x[5]) <<< 7
```

Will do long series of these simple modifications, as in TEA.

Modify other colu

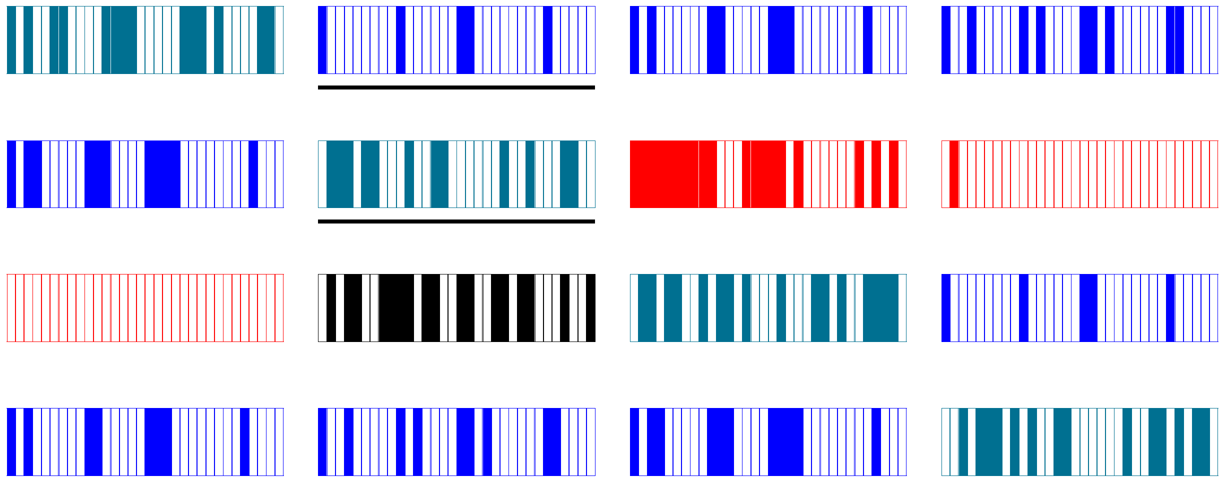


Columns wrap aro
from bottom to to

```
x[4] ^= (x[12]+
x[14] ^= (x[6]+
x[3] ^= (x[11]+
```

Total: 4 modificat

Modify one word using two others:

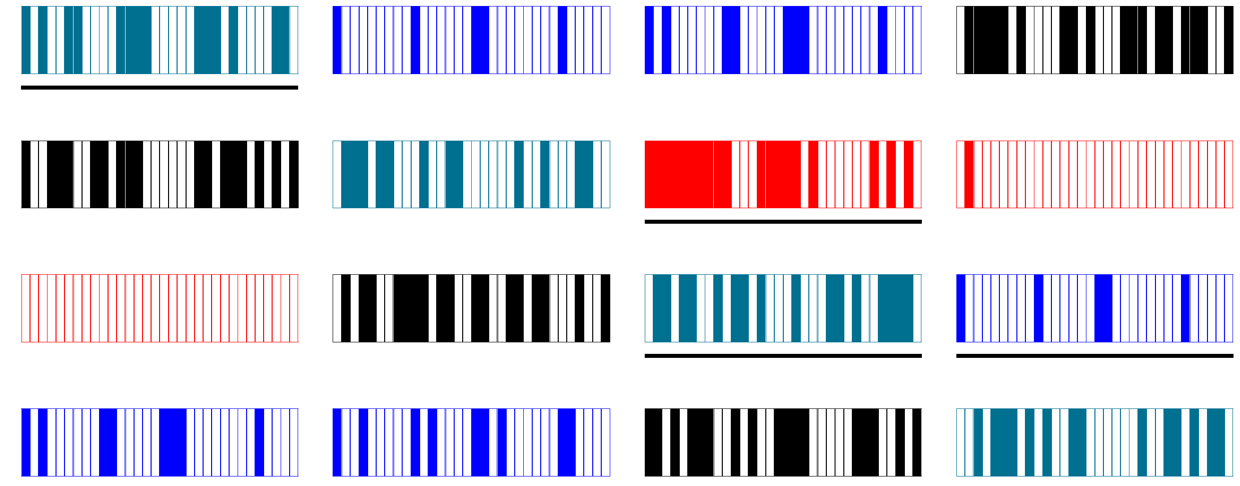


The modification is very simple:
add two underlined words;
rotate left by 7 bits;
xor into next word down.

```
x[9] ^= (x[1]+x[5]) <<< 7
```

Will do long series of these
simple modifications, as in TEA.

Modify other columns:

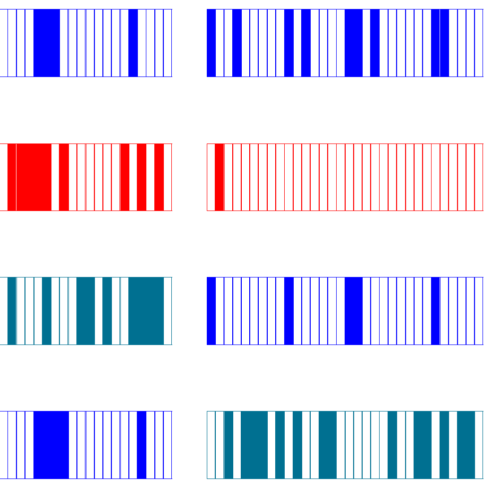


Columns wrap around
from bottom to top.

```
x[4] ^= (x[12]+x[0]) <<< 7
x[14] ^= (x[6]+x[10]) <<< 7
x[3] ^= (x[11]+x[15]) <<< 7
```

Total: 4 modifications.

using two others:



is very simple:

d words;

ts;

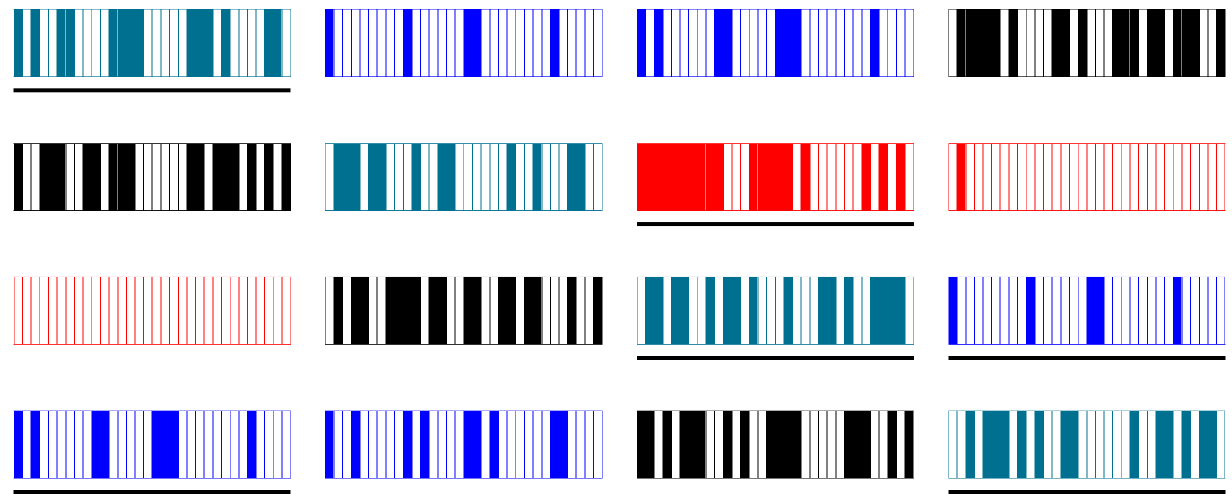
down.

```
[5]) <<< 7
```

of these

ns, as in TEA.

Modify other columns:



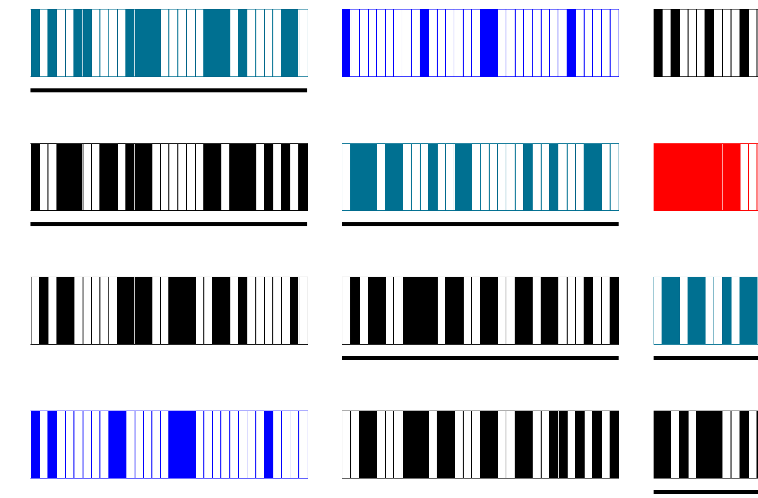
Columns wrap around

from bottom to top.

```
x[4] ^= (x[12]+x[0]) <<< 7
x[14] ^= (x[6]+x[10]) <<< 7
x[3] ^= (x[11]+x[15]) <<< 7
```

Total: 4 modifications.

Modify each colun

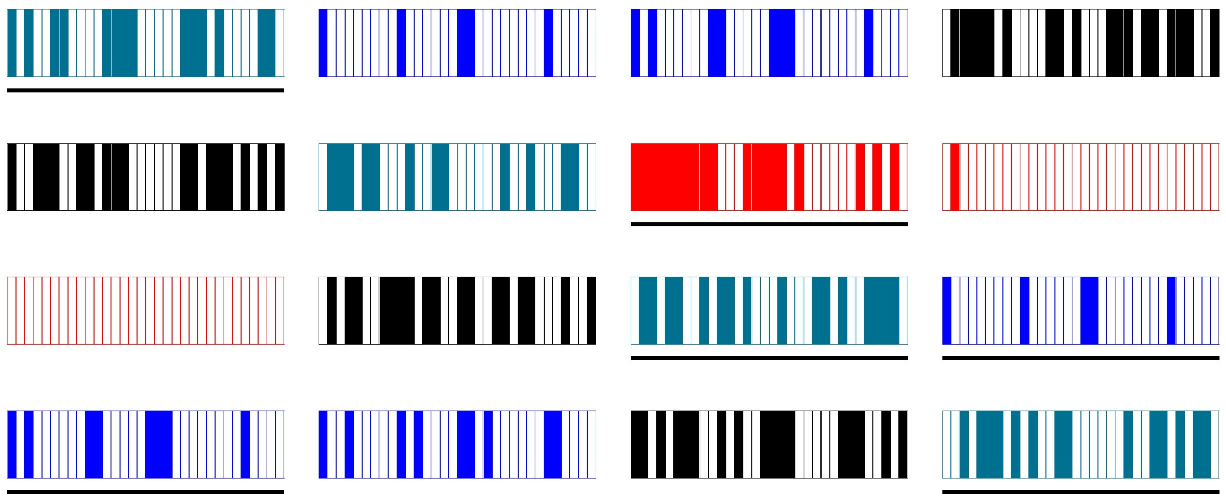


This time rotate b

```
x[8] ^= (x[0]+x
x[13] ^= (x[5]+
x[2] ^= (x[10]+
x[7] ^= (x[15]+
```

Total: 8 modificat

Modify other columns:

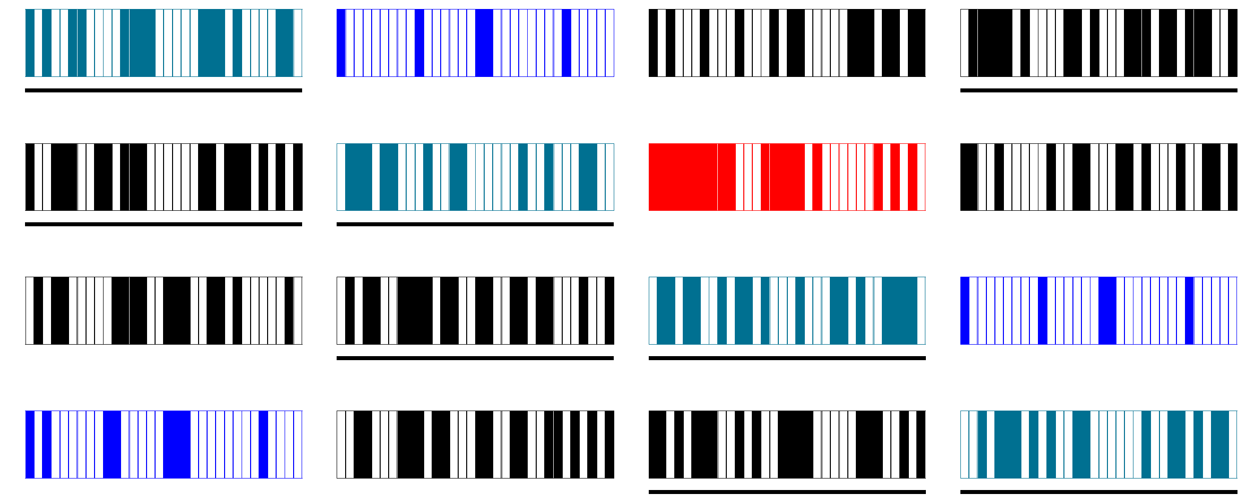


Columns wrap around
from bottom to top.

