Breaking the Stream Cipher Whitenoise

Hongjun Wu

Institute for Infocomm Research, Singapore
hongjun@i2r.a-star.edu.sg

Abstract. Whitenoise is a stream cipher with specification given at http://eprint.iacr.org/2003/249. In this paper, we show that Whitenoise is extremely weak. It can be broken by solving about 80,000 linear equations. And only about 80,000 bytes keystream are needed in the attack.

1 Introduction

The stream cipher Whitenoise was proposed at http://eprint.iacr.org/2003/249. A security evaluation has been given at http://eprint.iacr.org/2003/218. No security flaw of Whitenoise has been reported.

However we notice that the stream cipher Whitenoise is extremely weak. The poorly designed non-linear function in Whitenoise fails to resist the simple crypto attack. With about 80,000 bytes keystream, we can break that cipher by solving about 80,000 linear equations. We will describe the Whitenoise in Section 2 and give our attack in Section 3. We implemented the attacks on the reduced versions of Whitenoise and the results are given in Section 4.

2 The Cipher Whitenoise

The key schedule of Whitenoise generates some session key from the secret key and initialization vector. We will ignore the key schedule of Whitenoise since it is not relevant to our attack.

The session key is used to encrypt only one message and it consists of the following components.

1. n, a secret integer in the range [50,99].
2. n distinct secret integers ℓ^i $(1 \leq i \leq n)$. Each ℓ^i is a prime number not larger than 1021.
3. n subkeys s^i $(1 \leq i \leq n)$. Each s^i consists of ℓ^i bytes. The j-th byte of s^i is denoted as s^i_j $(0 \leq j \leq \ell^i - 1)$
4. a secret array S[256] that is a one-to-one mapping 8-bit-to-8-bit S-box.

Denote the key stream being generated as $Y = y_0 || y_1 || y_2 || \cdots\cdots$, where $||$ denotes concatenation. The key stream is generated as follows:

1. Let $z_j = s^1_{j \bmod \ell^1} \oplus s^2_{j \bmod \ell^2} \oplus s^3_{j \bmod \ell^3} \oplus \cdots s^n_{j \bmod \ell^n}$
2. The j-th byte of the keystream is given as $y_j = S[z_j]$.

3 Break the Whitenoise

There are two flaws in the design of Whitenoise.

Flaw 1: The S-box leaks a lot of infomation of the subkeys. Since the 8-bit-to-8-bit S-box in Whitenoise is one-to-one mapping, so if $y_j = y_k$, then $z_j = z_k$.

Flaw 2: The secret lengths of the subkeys, the ℓ^i $(1 \le i \le n)$, are pseudo-secret information. The subkeys in Whitenoise is a subset of all the possible subkeys. In our attack, we solve for all the subkeys, then the ℓ^i $(1 \le i \le n)$ become useless.

The attack to break Whitenoise is illustrated below.

There are 172 prime numbers not larger than 1021. Denote the t^i $(1 \le i \le 172)$ as the i-th prime, i.e. $t^1 = 2$, $t^2 = 3$, $t^3 = 5$, $\cdots$, $t^{172} = 1021$. Consider all the 172 possible subkeys $\bar{s}^i$ $(1 \le i \le 172)$, and each $\bar{s}^i$ with t^i bytes. Now observe the keystream, if $y_j = y_k$, then we know that $z_j = z_k$. Consequently, we know that

$$\bar{s}^1_{j \bmod 2} \oplus \bar{s}^2_{j \bmod 3} \oplus \bar{s}^3_{j \bmod 5} \oplus \cdots \oplus \bar{s}^{172}_{j \bmod 1021} = \\ \bar{s}^1_{k \bmod 2} \oplus \bar{s}^2_{k \bmod 3} \oplus \bar{s}^3_{k \bmod 5} \oplus \cdots \oplus \bar{s}^{172}_{k \bmod 1021}$$

This equation leaks one byte information of the subkeys. Note that there are $2+3+5+7+\cdots+1021 = 80189$ bytes in all the subkeys. So with slightly more than 80189 such equations, we can recover all the 172 subkeys and consequently the secret S-box. We note that to obtain slightly more than 80189 equations as above, we only need slightly more than 80445 bytes of keystream. To solve these linear equations, the complexity is about $2^{48.4}$. If we implement this attack on the current Pentium 4 processor, the time required to break Whitenoise is estimated to be at most a few days.

