A Security Evaluation of Whitenoise

Final Report

David Wagner

October 8, 2003

Executive Summary

This report studies the security of Whitenoise, a stream cipher invented by BSB Utilities Inc. I present the results of a limited time analysis of this scheme.

After a careful security analysis, I was unable to find any security weaknesses in the Whitenoise stream cipher. Whitenoise resists all of the attack methods I was able to think of. This provides evidence for the security of Whitenoise.

To be sure, this is not a guarantee that Whitenoise is secure. It is possible that someone else might be able to find some weakness that I overlooked. Cryptographic canon is that no new algorithm should be deployed until it has been examined carefully by many qualified cryptanalysts. For this reason, it would be premature to deploy Whitenoise in new systems at this point.

However, the security analysis conducted in this study is an important first step in the process of gaining confidence in the quality of the Whitenoise stream cipher. My analysis does provide considerable evidence that Whitenoise is worth further study; it appears to be a serious candidate, and in my opinion, it deserves to be further vetted by the security community at large.

1 Introduction

BSB has proposed a new stream cipher, called Whitenoise.

Scope of my analysis. In this report, I evaluate the security of Whitenoise against modern cryptanalytic techniques. In particular, my goal was to check whether Whitenoise provides confidentiality protection[1] against ciphertext-only, known-plaintext, and chosen-plaintext cryptanalytic attacks. I sought to evaluate whether Whitenoise provides protection commensurate with what one would expect from a modern stream cipher and at the level required by typical applications. These are very demanding requirements.

The purpose of this report was not to serve as the last word on the security of Whitenoise or to make any final pronouncements about its readiness for deployment. Rather, the goal was to evaluate whether the design has merit and is deserving of further vetting.

I evaluated only the Whitenoise algorithm, as specified in the following design document:

> "Software Specifications for Tinnitus Utilizing Whitenoise Substitution Stream Cipher," BSB Utilities Inc., Last modified 7/29/2003.

I only evaluated the Tinnitus version of Whitenoise, and the conclusions of this report are intended to apply only to that instantiation of the general Whitenoise concept.

I did not evaluate any implementations of this stream cipher. Accordingly, I assume throughout this report that the cipher has been properly and accurately implemented.

Conclusions. The results of the analysis were promising. The Whitenoise approach appears to have merit, and this evaluation provided evidence that Whitenoise deserves further study.

2 Description of the Whitenoisestream cipher

Terminology. BSB Utilities have proposed a general framework for building stream ciphers, which they call "Whitenoise." They have also proposed a specific instantiation of this concept into a concrete stream cipher, which might be dubbed

[1] In the sense of IND-CPA. Whitenoise is not intended to provide integrity protection on its own, and I did not investigate integrity in this work. Like all other stream ciphers, in real deployments Whitenoise should be combined with other mechanisms that provide message integrity.

"Whitenoise for Tinnitus." Because this report analyzes only the "Whitenoise for Tinnitus" instantiation, in interests of brevity I will use the term "Whitenoise" from here on to refer specifically to that concrete instantiation rather than to the general framework.

Whitenoise. The general Whitenoise schema is composed of two separate components:

- First, a key setup component, which has the following interface:

  INPUTS:
  a key $K = (\text{seed1}, \text{seed2})$
  a 32-bit counter T
  OUTPUTS:
  an integer n, in the range $50, 51, .., 99$
  a list $(\ell^{(1)}, \ell^{(2)}, .., \ell^{(n)})$ of n lengths
  a list $(s^{(1)}, s^{(2)}, .., s^{(n)})$ of n sub keys
  an array $S[256]$ of bytes

  The counter T is different for each message, counting from a random starting point upwards. The sub key $s^{(i)}$ consists of $\ell^{(i)}$ bytes. The array $S[]$ holds a permutation on bytes. All the outputs of the key setup phase are kept secret.

- Second, a simple output generation component, consisting of the following interface:

  INPUTS:
  a plaintext P
  an integer n, in the range $50, 51, .., 99$
  a list $(\ell^{(1)}, \ell^{(2)}, .., \ell^{(n)})$ of n lengths
  a list $(s^{(1)}, s^{(2)}, .., s^{(n)})$ of n sub keys
  an array $S[256]$ of bytes
  OUTPUTS:
  a ciphertext C, of the same length as P; it is the encryption of P

When these two components are composed, we obtain the Whitenoise stream cipher.