```
x[4] ^= (x[12]+x[0]) <<< 7
x[14] ^= (x[6]+x[10]) <<< 7
x[3] ^= (x[11]+x[15]) <<< 7
```

Total: 4 modifications.

Modify each column again:

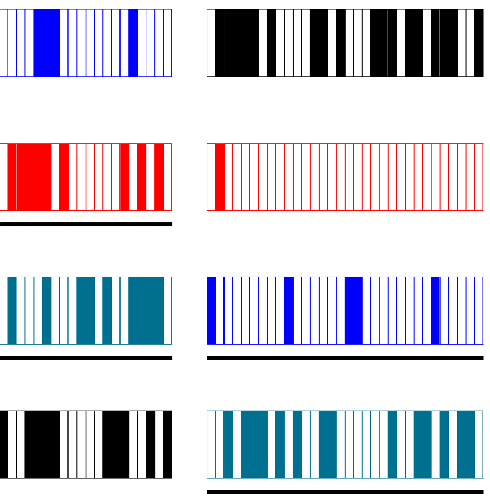


This time rotate by 9 bits.

```
x[8] ^= (x[0]+x[4]) <<< 9
x[13] ^= (x[5]+x[9]) <<< 9
x[2] ^= (x[10]+x[14]) <<< 9
x[7] ^= (x[15]+x[3]) <<< 9
```

Total: 8 modifications.

mns:



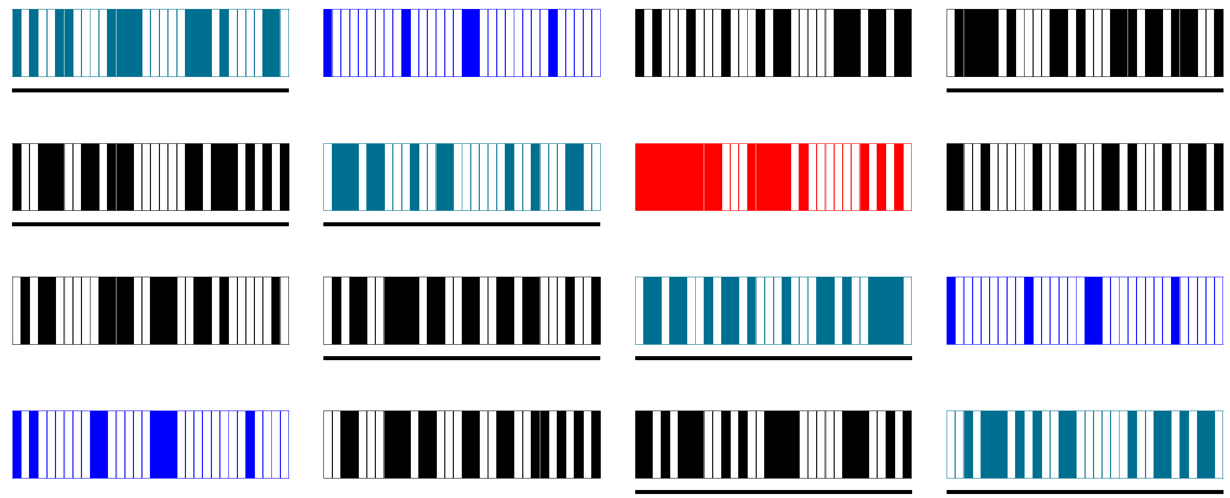
und

p.

```
x[0]) <<< 7
x[10]) <<< 7
x[15]) <<< 7
```

tions.

Modify each column again:

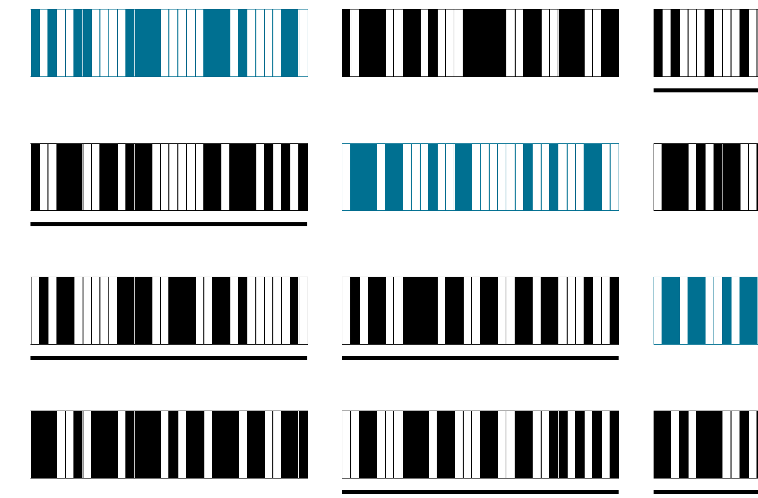


This time rotate by 9 bits.

```
x[8] ^= (x[0]+x[4]) <<< 9
x[13] ^= (x[5]+x[9]) <<< 9
x[2] ^= (x[10]+x[14]) <<< 9
x[7] ^= (x[15]+x[3]) <<< 9
```

Total: 8 modifications.

Modify each colun

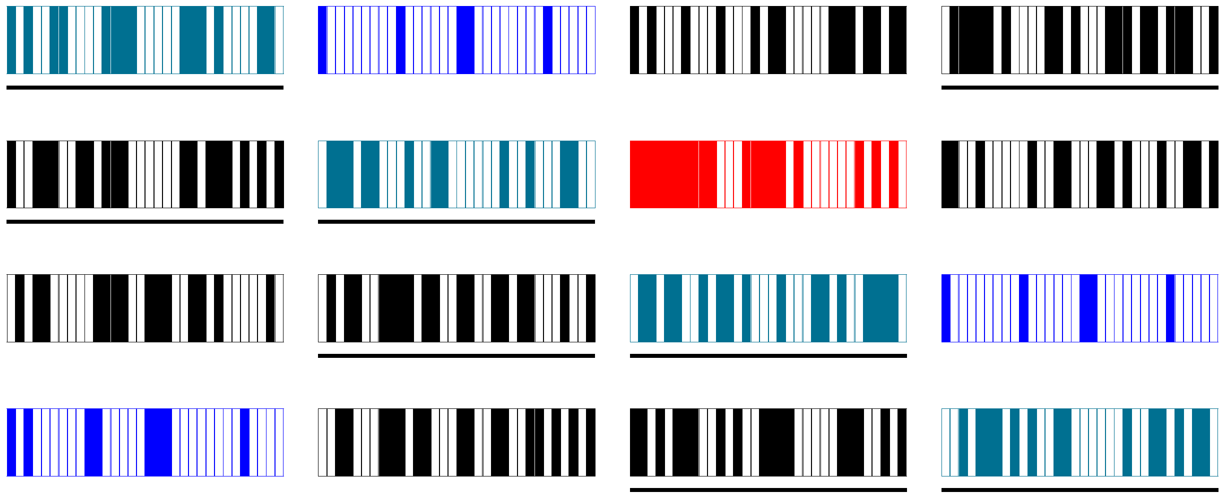


This time rotate b

```
x[12] ^= (x[4]+
x[1] ^= (x[9]+x
x[6] ^= (x[14]+
x[11] ^= (x[3]+
```

Total: 12 modifica

Modify each column again:

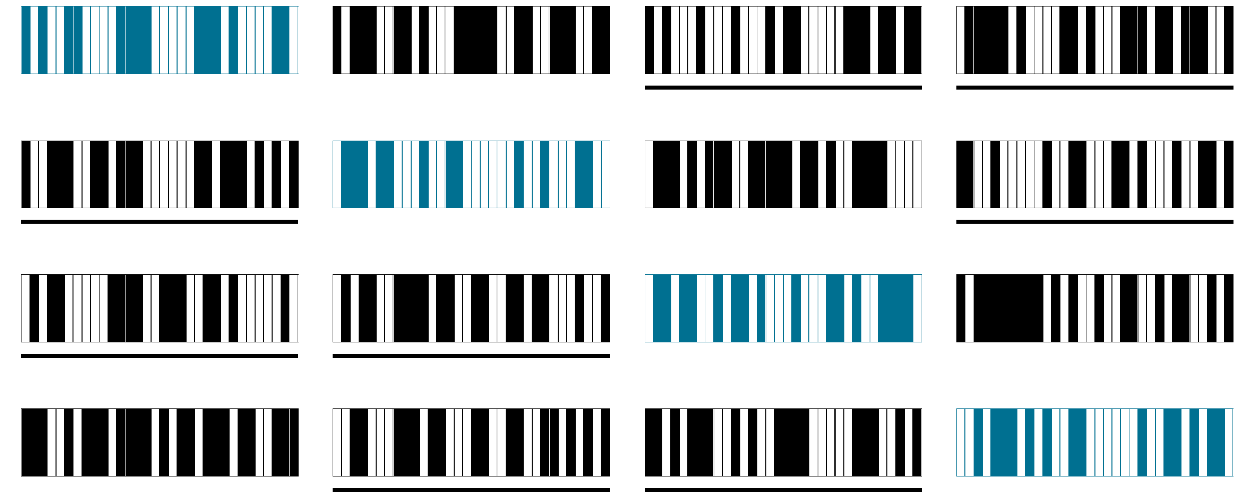


This time rotate by 9 bits.

```
x[8] ^= (x[0]+x[4]) <<< 9
x[13] ^= (x[5]+x[9]) <<< 9
x[2] ^= (x[10]+x[14]) <<< 9
x[7] ^= (x[15]+x[3]) <<< 9
```

Total: 8 modifications.

Modify each column again:

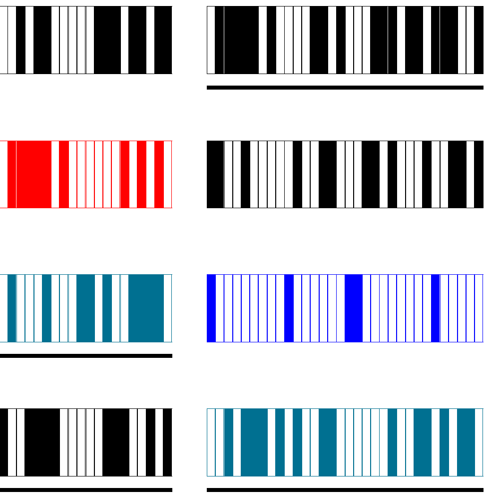


This time rotate by 13 bits.

```
x[12] ^= (x[4]+x[8]) <<< 13
x[1] ^= (x[9]+x[13]) <<< 13
x[6] ^= (x[14]+x[2]) <<< 13
x[11] ^= (x[3]+x[7]) <<< 13
```

Total: 12 modifications.

mn again:

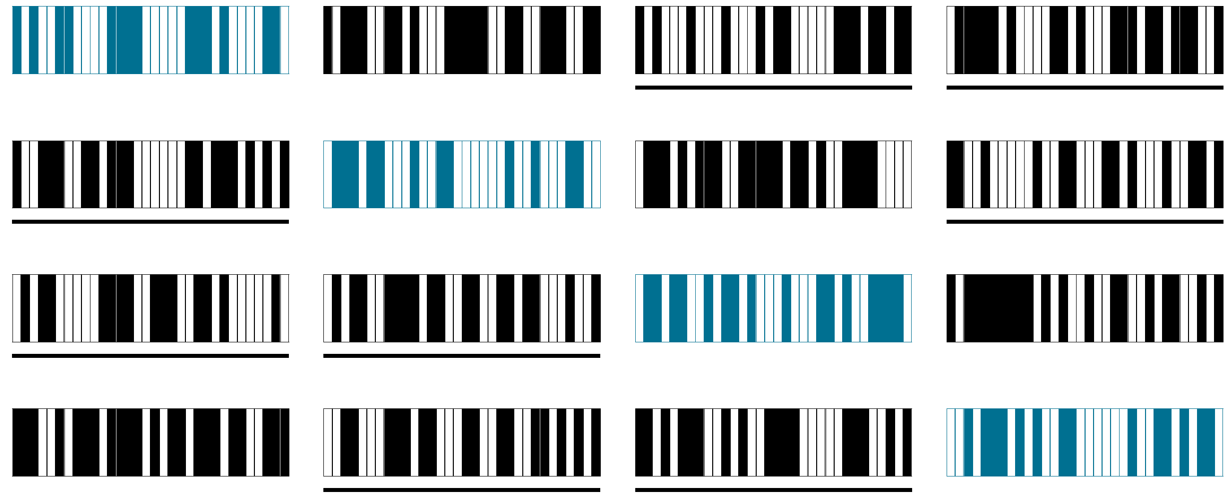


y 9 bits.

```
[4]) <<< 9
x[9]) <<< 9
x[14]) <<< 9
x[3]) <<< 9
```

tions.

Modify each column again:

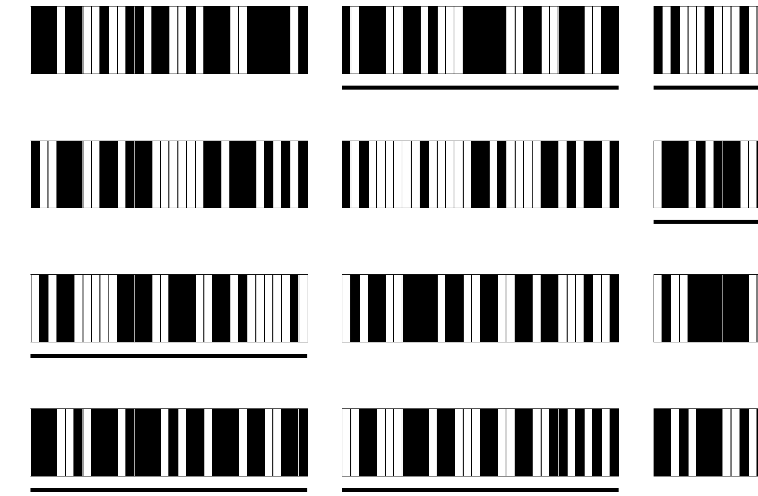


This time rotate by 13 bits.

