Analysis of Message Injection in Stream Cipher-Based Hash Functions

Yuto Nakano[1], Carlos Cid[2], Kazuhide Fukushima[1], and Shinsaku Kiyomoto[1]

[1] KDDI R&D Laboratories Inc.
2-1-15 Ohara, Fujimino, Saitama 356-8502, Japan
{yuto,ka-fukushima,kiyomoto}@kddilabs.jp
[2] Information Security Group,
Royal Holloway, University of London
Egham, TW20 0EX – UK
carlos.cid@rhul.ac.uk

Abstract. A common approach for the construction of cryptographic hash functions is to design the algorithm based on an existing symmetric encryption primitive. While there has been extensive research on the design of block cipher-based hash functions, little has been done on the study of design and security of stream cipher-based hash functions (SCH). In this paper we discuss the general construction of stream cipher-based hash functions, devoting special attention to one of the function's crucial components: the message injection function. We define two types of message injection functions, which may be appended to the keystream generator (e.g. a stream cipher) to build an SCH. Based on these constructions, we evaluate the security of simple SCHs whose stream cipher function consists of a LFSR-based filter generator. We see this as an initial step in the more formal study of the security of hash function constructions based on stream ciphers.

Keywords: hash function, stream cipher, collision resistance.

1 Introduction

There has been much recent activity in the area of cryptographic hash function design and analysis: popular hash functions such as MD5 [12] and SHA-1 [10] have been shown to have weaknesses, e.g. lack of collision resistance [16,17], and this has spurred much interest in research in hash functions, culminating in the establishment of the NIST-sponsored SHA-3 competition to select a new hash function standard. As a result, several new designs have been proposed in the past few years.

A common approach for the construction of cryptographic hash functions is to design the algorithm based on an existing symmetric encryption primitive. There are several advantages in this type of approach: it may for instance be possible to derive the security of the hash function from the underlying symmetric algorithm; moreover, such a construction may allow one to use the symmetric

J. Lopez and G. Tsudik (Eds.): ACNS 2011, LNCS 6715, pp. 498–513, 2011.

algorithm for both encryption and as the building block for the hash function. This may be particularly attractive when trying to reduce the total cost of implementation of a cryptographic system in which encryption and hashing are required (for example, implementations in resource-constrained devices).

While there has been extensive research on the design of block cipher-based hash functions, not much has been done on the study of design and security of stream cipher-based functions. The general construction of a stream cipher-based hash function (SCH) was first introduced by Golić [3], as a mode of operation of stream ciphers. An SCH consists of a keystream generator (e.g. a stream cipher) and an additional function, which inputs the message into the internal state of the stream cipher. Thus, we can model a stream cipher-based hash function as a message injection function and a stream cipher function. The stream cipher function is the core component of SCHs and an appropriate algorithm is selected from among existing stream cipher algorithms. The pre-computation/injection function is used to input the message into the internal state of the stream cipher algorithm.

Several *dedicated* hash functions such as Boole [13] and MCSSHA-3 [5] are based on stream ciphers, and although these have been shown to be insecure, they did present good performance on a variety of platforms. It is also not uncommon to encounter in practice *ad hoc* constructions where a stream cipher is used to build a hash-like function. Despite of that, to the authors' best knowledge a well-developed set of design principles and security evaluation criteria for SCHs has yet to be established.

Nakano *et al.* [8] proposed a model for SCHs and showed necessary conditions for the construction of secure stream cipher-based hash functions. They concentrated on the problem of how to construct SCHs that are collision resistant, as this is perhaps the most challenging task for a designer. Their considerations are however not yet sufficient and a proposal for a concrete construction technique remains an open problem. In this paper, we describe the construction of SCHs based on bit-oriented simple stream cipher algorithms and provide the security analysis of possible message injection functions for SCHs. As an initial work in the area, we do not claim to provide an exhaustive discussion of all relevant security and design aspects of SCHs. We only deal here with collision resistance, and an analysis of other security requirements such as pre-image and second pre-image resistance remains to be considered. Moreover, in order to make the discussion simple, we only deal with bit-oriented linear feedback shift registers for the stream cipher. Other stream cipher constructions (such as the ones based on sponge functions) are not discussed in this paper.

