## Title:

**On Studying Whitenoise Stream-Cipher Robustness against Power Analysis Attacks:**
**there is no provable or demonstrated break.**
*Vancouver, British Columbia, Canada*

## Background/Purpose

The purpose of this paper is to clarify that there is no break that was demonstrated or possible against Whitenoise as proposed by B. Zakeri. To make the claim based on this work that Whitenoise has been broken is not academically sufficient, ever demonstrated, or warranted. It is theoretical (his own words). It is misleading for non-expert readers.

B. Zakeri, *On Studying Whitenoise Stream-Cipher Robustness against Power Analysis Attacks*, MASc Thesis, University of Victoria, British Columbia, Canada, Dec. 2012.

http://dspace.library.uvic.ca/handle/1828/4360?show=full

This is important to address because of the general failure, inability, or knowledge of cryptography for evaluators to properly evaluate such research and claims made therein.  That said, unproven break statements can have a significant impact on the perception of specific cryptographic sciences and techniques.

The paper fails to set the proper context and give an indication of how rare side channel attacks are in the first place because they generally cannot be afforded by anyone but nation states. Neither does the original study address that in normal circumstances there is no access to devices or keys because they reside behind secure perimeters.

This paper compares the proper construction of Whitenoise keys as documented in global patents versus the attempted implementation of Whitenoise with circular shift registers that was conducted by Babak

Zakeri, at the University of Victoria. Mr. Zakeri has an undergraduate degree from the University of Tehran.

At no point in a Whitenoise patent, presentation or scope document are the words shift, register, line feed shift register, cycle or any other word which is used to characterize a circular shift register used.

That decision or choice, possibly because of constraints within the FPGA environment, made by Mr. Zakeri to treat Whitenoise as a circular shift register did not follow the patent or scope documents. Therefore the implementation he was evaluating did not following scope.

As such, the approach taken by Zakeri was actually testing against a completely different algorithm and is not representative or academically sufficient for any meaningful conclusions about the security of the Whitenoise algorithm. No implementation analysis is ever sufficient to make conclusions about an algorithm anyway because it is mixing apples and oranges. If done properly, it might only question the implementation approach.

The flawed study ironically does yield important results if framed properly.

As opposed to making an unsubstantiated claim against Whitenoise, Mr. Zakeri could have framed the problem as trying to improve the security of registers, counters, key rings and low-cost, low-level-security components. And, on that front, since he can not demonstrate breaking Whitenoise he has inadvertently shown a significant demonstration that the security of  circular shift registers (CSR), counters, line feed shift registers, and key rings can be dramatically improved by using Whitenoise.

A subsequently published research paper from the same University of Victoria, British Columbia labs makes recommendations that merit further research. The recommendations would make the unproven break undoable anyway and dramatically advances an approach to securing registers, counters and rings by using Whitenoise. It shows that using Whitenoise with circular shift registers, line shift registers, counters and key rings can secure this low-cost kind of components from side channel attacks which is a first. This moves the body of knowledge forward.
--
However, by necessity, the focus of this rebuttal paper is on the inaccurate assumptions made by Mr. Zakeri which completely invalidate any theoretical claim the original paper made of a successful attack against Whitenoise. It was NEVER demonstrated. One of the principal researchers said that after two years they were never able to demonstrate an “end-to-end break because they ran out of money.” And that is against a deployment that Mr. Zakeri claims would take .1 second to accomplish. Never is any mathematical data shown to back up his claim. Never does he show a broken key.

Because it was recognized that the break claims of the paper On Studying Whitenoise Stream-Cipher Robustness against Power Analysis Attacks could not be backed up, a $200,000 challenge specifically invited the author of the first paper to demonstrate his claim. He could not.

http://www.wnlabs.com/news/challengeDEFCON.php

This provides a sad comment on the consequences of misinformation or incomplete scientific method that we would have to provide a defence of Whitenoise in such a manner and outside of the proper research environment. We were left only with being able to say "prove it."

It also provides a sad comment on how decision makers read these kinds of papers.

It is interesting to consider that in a static environment a break was never demonstrated over two years and in real life the dynamic key would have changed millions of times. A hacker would not have two years to solve the problem or challenge. The hacker, using the proposed technique or not, would have had to start over every time the key dynamically changes.

http://www.wnlabs.com/news/DEFCON_Black_HAT_Challenge_Clock.php

In review of the Whitenoise patents filed and granted globally there is not a single description or reference that would lead one to believe that Whitenoise is a circular shift register.

Within those patents there is not a single instance where any of the following words or phrases are used or contemplated.

- register
- registers
- line shift feed register
- feed
- shift
- cycles
- internal registers

Conceptually it is often difficult to illustrate how algorithms behave; however, we will do so in this paper. And we will show that far from being able to create a demonstrable break, given the misunderstanding how Whitenoise operates and the deployment of the CSR approach, the researchers were unable to accurately identify any of the correct bytes in an initial Whitenoise key stream before application of a delinearization technique.

This error occurred not in any attempt to break (guess, discern) a key. It was done in the context of being unable to list key stream information correctly from information directly provided to them. This simple test was done to evaluate their correct understanding of Whitenoise.