```
x[12] ^= (x[4]+x[8]) <<< 13
x[1] ^= (x[9]+x[13]) <<< 13
x[6] ^= (x[14]+x[2]) <<< 13
x[11] ^= (x[3]+x[7]) <<< 13
```

Total: 12 modifications.

Modify each colum

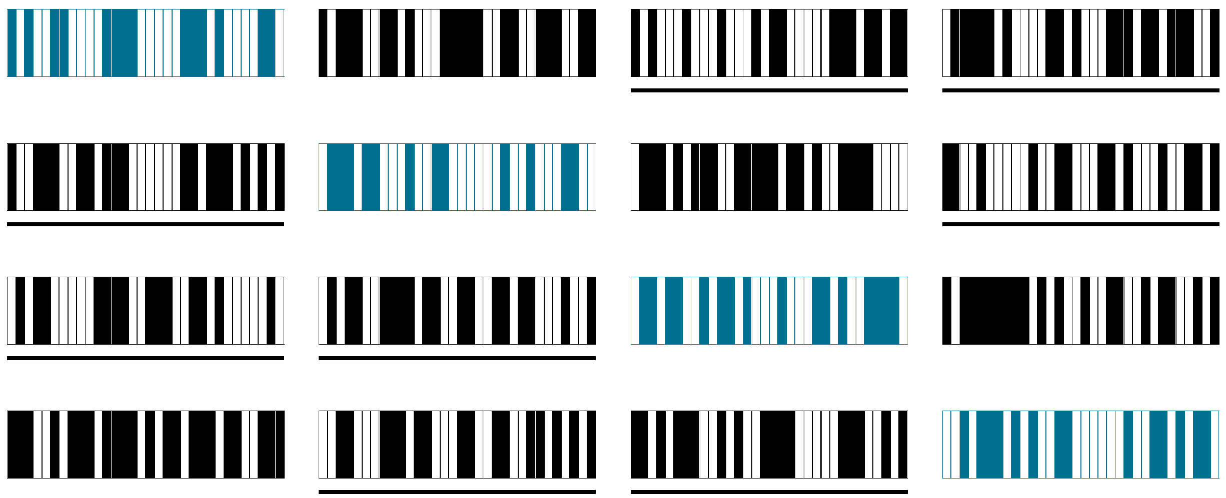


This time rotate b

```
x[0] ^= (x[8]+x
x[5] ^= (x[13]+
x[10] ^= (x[2]+
x[15] ^= (x[7]+
```

Total: 16 modifica

Modify each column again:

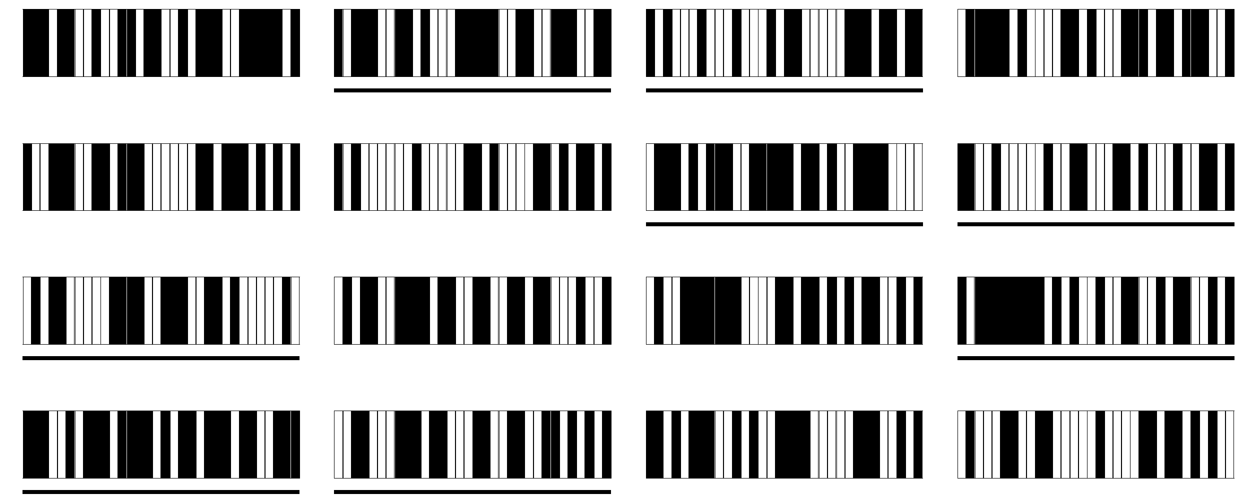


This time rotate by 13 bits.

```
x[12] ^= (x[4]+x[8]) <<< 13
x[1] ^= (x[9]+x[13]) <<< 13
x[6] ^= (x[14]+x[2]) <<< 13
x[11] ^= (x[3]+x[7]) <<< 13
```

Total: 12 modifications.

Modify each column again:

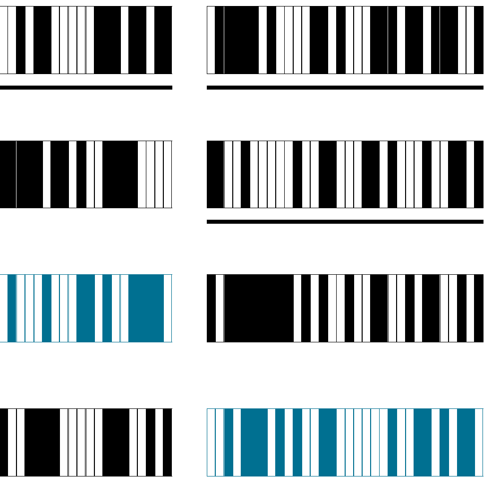


This time rotate by 18 bits.

```
x[0] ^= (x[8]+x[12]) <<< 18
x[5] ^= (x[13]+x[1]) <<< 18
x[10] ^= (x[2]+x[6]) <<< 18
x[15] ^= (x[7]+x[11]) <<< 18
```

Total: 16 modifications.

mn again:



y 13 bits.

```
x[8]) <<< 13
[13]) <<< 13
x[2]) <<< 13
x[7]) <<< 13
```

ations.

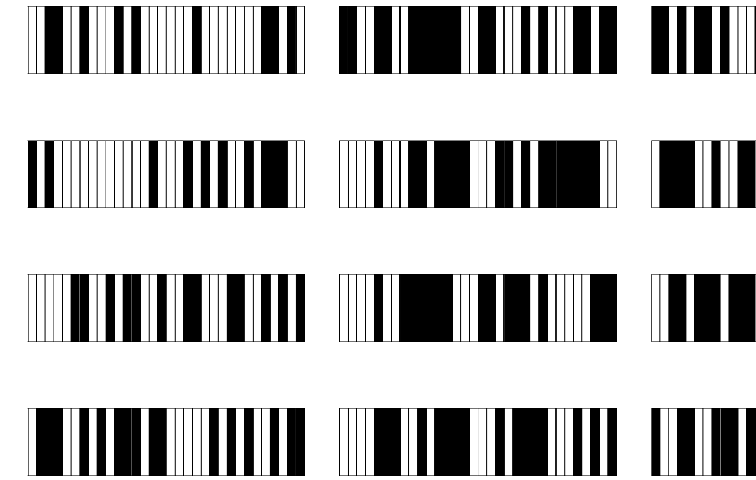
Modify each column again:

This time rotate by 18 bits.

```
x[0] ^= (x[8]+x[12]) <<< 18
x[5] ^= (x[13]+x[1]) <<< 18
x[10] ^= (x[2]+x[6]) <<< 18
x[15] ^= (x[7]+x[11]) <<< 18
```

Total: 16 modifications.

Modify rows by 7,



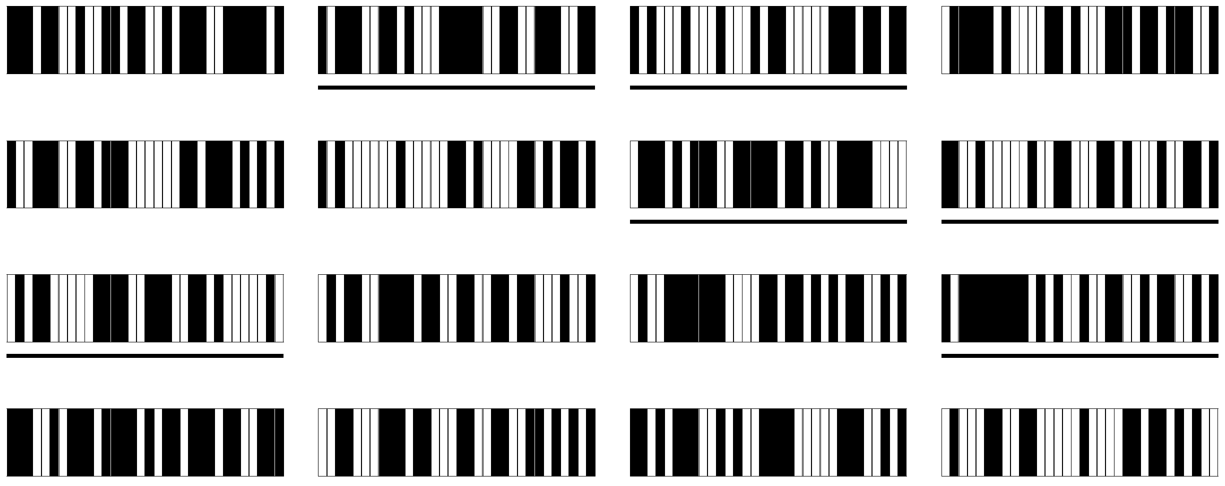
Now every word

has been modified

Total: 32 modifica

That's 2 rounds of

Modify each column again:

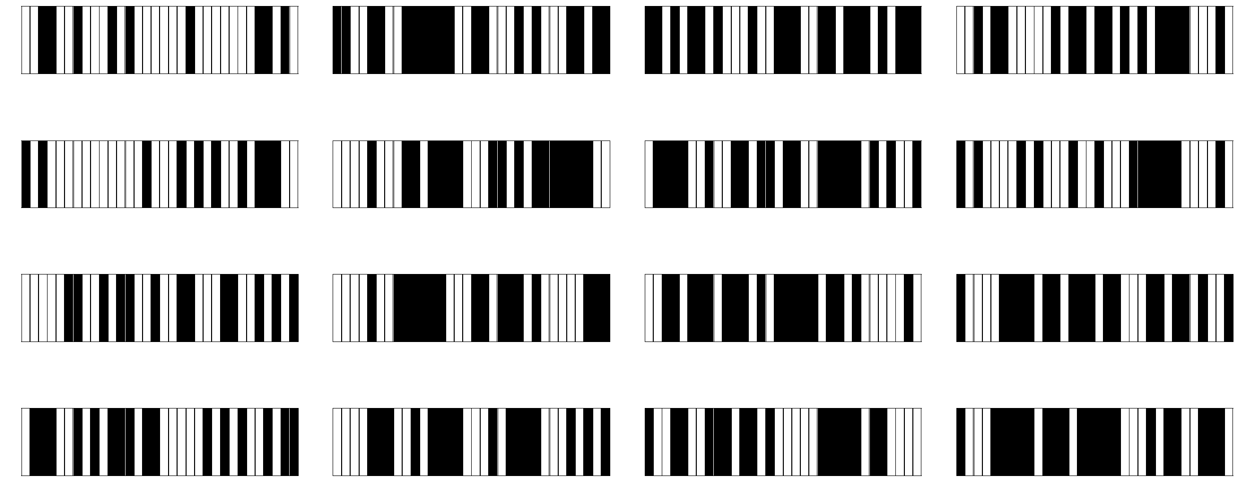


This time rotate by 18 bits.

```
x[0] ^= (x[8]+x[12]) <<< 18
x[5] ^= (x[13]+x[1]) <<< 18
x[10] ^= (x[2]+x[6]) <<< 18
x[15] ^= (x[7]+x[11]) <<< 18
```

Total: 16 modifications.

Modify rows by $7, 9, 13, 18$:

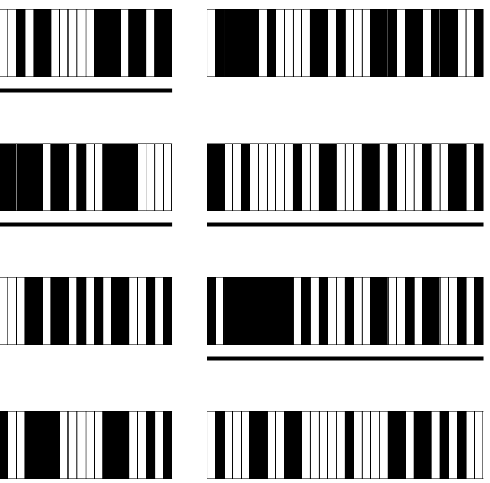


Now every word
has been modified twice.

Total: 32 modifications.

That’s 2 rounds of Salsa20.

nn again:

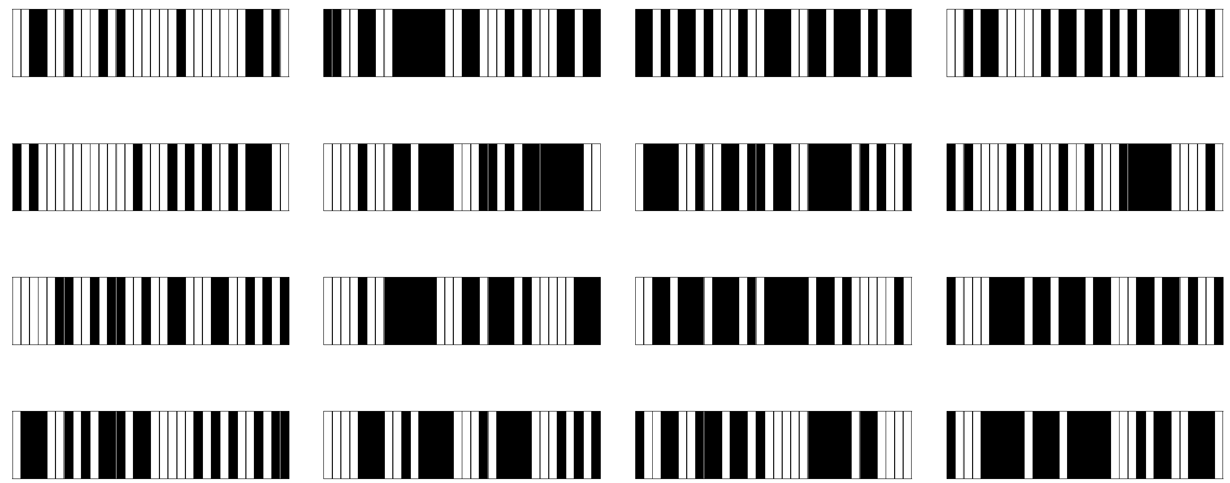


y 18 bits.

