Why Whitenoise creates a data array and not a LFSR

Background on WN deterministic random number generation for Whitenoise key creation 1

Creating a data array (data source) for constructing a WN key. 1

The use of registers or CSRs a visual analogy 4

Another visual graphic for illustration using your Illustration 1 6

Background on WN deterministic random number generation for Whitenoise key creation

In cryptography, creating strong highly random keys to use for various security controls is crucial. To create keys in any crypto-system or key generation technique a data source is required. This is why a plurality of computer based crypto-systems use deterministic random number generators (RNG).

The problem is most RNGs are not very random and we have seen PGP and Microsoft move away from NIST prescribed ones.

Telecoms often use highly randomized data created from radio active decay as a data source. Samples of electrons shedding from radio active decay has historically been considered the most random data source in nature but using this as a data source creates design problems because it is not deterministic.

When Whitenoise is used as a random number generator it creates a data source that is orders of magnitude more random than samples of radio active decay. It is deterministic which makes it ideal for crypto-systems. In a scientific first there was not even a single expected statistical failure against the NIST randomness test suite in the performance analysis conducted at the ECE of the University of Victoria, British Columbia, Canada in conjunction with the National Research Council of Canada.

Creating a data array (data source) for constructing a WN key.

As an analogy creating this data array is similar to creating an S-box and populating the subkey lengths within the array in a deterministic fashion with randomized data from a master key. Specific subkey lengths correspond to specific rows. These subkey lengths are not generally in an ascending or descending order and the randomized ordering of subkey lengths within the array further eliminate entire classes of mathematical attacks that require predictability.

The process uses a master key which is highly randomized (more random then radio active decay) to populate the subkey lengths. Picture the subkey lengths as troughs. As necessary these troughs can be flushed out and repopulated with new unused random data. This occurs when the data source would repeat itself and lose a characteristic of a one-time-pad.

In this small example it would happen at a point where the data array is 105 elements wide. [7 X 5 X 3 = 105].

The overall strength of CSRs and other kinds of counters is fatally flawed for several reasons:

- Those counters, registers or CSRs will continue to repeat themselves forever and after just 105 cycles (in this example) starts repeating itself and creating a pattern which is fatal in cryptography.

- Additionally, the register approach is further weakened by a constant, regular, patterned order of data within the register.

- And finally they are flawed by the relative shifting patterns created between the different CSRs by using a shift technique.

- Brute force attacks can be used against the CSR, register or counters approaches because of computer speeds and the relative weakness of these constructs.

The counter/CSR representation approach would create quadratic expressions for mathematical attack. This has been disproved as a valid attack by both a Communications Security Establishment crypto mathematician and cryptography icon David Wagner at the University of California, Berkeley. The results of the security analysis show there are no vulnerabilities to mathematical attacks.

Imagine a structure with subkeys of the lengths 3, 5, 7, 11, &13.

Quadratic expressions can be determined and might be solved if those subkeys occurred only in ascending or descending order i.e.

3, 5, 7, 11, & 13

13, 11, 7, 5, & 3

Whitenoise has a randomized order of subkeys lengths represented by specific rows within the source data array so those mathematical techniques are negated. I.E.

11, 3, 13, 5, & 7

Given all of the above, at the point that an array or data source begins repeating itself, its use should be discontinued because it would be less cryptographically secure and in the case of Whitenoise unnecessarily eliminate one of the characteristics required of one-time-pads.

In the data array RGN described herein, this example for Whitenoise would repeat itself at that same interval as the CSR representation and lose a characteristic of an OTP. At this point, the subkey lengths (troughs) can be flushed out and repopulated with randomized data from an unused portion of the Whitenoise master key.

Why Whitenoise creates a data array and not a LFSR

In most cases because of the exponentialism of resultant Whitenoise keys and following the scope recommendations of using between 10 and 20 subkeys to create a Whitenoise key, reaching this point of repetition in a Whitenoise data source array is infeasible during the life of the person or device utilizing that key structure and the resultant Whitenoise key.

Whitenoise creates a data array as a data source that is similar to constructing an S-box. It is not an CSR, register or counter.