The remaining of this paper is organised as follows: in Section 2 we present a brief discussion of related work. Section 3 provides definitions of message injection functions. We analyse the security of the message injection functions in Section 4. In Section 5, as an extension, we consider two LFSR-based SCHs. We present a discussion about the security and efficiency of the message injection functions in Section 6, and a comparison with existing algorithms is made in Section 7. Finally we conclude our paper in Section 8.

2 Related Work

We introduce below the necessary definitions and briefly describe research results of relevance to the construction of stream cipher-based hash functions.

2.1 Security Definitions of Hash Functions

The conventional security requirements for cryptographic hash functions are pre-image resistance, second pre-image resistance, collision resistance [7]; more recently length-extension security [4] has also been proposed as a security requirement for hash functions. Let $\mathcal{H}$ be a hash function, n be the hash output length, M and M' be messages; furthermore, let the symbol $||$ denote the concatenation of data.

Pre-image Resistance: given $h = \mathcal{H}(M)$ for some (unknown, randomly generated) M, finding any M' such that $\mathcal{H}(M') = h$ requires on average the work effort on the order of 2^n hash operations.

Second Pre-image Resistance: given a randomly generated M and $h = \mathcal{H}(M)$, finding any $M' \neq M$ such that $\mathcal{H}(M') = h$ requires on average the work effort on the order of 2^n hash operations.

Collision Resistance: finding M and M' such that $\mathcal{H}(M) = \mathcal{H}(M')$ and $M \neq M'$ requires on average the work effort on the order of $2^{n/2}$ hash operations.

Length-extension Security[1]: given $\mathcal{H}(M)$, the complexity of finding (z, x) such that $x = \mathcal{H}(M||z)$ should be greater than or equal to either the complexity of guessing M itself or 2^n.

2.2 Hash Function Constructions

The great majority of hash functions are of dedicated design. In this section, we briefly discuss the design of hash functions based on symmetric encryption algorithms, as well as some related SHA-3 proposals.

Block Cipher-Based Hash Functions. Preneel *et al.* studied general constructions of block cipher-based hash functions in [11]; they found that 12 out of all 64 possible constructions are secure. Later, Black *et al.* [2] extensively analysed the constructions. Stam [15] extended the work taking pre- and post-processing into consideration. We note that these results on the security of block cipher-based hash functions are based on the assumption that the primitive used is an *ideal block cipher* (rather than a specific algorithm).

Golić's Construction. Golić [3] studied modes of operation for stream ciphers, and showed how to convert a keystream generator into a stream cipher

[1] This requirement has been proposed in the NIST SHA-3 competition.

with memory (SCM), and how to build a hash function from an SCM. When a feedback shift register (FSR) based keystream generator is used, the SCM mode can be easily converted into a hash function by adding the plaintext bit to the feedback bit of the FSR. The SCM mode is clocked m times with an m-bit message and the corresponding m-bit ciphertext is stored in memory; then the SCM mode is clocked another m times with the m-bit ciphertext being input in reverse order. Finally, the SCM mode is clocked αm times where α is a small positive integer (e.g., three), and the last h successive ciphertext bits (or keystream bits) are output as the hash value. As the ciphertext in reverse order is used, the scheme requires an amount of memory that is equivalent to the message size.

SHA-3 Candidates. The NIST-sponsored SHA-3 competition kicked off in late 2008, and is expected to announce the new hash function standard in 2012. In total 64 algorithms were submitted to the competition, although only 56 of them are publicly known. Seven of the original submissions have similar structure to a stream cipher, and are thus of relevance to our work. Below we give a brief description of three of them.

Abacus. The Abacus [14] hash function has four registers (`ra, rb, rc, rd`). The message injection phase processes one byte in each round. First, the values of the four registers and one byte of the message are XORed and the result goes through S-Boxes. In the next step, four counters are combined with registers. Then, a maximum distance separable (MDS) matrix-based function is applied and four bytes are output. Finally, another S-Box operation is applied to the four bytes of registers. Two second pre-image attacks on Abacus were independently proposed by Nikolić *et al.*[9] and Wilson [18]. Wilson also showed a collision attack [18].