The size of the key K is a security parameter that can be varied based on the application and the level of security desired. We will assume that all parameters are chosen according to the guidelines set out in the specification document. As always, we make the standard assumption that all security rests in the secret key, i.e., the two "seeds."

The output generation component. The mechanism for computing the "SuperKey" from the "sub keys" works as follows. Let $s_j^{(i)}$ denote the j-th byte of the i-th "sub key". Let $\ell^{(i)}$ denote the length of the i-th "sub key." Associate to "sub key" i the unending sequence of bytes

$$s_1^{(i)}, s_2^{(i)}, s_3^{(i)}, \ldots, s_{\ell^{(i)}}^{(i)}, s_1^{(i)}, s_2^{(i)}, \ldots, s_{\ell^{(i)}}^{(i)}, s_1^{(i)}, s_2^{(i)}, \ldots$$

By abuse of notation, we can let $s_j^{(i)}$ denote the j-th byte of the above sequence, if j is any natural number; or, in other words, we implicitly reduce the subscript of $s_j^{(i)}$ modulo $\ell^{(i)}$. Then, the j-th byte of the "SuperKey," call it z_j, is defined by

$$z_j \stackrel{\text{def}}{=} s_j^{(1)} \oplus s_j^{(2)} \oplus \cdots \oplus s_j^{(n)}.$$

Here, "$\oplus$" denotes the XOR operation. In other words, to be more explicit,

$$z_j \stackrel{\text{def}}{=} s_{j \bmod \ell^{(1)}}^{(1)} \oplus s_{j \bmod \ell^{(2)}}^{(2)} \oplus \cdots \oplus s_{j \bmod \ell^{(n)}}^{(n)}$$

where $j \bmod \ell$ returns an integer in the range $1, 2, \ldots, \ell$. Finally, we compute

$$C_j \stackrel{\text{def}}{=} S[z_j] \oplus P_j$$

The key setup component. The key setup component is fully described in the specification document.

3 Attacks that are not a threat

I tried a number of different mathematical shortcut attacks to see if any them could perhaps be used to break Whitenoise. However, none of them were successful; Whitenoise provided more than sufficient security against these attacks. In this section, I document these failed attempts to break Whitenoise.

Exhaustive keysearch is not a threat. With the recommended parameters, Whitenoise uses keys with at least 1600 bits of randomness. Exhaustive search of 1600-bit keys is completely and absolutely infeasible. Even if we hypothesized the existence of some magic computer that could test a trillion trillion key trials per second (very unlikely!), and even if we could place a trillion trillion such computers somewhere throughout the universe (even more unlikely!), and even if we were willing to wait a trillion trillion years (not a chance!), then the probability that we would discover the correct key would be negligible (about $1/2^{1340}$, which is unimaginably small).

Hence, if keys are chosen appropriately and Whitenoise is implemented correctly, exhaustive keysearch is not a threat.

Keystream reuse attacks are not a threat. The period of Whitenoise is very large: it is at least $\text{lcm}(\ell^{(1)}, \ldots, \ell^{(n)}) \geq 2 \times 3 \times \cdots \times 229 \geq 2^{303}$. Also, a different counter value is used for each message encrypted[2]. Hence, there is no risk that the SuperKey or the keystream will repeat at any time. This means that naive attacks based on hoping that the keystream will eventually repeat itself are completely and totally infeasible.

Attacks on two-tape systems are not a threat. Note that the lengths of the subkeys are kept secret from the cryptanalyst. Also, the SuperKey is post-processed using a nonlinear substitution $S[]$. Hence, the standard linear algebra attacks on multiple-loop Vigenere ciphers do not pose any risk to Whitenoise.

Berlekamp-Massey attacks are not a threat. If the nonlinear substitution $S[]$ were omitted, the system would be insecure. Because everything else in the output generation component is linear, and because the SuperKey is constructed by xoring together n low-complexity sequences, the SuperKey has low linear complexity and can be equivalently generated by some fairly small (key-dependent) LFSR. If we were unwise enough to xor the SuperKey directly against the plaintext, then the system would become vulnerable to Berlekamp-Massey attacks.

However, Whitenoise is not vulnerable to these attacks. The presence of the substitution $S[]$ in Whitenoise defends against Berlekamp-Massey, because it introduces nonlinearity. This nonlinearity dramatically increases the linear complexity of the sequence xor-ed against the plaintext; thus, Berlekamp-Massey attacks would apparently require an astronomically infeasible quantity of known plaintext, which means that Berlekamp-Massey attacks will be unsuccesfull in practice. Thus, it seems that Whitenoise provides more than adequate security against Berlekamp-Massey attacks.

