Chosen Ciphertext Attack on SSS

Joan Daemen[1], Joseph Lano[2] *, and Bart Preneel[2]

[1] STMicroelectronics Belgium
joan.daemen@st.com
[2] Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC
{joseph.lano,bart.preneel}@esat.kuleuven.ac.be

Abstract. The stream cipher Self-Synchronizing Sober (SSS) is a candidate in the ECRYPT stream cipher competition. In this paper, we describe a chosen ciphertext attack on SSS. Our implementation of the attack recovers the entire secret state of SSS in around 10 seconds on a 2.8 GHz PC, and requires a single chosen ciphertext of less than 10 kByte. The designers of SSS state that chosen ciphertext attacks were considered to fall outside of the threat model. Hence the relevance of such attacks is also discussed in this paper.

1 Introduction

In the ECRYPT Stream Cipher Project [6], 34 stream cipher primitives have been submitted for evaluation. Of these 34 proposals, 31 are synchronous, 2 are self-synchronizing, and one design, Phelix, is neither synchronous nor self-synchronizing. This division reflects the fact that synchronous stream ciphers have been more widely studied in the past years.

In a synchronous stream cipher, the internal state of the stream cipher is independent of the plaintext and ciphertext. Hence the only relevant attack model is the known plaintext attack. Also, the attacker can influence the internal state through a resynchronization attack with chosen or known IV [3, 1]. A strong resynchronization mechanism is therefore needed to prevent such attacks.

Another attack model applies to the self-synchronizing stream ciphers, where the ciphertext needs to enter the state to ensure the self-synchronization property. This makes chosen plaintext (at encryption) and chosen ciphertext (at decryption) attacks interesting. Because of this property, the design and analysis of self-synchronizing stream ciphers is much closer related to the field of block ciphers than to the field of synchronous stream ciphers [4]. Note that the same applies to the design of

* Research financed by a Ph.D. grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen)

a good resynchronization mechanism for synchronous stream ciphers, as evidenced by several ECRYPT candidates.

In this paper, we describe such a chosen ciphertext attack on the ECRYPT candidate Self-Synchronizing SOBER (SSS) [8]. We also implemented the attack in C. Our implementation recovers the secret key with a chosen ciphertext of around 10 kByte and runs in 10 seconds on a 2.8 GHz PC.

The designers of SSS state that chosen ciphertext attacks were considered to fall outside of the security model. However, chosen ciphertext attacks have previously been considered when evaluating the security of self-synchronizing stream ciphers. Self-synchronizing stream encryption with DES in CFB mode was analyzed with respect to chosen ciphertext attacks in [7]. The stream cipher KNOT [2] has been broken by differential attacks using chosen ciphertext in [5]. The other self-synchronizing ECRYPT stream cipher candidate is MOSQUITO, the successor of KNOT. In the paper on MOSQUITO [4], the security analysis is mainly devoted to differential and linear attacks using chosen ciphertext. We will give arguments for the importance of this type of attacks in this paper.

The outline of this paper is as follows. A brief explanation on self-synchronizing stream ciphers is given in Sect. 2 and the design of SSS is briefly presented in Sect. 3. A chosen ciphertext attack on SSS is described in Sect. 4, and the relevance of such an attack on self-synchronizing stream ciphers is discussed in Sect. 5. The paper concludes in Sect. 6.

2 Self-Synchronizing Stream Ciphers

A simplified representation of a self-synchronizing stream cipher is given in Fig. 1. In such a design, the next key stream bit z_t is fully determined by the last n_m ciphertext bits and the cipher key K. This can be modelled as the key stream symbol being computed by a keyed cipher function f_c operating on a shift register that contains the last n_m ciphertexts. This conceptual model can be implemented in various ways, with the design of SSS described in Sect. 3 as an example.

For the first n_m plaintext or ciphertext symbols, the previous n_m ciphertexts do not exist. Hence the self-synchronizing stream cipher must be initialized by loading n_m *dummy* ciphertext symbols, called the initialization vector IV.