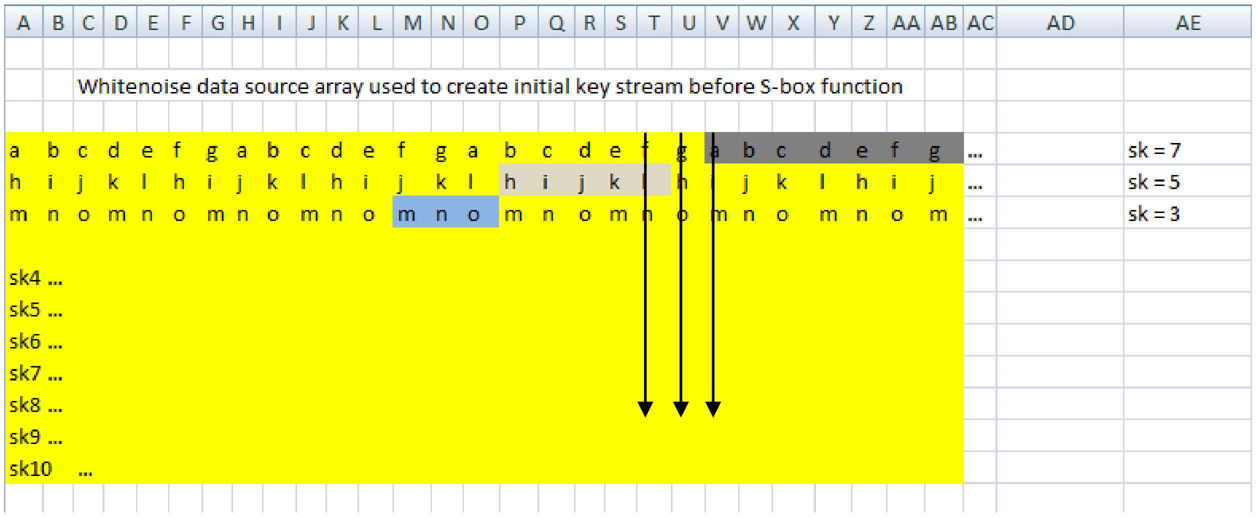
Row 4 from the spreadsheet illustration is populated with data from subkey 1 and the subkey length is 7 bytes so moving horizontally along that row on the data array we see a, b, c, d, e, f, g repeating itself horizontally until that row of the array is filled.

Row 5 is populated with data from subkey 2. The subkey length is 5 bytes long in this example and to distinguish the bytes as coming from this unique subkey length we are labeling it h, I, j, k, l repeating itself horizontally until that row of the array is filled.

Row 6 is populated with data from subkey 3. The subkey length is 3 bytes long in this example and to distinguish the bytes as coming from this unique subkey length we are labeling it m, n, o repeating itself horizontally until that row of the array is filled.

For ease in seeing the combination of bytes, each box has been labeled with a letter in an ascending and regular fashion with letters from the alphabet. These letters are illustrative and do not represent specific or static values as the subkey lengths are populated with randomized data from a master key.

For each subkey and its corresponding row a block of the subkey is highlighted so that it is easier to see how that repeats within a row to populate it.



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE
		Whitenoise data source array used to create initial key stream before S-box function																												
a	b	c	d	e	f	g	a	b	c	d	e	f	g	a	b	c	d	e	f	g	a	b	c	d	e	f	g	...		sk = 7
h	i	j	k	l	h	i	j	k	l	h	i	j	k	l	h	i	j	k	l	h	i	j	k	l	h	i	j	...		sk = 5
m	n	o	m	n	o	m	n	o	m	n	o	m	n	o	m	n	o	m	n	o	m	n	o	m	n	o	m	...		sk = 3
sk4 ...																														
sk5 ...																														
sk6 ...																														
sk7 ...																														
sk8 ...																														
sk9 ...																														
sk10	...																													

XOrs are done in a vertical fashion for all columns as seen in columns T, U and V arrows.

Why Whitenoise creates a data array and not a LFSR

The first twelve bytes of the initial key stream starting from the first column and moving across the array before subsequent S-box functions are:

(ahm) (bin) (cjo) (dkm) (eln) (fho) (gim) (ajn) (bko) (clm) (dhn) (eio) …

It is a different process generating different results than the bytes that would be generated by an CSR, register or counter. An CSR would generate the following first twelve bytes from the same register lengths with the same labels because it is operating differently with a shift after each XOr function:

(ahm) (glo) (fkn) (e,j,m) (d,i,o) (c,h,n) (b,l,m) (a,k,o) (g,j,n) (f,i,m) (e,h,o) (d,l,n)

(AHM) (GLO) (FKN) (EJM) (DIO) (CHN) (BLM) (AKO) (GJN) (FIM) (EHO) (DLN) (CKM) [Mihai, this is what you just sent and confirms what I anticipated.