In the original report, we stated that the subkeys can be obtained by solving those equations. Wagner has pointed out that unlikely the original subkeys could be recovered. Our experiments on the reduced versions of Whitenoise confirm that Wagner's observation is correct. Instead, we can only recover some equivalent subkeys or partial-equivalent subkeys to break the Whitenoise cipher.

We implemented the experiments on the reduced versions of Whitenoise. Our experiments show that it is quite easy to break Whitenoise. After solving the linear equations above, we can express all the subkeys in terms of some variables (we denote those variables as a set V). Randomly set those variables in the set V as different values, then we obtain a set of subkeys. Only in the very rare case, the sequence generated from such subkeys is with only one value in it, i.e., all the outputs are the same. For most of the subkeys, this set of subkeys is either a set of equivalent subkeys or a set of partial-equivalent subkeys. Here **the concept of equivalent subkeys is that from the equivalent subkeys, we can generate a sequence that can be transformed into the original keystream with a permutation table** (This concept is different from the direct explanation that the sequence generated from the equivalent subkeys must be the same as those generated from the original subkeys). If we obtained such a set of equivalent subkeys, we can generate the rest of the keystream, as illustrated in Subsection 4.1. The concept of partial-equivalent subkeys is that from the equivalent subkeys, we can generate a sequence such that the original keystream can be transformed into this sequence with an n-to-1 ($1 < n \leq 128$) mapping table. With several sets of partial-equivalent subkeys, we are able to recover the rest of the keystream, as illustrated in Subsection 4.2.

4 Experiments

In the experiments, we deal with the reduced versions of Whitenoise. We use 4-bit data instead of the 8-bit data in the original Whitenoise cipher. In Subsection 4.1, 4 subkeys are used. In Subsection 4.2, 6 subkeys are used. In these two experiments, it is assumed that the attacker knows a small piece of the keystream, then the attacker tries to recover the rest of the keystream. The experiments show that it is quite easy to recover the rest of the keystream.

4.1 Experiment one – using one set of equivalent subkeys

Suppose that 4 subkeys are used. The first subkey is with value 0. Each element of a subkey is a 4-bit number. We randomly set the values of those subkeys as:

A = {0, 0, 0};
B = {2, 12, 4, 9, 3};
C = {6, 1, 3, 9, 0, 2, 3};
D = {1, 6, 15, 13, 10, 13, 0, 6, 12, 6, 9};

We randomly set the secret permutation as:

S = {4, 15, 0, 9, 13, 5, 3, 7, 14, 1, 11, 6, 10, 2, 12, 8};

Generate the first 256 outputs as:

L = { 5, 6, 14, 2, 1, 2, 8, 13, 13, 3, 0, 2, 4, 5, 14, 1,
0, 2, 8, 2, 7, 9, 13, 10, 5, 8, 13, 11, 8, 13, 2, 9,
2, 11, 3, 6, 4, 2, 2, 9, 3, 9, 13, 15, 15, 2, 9, 6,
4, 14, 9, 1, 15, 8, 14, 4, 10, 11, 7, 4, 8, 12, 15, 9,
13, 14, 13, 0, 13, 2, 12, 4, 7, 3, 8, 3, 3, 9, 12, 8,
3, 3, 6, 11, 9, 8, 1, 13, 14, 7, 12, 7, 8, 7, 11, 13,
0, 15, 3, 9, 7, 11, 1, 15, 2, 13, 6, 6, 3, 11, 15, 1,
2, 5, 11, 3, 10, 4, 3, 9, 11, 12, 6, 3, 10, 6, 7, 5,
10, 3, 13, 7, 3, 1, 2, 10, 8, 1, 6, 3, 14, 6, 12, 15,
5, 8, 0, 14, 5, 4, 2, 4, 4, 9, 13, 5, 4, 4, 9, 10,
15, 10, 1, 10, 9, 9, 14, 14, 0, 14, 10, 5, 0, 7, 3, 2,