Fundamentally, you will see that it is impossible for a single CSR to create the same information that the Whitenoise algorithm generates because they are different things.

In the study Mr. Babak Zakeri provides the following illustration for how he erroneously thinks a Whitenoise key is made or can be represented. He chose to implement it within an FPGA in this fashion because of limited options. Or, he assumed that the algorithm operated in this fashion which is why they were unable to demonstrate a break.

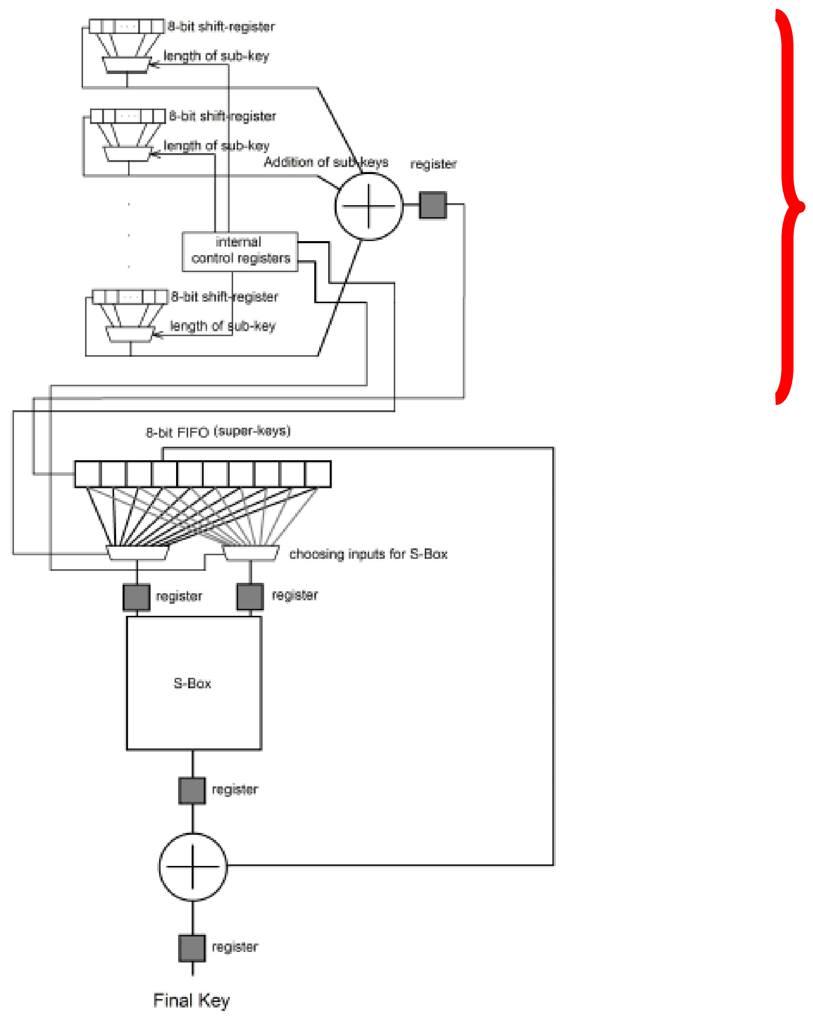
## ***Fatal assumption by Zakeri***

*We can readily see with Mr. Zakari's own words where he embarks on an implementation* that is not Whitenoise:

"This is built as such for implementation purposes, so the length can fit in a single byte."

*He has fundamentally altered the Whitenoise algorithm so that a single sub-key length might be represented by a single byte, 8-bit register.*

*This is a classic case of confusing bits and bytes. And in doing so the construct is NOT Whitenoise. If you return to Illustration 1, you will see the top third of the graphic represents the data source and it is generated from two 8-bit registers (that is one byte.)*



The top third represented by the bracket is the data source. Mr. Zakeri is using two, 8-bit registers with pre-programmed data to create data that will then be used to both determine the length of sub-keys and populate them to create the initial key stream.

Eight bits is the equivalent of 1 byte or 1 character. Mr. Zakeri's process uses these registers to generate sub-key lengths that are populated with data for the register.

The very first step from the Whitenoise patent on generating and populating subkey lengths is:

1. Treat seed1 as the decimal representation of an integer in the range of 500-700 digits.
2. Let X := seed1
3. Let Y := $\sqrt{X}$ is the irrational number generated by square rooting X

Mr. Zakeri writes: "For generating the sub-keys, the algorithm uses two 32-bit seeds. The square root of one of the seeds is computed, and the digits after the decimal points are used for computing the values of each byte of every sub-key."

*His description is inaccurate. In fact the Whitenoise algorithm specifically states that the primary seed is a minimum of 500 to 700 digits or between 4000 and 5600 bits (number of digits X 8 bits per byte).*

*Another seed is 32 bits but that is a restriction of computers at that time that could only accept 32-bit numbers to seed the rand function on most computers. (This is the same problem as IPv4 being 32-bits and IPv6 being 64 bits or greater.)*

*Additionally, the use of the second 32 bit number is simply to choose a random starting point. Otherwise there is no other interaction between this value and the determination of subkey length or the data with which it is populated.*

From the Whitenois patent:

5. Call srand(seed2). // only the first time
6. Call rand() to get the irrational starting point, start.

*We can readily see with Mr. Zakeri's own words where he embarks on an implementation* that is not Whitenoise:

Mr. Zakeri writes:

"For a simple visualization, one can assume each sub-key as a circular shift-register and in every cycle one shift operation to left (or right) is performed, while some entry with fixed position in the shift-register is used for the summation."