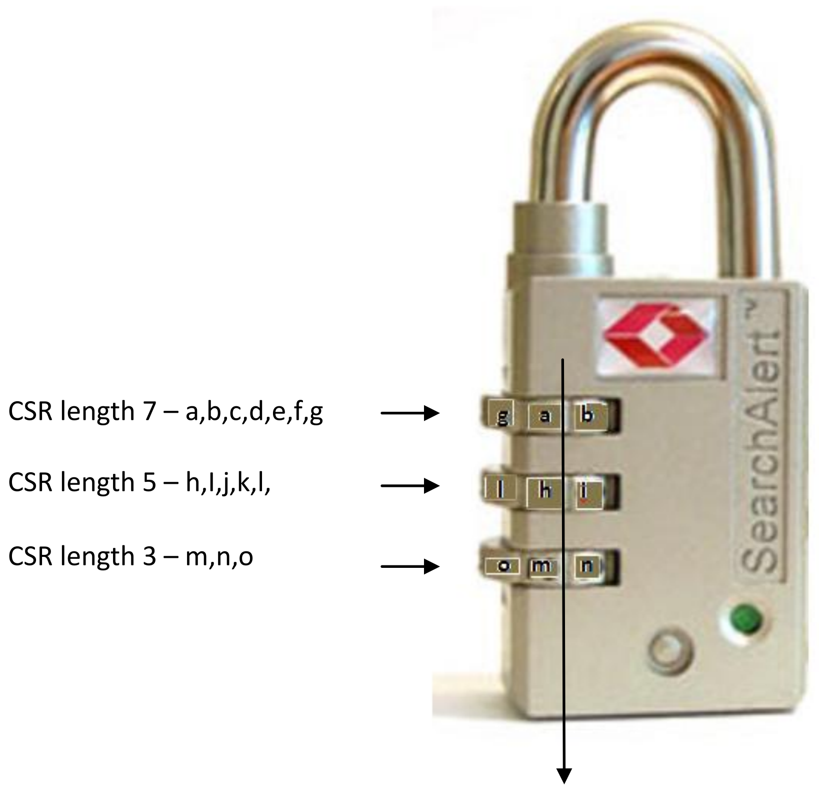
The only value that would be identical between the two distinct key generation approaches in this example would be the first value (ahm) because that was the arbitrary starting point. In use, this point represents a Current Dynamic offset which is an index into a key stream.

--

The use of Circular Shift Registers a visual analogy

Cycle 1

In this illustration assume that the different CSR lengths are different sizes of 7, 5, and 3 bytes.

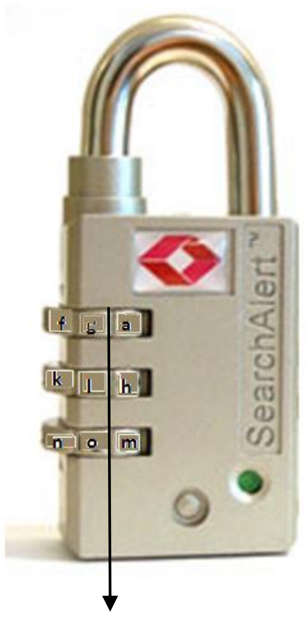


The XOr function performed is vertical. The XOr value of the first byte would be **(a, h, m).**

Why Whitenoise creates a data array and not a LFSR

In this scenario, after each XOr calculation each counter, register or CSR would immediately increment either clockwise or counter clockwise by 1 byte.

The result to set up the second cycle state would look like this by rotating the counters one shift to the right:



The XOr function performed is vertical. The XOr value of the second byte would be **(g,l,o)**.

We can see our initial starting point has shifted one "tick" to the right. The counters are rotating horizontally in this illustration.

Following this process with an CSR the first 12 bytes of a key stream would be:

(ahm) (glo) (fkn) (e,j,m) (d,i,o) (c,h,n) (b,l,m) (a,k,o) (g,j,n) (f,i,m) (e,h,o) (d,l,n)

This is creating predictable quadratic equations. A Communications Security Establishment crypto mathematician has proven that this is not the case and this technique to try to break a Whitenoise key is not valid.

The first twelve bytes of key stream resulting from a whitenoise data array comprised of identical subkey lengths and labeled in the same fashion would be:

(ahm) (bin) (cjo) (dkm) (eln) (fho) (gim) (ajn) (bko) (clm) (dhn) (eio) …

The process is completely different and therefore the key stream data is completely different. Importantly, the data's internal characteristics are completely different as well and cannot be determined or predicted using quadratic expression mathematical techniques.

Only the first value beginning from an arbitrary starting point for this example is identical. No other expected values for the next byte value in a properly constructed Whitenoise key stream will ever be the same as predicted by the CSR approach.

Whitenoise is not an CSR.

Whitenoise is not operating like a counter or a register in a way similar to an CSR. It creates a deterministic data source array to be used for key creation.