```
[12]) <<< 18
x[1]) <<< 18
x[6]) <<< 18
x[11]) <<< 18
```

ations.

Modify rows by 7, 9, 13, 18:

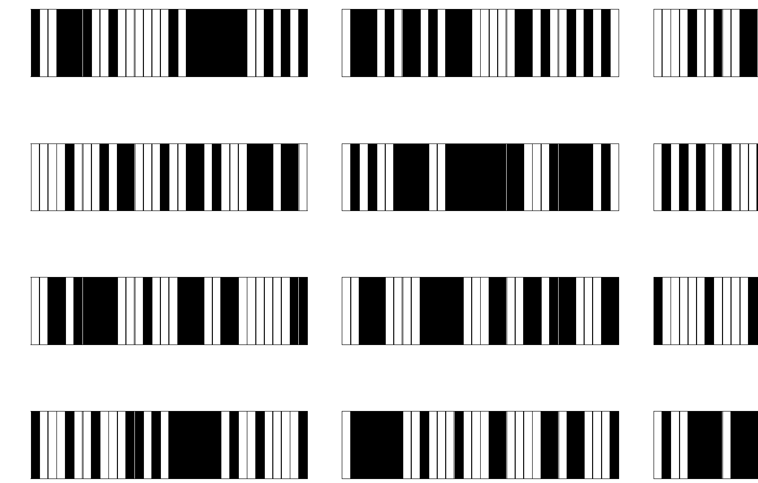


Now every word
has been modified twice.

Total: 32 modifications.

That's 2 rounds of Salsa20.

Repeat column mo

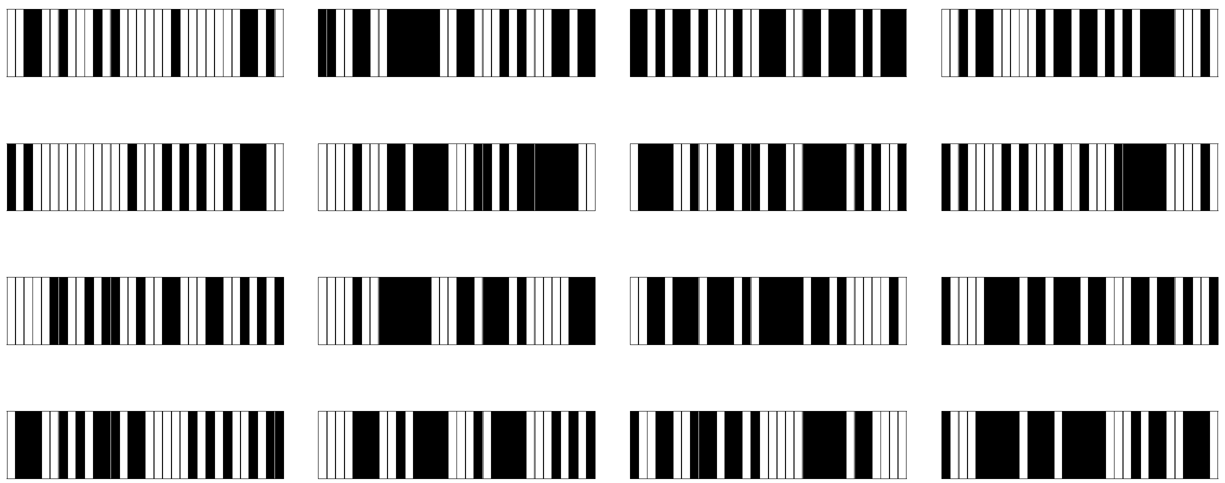


Now every word
has been modified

Total: 48 modifica

That's 3 rounds of

Modify rows by $7, 9, 13, 18$:



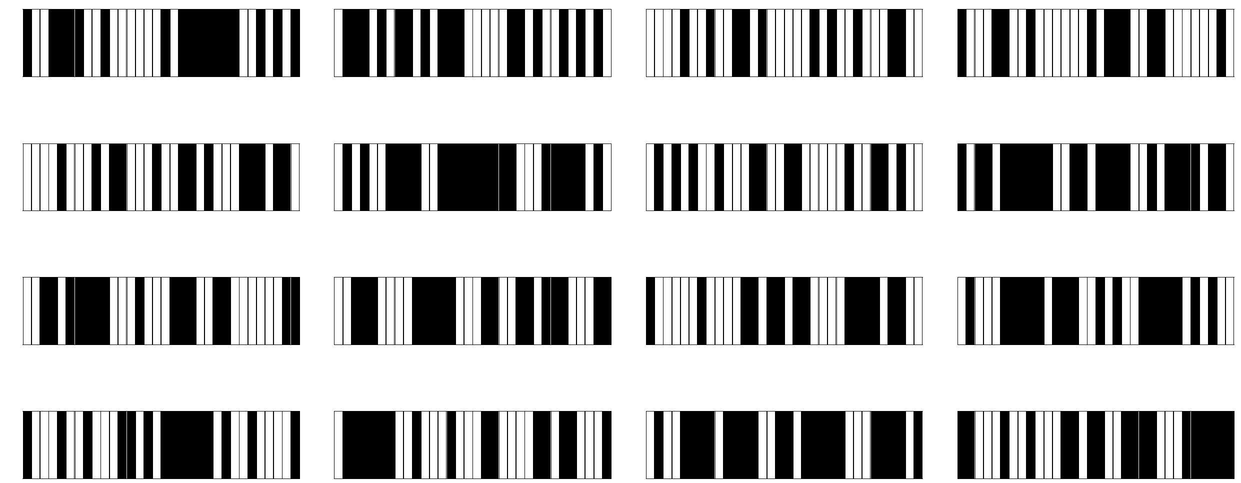
Now every word

has been modified twice.

Total: 32 modifications.

That's 2 rounds of Salsa20.

Repeat column modifications:



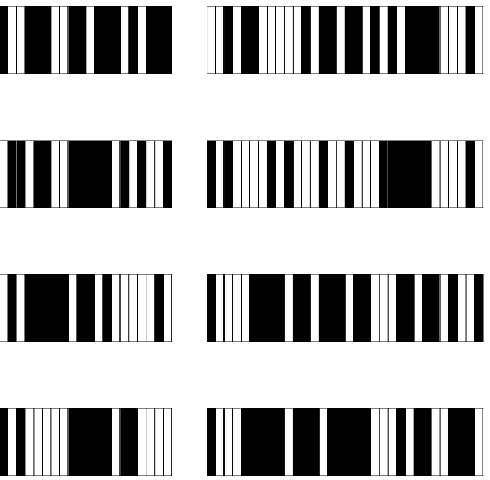
Now every word

has been modified 3 times.

Total: 48 modifications.

That's 3 rounds of Salsa20.

9, 13, 18:

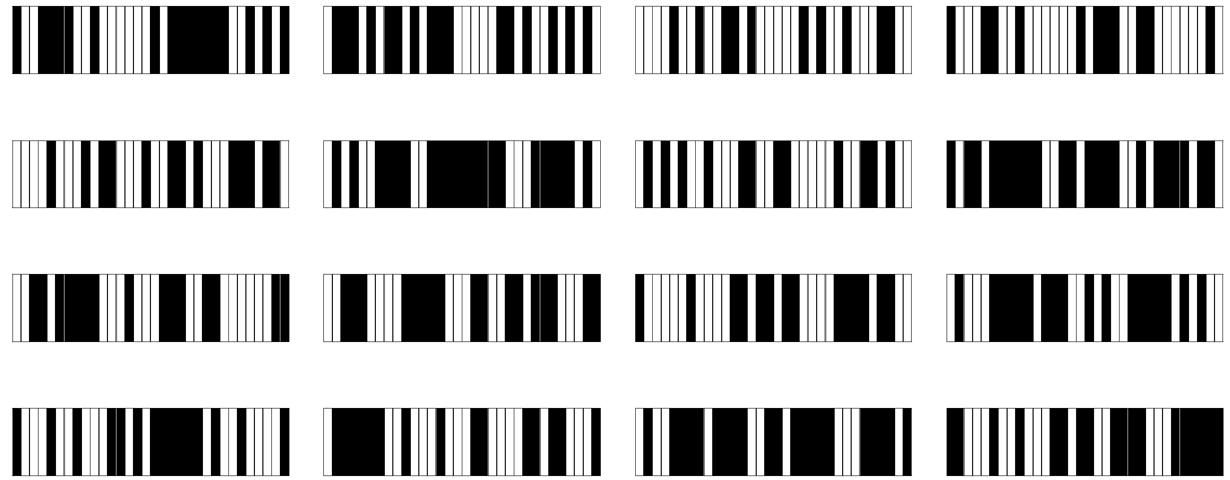


twice.

ations.

f Salsa20.

Repeat column modifications:

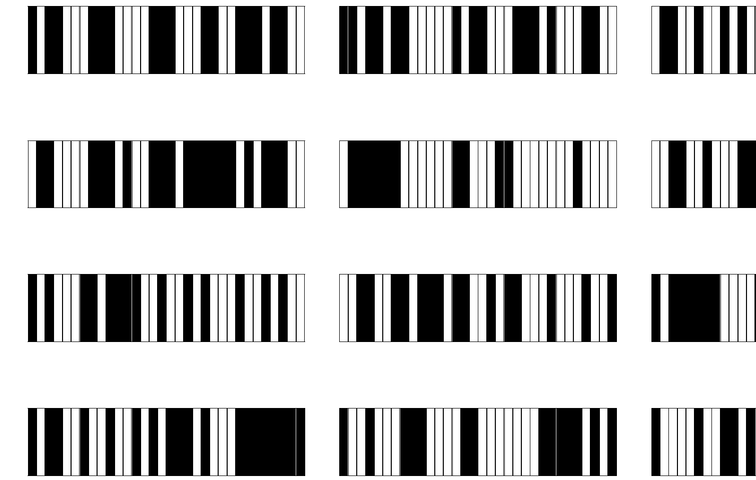


Now every word
has been modified 3 times.

Total: 48 modifications.

That's 3 rounds of Salsa20.

Repeat row modifi

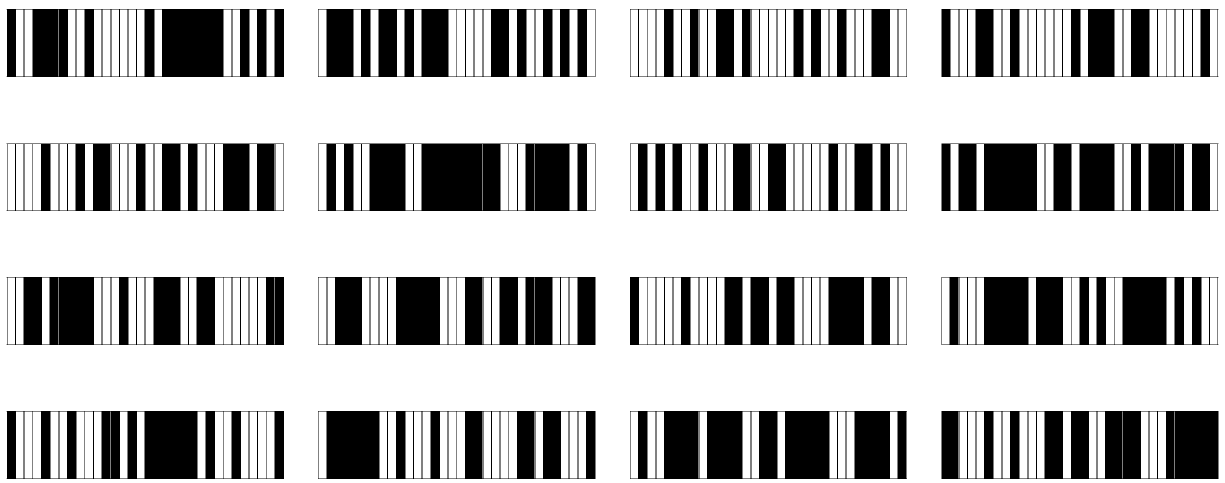


Now every word
has been modified

Total: 64 modifica

That's 4 rounds of

Repeat column modifications:



Now every word
has been modified 3 times.

Total: 48 modifications.

That's 3 rounds of Salsa20.

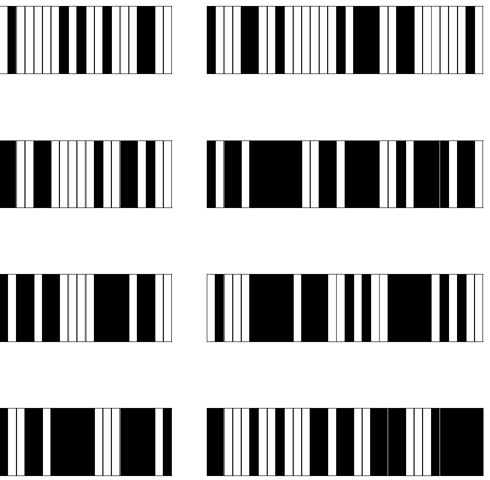
Repeat row modifications:

Now every word
has been modified 4 times.

Total: 64 modifications.

That's 4 rounds of Salsa20.

odifications:

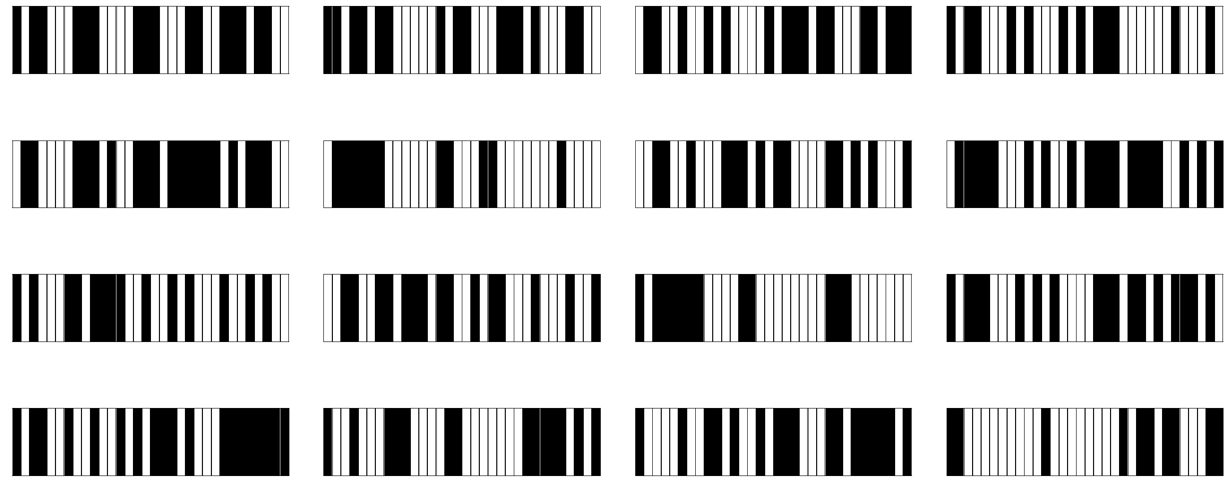


3 times.

ations.

f Salsa20.

Repeat row modifications:



Now every word
has been modified 4 times.

Total: 64 modifications.

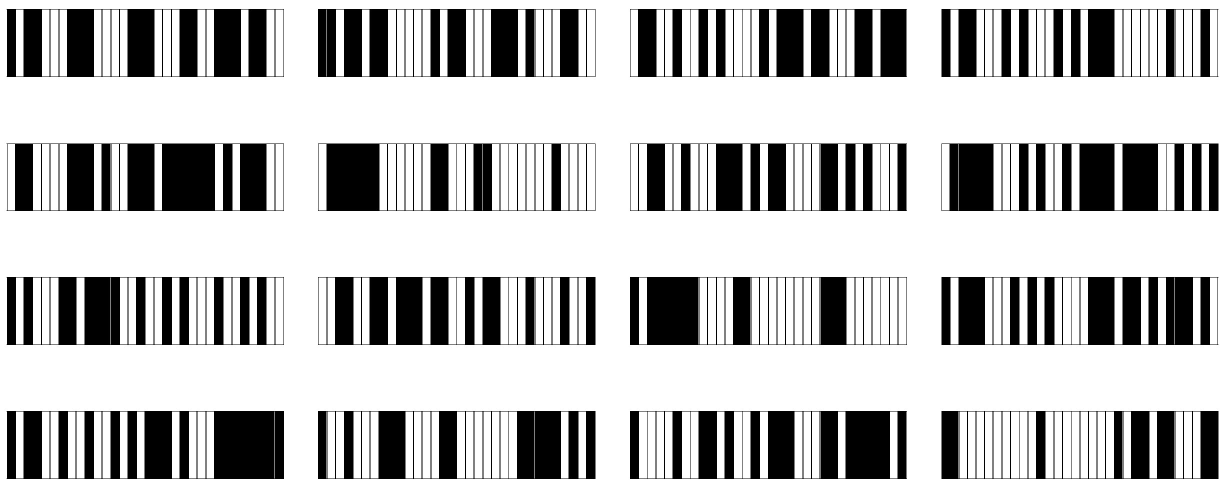
That's 4 rounds of Salsa20.

Continue for 20 ro
columns, rows, col
columns, rows, col
columns, rows, col
columns, rows, col
columns, rows, col

First block of Sals
final array plus orig
`x[0]+z[0],...,x`

For subsequent blo
with block counter
Parallelizable. Ver

Repeat row modifications:



Now every word
has been modified 4 times.

Total: 64 modifications.

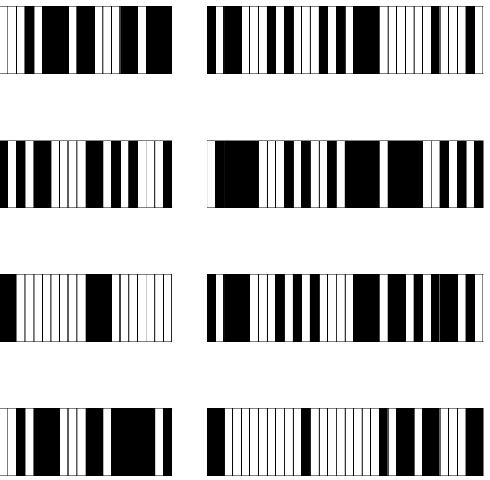
That's 4 rounds of Salsa20.

Continue for 20 rounds total:
columns, rows, columns, rows,
columns, rows, columns, rows,
columns, rows, columns, rows,
columns, rows, columns, rows,