*In no Whitenoise patent or document is the use of registers or shifts ever discussed. This is his creation. A description of the Whitenoise key creation process will be found in the addendum to this document.*

Mr. Zakeri writes: "In Whitenoise algorithm up to 10 such sub-keys can be chosen."

*This is inaccurate. All scope documents and patents state that a **MINIMUM** of 10 sub-keys is used with a recommendation to use between 10 and 20 sub-keys of a recommended minimum length.*

“and length of each one can be a prime number between 2 and 255[5].”

*This is incorrect. The length of each subkey is not restricted to the weak universe of prime numbers between 2 and 255. This inaccurately restricts the number of possible subkeys lengths or sizes to <u>just the 54 smallest  prime number lengths</u>.*

```
     2      3      5      7     11     13     17     19     23     29
    31     37     41     43     47     53     59     61     67     71
    73     79     83     89     97    101    103    107    109    113
   127    131    137    139    149    151    157    163    167    173
   179    181    191    193    197    199    211    223    227    229
   233    239    241    251
```

## *Drilling down to the math:*

From the Whitenoise patent:

```
13.     Let tmp be the next byte from the Z stream.
 14.    Let tmp := $1000*Z_{j+1} + 100*Z_{j+2} + 10Z_{j+3} + Z_{j+4}$
 15.    Let t := 1862 - (tmp mod 1862).  t is in the range 1,2,.., 1862.
 16.    Let u be the t$^{th}$ prime among the sequence 2,3,5,.., 1862.
 17.    If u is equal to any of $l^1, l^2, .., l^{\{i-1\}}$, set t to (t+1)mod 1862 goto 16
 18.    Set $l^i$ = u.
 19. Next I : goto 11 until all 10 subkey sizes are set.
 20. Then get remainder of lengths
 21. For i := 11,..,n, do:
 22.    Let j = 5*(i-1)
 23.    Let tmp be the next byte from the Z stream.
 24.    Let tmp := $10000*Z_{j+1} + 1000*Z_{j+2} + 100Z_{j+3} + 10*Z_{j+4} + Z_{j+5}$
```

Mr. Zakeri states:

“The idea behind Whitenoise stream-cipher comes from this basic theorem in number theory that if there are n distinct prime numbers p1, p2, ..., pn, the least common multiple of them would be p1p2...pn. Therefore, if there are n sub-keys (intermediate sets of key produced by some seed keys) k1, k2, ..., kn, each of the length pi bytes, the summation of them given as: S(x) = k1(x mod p1) + k2(x mod p2) + ... + kn(x mod pn) (3.1) has a repeating period of p1p2...pn for all $x \geq 0$. x denotes the cycle of execution, and for each x, k(x mod pi) represents the next entry of each sub-key, that is chosen for the summation.”

This is how the next byte of an initial Whitenoise key stream is created.

**Let tmp be the next byte from the Z stream.**

Let tmp := $10000*Z_{j+1} + 1000*Z_{j+2} + 100Z_{j+3} + 10*Z_{j+4} + Z_{j+5}$

Mr. Zakeri has implemented an unspecified approach and completely different mathematical statement that he then treats incorrectly as being equivalent. We will next see how this creates completely different data with different characteristics impacts its inherent security, or lack of it.

This is how the incorrect CSR (circular shift register) approach is creating the next entry (byte) of each sub-key.

$S(x) = k1(x \bmod p1) + k2(x \bmod p2) + \ldots + kn(x \bmod pn)$

We immediately see these are fundamentally two different processes and two different mathematical expressions. Zakeri is using his owned defined constructions at this point and calling it Whitenoise. We readily see it is deploying a mod function at this point of his algorithm and Whitenoise does not. This is one of Zakeri's misguided assumptions. It is a different mathematical representation of how to create and populate a subkey length.

## *Fatal exclusion by Zakeri*

Previously the one-way functions inherent in the Whitenoise algorithm were listed.

The side channel attack tests the idea that output (cipher text) and the S-box one-way-function can be ignored and "hopped over" by evaluating the power consumption of the circular shift registers that have been deployed and are already populated and the subkey lengths are in ascending or descending order and the data within is already fixed. If that assumption were valid it would be a product of the change of physical characteristics not present in software deployments. Again, for clarity: if there are any valuable things learned here it is only about the physical use of CSRs and their implementation and not the Whitenoise algorithm.

In Mr. Zakeri's own words he states the theoretical break requires the following conditions for it to work.

"There is one requirement though, as equation 3.11 denotes, which should hold for all of the mi values and it is as follows:

$$2m9p9 \leq m10p10$$
$$2m8p8 \leq m9p9$$
$$2m7p7 \leq m8p8$$
$$\ldots$$
$$2m1p1 \leq m3p3$$

and

$$p1 \leq m2p2$$

This states that subkey lengths and their resulting definitions MUST be in ascending or descending order or order for the theoretical break proposal to have any validity.