We can estimate how much known plaintext would be required, based on heuristic assumptions. Suppose we examine just a single bit of the 8-bit output of $S[]$, say, the high bit in every position. This gives a bit stream (decimated from the original keystream by a factor of 8) whose linear complexity we can then estimate. Notice that this stream is obtained by applying some random nonlinear function $f : \{0,1\}^8 \to \{0,1\}$ to the input stream of S; in fact, $f(z_i)$ is just the high bit of $S[z_i]$. Also, each of the 8 bits of input z_i to f can be viewed as coming from a separate one of 8 independent LFSRs. (This view is valid because there is no mixing between the bit positions of the internal state; the high bit of $s^{(i)}$ evolves separately from the low bit, for instance.) Thus, we can view this stream as coming

[2] I assume that at most 2^{32} messages can be encrypted, and that the counter T is not allowed to wrap around.

from a nonlinear combination generator, a construct that has been well-analyzed in the literature. The following is a standard result regarding the linear complexity of nonlinear combination generators, quoted from Fact 6.49 of the *Handbook of Applied Cryptography*:

Theorem 1. *Let $f(x_1,\ldots,x_8)$ be a nonlinear function expressed in algebraic normal form. Suppose that we have a nonlinear combination generator with n LFSRs of length $L_1,\ldots,L_8$ combined by f. If the lengths are pairwise distinct and all greater than two, then the linear complexity of the keystream is $f(L_1,\ldots,L_8)$ (evaluated over the integers).*

In the case of Whitenoise, we have $L_1 = \cdots = L_8 = \ell^{(1)} + \ldots + \ell^{(n)}$, where n is uniformly distributed on $\{50,\ldots,99\}$ and $\ell^{(i)}$ is uniformly distributed on the set of primes $\{2,3,5,7,11,\ldots,1021\}$. Hence the lengths L_i are likely to be on the order of $20,000$ to $40,000$. Since f behaves like a random function, the nonlinear order of f is likely to be 7 or 8, hence $f(L_1,\ldots,L_8)$ can be expected to be at least $(20,000)^7 \approx 2^{100}$ in typical practice.

This suggests that Berlekamp-Massey attacks on Whitenoise might require 2^{100} bits of known plaintext or more, which would make this class of attacks completely impractical. There are two facts that make this analysis heuristic, though. First, we focused on looking at just one bit of each byte of output; this analysis does not consider the information one might be able to gain by looking at the entire byte as a whole. Second, the Theorem listed above does not actually apply to Whitenoise, because Whitenoise's lengths are not distinct. I expect this latter exception to be unimportant, though; it seems there is no strict requirement for distinct lengths, and the appearance of this condition in the statement of the Theorem seems to be motivated primarily by a desire to avoid complicating the statement of the Theorem. When lengths are not distinct and chosen as Whitenoise does, the probability (taken over the choice of key) that the linear complexity is significantly lower than expected is negligible. For these reasons, I do not expect Berlekamp-Massey attacks to succeed.

Divide-and-conquer attacks are not a threat. Divide-and-conquer attacks work by guessing a part of the secret internal state of the stream cipher and then attempting to analytically deduce the rest of the state. However, because Whitenoise mixes together so much secret key material, guessing any small portion of the internal state does not appear to help. Therefore, divide-and-conquer attacks seem unlikely to work.

Correlation attacks do not seem to be a threat. Correlation attacks are applicable on nonlinearly filtered systems when some significant linear correlation can

be found between the bits of the inputs and outputs to the nonlinear function. For instance, if we could find linear functions $\ell, \ell' : \{0,1\}^8 \rightarrow \{0,1\}$ such that

$$|\Pr[\ell'(S[x]) = \ell(x)] - \frac{1}{2}|$$

is large (where the probability is taken over the choice of a uniformly random value x), this would provide a linear correlation in $S[]$ that one might be able to exploit in a correlation attack on Whitenoise.

However, several factors make correlation attacks appear unlikely to succeed.

1. The nonlinear substitution $S[]$ is chosen pseudorandomly. Hence, if there is a linear correlation, the bias is unlikely to be exceptionally large, and this means that a good deal of known plaintext would be required. (However, see the bias mentioned below for one countervailing effect that might increase the magnitude of the bias.)

2. It is not clear how to find ℓ, ℓ'. Any correlation attack would need to know, at a minimum, the linear function ℓ'. However, the choice of an effective ℓ' is likely to depend on the specific $S[]$ in use, and $S[]$ is kept secret from the attacker. An attacker might guess ℓ' blindly, or try all possibilities, but this would increase the attacker's workfactor.