columns, rows, columns, rows.

First block of Salsa20 output is
final array plus original array:
`x[0]+z[0],...,x[15]+z[15].`

For subsequent blocks: Repeat
with block counter 1, 2, etc.
Parallelizable. Very small state.

fications:



4 times.

ations.

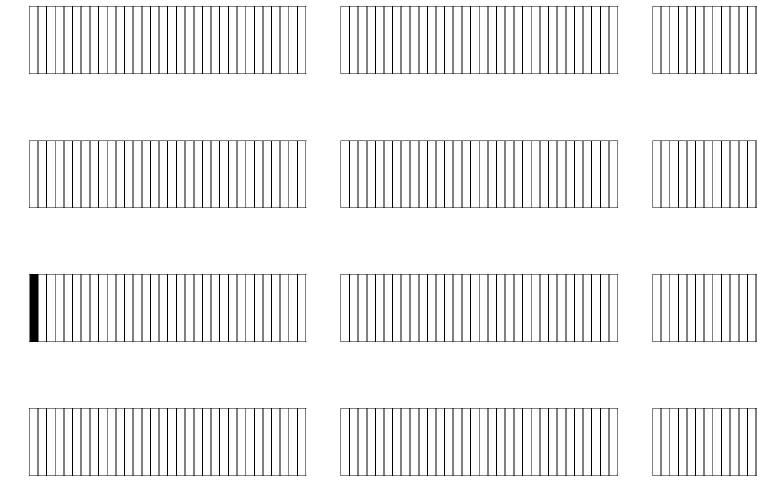
f Salsa20.

Continue for 20 rounds total:
columns, rows, columns, rows,
columns, rows, columns, rows,
columns, rows, columns, rows,
columns, rows, columns, rows,
columns, rows, columns, rows.

First block of Salsa20 output is final array plus original array: `x[0]+z[0], ..., x[15]+z[15]`.

For subsequent blocks: Repeat with block counter 1, 2, etc. Parallelizable. Very small state.

Change in starting



Let's watch how th
affects subsequent

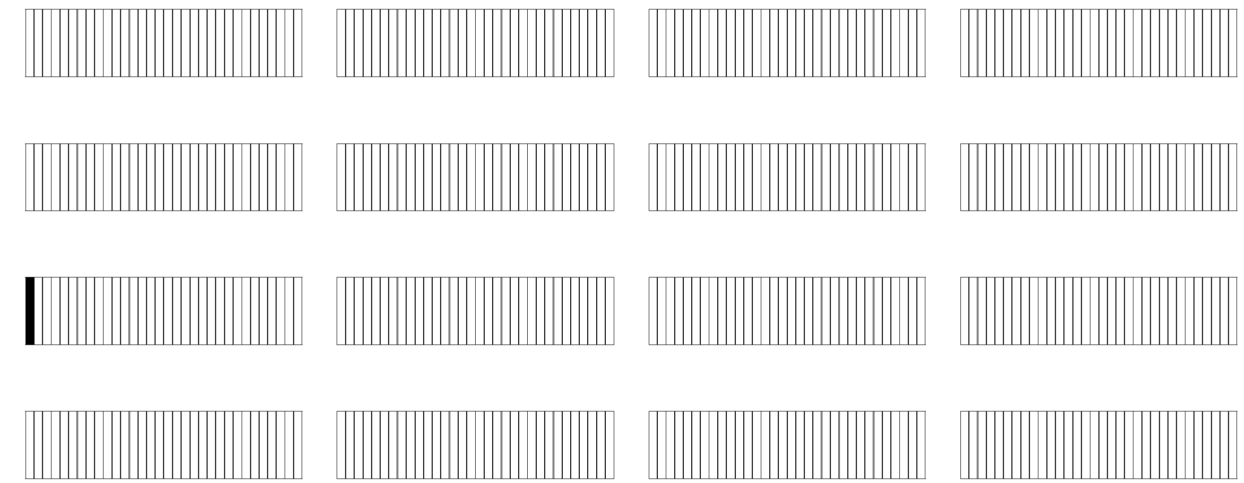
Changes shown he

Continue for 20 rounds total:
columns, rows, columns, rows,
columns, rows, columns, rows,
columns, rows, columns, rows,
columns, rows, columns, rows,
columns, rows, columns, rows.

First block of Salsa20 output is final array plus original array: `x[0]+z[0]`, . . . , `x[15]+z[15]`.

For subsequent blocks: Repeat with block counter 1, 2, etc. Parallelizable. Very small state.

Change in starting array for block 1:



Let's watch how this change affects subsequent rounds.

Changes shown here by xor.

ounds total:

umns, rows,

umns, rows,

umns, rows,

umns, rows,

umns, rows.

a20 output is

ginal array:

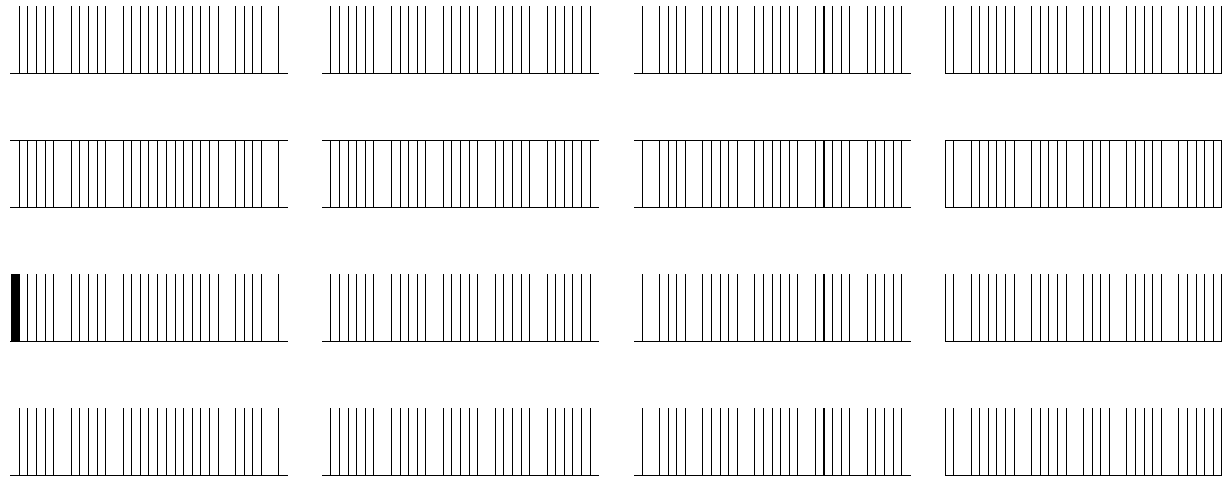
[15]+z[15].

ocks: Repeat

r 1, 2, etc.

y small state.

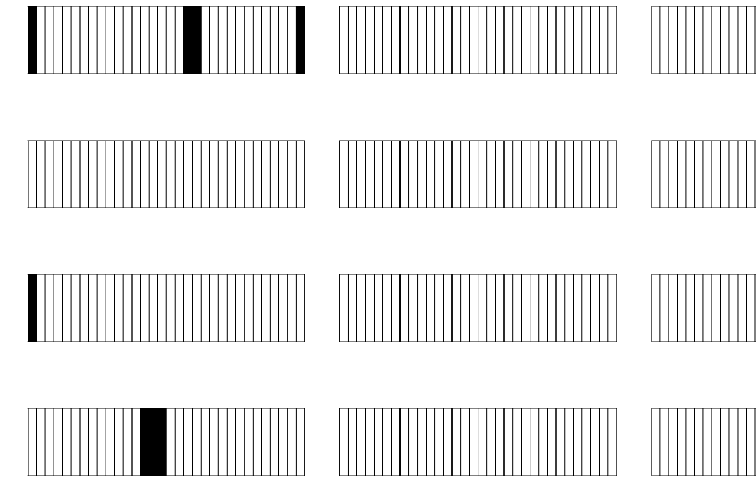
Change in starting array for block 1:



Let's watch how this change affects subsequent rounds.

Changes shown here by xor.

Change after one r



Difference has pro

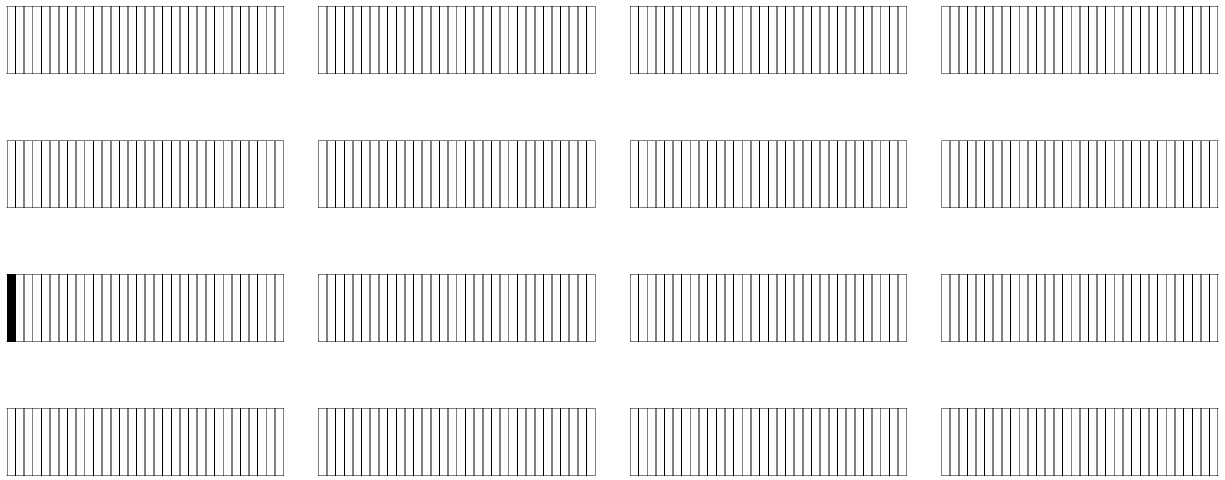
to two other entrie

in the same colum

Depends on a few

but still highly pre

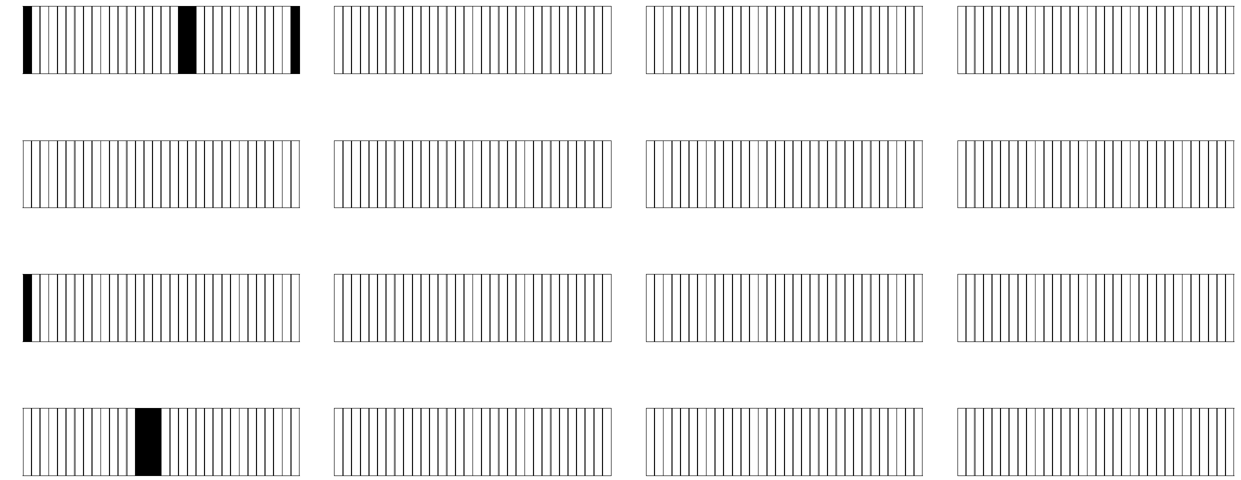
Change in starting array for block 1:



Let's watch how this change affects subsequent rounds.

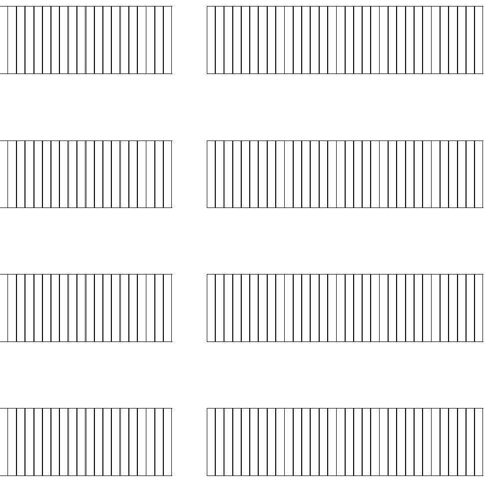
Changes shown here by xor.

Change after one round:



Difference has propagated
to two other entries
in the same column.
Depends on a few carries,
but still highly predictable.