The Whitenoise algorithm has no requirement for ascending or descending subkey lengths and actually recommends that these subkey lengths are randomized according to size as well as the data populating those lengths.

The theoretical break has already been disproven by a proof written in conjunction with a CSE crypto-mathematician. http://www.wnlabs.com/pdf/Response.pdf (Communications Security Establishment)

Additionally, no repeating data (data source) is ever used. After all combinations in a CSR are used once, it would be flushed out, and the data within the registers would be repopulated.

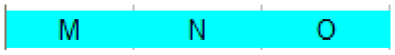
There is no information that the theoretical technique, nor any other, can get about the master key used to populate the registers.

At best, the hacker needs to start over and over and over and over. This introduces a dynamic one-time-pad element to the use of the CSR. Given cost, time, speed, rarity of attack use, etc. any attack scenario on the register constructed in this fashion is not feasible. And we know a true one-time-pad is unbreakable.

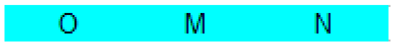
--

## Visual models to compare Whitenoise and an CSR Circular Shift Register

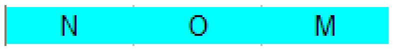
In our working example in the final paper available from the Side Channel Research Analysis web page, one of the 3 registers is three bytes long. These are fixed values and are labelled in our example as M, N, and O.



Since a register has fixed values populating them, the only possible way to try to randomize them is by utilizing a shift which leaves the data populating the register identical sequentially but unique by shifting and starting the register with one of the other values. This register with a counter clockwise shift of 1 byte would yield:



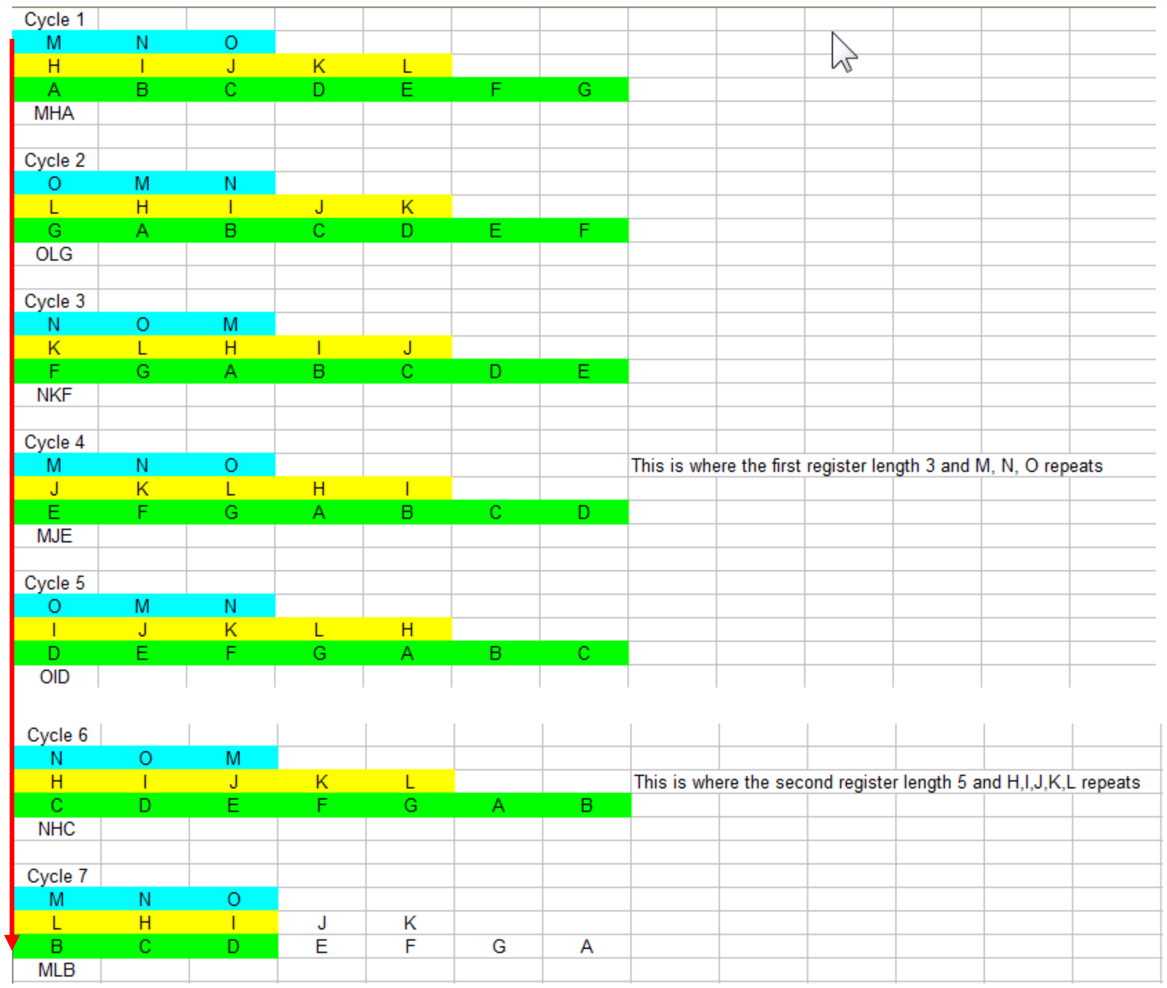
And finally, the last iteration possible yields:



There are only three possible iterations for a register with a length of 3 bytes.