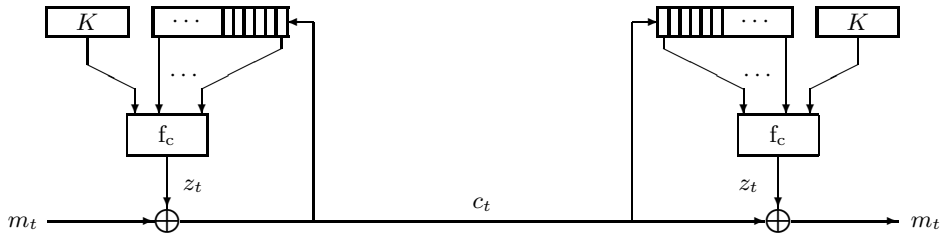

Fig. 1. Self-synchronizing stream encryption.

3 Brief Description of SSS

We only describe the aspects of the design that are relevant for the analysis performed in this article. For a complete description of the design, including the initialization and authentication mechanism, we refer to [8].

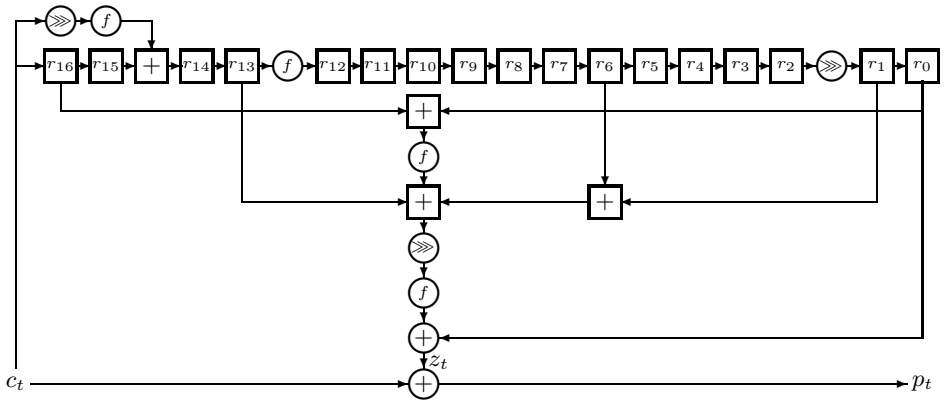

Fig. 2. Layout of SSS at the decryption side

A layout of SSS at the decryption side can be found in Fig. 2. In this figure, $\oplus$ represents exclusive or, $\boxplus$ represents addition and $\ggg$ represents a rotation by 8 positions to the right (or byteswap). The internal state of SSS consists of a 17-word shift register $r_0, \ldots, r_{16}$, where each word is 16 bits in size. The main building block is the key-dependent function f, which can be seen as a key-dependent permutation of a 16-bit word. The function f is built as follows:

$$f(x) = SBox(x_H) \oplus x\,, \tag{1}$$

where x_H stands for the Most Significant Byte (MSB) of x and where SBox is a key-dependent substitution box from 8 to 16 bits determined at key setup. In the rest of the paper we will assume that this SBox is a random unknown table of 256 16-bit words.

4 The Chosen Ciphertext Attack on SSS

From the description of SSS follows that its secret key consists of a table of 256 values of 16 bits. The aim of our attack is to recover this secret table.

We are going to decrypt a single ciphertext string that consists of a succession of 263 similar patterns and obtain the corresponding plaintext (and hence the key stream). The pattern i ($i = 0, 1, 2, \ldots, 263$) consists of 18 16-bit words and always has the following format:

$$\begin{cases} c_t^i = 0 \text{ for } t = 0, \ldots, 12, 14, \ldots 18 \\ c_{13}^i = b^i, \end{cases} \tag{2}$$

where b^i takes some value in each pattern, to be determined as explained below. Note that the values that we have chosen to be 0 could take any value for our attack to work, as long as they are constant across all patterns.

