Whitenoise Labs Inc.

Whitenoise stream Cipher still unbroken

Preface: The following was written as a response to the purported break of Whitenoise by Wu. To put things into the proper context a little history is required. When Whitenoise was first designed, 3 distinct variants or versions were proposed, each with the same core whitenoise internal keystream generation but different de-linearization layers. The second variant uses 3 bytes in its de-linearization layer, using a suggestion from Wu to solidify that the output was secure. This is what we use in our applications. Version 1, the one attacked by Wu, had a simple substitution as the de-linearization layer and was only proposed to simplify cryptanalysis and academic interest.

Abstract: Whitenoise is a stream cipher with specification given at http://eprint.iacr.org/2003/249. In the proposed break by Wu specified at http://eprint.iacr.org/2003/250 there is one fundamental mistake in the attack. The fundamental mistake is that the internal keystream byte values are known. This is not true as specified below, therefore without that information you cannot build the linear systems required to break the cipher.

Written by
Stephen Boren email: sboren@bsbutil.com
Andre Brisson email: brisson@bsbutil.com

1. Introduction

There is one critical flaw in the break suggested by Mr. Wu. That flaw is the fact that you can get an association of similar bytes of the inner Superkey stream but despite that, you can not get the actual byte represented. In formulating the linear system suggested by Mr. Wu to break the key, the resulting output of each linear equation is zero. As such, there are then many solutions to the system. One solution is where all variables are zero, another solution is where all subkey values are 1's, and there are in fact many combinations of those variables that are solvable in this context but do not actually solve the system.

There is a simple proof that can illustrate that the attack suggested by Mr. Wu does not work at the stage of solving the linear systems. If we look at the linear equations that are generated as binary equations for each individual bit, there would be eight identical linear equation sets that would be generated. Each would be required to generate a very different bit sequence and this is impossible since they are the same linear systems. In fact it is impossible without having the internal Superkey bit stream. To guess the bit stream would require guessing the binary byte representation of each of the 256 different groups and would only work if you guessed 100% correctly. There are 256 factorial combinations that could be represented or 8.578×10^{506} combinations. It is infeasible to test every combination, especially since each test would take four to five days.

That said, there seems to be no way of building the linear system to generate these internal key stream bytes because you need the inner Superkey stream bytes to build the linear system.

Looking at it from another standpoint as suggested by Mr. Wu, you can set up sets of equations that are equivalent. Then using row substitution you can reduce the number of variables that are completely unknown but given the size of the system the best that you can do is approximately 30% of the actual internal key structure. In the simplest example presented that amounts to in excess of 100 unknown bytes. This would require testing of 2^{100} possibilities to attempt to solve. The following is a simplified example of this:

```
Suppose the lengths used are {3,5,11} and 2-bit
values {0,1,2,3} are used instead of bytes.

Subkeys B and C are used (i.e. with lengths 5
and 11). Generate 2-bit entries of A and C

A = [A0,A1,A2]        not used, so set to [0,0,0]
B = [B0,B1,B2,B3,B4]  = [0,1,3,2,1]
C = [C0,...,C9,C10]    = [2,3,1,1,2,0,1,0,3,1,2]
```

Suppose we have the first 23 stream values (to get 16 equations in 16 unknowns). Note that we don't know the z values, but do know which equations give equal z values.

	z values (before S box input)
A0 ^ B0 ^ C0 = 0 ^ 0 ^ 2 =	2
A1 ^ B1 ^ C1 = 0 ^ 1 ^ 3 =	2
A2 ^ B2 ^ C2 = 0 ^ 3 ^ 1 =	2
A0 ^ B3 ^ C3 = 0 ^ 2 ^ 1 =	3
A1 ^ B4 ^ C4 = 0 ^ 1 ^ 2 =	3
A2 ^ B0 ^ C5 = 0 ^ 0 ^ 0 =	0
A0 ^ B1 ^ C6 = 0 ^ 1 ^ 1 =	0
A1 ^ B2 ^ C7 = 0 ^ 3 ^ 0 =	3
A2 ^ B3 ^ C8 = 0 ^ 2 ^ 3 =	1
A0 ^ B4 ^ C9 = 0 ^ 1 ^ 1 =	0
A1 ^ B0 ^ C10 = 0 ^ 0 ^ 2 =	2
A2 ^ B1 ^ C0 = 0 ^ 1 ^ 2 =	3
A0 ^ B2 ^ C1 = 0 ^ 3 ^ 3 =	0
A1 ^ B3 ^ C2 = 0 ^ 2 ^ 1 =	3
A2 ^ B4 ^ C3 = 0 ^ 1 ^ 1 =	0
A0 ^ B0 ^ C4 = 0 ^ 0 ^ 2 =	2
A1 ^ B1 ^ C5 = 0 ^ 1 ^ 0 =	1
A2 ^ B2 ^ C6 = 0 ^ 3 ^ 1 =	2
A0 ^ B3 ^ C7 = 0 ^ 2 ^ 0 =	2
A1 ^ B4 ^ C8 = 0 ^ 1 ^ 3 =	2
A2 ^ B0 ^ C9 = 0 ^ 0 ^ 1 =	1
A0 ^ B1 ^ C10 = 0 ^ 1 ^ 2 =	3
A1 ^ B2 ^ C0 = 0 ^ 3 ^ 2 =	1

Now we try to solve these equations the A, B, C values

First reorder and pair up the equations based on their 2-bit output value (0,1,2 or 3)

A2 B0 C5 = A0 B1 C6
A0 B1 C6 = A0 B4 C9
A0 B4 C9 = A0 B2 C1
A0 B2 C1 = A2 B4 C3

A2 B3 C8 = A1 B1 C5
A1 B1 C5 = A2 B0 C9
A2 B0 C9 = A1 B2 C0

A0 B0 C0 = A2 B2 C2

```
A2 B2 C2 = A1 B1 C1
A1 B1 C1 = A1 B0 C10
A1 B0 C10 = A0 B0 C4
A0 B0 C4 = A2 B2 C6
A2 B2 C6 = A0 B3 C7
A0 B3 C7 = A1 B4 C8

A0 B3 C3 = A1 B4 C4
A1 B4 C4 = A1 B2 C7
A1 B2 C7 = A2 B1 C0
A2 B1 C0 = A1 B3 C2
A1 B3 C2 = A0 B1 C10

We get 16 equations in 16 unknowns

After a lot of row reduction etc., you get:

A0 = A2
A1 = A2

B0 = B4
B1 = B4 C4 C5
B2 = B4 C4 C5
B3 = B4 C4    C6

C0 =       C6
C1 = C4
C2 = C4 C5 C6
C3 =       C6

Now you can use 5 parameters {A2, B4, C4, C5, C6} to
express the 15 variables

You'll note that these equations tell you much of
the relationship within each subkey
(i.e. A0=A1=A2, B0=B4, B1=B2, C0=C3=C6, C1=C4)
```

This defines a subspace of potential solutions.
This subspace includes the "true" solution, the
all-zero solution and many more potential solutions.
With a few extra equations you may (or maybe not)
be able to reduce the subspace a bit more.

Tinnitus and the Whitenoise algorithm are readily available upon request through www.bsbutil.com and the Peer Review section for academic and scholarly review