10, 15, 8, 12, 11, 0, 0, 12, 10, 2, 5, 7, 10, 11, 12, 5,
4, 1, 7, 2, 10, 10, 6, 10, 2, 9, 2, 0, 0, 7, 1, 0,
0, 15, 0, 0, 11, 11, 12, 4, 1, 12, 12, 1, 11, 11, 1, 7,
8, 12, 8, 6, 5, 7, 14, 6, 9, 5, 7, 14, 9, 7, 1, 12,
7, 10, 5, 2, 8, 1, 7, 11, 4, 5, 10, 1, 6, 1, 3, 13}

Assume that the attacker knows the values of the first 60 outputs. Then the attacker wants to generate the rest of the keystream. From the first 60 outputs, the attacker obtains 45 equations. Solve those equations, we obtain the following relations:

```
A[0]=A[1];
A[2]=A[1];
C[0]=B[0]^B[1]^B[3]^B[4]^C[5];
C[1]=B[0]^B[1]^B[2]^B[3]^C[5];
C[2]=B[0]^B[4]^C[5];
C[6]=B[0]^B[4]^C[5];
C[3]=B[0]^B[3]^C[5];
C[4]=B[1]^B[2]^B[3]^B[4]^C[5];
D[10]=B[1]^B[2]^D[0];
D[1]=B[2]^B[4]^D[0];
D[9]=B[2]^B[4]^D[0];
D[8]=B[2]^B[3]^D[0];
D[3]=B[0]^B[2]^B[3]^B[4]^D[0];
D[5]=B[0]^B[2]^B[3]^B[4]^D[0];
D[6]=B[0]^B[4]^D[0];
D[4]=B[0]^B[3]^D[0];
D[7]=B[2]^B[4]^D[0];
D[2]=B[0]^B[1]^D[0];
```

That is to say, all variables can be expressed in terms of 8 variables $\{$A[1], B[0],B[1],B[2], B[3],B[4], C[5], D[0] $\}$. We also know that the value of A is 0 since A[0] =A[1]=A[2].

Denote the equivalent subkeys as A', B', C' and D'. Set

```
A'[1] = 3;
B'[0] = 1;
B'[1] = 2;
B'[2] = 4;
B'[3] = 8;
B'[4] = 9;
C'[5] = 11;
D'[0] = 12;
```

Compute A', B', C', D' from these 8 variables, we obtain a set of equivalent subkeys:

A' = {3, 3, 3};
B' = {1, 2, 4, 8, 9};
C' = {9, 4, 3, 2, 12, 11, 3};
D' = {12, 1, 15, 8, 5, 8, 4, 1, 0, 1, 10};

From A', B', C' and D', generate keystream (without using the S-box). The first 256 outputs are

L' = {7, 4, 11, 1, 3, 1, 6, 15, 15, 8, 10, 1, 13, 7, 11, 3,
10, 1, 6, 1, 0, 2, 15, 9, 7, 6, 15, 12, 6, 15, 1, 2,
1, 12, 8, 4, 13, 1, 1, 2, 8, 2, 15, 5, 5, 1, 2, 4,
13, 11, 2, 3, 5, 6, 11, 13, 9, 12, 0, 13, 6, 14, 5, 2,
15, 11, 15, 10, 15, 1, 14, 13, 0, 8, 6, 8, 8, 2, 14, 6,
8, 8, 4, 12, 2, 6, 3, 15, 11, 0, 14, 0, 6, 0, 12, 15,
10, 5, 8, 2, 0, 12, 3, 5, 1, 15, 4, 4, 8, 12, 5, 3,
1, 7, 12, 8, 9, 13, 8, 2, 12, 14, 4, 8, 9, 4, 0, 7,
9, 8, 15, 0, 8, 3, 1, 9, 6, 3, 4, 8, 11, 4, 14, 5,
7, 6, 10, 11, 7, 13, 1, 13, 13, 2, 15, 7, 13, 13, 2, 9,
5, 9, 3, 9, 2, 2, 11, 11, 10, 11, 9, 7, 10, 0, 8, 1,
9, 5, 6, 14, 12, 10, 10, 14, 9, 1, 7, 0, 9, 12, 14, 7,
13, 3, 0, 1, 9, 9, 4, 9, 1, 2, 1, 10, 10, 0, 3, 10,
10, 5, 10, 10, 12, 12, 14, 13, 3, 14, 14, 3, 12, 12, 3, 0,
6, 14, 6, 4, 7, 0, 11, 4, 2, 7, 0, 11, 2, 0, 3, 14,
0, 9, 7, 1, 6, 3, 0, 12, 13, 7, 9, 3, 4, 3, 8, 15 };