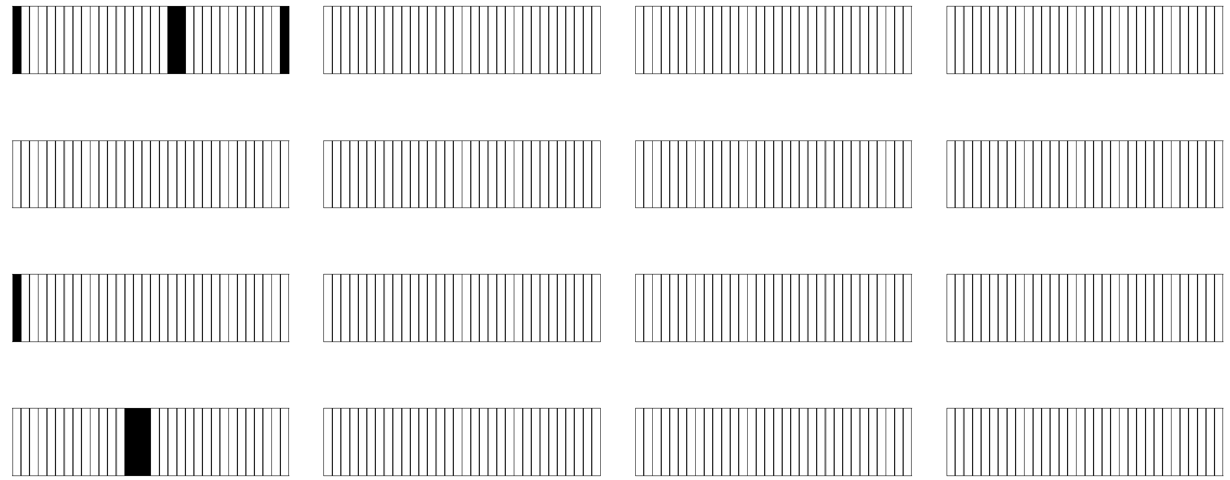
g array for block 1:



his change
rounds.

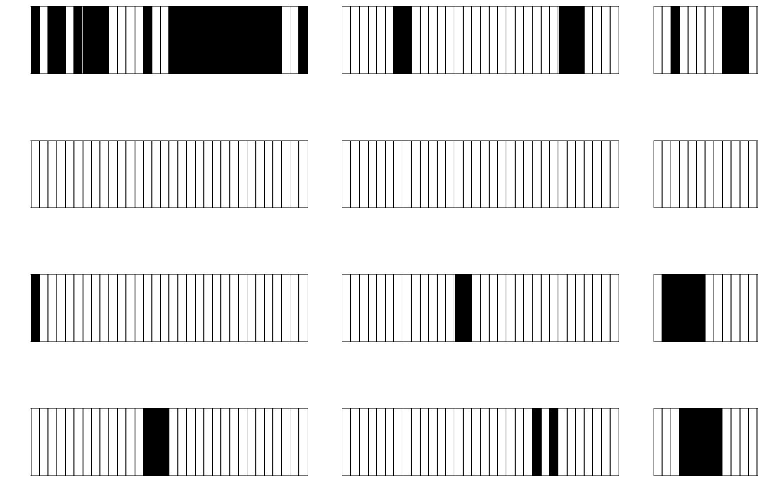
ere by xor.

Change after one round:



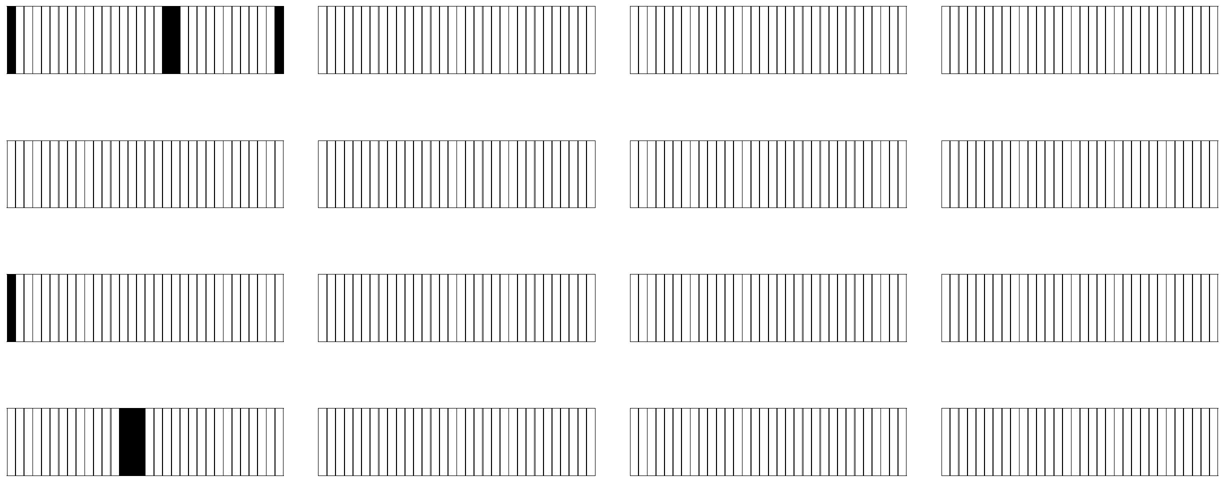
Difference has propagated
to two other entries
in the same column.
Depends on a few carries,
but still highly predictable.

Change after two r



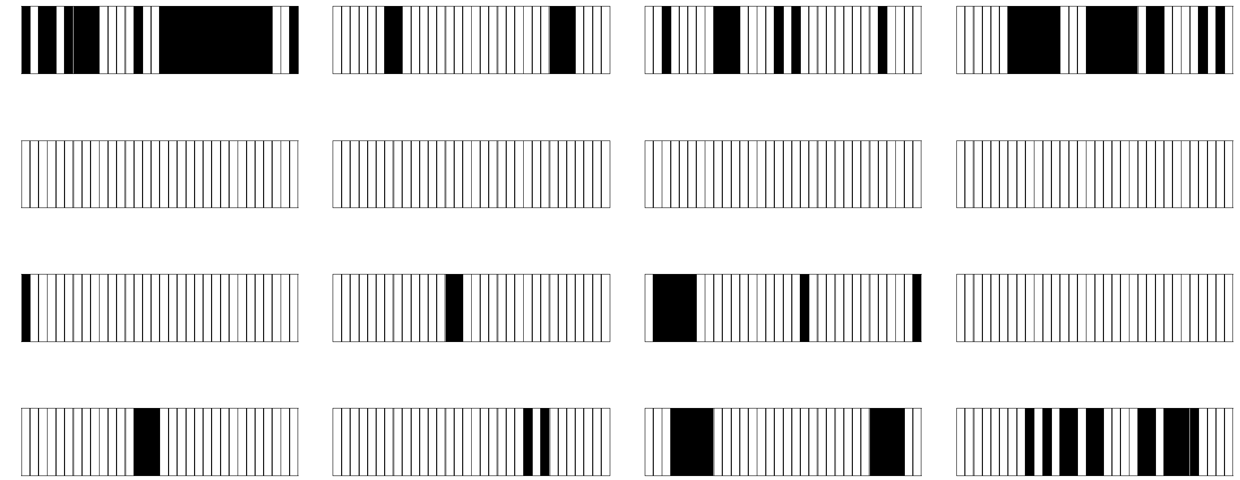
Difference has pro
across columns.

Change after one round:



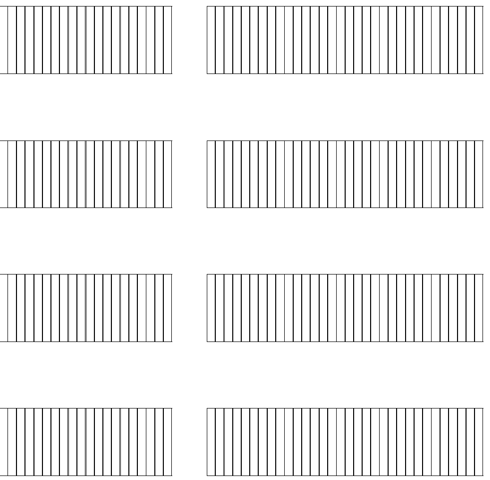
Difference has propagated
to two other entries
in the same column.
Depends on a few carries,
but still highly predictable.

Change after two rounds:



Difference has propagated
across columns.

round:



pagated

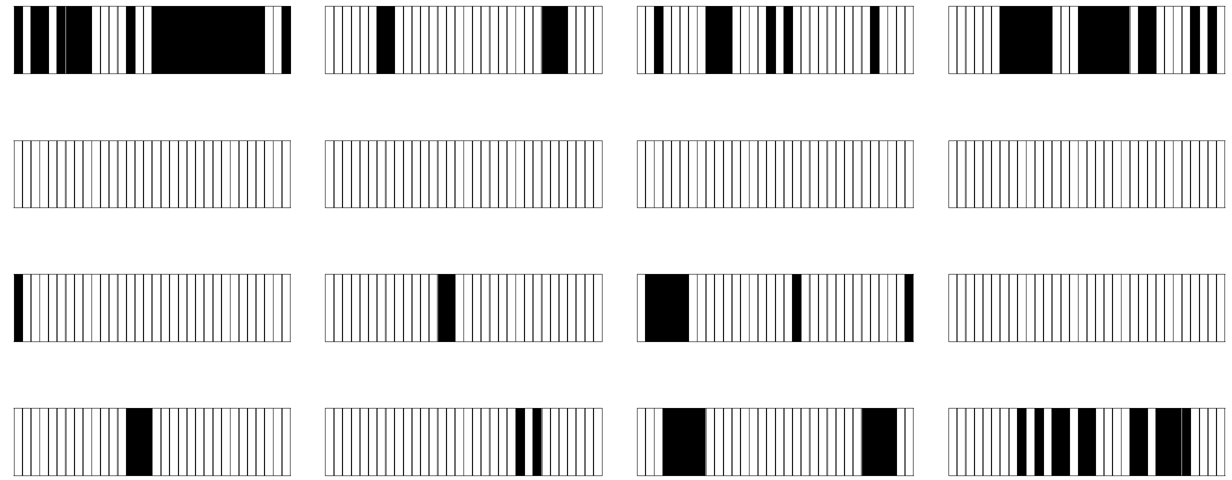
es

n.

carries,

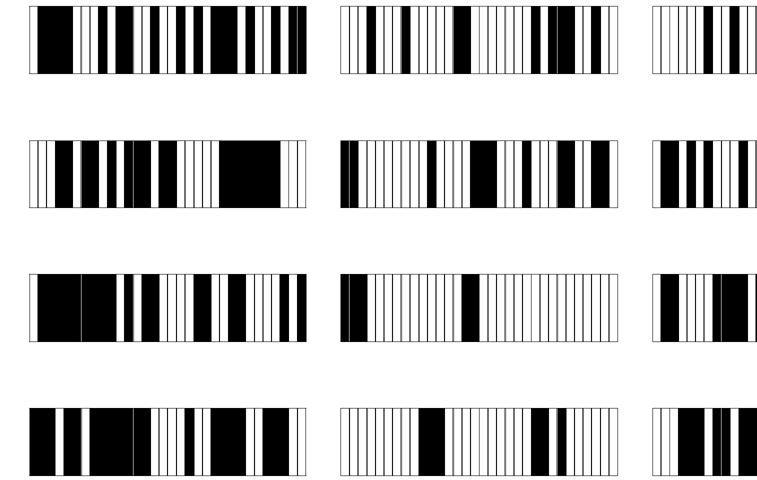
dictable.

Change after two rounds:



Difference has propagated across columns.

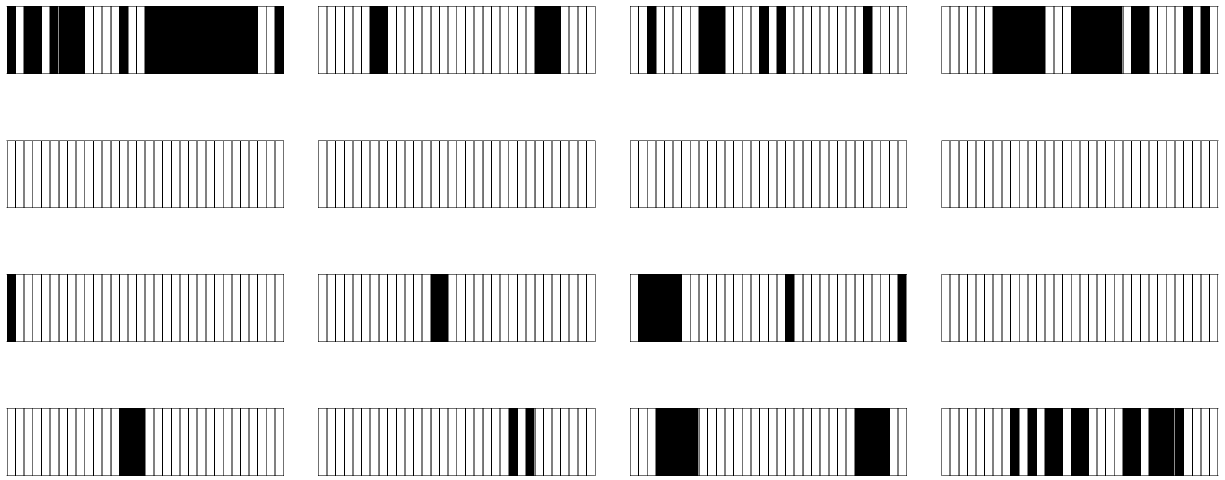
Change after three



Every word has be

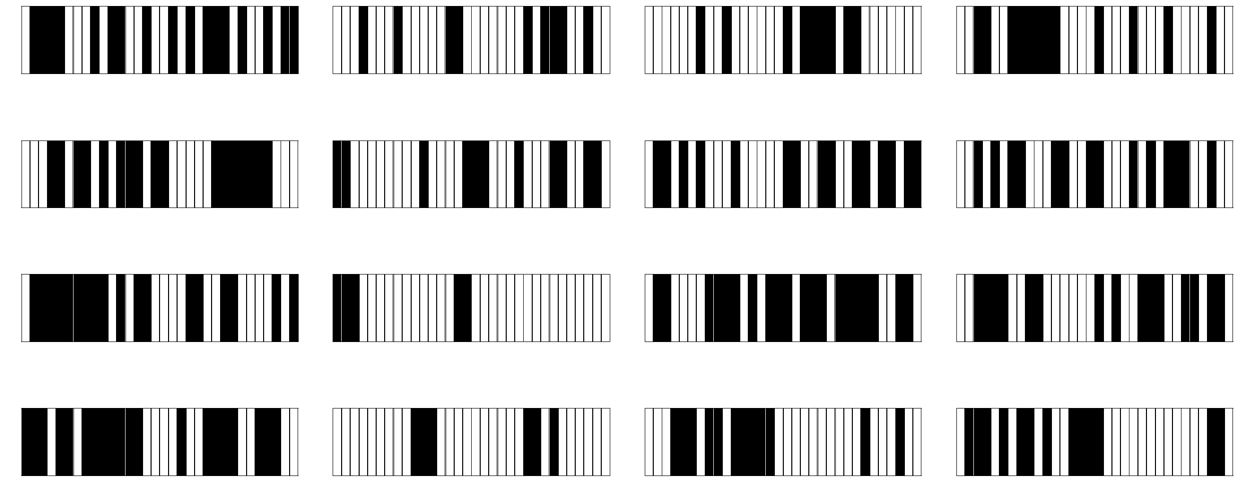
A substantial fract are now active.

Change after two rounds:



Difference has propagated across columns.

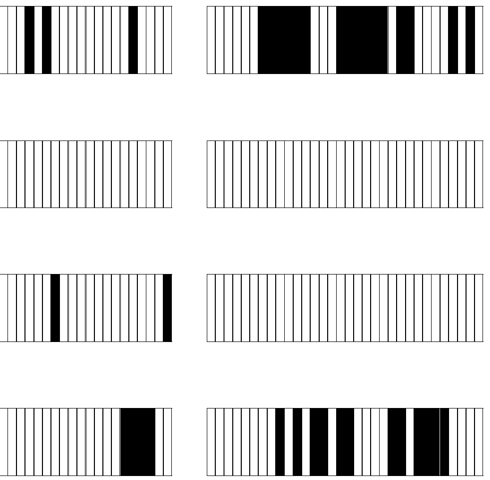
Change after three rounds:



Every word has been affected.

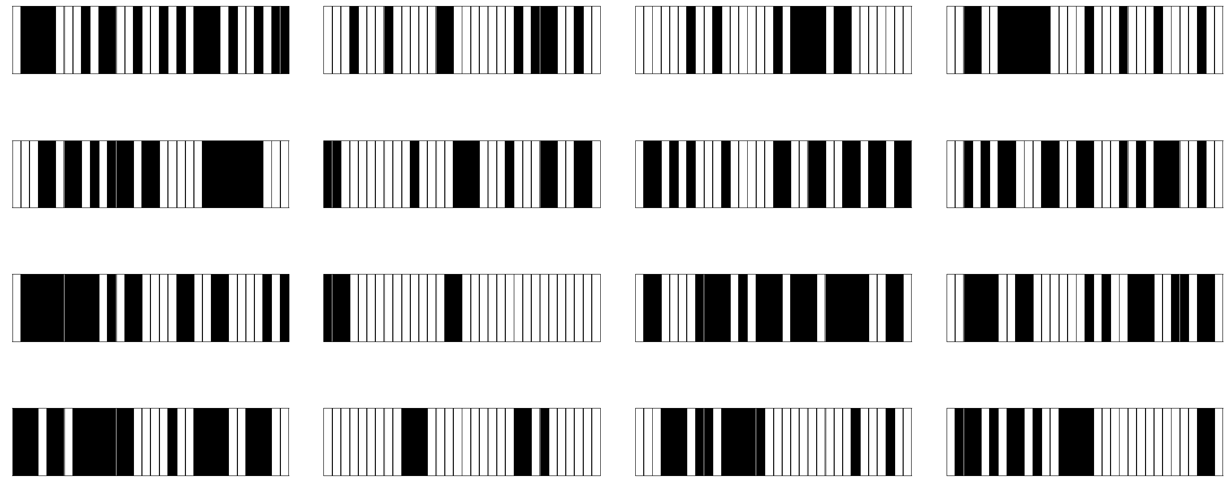
A substantial fraction of bits are now active.

rounds:



pagated

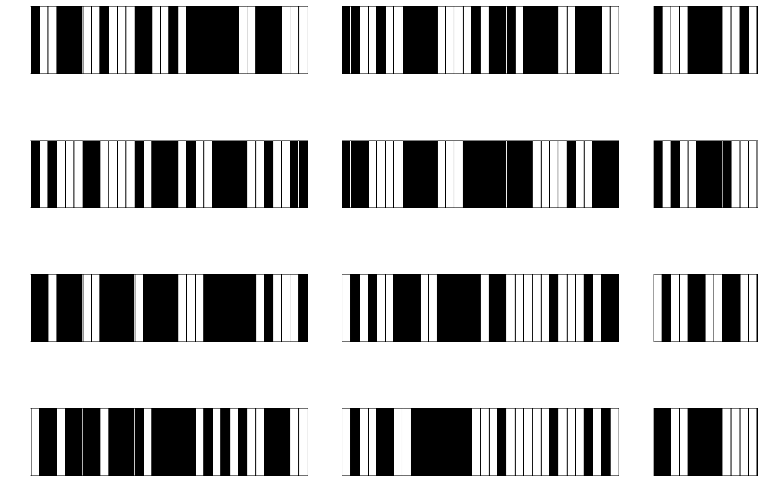
Change after three rounds:



Every word has been affected.

A substantial fraction of bits are now active.

Change after four



Hundreds of active

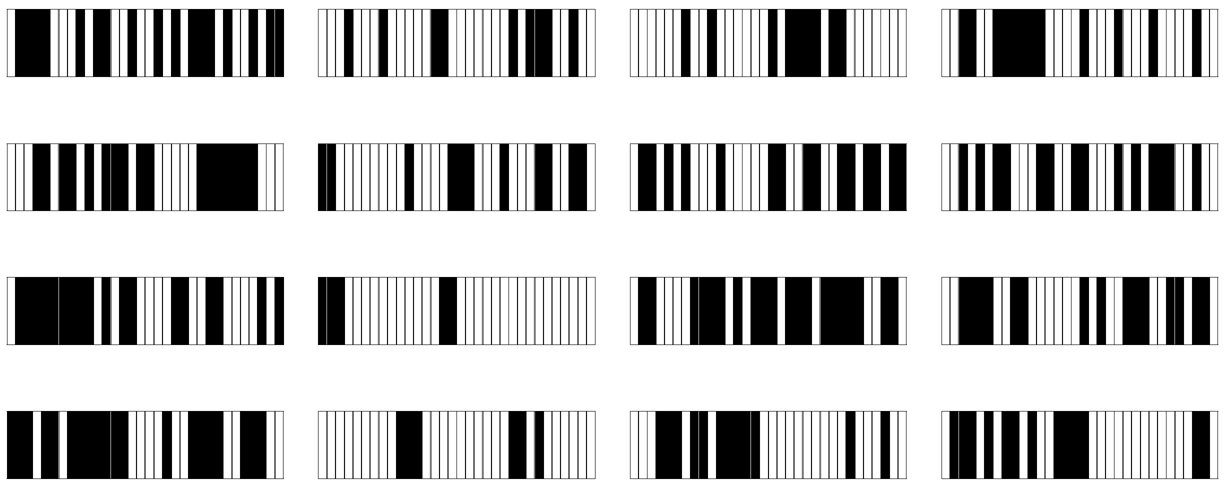
in every subsequen

Total $>$ 4000 activ

interacting with ca

in a random-lookin

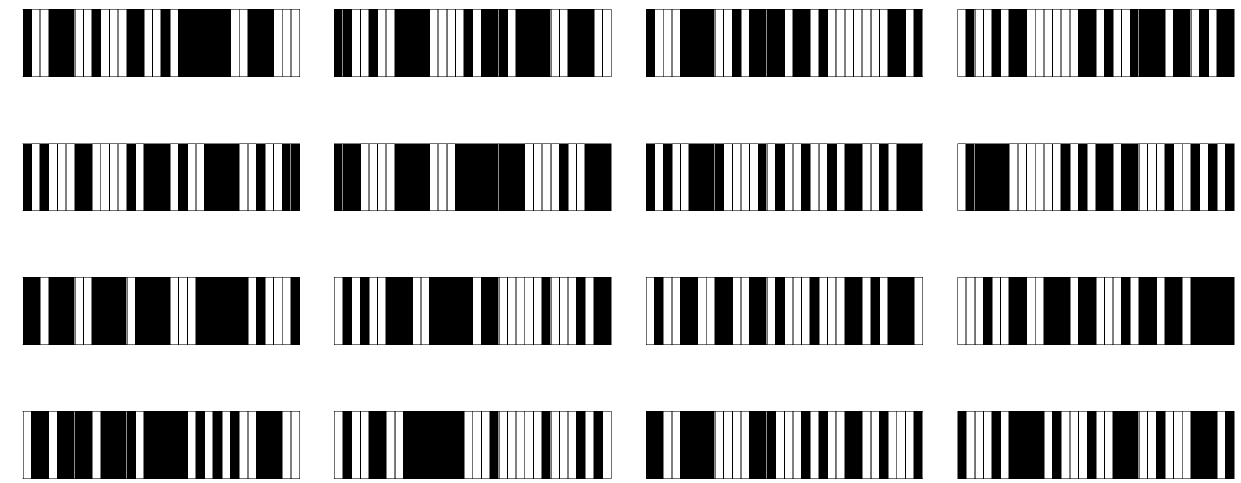
Change after three rounds:



Every word has been affected.

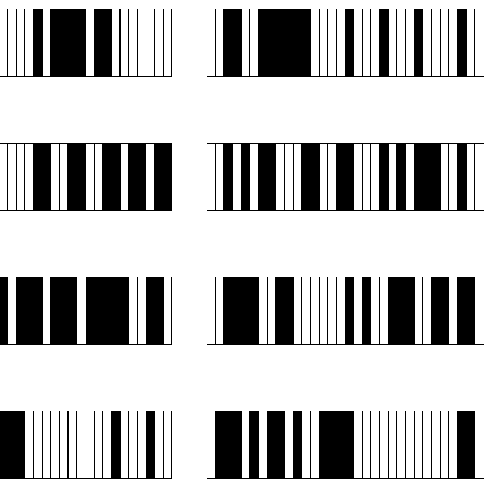
A substantial fraction of bits are now active.

Change after four rounds:



Hundreds of active bits in every subsequent round.
Total $>$ 4000 active bits interacting with carries in a random-looking way.

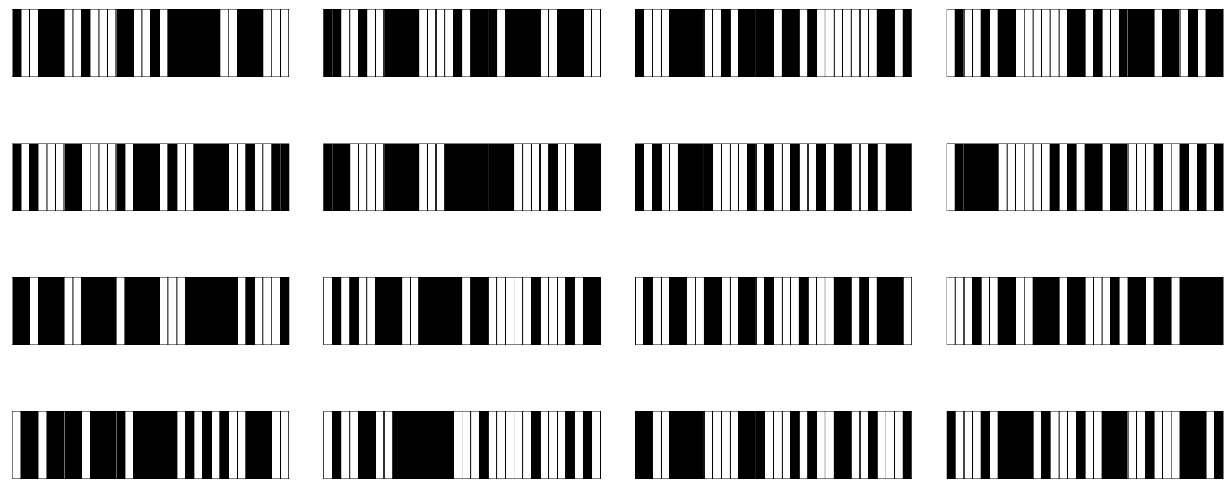
e rounds:



een affected.

tion of bits

Change after four rounds:

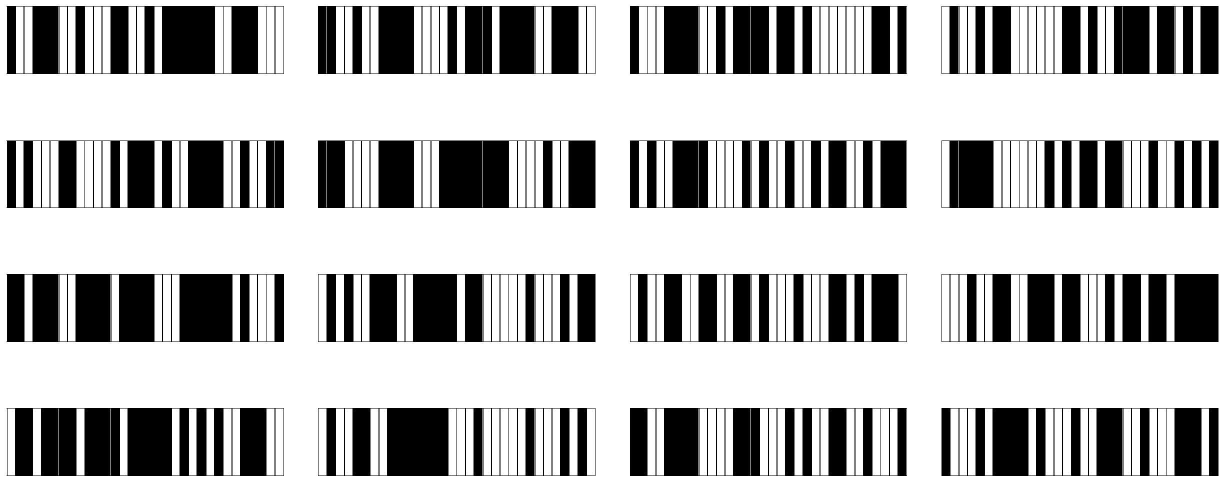


Hundreds of active bits
in every subsequent round.