Those shifts are cycles. When we continue with our example of registers of 3, 5, and 7 represented by separate rows the first 7 cycles would yield the following results. The XOr with Line Feed Shift Register can only occur at one specific location, in this case column A and represented by the red, downward XOr arrow.

The only way to involve the other data in the other larger registers and satisfying the requirement that all registers are interacting with all the data of the other registers is by implementing this construct with a shift.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Cycle 1 | | | | | | | |
| M | N | O | | | | | |
| H | I | J | K | L | | | |
| A | B | C | D | E | F | G | |
| MHA | | | | | | | |
| Cycle 2 | | | | | | | |
| O | M | N | | | | | |
| L | H | I | J | K | | | |
| G | A | B | C | D | E | F | |
| OLG | | | | | | | |
| Cycle 3 | | | | | | | |
| N | O | M | | | | | |
| K | L | H | I | J | | | |
| F | G | A | B | C | D | E | |
| NKF | | | | | | | |
| Cycle 4 | | | | | | | |
| M | N | O | | | | | This is where the first register length 3 and M, N, O repeats |
| J | K | L | H | I | | | |
| E | F | G | A | B | C | D | |
| MJE | | | | | | | |
| Cycle 5 | | | | | | | |
| O | M | N | | | | | |
| I | J | K | L | H | | | |
| D | E | F | G | A | B | C | |
| OID | | | | | | | |
| Cycle 6 | | | | | | | |
| N | O | M | | | | | |
| H | I | J | K | L | | | This is where the second register length 5 and H,I,J,K,L repeats |
| C | D | E | F | G | A | B | |
| NHC | | | | | | | |
| Cycle 7 | | | | | | | |
| M | N | O | | | | | |
| L | H | I | J | K | | | |
| B | C | D | E | F | G | A | |
| MLB | | | | | | | |

Note the XOr function between values of sub-keys can only occur at one static point at column A. Because of this the register data must circulate around this static point.

In order to test and confirm that the principal researchers did not implement Whitenoise properly and had made a faulty assumption, they were asked to list the first dozen bytes of a Whitenoise key stream generated from subkeys of 3, 5, and 7 bytes in length. The sub-keys were provided labelled values seen below.

The researcher was unable to predict a single anticipated byte of key stream correctly based on their faulty assumption that the relationship of the bytes operated in a manner described by a circular-shift register. In correspondence they indicated that the bytes relative to a line feed shift register for the first seven cycles would be those listed above:

**MHA, OLG, NKF, MUE, OID, NHC, MLB**

Whitenoise is not a circular shift register. Our goal conceptually is to create a data array using the same sub-key lengths for specific rows within a single data array that would be 105 bytes (columns – 3X5X7) wide by 3 rows deep (the number of sub-keys in our example.)

| Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | | | | | | ... | Byte 105 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M | N | O | M | N | O | M | N | O | M | N | O | | |
| H | I | J | K | L | H | I | J | K | L | H | I | | |
| A | B | C | D | E | F | G | A | B | C | D | E | | |
| MHA | NIB | OJC | MKD | NLE | OHF | MIG | | | | | | | |

The correct first 7 key stream bytes from the Whitenoise algorithm would be:

**MHA, NIB, OJC, MKD, NLE, OHF, MIG**

Only the first value MHA is correct between the two different approaches but only because the arbitrary starting point of this example are the same for both. Every other value the researcher anticipated was incorrect because they are just completely different processes.

--

To see this more clearly, let us construct this step-by-step so we can see the differences in the processes, how they are not equivalent and how they generate different results.

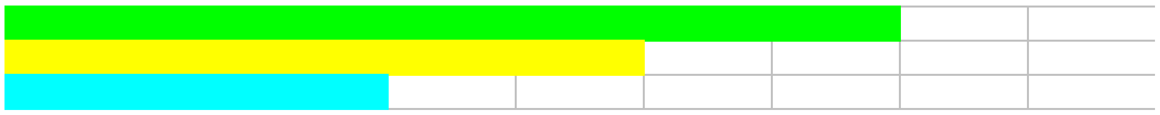
This illustration represents a circular line feed shift register with fixed labels or values provided for counters (subkeys) of fixed lengths of 3, 5, and 7 bytes.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| M | N | O | | | | | | skl=3 |
| H | I | J | K | L | | | | skl=5 |
| A | B | C | D | E | F | G | | skl=7 |

For simplicity so that we can see later how data pours into WN subkey lengths we are just reordering the sequence in a descending length fashion. In this view we can see how the values are contiguous and sequential, in this case following the order of the alphabet.

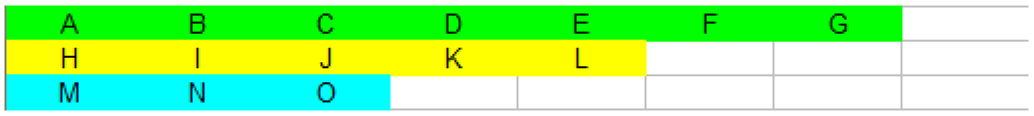
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | | skl=7 | |
| H | I | J | K | L | | | | skl=5 | |
| M | N | O | | | | | | skl=3 | |

The next illustration represents a Whitenoise subkey length before it is populated with data.