Comparing the first 60 outputs of L' with that of L, we obtain the following permutation table S' that transforms the first 60 outputs of L' into that of L.

S' = { 7, 2, 9, 1, 6, 15, 8, 5, 3, 10, 0, 14, 11, 4, 12, 13 };

Apply this S' to all the elements L', we obtained a sequence that is exactly the same as L.

So from the first 60 outputs of the keystream L, we obtain the following set of equivalent subkeys that breaks the Whitenoise:

A' = {3, 3, 3};
B' = {1, 2, 4, 8, 9};
C' = {9, 4, 3, 2, 12, 11, 3};
D' = {12, 1, 15, 8, 5, 8, 4, 1, 0, 1, 10};
S' = {7, 2, 9, 1, 6, 15, 8, 5, 3, 10, 0, 14, 11, 4, 12, 13};

4.2 Experiment two – using two sets of partial-equivalent subkeys

Suppose that 6 subkeys are used. The second subkey is with value 0. Each element of a subkey is a 4-bit number. We randomly set the values of those subkeys as:

A = {4, 11, 11};
B = {0, 0, 0, 0, 0};
C = {7, 4, 11, 6, 12, 6, 6};
D = {13, 12, 11, 4, 8, 15, 14, 2, 2, 1, 15};
E = {2, 0, 4, 15, 8, 8, 5, 14, 2, 8, 5, 14, 1};
F = {11, 4, 5, 7, 8, 15, 4, 13, 6, 3, 8, 2, 0, 2, 13, 13, 13};

We randomly set the secret permutation as:

S = {2, 12, 7, 11, 15, 3, 8, 10, 5, 4, 9, 0, 13, 1, 6, 14 };

Generate the first 256 outputs as:

L = {10, 10, 9, 6, 14, 3, 1, 1, 4, 3, 14, 8, 14, 8, 3, 12,
1, 2, 8, 14, 3, 8, 3, 1, 9, 15, 15, 3, 15, 7, 10, 4,
2, 13, 5, 0, 15, 12, 0, 0, 0, 8, 0, 0, 1, 4, 7, 4,
14, 0, 7, 10, 4, 11, 6, 10, 10, 10, 13, 1, 8, 11, 5, 14,
1, 2, 7, 7, 7, 9, 4, 7, 10, 7, 4, 0, 9, 11, 8, 4,
1, 1, 10, 8, 4, 5, 10, 2, 5, 10, 10, 6, 9, 7, 14, 3,
9, 4, 14, 7, 4, 6, 4, 2, 3, 9, 12, 7, 14, 1, 11, 11,
1, 12, 7, 6, 3, 14, 7, 7, 15, 2, 6, 6, 5, 11, 4, 7,
14, 4, 8, 7, 4, 7, 12, 6, 0, 7, 0, 2, 11, 2, 9, 14,
7, 12, 6, 12, 5, 4, 11, 9, 4, 11, 0, 10, 12, 12, 15, 8,
8, 2, 15, 10, 7, 1, 8, 3, 0, 5, 15, 13, 14, 2, 11, 4,
5, 13, 5, 3, 13, 12, 2, 2, 0, 6, 7, 11, 2, 11, 6, 5,
10, 11, 11, 15, 6, 13, 14, 4, 8, 6, 8, 13, 1, 11, 15, 14,
5, 14, 14, 8, 9, 12, 3, 14, 7, 15, 8, 11, 13, 7, 1, 5,
13, 14, 4, 6, 4, 5, 4, 11, 14, 9, 8, 7, 0, 6, 1, 12,
6, 0, 2, 4, 10, 0, 14, 6, 0, 13, 2, 8, 4, 9, 14, 2};