Total $>$ 4000 active bits
interacting with carries
in a random-looking way.

Surprise: Salsa20

My current public-
26.75 Athlon cycle
37.5 Pentium III c
48 Pentium 4 f12
33.75 Pentium M
24.5 PowerPC 741
33 PowerPC RS64
40.5 UltraSPARC
41 UltraSPARC III

Change after four rounds:



Hundreds of active bits
in every subsequent round.
Total $>$ 4000 active bits
interacting with carries
in a random-looking way.

Surprise: Salsa20 is fast!

My current public-domain software:

26.75 Athlon cycles/round.

37.5 Pentium III cycles/round.

48 Pentium 4 f12 cycles/round.

33.75 Pentium M cycles/round.

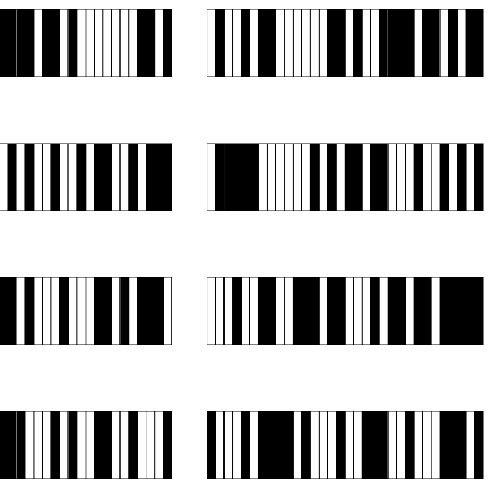
24.5 PowerPC 7410 cycles/round.

33 PowerPC RS64 IV cycles/round.

40.5 UltraSPARC II cycles/round.

41 UltraSPARC III cycles/round.

rounds:



e bits
nt round.
ve bits
arries
ng way.

Surprise: Salsa20 is fast!

My current public-domain software:
26.75 Athlon cycles/round.
37.5 Pentium III cycles/round.
48 Pentium 4 f12 cycles/round.
33.75 Pentium M cycles/round.
24.5 PowerPC 7410 cycles/round.
33 PowerPC RS64 IV cycles/round.
40.5 UltraSPARC II cycles/round.
41 UltraSPARC III cycles/round.

Multiply by 20 for
divide by 64 for cy
. . . but rounds are

I still need to opti
for block counting
combine with Poly
do comprehensive

But it's clear that
will be at least as
sometimes much f
depending on the

Surprise: Salsa20 is fast!

My current public-domain software:
26.75 Athlon cycles/round.
37.5 Pentium III cycles/round.
48 Pentium 4 f12 cycles/round.
33.75 Pentium M cycles/round.
24.5 PowerPC 7410 cycles/round.
33 PowerPC RS64 IV cycles/round.
40.5 UltraSPARC II cycles/round.
41 UltraSPARC III cycles/round.

Multiply by 20 for 20 rounds,
divide by 64 for cycles/byte
. . . but rounds aren't everything.

I still need to optimize code
for block counting, xor, etc.;
combine with Poly1305;
do comprehensive benchmarks.

But it's clear that Salsa20
will be at least as fast as AES,
sometimes much faster,
depending on the CPU.

is fast!

-domain software:
es/round.
ycles/round.
cycles/round.
cycles/round.
.0 cycles/round.
IV cycles/round.
II cycles/round.
I cycles/round.

Multiply by 20 for 20 rounds,
divide by 64 for cycles/byte