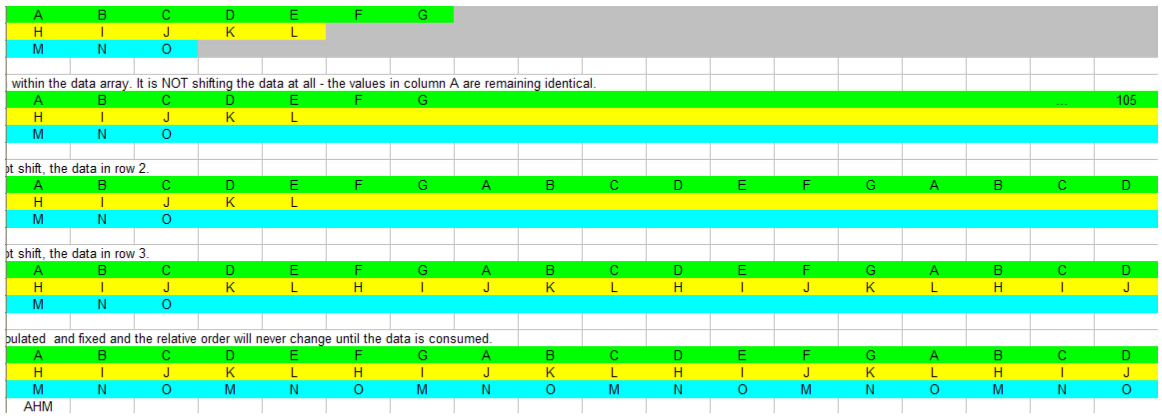
The next illustration again shows how these keys would be populated with data poured in from a Whitenoise key Z. You can see they are sequential.

This represents populating WN subkey lengths with data from another master key.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | |
| H | I | J | K | L | | | |
| M | N | O | | | | | |

The next illustration shows how we are creating a data array equivalent to 105 columns wide (the point where the data array would repeat) by 3 rows (subkeys) deep. Step by step, we see how the grey area might represent the balance of the array that needs to be populated. Each subkey row just REPEATS the data to fill out the balance of the row. Note: Repeating data in this visual analogy is not equivalent to cycling and shifting data within a circular shift register. So for a Whitenoise data source array row by row we see it populating with appropriate data.



| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | | | | | | | | | | | |
| H | I | J | K | L | | | | | | | | | | | | | |
| M | N | O | | | | | | | | | | | | | | | |

within the data array. It is NOT shifting the data at all - the values in column A are remaining identical.

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | | | | | | | | | | ... | 105 |
| H | I | J | K | L | | | | | | | | | | | | | |
| M | N | O | | | | | | | | | | | | | | | |

pt shift, the data in row 2.

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | A | B | C | D | E | F | G | A | B | C | D |
| H | I | J | K | L | | | | | | | | | | | | | |
| M | N | O | | | | | | | | | | | | | | | |

pt shift, the data in row 3.

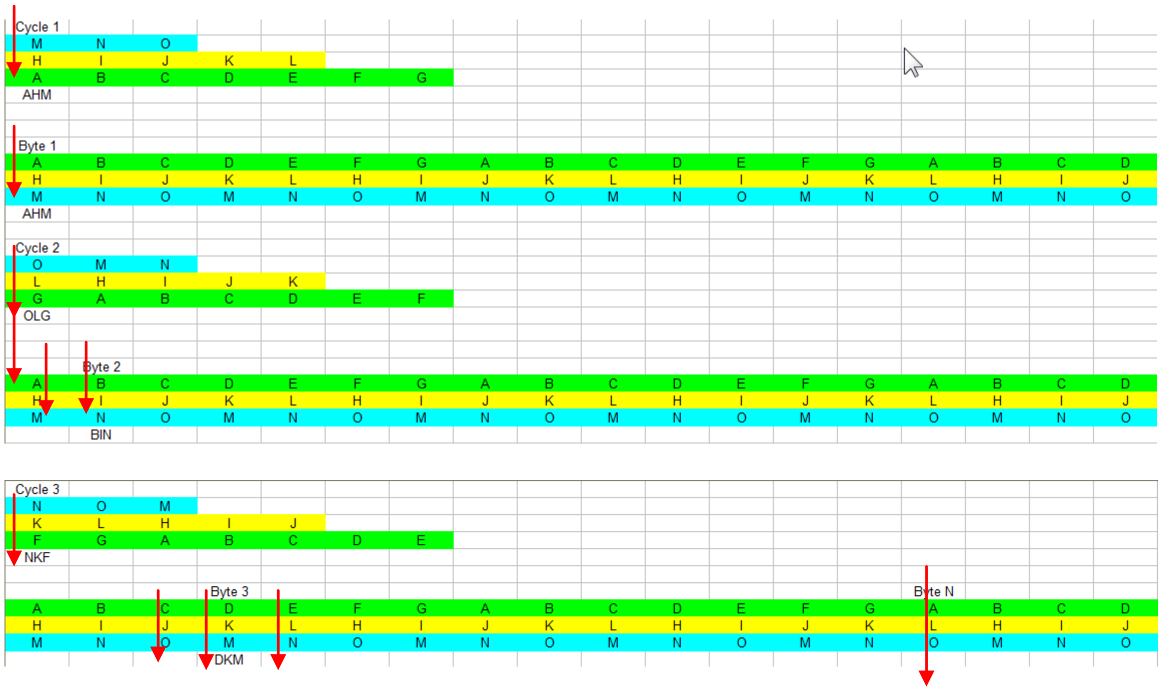
| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | A | B | C | D | E | F | G | A | B | C | D |
| H | I | J | K | L | H | I | J | K | L | H | I | J | K | L | H | I | J |
| M | N | O | | | | | | | | | | | | | | | |