Assume that the attacker knows the first 90 outputs. Then the attacker wants to obtain the rest of the keystream. From the first 90 outputs, obtain 74 equations. Solve those equations, we obtain that

```
F[3]=C[0]^C[1]^F[1];
F[8]=C[1]^C[3]^F[1];
F[2]=C[0]^C[3]^F[1];
F[7]=C[0]^C[1]^C[3]^C[4]^F[1];
F[0]=A[0]^A[1]^F[1];
E[10]=C[0]^C[4]^E[7];
F[13]=A[0]^A[1]^C[0]^C[1]^C[3]^C[4]^F[1];
F[10]=A[0]^A[1]^C[0]^C[1]^F[1];
F[6]=F[1];
F[4]=A[0]^A[1]^C[0]^C[1]^F[1];
```

```
F[14]=C[0]^C[1]^C[3]^C[4]^F[1];
E[6]=C[0]^C[4]^E[7];
E[2]=C[3]^C[4]^E[7];
D[2]=A[0]^A[1]^C[1]^C[4]^D[1];
E[5]=A[0]^A[1]^C[0]^C[1]^C[3]^C[4]^E[7];
E[9]=A[0]^A[1]^C[0]^C[1]^C[3]^C[4]^E[7];
E[0]=A[0]^A[1]^C[0]^C[1]^E[7];
D[6]=C[1]^C[3]^D[1];
D[8]=A[0]^A[1]^C[0]^C[3]^D[1];
C[6]=C[3];
B[4]=B[0];
C[5]=C[3];
B[1]=B[0];
A[2]=A[1];
B[2]=B[0];
B[3]=B[0];
C[2]=A[0]^A[1]^C[1];
D[9]=A[0]^A[1]^C[1]^C[3]^D[1];
D[5]=C[0]^C[1]^D[1];
D[7]=A[0]^A[1]^C[0]^C[3]^D[1];
D[0]=C[0]^C[3]^D[1];
D[3]=C[1]^C[4]^D[1];
E[4]=A[0]^A[1]^C[0]^C[1]^C[3]^C[4]^E[7];
E[1]=A[0]^A[1]^C[0]^C[3]^E[7];
E[8]=A[0]^A[1]^C[0]^C[1]^E[7];
E[3]=C[0]^C[3]^E[7];
D[4]=A[0]^A[1]^C[0]^C[4]^D[1];
E[12]=A[0]^A[1]^E[7];
F[9]=A[0]^A[1]^C[1]^C[4]^F[1];
E[11]=E[7];
F[11]=A[0]^A[1]^C[0]^C[1]^C[3]^C[4]^F[1];
D[10]=C[0]^C[1]^D[1];
F[5]=C[0]^C[4]^F[1];
F[16]=C[0]^C[1]^C[3]^C[4]^F[1];
F[15]=C[0]^C[1]^C[3]^C[4]^F[1];
F[12]=A[0]^A[1]^C[0]^C[4]^F[1];
```

It is the same as to say that all the variables can be expressed in terms of ten variables {A[0], A[1], B[0], C[0], C[1], C[3], C[4], D[1], E[7], F[1]}. Here we know the value B is 0 since all its elements are equal.

Denote the equivalent subkeys as A', B', C', D' and E'. Randomly set

```
B'[0] = 0;
A'[0] = 1;
A'[1] = 2;
```

```
C'[0] = 4;
C'[1] = 8;
C'[3] = 9;
C'[4] = 10;
D'[1] = 11;
E'[7] = 12;
F'[1] = 13;
```

We obtain the following equivalent subkeys

A' = {1, 2, 2 };
B' = {0, 0, 0, 0, 0};
C' = {4, 8, 11, 9, 10, 9, 9};
D' = {6, 11, 10, 9, 6, 7, 10, 5, 5, 9, 7};
E' = {3, 2, 15, 1, 0, 0, 2, 12, 3, 0, 2, 12, 15};
F' = {14, 13, 0, 1, 2, 3, 13, 2, 12, 12, 2, 1, 0, 1, 2, 2, 2};