. . . but rounds aren't everything.

I still need to optimize code
for block counting, xor, etc.;
combine with Poly1305;
do comprehensive benchmarks.

But it's clear that Salsa20
will be at least as fast as AES,
sometimes much faster,
depending on the CPU.

Here AES has 16-l
slower with 32-byt
Salsa20 has no suc

AES becomes ever
key is not pre-expa
Salsa20 has no pre

AES has serious ti
see `http://cr.yp`
`/papers.html#ca`
for successful AES
Constant-time AES
Salsa20 has no tim

Multiply by 20 for 20 rounds,
divide by 64 for cycles/byte
. . . but rounds aren't everything.

I still need to optimize code
for block counting, xor, etc.;
combine with Poly1305;
do comprehensive benchmarks.

But it's clear that Salsa20
will be at least as fast as AES,
sometimes much faster,
depending on the CPU.

Here AES has 16-byte key;
slower with 32-byte key.
Salsa20 has no such slowdown.

AES becomes even slower if
key is not pre-expanded.
Salsa20 has no precomputation.

AES has serious timing leaks:
see `http://cr.yp.to/papers.html#cachetiming`
for successful AES key extraction.
Constant-time AES is very slow.
Salsa20 has no timing leaks.

20 rounds,
cles/byte
en't everything.

mize code
, xor, etc.;
1305;
benchmarks.

Salsa20
fast as AES,
aster,
CPU.

Here AES has 16-byte key;
slower with 32-byte key.
Salsa20 has no such slowdown.

AES becomes even slower if
key is not pre-expanded.
Salsa20 has no precomputation.

AES has serious timing leaks:
see `http://cr.yp.to/papers.html#cachetiming`
for successful AES key extraction.
Constant-time AES is very slow.
Salsa20 has no timing leaks.

I offer $1000 prize
the public Salsa20
that I consider mo
Awarded at the en

Send URLs of you
`snuffle@box.cr.`

Here AES has 16-byte key;
slower with 32-byte key.
Salsa20 has no such slowdown.

AES becomes even slower if
key is not pre-expanded.
Salsa20 has no precomputation.

AES has serious timing leaks:
see `http://cr.yp.to/papers.html#cachetiming`
for successful AES key extraction.
Constant-time AES is very slow.
Salsa20 has no timing leaks.

I offer $1000 prize for
the public Salsa20 cryptanalysis
that I consider most interesting.
Awarded at the end of 2005.

Send URLs of your papers to
`snuffle@box.cr.yp.to`.