3. More seriously, the size of the internal linear system is likely to pose significant challenges for any nonlinear correlation attack. The complexity of known methods increases rapidly with the size of the internal state of the system. I am not aware of any successful linear correlation attack applied to a system with 320,000 bits of internal state (which is the case for Whitenoise).

4. Most dauntingly, the linear recurrence for the internal state is not known. I do not know of any efficient technique for mounting fast correlation attacks when the linear recurrence (or feedback polynomial) is unknown.

In short, as far as I can tell, mounting a correlation attack on Whitenoise would likely require major advances in cryptanalysis. Consequently, correlation attacks do not appear to be a significant threat to the security of Whitenoise.

Coppersmith-style linear algebra attacks are not a threat. In some of his research, Coppersmith has pioneered a clever method for removing the effect of unknown linear substitutions. I tried applying his method to Whitenoise, but found that the approach fails due to the key-dependent linear recurrence used to update the internal state.

To illustrate its relevance to Whitenoise, suppose that we knew the lengths $\ell^{(1)}, \ldots, \ell^{(n)}$, so that the linear recurrence underlying the state update function was known. Let $S^{-1}[]$ denote the inverse of the $S[]$ substitution, i.e., $S^{-1}[y]$ represents the unique value x such that $S[x] = y$. In a known-plaintext attack, for each byte of known plaintext we could write an equation of the form

$$S^{-1}[P_j \oplus C_j] = s^{(1)}_{j \bmod \ell^{(1)}} \oplus s^{(2)}_{j \bmod \ell^{(2)}} \oplus \cdots \oplus s^{(n)}_{j \bmod \ell^{(n)}}.$$

I claim this equation can be linearized. Introduce 256 formal unknowns for the values $S^{-1}[0]$, $S^{-1}[1]$, etc. In a known-plaintext attack, $P_j \oplus C_j$ is known, hence the left-hand side of the equation above is one of our 256 unknowns (and we can tell which of these unknowns is implicated in the equation). Next introduce $\ell^{(1)}$ formal unknowns for the values $s^{(1)}_1$, $s^{(1)}_2$, etc., $\ell^{(2)}$ formal unknowns for $s^{(2)}$, and so on. If all lengths $\ell^{(i)}$ are known, then the right-hand side of the above equation is the sum of n of our unknowns (and we can tell which of the unknowns are implicated). Here we have used that the XOR is the same as the sum in $GF(2)$. The result of all this maneuvering is that we obtain one equation per byte of known plaintext, and this is a linear equation. After obtaining sufficiently many bytes of known text, we will have an over-determined system of linear equations, and then we can attempt to solve for the unknowns. There remain many nagging details to work out, but if the stars all align and the attacker is fortunate, the solution to this system of equations may reveal the internal state and thereby break the cipher.

However, this entire attack is premised on knowledge of the lengths $\ell^{(1)}, \ldots, \ell^{(n)}$. In Whitenoise, these lengths are not known to the attacker: they are chosen randomly in a key-dependent manner and kept secret. An attacker could attempt to guess the set of lengths $\ell^{(1)}, \ldots, \ell^{(n)}$, and for each guess, try the above attack. However, in Whitenoise, there are far too many possibilities for such an enumeration attack to have any hope of succeeding. Indeed, there are $\binom{172}{n} \geq \binom{172}{50} \approx 2^{150}$ possibilities, and so any such attack would be completely infeasible.

I don't see any obvious way to exploit biases in the key setup procedure. The key setup procedure does have certain biases inherent in the way it selects the lengths $\ell^{(1)}, \ldots, \ell^{(n)}$ as well the substitution function $S[]$. The key setup algorithm works like this:

1. For $i := 0, 1, \ldots, 255$, do:
2. Choose $S[i]$ pseudorandomly from $\{0, 1, \ldots, 255\}$.
3. While $S[i]$ is equal to one of $S[0], \ldots, S[i-1]$, do:
4. Set $S[i] := S[i] + 1 \bmod 256$.