From these subkeys, we generate 256 outputs

L' = {14, 14, 12, 1, 12, 15, 13, 13, 0, 15, 12, 3, 12, 3, 15, 2,
13, 15, 3, 12, 15, 3, 15, 13, 12, 2, 2, 15, 2, 14, 14, 0,
15, 0, 13, 1, 2, 2, 1, 1, 1, 3, 1, 1, 13, 0, 14, 0,
12, 1, 14, 14, 0, 3, 1, 14, 14, 14, 0, 13, 3, 3, 13, 12,
13, 15, 14, 14, 14, 12, 0, 14, 14, 14, 0, 1, 12, 3, 3, 0,
13, 13, 14, 3, 0, 13, 14, 15, 13, 14, 14, 1, 12, 14, 12, 15,
12, 0, 12, 14, 0, 1, 0, 15, 15, 12, 2, 14, 12, 13, 3, 3,
13, 2, 14, 1, 15, 12, 14, 14, 2, 15, 1, 1, 13, 3, 0, 14,
12, 0, 3, 14, 0, 14, 2, 1, 1, 14, 1, 15, 3, 15, 12, 12,
14, 2, 1, 2, 13, 0, 3, 12, 0, 3, 1, 14, 2, 2, 2, 3,
3, 15, 2, 14, 14, 13, 3, 15, 1, 13, 2, 0, 12, 15, 3, 0,
13, 0, 13, 15, 0, 2, 15, 15, 1, 1, 14, 3, 15, 3, 1, 13,
14, 3, 3, 2, 1, 0, 12, 0, 3, 1, 3, 0, 13, 3, 2, 12,
13, 12, 12, 3, 12, 2, 15, 12, 14, 2, 3, 3, 0, 14, 13, 13,
0, 12, 0, 1, 0, 13, 0, 3, 12, 12, 3, 14, 1, 1, 13, 2,
1, 1, 15, 0, 14, 1, 12, 1, 1, 0, 15, 3, 0, 12, 12, 15 };

Here, there are only 8 different values in L', the mapping L' to L is not one-to-one. We obtained the following mapping

```
S'={
0  --> 4 or 13;
1  --> 0 or 6;
2  --> 12 or 15;
3  --> 8 or 11;
12 --> 9 or 14;
13 --> 1 or 5;
```

```
14 --> 7 or 10;
15 --> 2 or 3;}
```

Apply this mapping S' to L', we could not obtain the exact L since every element in L' has two possible values after the mapping. So we need to filter out the wrong values by using one more set of partial-equivalent subkeys.

To do that, we choose another set of subkeys A", B", C", D", E" and F". Set

```
B''[0] = 0;
A''[0] = 8;
A''[1] = 4;
C''[0] = 2;
C''[1] = 1;
C''[3] = 9;
C''[4] = 10;
D''[1] = 11;
E''[7] = 12;
F''[1] = 13;
```

Compute those subkeys from these 10 variables, we obtain the subkeys

A" = {8, 4, 4};
B" = {0, 0, 0, 0, 0};
C" = {2, 1, 13, 9, 10, 9, 9};
D" = {0, 11, 12, 0, 15, 8, 3, 12, 12, 15, 8};
E" = {3, 11, 15, 7, 0, 0, 4, 12, 3, 0, 4, 12, 0};
F" = {1, 13, 6, 14, 2, 5, 13, 13, 5, 10, 2, 1, 9, 1, 13, 13, 13};