Another visual graphic for illustration using your Illustration 1

Line Feed Shift Register deployment (not Whitenoise)

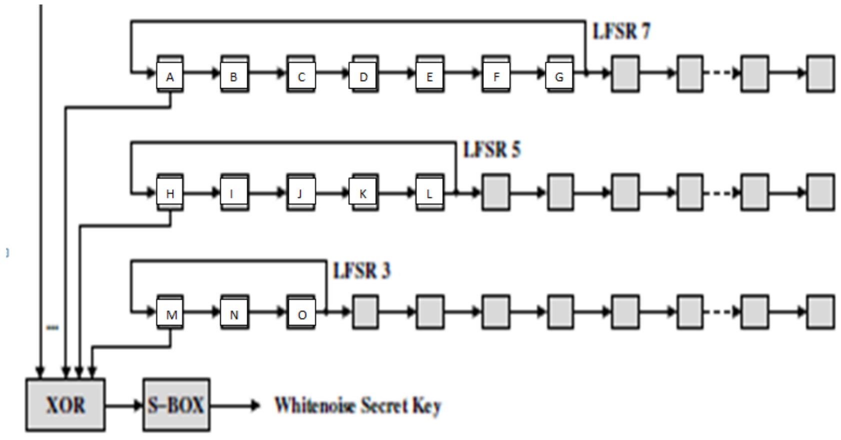


Fig. 1: The Whitenoise encryption algorithm.

In the in CSR the counters representing subkeys (sic) have a loop instead of just continuing horizontally to create and populate an array instead of creating a counter or CSR function. And the tick or shift occurs immediately each time the XOr function is performed. The counter ticking or shifting in this manner is creating a relative pattern between changing counters.

And repeating, the first 12 anticipated bytes from this approach would be:

(ahm) (glo) (fkn) (e,j,m) (d,i,o) (c,h,n) (b,l,m) (a,k,o) (g,j,n) (f,i,m) (e,h,o) (d,l,n)

When the Whitenoise data source is properly used as an array and not a counter XOr'ing the corresponding bytes between subkeys yields the correct anticipated results. To create this array we are eliminating the loop. We are eliminating the CSR XOr approach in the red box. The black line signifies the end of the array box. Subkeys 2 and 3 have already begun repeating because they are smaller than subkey 1. The XOr function is properly illustrated within this array and performed in a vertical fashion.

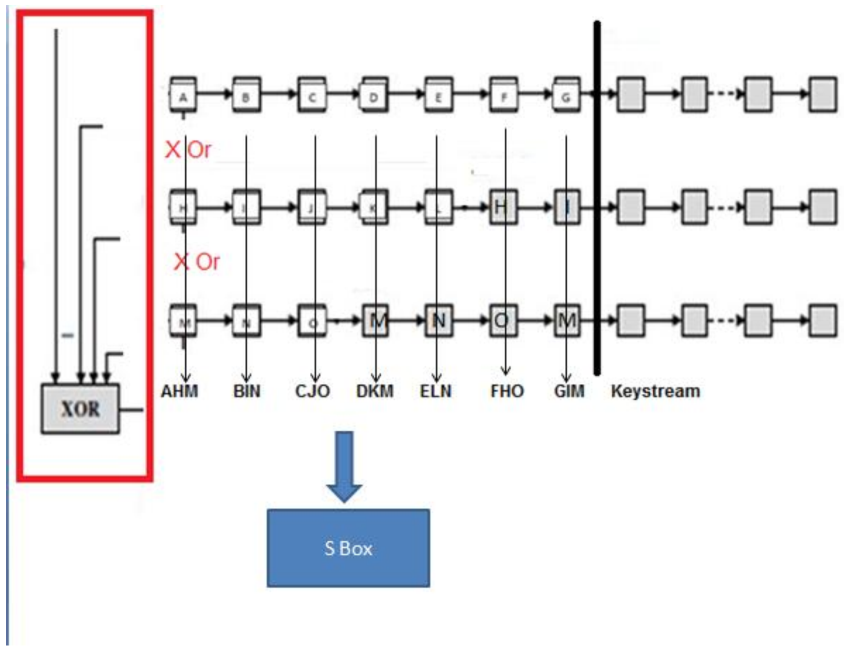
Without a shift Whitenoise cannot be a line feed **shift** register.

Why Whitenoise creates a data array and not a LFSR

Since Whitenoise is not an CSR the postulated attack is invalid. Without any shifting no relative pattern information between “registers” or subkeys is available for attack use. In an above illustration we have seen that the correct anticipated next 12 bytes using subkey lengths of 7, 5, and 3 yields:

(ahm) (bin) (cjo) (dkm) (eln) (fho) (gim) (ajn) (bko) (clm) (dhn) (eio) …

And if we did it in the manner just described we can see the correct results.


A
B
C
D
E
F
G
X Or
H
I
J
K
L
H
I
X Or
M
N
O
M
N
O
M
XOR
AHM
BIN
CJO
DKM
ELN
FHO
GIM
Keystream
S Box