When generating key stream word z_{18}^i, we can see from Fig. 2 that the following words are needed[3]:

$$z_{18}^i = f((f(r[0] + r[16]) + r[1] + r[6] + r[13]) \ggg 8) \oplus r[0]\,. \tag{3}$$

It is easy to derive that these registers have the following content at $t = 18$:

$$\begin{cases} r[0] = f^2(0) \ggg 8 \\ r[1] = f^2(0) \ggg 8 \\ r[6] = f^2(0) \\ r[13] = f(0) + b^i \\ r[16] = 0\,. \end{cases} \tag{4}$$

In other words, all these registers are constant for each pattern i except for register $r[13]$. We can hence regroup all the constants inside $f()$ of (3) into a single (yet unknown) constant a as follows:

$$\begin{aligned} a &= f(r[0] + r[16]) + r[1] + r[6] + f(0) \\ &= f(f^2(0) \ggg 8) + (f^2(0) \ggg 8) + f^2(0) + f(0)\,, \end{aligned} \tag{5}$$

[3] At first sight, one may think that this should be z_{17}, but the designers have built a delay into their design, as can be deducted from the source code, which can be obtained at [8].

and then (3) simplifies to:

$$z_{18}^i = f((a + b^i) \ggg 8) \oplus r_0\,. \tag{6}$$

We use the notation r_0 to indicate that the content of $r[0]$ is also constant for all patterns. a is a two-byte word and we denote its MSB byte by a_H and its LSB byte by a_L. In the same way we split b^i in its two bytes b_H^i and b_L^i.

In a first phase, we will determine the 7 least significant bits of a_H. We will not be able to recover the most significant bit of a_H, but this is not a problem as the value of this bit is irrelevant to our analysis. To recover these 7 bits, we need 8 patterns of the type described above with $b_L^i = 0$, $b_H^0 = 0$ and $b_H^i = 2^{i-1}$ for $i = 1, 2, \ldots, 7$.

We now rewrite the above equation by splitting up the f function and some words of interest to get:

$$z_{18}^i = SBox(a_L) \oplus (2^8 \cdot a_L + a_H + 2^{i-1}) \oplus r_0\,, \tag{7}$$

By XORing the above equation for each $i \neq 0$ with the equation for $i = 0$ and eliminating terms we obtain:

$$z_{18,L}^i \oplus z_{18,L}^0 = a_H \oplus (a_H + 2^{i-1})\,. \tag{8}$$

In these equations a_H is the only unknown, and we can easily deduce its 7 least significant bits from the above equations bit by bit: A difference in $v_{18,L}^i \oplus v_{18,L}^0$ equal to 2^{i-1} implies that the corresponding bit of a_H is 0, otherwise it is 1.

Now that the relevant bits of a_H have been recovered, we will try to extract the entire secret $SBox$ table in the second phase of our attack. In short, this phase operates as follows. First we guess the value of a_L and $SBox(a_L)$ (24 bits in total). Then we reconstruct the remaining 255 entries of $SBox$ using key stream symbols $z_{18,L}^j$ obtained from decrypting 256 patterns. We then use this value to decrypt some ciphertext from the above patterns and check whether the plaintext matches. We now describe this reconstruction phase into more detail.

Our ciphertext contains 256 patterns that have $b_L^j = j$ and $b_H^j = 0$ for $j = 0, 1, \ldots, 255$. We obtain the following equations, again after XORing with the equation for $j = 0$:

$$z_{18}^j \oplus z_{18}^0 = \begin{array}{l} SBox(a_L) \oplus SBox(a_L + j) \oplus \\ (((2^8 \cdot a_H + a_L) \oplus (2^8 \cdot a_H + a_L + j)) \ggg 8)\,. \end{array} \tag{9}$$

Assuming a guess for a_L and $SBox(a_L)$ we can deduce $SBox(a_L + j)$ from this equation:

$$SBox(a_L + j) = \begin{array}{l} z_{18}^{j} \oplus z_{18}^{0} \oplus SBox(a_L) \oplus \\ (((2^8 \cdot a_H + a_L) \oplus (2^8 \cdot a_H + a_L + j)) \ggg 8)\,. \end{array} \quad (10)$$

Because j takes all 255 nonzero values we recover the entire SBox. We then verify whether, for the current guess of a_L and of $SBox(a_L)$ and the deduced $SBox$, the ciphertext decrypts to the corresponding plaintext. If it does, we have found the entire secret key.

We have implemented this attack in C. It recovers the entire secret key in on average 10 seconds on a 2.8 GHz Pentium IV PC running gcc under Linux. The single chosen ciphertext consists of 263 patterns of 36 bytes (note that the patterns for $i = 0$ and for $j = 0$ are the same), or 9468 byte in total. It is possible to reduce the data complexity even further by overlapping the patterns.