From these subkeys, we generate the 256 outputs

L" = { 8, 8, 12, 8, 3, 0, 11, 11, 15, 0, 3, 3, 3, 3, 0, 4,
11, 15, 3, 3, 0, 3, 0, 11, 12, 11, 11, 0, 11, 7, 8, 15,
15, 0, 4, 7, 11, 4, 7, 7, 7, 3, 7, 7, 11, 15, 7, 15,
3, 7, 7, 8, 15, 12, 8, 8, 8, 8, 0, 11, 3, 12, 4, 3,
11, 15, 7, 7, 7, 12, 15, 7, 8, 7, 15, 7, 12, 12, 3, 15,
11, 11, 8, 3, 15, 4, 8, 15, 4, 8, 8, 8, 12, 7, 3, 0,
12, 15, 3, 7, 15, 8, 15, 15, 0, 12, 4, 7, 3, 11, 12, 12,
11, 4, 7, 8, 0, 3, 7, 7, 11, 15, 8, 8, 4, 12, 15, 7,
3, 15, 3, 7, 15, 7, 4, 8, 7, 7, 7, 15, 12, 15, 12, 3,
7, 4, 8, 4, 4, 15, 12, 12, 15, 12, 7, 8, 4, 4, 11, 3,
3, 15, 11, 8, 7, 11, 3, 0, 7, 4, 11, 0, 3, 15, 12, 15,
4, 0, 4, 0, 0, 4, 15, 15, 7, 8, 7, 12, 15, 12, 8, 4,
8, 12, 12, 11, 8, 0, 3, 15, 3, 8, 3, 0, 11, 12, 11, 3,
4, 3, 3, 3, 12, 4, 0, 3, 7, 11, 3, 12, 0, 7, 11, 4,
0, 3, 15, 8, 15, 4, 15, 12, 3, 12, 3, 7, 7, 8, 11, 4,

8, 7, 15, 15, 8, 7, 3, 8, 7, 0, 15, 3, 15, 12, 3, 15};

Then we obtained the following mapping that transforms L" into L:

```
S'' = {
0 --> 3 or 13;
3 --> 8 or 14;
4 --> 5 or 12;
7 --> 0 or 7;
8 --> 6 or 10;
11 --> 1 or 15;
12 --> 9 or 11;
15 --> 2 or 4;}
```

Apply this mapping S" to L", we could not obtain the exact L since every element in L" has two possible values after the mapping.

But with (L', S') and (L", S"), we are able to recover the exact L. We illustrate it with examples.

To recover L[253],
From L', we know that L'[253] = 12.
Transformed by S', L[253] would be 9 or 14.
From L", we know that L"[253] = 12.
Transformed by S", L[253] would be 9 or 11.
So the wrong values are filtered out, and we obtain that L[253] = 9.
Looking up the original keystream L in Subsection 4.2, L[253] = 9.
The attack is successful.

To recover L[254],
From L', we know that L'[254] = 12.
Transformed by S', L[254] would be 9 or 14.
From L", we know that L"[254] = 3.
Transformed by S", L[254] would be 8 or 14.
So the wrong values are filtered out, and we obtain that L[254] = 14.
Again the attack is successful.

To recover L[255],
From L', we know that L'[255] = 15.
Transformed by S', L[255] would be 2 or 3.
From L", we know that L"[255] = 15.
Transformed by S", L[255] would be 2 or 4.
So the wrong values are filtered, and we obtain that L[255] = 2.
The attack is successful.

Repeat the similar process, we can recover all the elements of L.

Remarks. We discuss here how many partial-equivalent subkeys are required in the attack. Assume that the one-to-n mappings (such as the S' and S" in Subsection 4.2) are random. Then for the 8-bit data, eight sets of partial-equivalent subkeys would be sufficient to retrieve the original keystream. The reason is given below. The worst set partial-equivalent subkeys is that, there are only two different values in the output key stream. (Here we do not consider the subkeys from which the keystream generated are with all the outputs being the same as partial-equivalent subkeys). In that case, the mapping table maps each value in the new sequence into 128 values in the original keystream. With two sets of partial-equivalent keys, in average we are able to filter out 64 wrong values, i.e., each set of partial-equivalent subkeys is able to filter out half of the wrong values in average. So eight sets such partial-equivalent subkeys would be able to to filter out the wrong values. We thus conjecture that at most eight partial-equivalent keys are needed to break Whitenoise.

5 Conclusion

In this paper, we show that the Whitenoise cipher is insecure. And our experiments on the reduced versions of Whitenoise cipher show that the attack works well.