pulated and fixed and the relative order will never change until the data is consumed.

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | A | B | C | D | E | F | G | A | B | C | D |
| H | I | J | K | L | H | I | J | K | L | H | I | J | K | L | H | I | J |
| M | N | O | M | N | O | M | N | O | M | N | O | M | N | O | M | N | O |
| AHM | | | | | | | | | | | | | | | | | |

The following illustration follows the LFSR circular shift register approach by cycles and directly underneath it compares the Whitenoise process scrolling along a data source array. But are supposed to be creating identical bytes of information but they do not because they are different processes. Scrolling along a data array is not equivalent to shifting within a register. This is seen by the index point where the XOr functions are performed between subkeys. This is a fixed point in LFSRs and represented by Column A. And we see the data incorrectly circling around that point.

In Whitenoise we can see that index pointer into the array (the result of another key stream) is moving to different columns to create different and subsequent bytes of the initial key-stream and generates a different result.



Cycle 1

| | | | | | | |
|---|---|---|---|---|---|---|
| M | N | O | | | | |
| H | I | J | K | L | | |
| A | B | C | D | E | F | G |
| AHM | | | | | | |

Byte 1

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | A | B | C | D | E | F | G | A | B | C | D |
| H | I | J | K | L | H | I | J | K | L | H | I | J | K | L | H | I | J |
| M | N | O | M | N | O | M | N | O | M | N | O | M | N | O | M | N | O |
| AHM | | | | | | | | | | | | | | | | | |

Cycle 2

| | | | | | | |
|---|---|---|---|---|---|---|
| O | M | N | | | | |
| L | H | I | J | K | | |
| G | A | B | C | D | E | F |
| OLG | | | | | | |

Byte 2

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | A | B | C | D | E | F | G | A | B | C | D |
| H | I | J | K | L | H | I | J | K | L | H | I | J | K | L | H | I | J |
| M | N | O | M | N | O | M | N | O | M | N | O | M | N | O | M | N | O |
| | BIN | | | | | | | | | | | | | | | | |

Cycle 3

| | | | | | | |
|---|---|---|---|---|---|---|
| N | O | M | | | | |
| K | L | H | I | J | | |
| F | G | A | B | C | D | E |
| NKF | | | | | | |

| | | | Byte 3 | | | | | | | | | | | Byte N | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | A | B | C | D | E | F | G | A | B | C | D |
| H | I | J | K | L | H | I | J | K | L | H | I | J | K | L | H | I | J |
| M | N | O | M | N | O | M | N | O | M | N | O | M | N | O | M | N | O |
| | | | DKM | | | | | | | | | | | | | | |

Note: subsequent to this discussion, and in a follow up paper, the use of multiple registers to perform the XOr functions was proposed. This would generate the same deterministic data but is a complete change of design from the original study and would affect the eventual manufacturing costs (and complexity) by requiring more and more registers to perform one-off XOr calculations.