Boole. Boole [13] is constructed from a non-linear feedback shift register, input accumulators, and an output filter function. Boole consists of three phases: an input phase, a mixing phase, and an output phase. The state update function of the register, referred to as a cycle, transforms the state S_t into S_{t+1}. A message word is input to three accumulators, which are then updated. The register is then cycled once. After the input phase, the mixing phase is applied: the register is mixed with three accumulators, and the register is cycled 16 times. Finally the hash value is generated in the output phase. A pre-image attack and a collision attack against Boole were proposed by Nikolić [6].

MCSSHA-3. The MCSSHA-3 [5] hash function has a non-linear shift register whose size is the same as the hash value. A message is added with the feedback from the shift register. Before addition of the message, an 8×8 substitution is applied to the feedback. In MCSSHA-3, the message injection is performed every four clocks. Once a message block is input, the shift register is clocked without a message for another three clocks. After the whole message is processed, the h-bit internal state S is obtained, where h denotes the size of a hash value. The register is then clocked $h/2$ times with a $4h$-bit sequence which is a concatenation of the state S. Finally, the internal state is output as a hash value. A collision

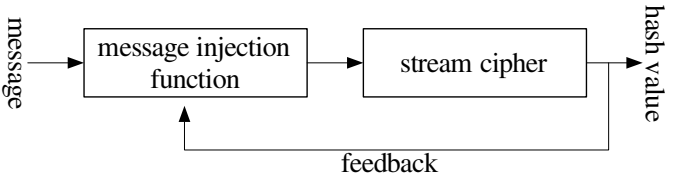

Fig. 1. Model of SCH

attack on MCSSHA-3 was presented by Aumasson and Naya-Plasencia[1]; they also demonstrated a second pre-image attack on MCSSHA-3 and a pre-image attack on the tweaked version: MCSSHA-4.

Model of SCHs. Nakano *et al.* [8] presented a general model of SCH, which is shown in Figure 1. They showed that an SCH can be modelled with two components: a message injection function and a stream cipher function.

A message injection function is appended to a stream cipher to construct an SCH. The message injection function is the component that takes the message and feedback from stream cipher as input, and determines the internal state of the stream cipher. The stream cipher diffuses the message over its internal state. The hash value is then generated as a certain length of the keystream. Generally, keys and IVs (initialization vectors) are set to constant values, usually to zero, and the message is loaded to the internal state of the stream cipher.

SCHs execute three phases: message injection, blank rounds, and hash generation. The message is loaded into the internal state of the stream cipher in the message injection phase. The internal state is updated from the message, previous state, and output feedback in the message injection phase. Blank rounds are the iteration of the state update (e.g. clocking the stream cipher) for diffusing the last input message words over the entire state. In the blank round phase, the internal state is updated from the previous state and output feedback. This phase is similar to the initialization of the stream cipher. Finally the stream cipher outputs a keystream as a hash value in the hash generation phase. During hash generation, only the previous state is used to update the state and the hash value is generated. In [8], the security analysis of SCHs based on the model above was provided, and suggested that secure message injection is a critical stage for achieving collision resistance. In this paper we further consider the problem of how to design a collision resistant message injection for stream cipher-based hash functions.

3 Definition of Message Injection Function

In this section, we present two message injection functions for a bit-oriented stream cipher.

3.1 Stream Cipher Model

Before defining the message injection function, let us define the stream cipher considered in this paper. We deal with a simple stream cipher that consists of

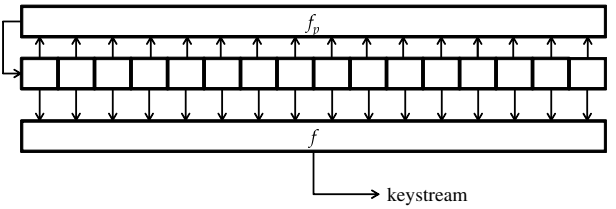

Fig. 2. An LFSR and a filter function

an LFSR and a filter function as in Figure 2, and an extension that consists of two LFSRs and a filter function as in Figure 5. Modern stream ciphers require an initialization operation for loading and mixing the encryption key and initialization vector into the internal state before producing the keystream; in the constructions discussed, the output of the non-linear filter can be fed back to the end of the LFSR as part of the initialization process. We use the initialization operation for message injection of the SCH.