Suppose in the first iteration we select, say, $S[0] = 157$. What is the distribution of $S[1]$? In fact, $S[1] = 158$ happens with probability $2/256$, because there are two ways it can happen (either we initially pick 158, or we initially pick 157 and then are forced to increment it), while all other values of $S[1]$ appear with probability $1/256$. This is a bias, and the bias can grow stronger with more iterations of the algorithm. At the *i*-th iteration, if all of the numbers $157, 158, \ldots, 171$ (for example) appear in $S[0], \ldots, S[i-1]$, then the number 172 has a $16/256$ chance of being selected for $S[i]$. In general, if we've previously selected a string of *m* consecutive integers, then the immediately following integer has a $(m+1)/256$ probability of selection, which for large *m* is a large bias. Also, such long strings have a tendency to grow longer.

Nonetheless, despite the existence of this bias, I was unable to find any way to exploit the bias. It is possible that this bias in the selection of $S[]$ might increase the magnitude of the bias of the best linear relation for $S[]$; I did not test this. However, in any case, I could not find any way to use this bias in an attack on Whitenoise.

If this property is judged undesirable, it would be easy to modify the key setup procedure to eliminate this bias. (See, for instance, the discussion of how to generate a uniformly random permutation, also known as fair shuffling, in Knuth, "The Art of Computer Programming," Volume 2.)

4 Discussion

4.1 My attack attempts all failed

In a concerted attempt to break Whitenoise, I tried many creative attacks, including both standard ideas as well as off-the-wall non-standard methods. I was unable to come up with any approach that could make a dent in Whitenoise. I cannot think of any method that holds promise for attacking Whitenoise.

Of course, this is no guarantee that Whitenoise resists all attacks; maybe there is some exceptionally clever attack that I was simply unable to discover. However, it seems that some new cryptanalytic ideas would likely be needed before there is any chance that Whitenoise could be broken.

4.2 Whitenoise appears to be engineered with a margin of safety

None of the attacks I tried against Whitenoise even came close to succeeding. With 1600–2400-bit keys and 50–100 sub keys, Whitenoise has a large margin of safety against all the attacks I could think of.

Even if we drastically simplified the cipher, it would still appear to be resist all the attacks I can think of. For instance, suppose we modified Whitenoise to use

only 10 sub keys, instead of 50 or more sub keys. Intuitively, 50 sub keys ought to be much harder to break than 10 sub keys. However, even only 10 sub keys are enough to resist all the attacks I could think of. Note, for instance, that $\binom{172}{10} \approx 2^{52}$, hence (even after taking into account the possibility of sorting the list of lengths) there are too many possibilities to guess the list of lengths $(\ell^{(1)}, \ell^{(2)}, .., \ell^{(n)})$ and then try to perform some attack (e.g., a Coppersmith-style attack) based on knowledge of this list. Likewise, even with only 10 sub keys, the period of the SuperKey will still be very large: I estimate that a period less than 2^{50} will occur for at most 2^{-40} of the keys, and probably far less often than this.

Consequently, it may be possible to greatly reduce the security parameters in the Whitenoise stream cipher and still retain adequate security. Such a reduction could make the performance of Whitenoise quite attractive.

4.3 The performance of Whitenoise may be attractive

I did not attempt to evaluate the expected performance level of software or hardware implementations of Whitenoise. Producing such estimates is not my expertise. However, an initial off-the-cuff estimate suggests that, if we used only 10 sub keys, it might be possible to encrypt at speeds of 5–10 clock cycles per byte on a modern 32-bit processor, and even faster on 64-bit processors. This compares favorably to competitors such as AES and 3DES, and it is in the same ballpark as the fastest known stream ciphers (e.g., RC4, SEAL2, SNOW). It is possible that the simple structure of Whitenoise might provide additional opportunities for optimization.

Because I am not an expert in performance evaluation, these estimates can only be viewed as suggestive speculations. However, they make Whitenoise look potentially interesting for high-bandwidth, speed-sensitive applications and might serve as a sensible motivation to further study the security of Whitenoise.

5 Conclusions

5.1 Whitenoise appears adequate for cryptographic purposes

In this report, I tried every attack I could think of. All of them failed; Whitenoise resisted them. This provides evidence for the hypothesis that Whitenoise is cryptographically secure.

5.2 Whitenoise appears adequate for non-cryptographic purposes

Whitenoise also appears adequate as a pseudorandom number generator (PRNG) for statistical, Monte Carlo, and related purposes. Most statistical applications are less demanding than cryptographic applications, hence any generator that is cryptographically secure will be adequate for statistical applications. To the best of my knowledge, the Whitenoise concept appears more than adequate for most statistical (non-cryptographic) applications, so long as it is seeded and implemented properly.

5.3 Summary

This report gave a first security analysis of the Whitenoise stream cipher. The results were encouraging. More study and attention seems warranted.