# Addendum

## The Whitenoise patent describes the algorithm for generating the sub-keys and the next byte of information to be contained within:

```
1. Treat seed1 as the decimal representation of an integer in the range of 500-700 digits.
2. Let X := seed1
3. Let Y := $\sqrt{X}$ is the irrational number generated by square rooting X
4. Let $Z_1$, $Z_2$, $Z_3$, $Z_4$, .. be the digits after the decimal point in the decimal representation of Y.  Each $Z_i$ is in the range 0,..,9.
5. Call srand(seed2). // only the first time
6. Call rand() to get the irrational starting point, start.
7.  Let start := rand() mod 100.  start is in the range 0,1,..,99.
8.  Throw away $Z_1$ and $Z_2$ all the way to $Z_{start}$.
9.  Let tmp := 10*$Z_{(start +1)}$ + $Z_{(start +2)}$.  Throw away those used values.
10. Let n := 11 + (tmp mod 20).  n is in the range 11,12,..,30.
11. For i := 1,2,..,10, do:
12.      Let j = 4*(i-1)
13.      Let tmp be the next byte from the Z stream.
14.      Let tmp := 1000*$Z_{j+1}$ + 100*$Z_{j+2}$ + 10$Z_{j+3}$ + $Z_{j+4}$
15.      Let t := 1862 - (tmp mod 1862).  t is in the range 1,2,.., 1862.
16.      Let u be the $t^{th}$ prime among the sequence 2,3,5,.., 1862.
17.      If u is equal to any of $l^1$, $l^2$, .., $l^{\{i-1\}}$, set t to (t+1)mod 1862 goto 16
18.      Set $l^i$ = u.
19. Next I : goto 11 until all 10 subkey sizes are set.
20. Then get remainder of lengths
21. For i := 11,..,n, do:
22.      Let j = 5*(i-1)
23.      Let tmp be the next byte from the Z stream.
24.      Let tmp := 10000*$Z_{j+1}$ + 1000*$Z_{j+2}$ + 100$Z_{j+3}$ + 10*$Z_{j+4}$ + $Z_{j+5}$
25.      Let t := 16000 - (tmp mod 16000).  t is in the range 1,2,..,16000.
26.      Let u be the t.
27.      If u is divisible by any of $l^1$, $l^2$, .., $l^{\{i-1\}}$, set t to (t+1)mod 16000 goto 26
28.      Set $l^i$ = u.
29. Next I : goto 21 until all subkey sizes are set.
30. For i := 1,2,..,n, do:
31.   For j := 0,1,2,..,$l^i$, do:
32.     Let k := 4*j
33.     Let tmp be the next byte from the Z stream.
34.     Let tmp := (1000*$Z_k$ + 100*$Z_{k+1}$ + 10*$Z_{k+2}$ + $Z_{k+3}$) mod 256
35.     Let $s_j^i$ := tmp
36.   Next j : Next subkey byte
37. Next I : Next subkey
38. For i := 0,1,2,..,65535, do:
```

```
39.   Let j := 4*i
40.   Let tmp := (1000*$Z_j$ + 100*$Z_{j+1}$ + 10*$Z_{j+2}$ + $Z_{j+3}$) mod 256
41.   If tmp is equal to any of S[0], S[1], .., S[i-1], set to (tmp+1)mod 256 goto 41
42.   Set S[i] := tmp.
43. Next i
44. Let offset := $Z_iZ_{i+1}\ldots Z_{i+9}$
45. Return n, ($l^1,l^2,..,l^n$), ($s^1,s^2,..,s^n$), S[65536] and offset.
46. Save in keyfile and add seed1 and start value to DB
47. Increment seed1 and goto 2 //repeat until enough keys are created
```

Errors in assumptions used for this study invalidate any results or claims. It was never demonstrated anyway.

## From the study in Mr. Zakeri's own words

This section is included to address the tendency of persons to evaluate work based on what they want to hear or see and not based on what was actually said or written.

*"the new scenario would also be* ***presented to completely hack the implementation****. However it would also be shown that the complete hack requires very accurate equipment, large number of computations and applying a lot of tests and thus Whitenoise* ***seems fairly strong against this specific group of attacks."***

This statement is completely self-contradictory: completely hack the implementation versus Whitenoise seems fairly strong against this specific group of attacks.

*"The* ***shift****-enable pin enables shifting in the shift-registers in both modes, and also* ***shifting*** *in FIFO in run mode. Later the importance of this implementation technique of circuit will be seen, as it is used in the proposed attack scenario."*

There is no shift in the Whitenoise algorithm.

*"To avoid this problem, there is a need to be able to separate the portion of consumed power related to shift-registers from the rest of the Therefore, the total number of cycles for performing the attack scenario in this case is about 110, 000 cycles. In a slow encryption system with the clock frequency of 1MHz, this means* ***only 0.1 second of run time which is a totally reasonable time for performing an attack."***

After two years why did the researchers not take a tenth of a second to demonstrate their claims and show us a broken key?

*"The shortest possible prime number is 2 and the largest is 251 with about 50 other possibilities for prime numbers between."*

This is incorrect and a misrepresentation of Whitenoise limiting length of subkeys to prime number values between 2 and 251.

*"As a short summary of what is discussed and the final conclusions, it was mentioned* that ***Whitenoise seems resistant to common attacks. The proposed scenario along with its improvement can break the implementation and find the sub-keys,*** *but since it requires a large number of computations and tests, and it also requires advanced measurement capabilities,* ***Whitenoise seems a fairly strong stream-cipher.***

This is self contradictory.

***<u>"The proposal and its improvement can merely be seen as a theoretical proof</u>*** *that the implementation is vulnerable, and as mentioned, how seriously they are going to be taken is dependant to the application of Whitenoise, and the designer. The more serious the application is, the more likely that the attacker is willing to use high precision equipment."*

Theoretical proof is NOT scientific proof.

*9. It was concluded that considering the fact that* ***Whitenoise is resistant to common attacks****, and the proposed scenario requires very accurate equipment, large amount of calculations and a lot of tests to apply and obtain samples,* ***the implementation is fairly strong.***

This is in contradiction to a complete break statement.

*"As the final conclusions and possible future works, as mentioned before, the author highly recommends paying more attention to the separable portion of power consumption in the Whitenoise implementation. If the designer can come up with a new method to go around this problem, it would be a noticeable improvement. Although other techniques such as masking and noise addition can also be applied.*
*But if this weakness is resolved,* ***Whitenoise seems strong enough by itself."***

This is a path forward and silences all those whose narrative requires an unfounded "Aha, got you!" moment.