5 On the Relevance of Chosen Ciphertext Attacks

In the security claims for SSS [8], the authors state that they did not consider chosen ciphertext attacks in their threat model. One of the security requirements for their design is that "the result of decrypting altered ciphertext is not made available to the attacker". They motivate this requirement as follows: "This should be a standard requirement for any self-synchronizing stream cipher, since the attacker has complete control over the state of the cipher." However, it seems to be logical to us to include chosen ciphertext attacks in the security model of a self-synchronizing stream cipher, both from a theoretical as from a practical perspective.

From a theoretical perspective, a self-synchronizing stream cipher is functionally equivalent to a block cipher used in CFB mode. Chosen ciphertext attacks do apply on this mode of operation of a block cipher, an example is a chosen ciphertext attack on DES in CFB mode [7]. To enable a fair comparison of primitives aiming at the same applications, we believe that a uniform threat model should apply.

From a practical perspective, we see several scenarios where chosen ciphertext attacks can apply, just like with block ciphers. Preventing such an attack would require authenticating the plaintext before it is released. This suffers from two serious problems. First, buffering and secure storage of large amounts of texts is necessary, and this is impractical in several environments of interest. Second, this authentication requirement is orthogonal to the concept of self-synchronization: we do not see the point

of designing self-synchronizing stream ciphers when transmission errors are not allowed.

Another remark is that a self-synchronizing stream cipher resistant to chosen ciphertext attacks will result in a more elegant design. No special IV loading mechanism will be necessary as in SSS; loading a nonce into the state will be sufficient to start encryption and decryption.

6 Conclusion

In this note, we have described an attack on the ECRYPT candidate SSS, a self-synchronizing stream cipher. Our attack recovers the secret key of the design with a single chosen ciphertext of less than 10 kByte in about ten seconds on a modern PC. We believe that our attack is a practical attack on SSS. SSS is hence insecure and should not be used.

References

1. Frederik Armknecht, Joseph Lano, and Bart Preneel. Extending the resynchronization attack. In Helena Handschuh and Anwar Hasan, editors, *Selected Areas in Cryptography, SAC 2004*, number 3357 in Lecture Notes in Computer Science, pages 19–38. Springer-Verlag, 2004.
2. Joan Daemen, Rene Govaerts, and Joos Vandewalle. A practical approach to the design of high speed self-synchronizing stream ciphers. In O. Hirota and P. Y. Kam, editors, *Singapore ICCS/ISITA '92*, pages 279–293. IEEE, 1992.
3. Joan Daemen, Rene Govaerts, and Joos Vandewalle. Resynchronization weaknesses in synchronous stream ciphers. In T. Helleseth, editor, *Advances in Cryptology - EUROCRYPT 1993*, number 765 in Lecture Notes in Computer Science, pages 159–167. Springer-Verlag, 1993.
4. Joan Daemen and Paris Kitsos. Submission to ECRYPT call for stream ciphers: the self-synchronizing stream cipher MOSQUITO. ECRYPT Stream Cipher Project Report 2005/018, 2005. http://www.ecrypt.eu.org/stream.
5. Antoine Joux and Frederic Muller. Loosening the KNOT. In Thomas Johansson, editor, *Fast Software Encryption, FSE 2003*, number 2887 in LNCS, pages 87–99. Springer, 2003.
6. ECRYPT Network of Excellence in Cryptology. ECRYPT stream cipher project, 2005. http://www.ecrypt.eu.org/stream/.
7. Bart Preneel, Marnix Nuttin, Vincent Rijmen, and Johan Buelens:. Cryptanalysis of the CFB mode of the DES with a reduced number of rounds. In D.R. Stinson, editor, *Advances in Cryptology - CRYPTO 1993*, number 773 in Lecture Notes in Computer Science, pages 212–223. Springer-Verlag, 1994.
8. Gregory Rose, Philip Hawkes, Michael Paddon, and Miriam Wiggers de Vries. Primitive specification for SSS. ECRYPT Stream Cipher Project Report 2005/028, 2005. http://www.ecrypt.eu.org/stream.