Let l be the length of the LFSR, and f denote the filtering function. We assume that the feedback polynomial of the LFSR is primitive, and that the function f is balanced. Then, the LFSR reaches all internal states except all-zero and the period of state transition is given by $2^l - 1$. By clocking the LFSR $2^l - 1$ times, '1' is fed back 2^{l-1} times and '0' is fed back $2^{l-1} - 1$ times. The probability '1' is fed back and that of '0' are given by

$$\Pr[1 \text{ is fed back}] = \frac{2^{l-1}}{2^l - 1} \approx \frac{1}{2},$$
$$\Pr[0 \text{ is fed back}] = \frac{2^{l-1} - 1}{2^l - 1} \approx \frac{1}{2}.$$

These probabilities can be developed to the probability which feedback has a difference. Let two distinct internal states be S_1 and S_2, and $HW(x)$ be the Hamming weight of a binary string x. We say two internal states have a difference $\Delta(= S_1 \oplus S_2)$ when $HW(\Delta) \geq 1$. We denote the feedback derived from internal states S_1 and S_2 as b_{fdbk1} and b_{fdbk2}, respectively. We also say the feedback of the LFSR has a difference $\Delta_1(= b_{fdbk1} \oplus b_{fdbk2})$ when $HW(\Delta_1) = 1$. Let us denote registers with a difference as '1' and without the difference as '0'.

Remark 1. *The probability that the feedback has a difference is* $1/2$.

Please note that Remark 1 holds if the length l of the LFSR is large enough.

Let g be a message injection function. The message injection function is a map $g : \{0,1\}^l \times \{0,1\}^m \to \{0,1\}^l$, where l and m denote the internal state size and the message block size. Then the message injection function can be given by

$$S'_t = g(S_t, m_t).$$

Let $f_p(x) = c_1 x_{t,1} \oplus c_2 x_{t,2} \oplus \cdots \oplus c_l x_{t,l}$ be the linear recursion function of the LFSR, then the update of the register can be described as:

$$s_{t+1,i} = \begin{cases} s_{t,i+1} & (1 \le i \le l-1), \\ f_p(s_{t,1}, \ldots, s_{t,l}) & (i = l). \end{cases} \quad (1)$$

The keystream z_t is given by

$$z_t = f(d_1 s_{t,1}, \ldots, d_l s_{t,l}),$$

where the constants $d_j \in \{0, 1\}$ for $1 \le j \le l$ choose the registers to be input to the filter function.

We define two message injections as: the message is XORed with the keystream and this result is XORed with feedback of the LFSR; or the message is XORed with the keystream and the result is XORed with the internal state. We describe both injections in more detail in the following sections.

3.2 Inject into Feedback

The SCH with this message injection function is shown in Figure 3. This is the most natural way to inject the message into the internal state, and a subcase of the "inject into the internal state" mode, which we explain later. The construction proposed by Golić applies this method. MD5 and SHA-1 can also be categorized as this type since the step operation of these hash functions update only one chaining variable and the others are just shifted[2]. Once the value of the register is fixed, it is not updated until it is fed back again. Furthermore, the feedback can be controlled by the message. It is possible for an adversary to control the entire internal state. Note that we consider the bit-oriented LFSR only and do not consider the message expansion.

The message injection function can be described as follows.

$$s_{t+1,i} = \begin{cases} s_{t,i+1} & (1 \le i \le l-1) \\ f_p(s_{t,1}, \ldots, s_{t,l}) \oplus (f(d_1 s_{t,1}, \ldots, d_l s_{t,l}) \oplus m_t) & (i = l). \end{cases} \quad (2)$$

3.3 Inject into the Internal State

We show the scheme of this message injection function in Figure 4. This message injection function updates the internal state by XORing the value of the internal state with the message and the keystream. Since we consider the bit-oriented LFSR only, the same message data is XORed in different positions. This scheme requires selectors $\sigma_i \in \{0, 1\}$ that determine the registers to be updated. The message injection function can be described as:

$$s_{t+1,i} = \begin{cases} s_{t,i+1} \oplus \sigma_i(z_t \oplus m) & (1 \le i \le l-1) \\ f_p(s_{t,1}, \ldots, s_{t,l}) & (i = l). \end{cases} \quad (3)$$

The security of the message injection depends not only on the stream cipher but also where to inject the message. As we discuss later in Section 4, the number of registers updated by the message is the important factor for the message injection.

[2] Although these hash functions also include a message expansion mechanism.

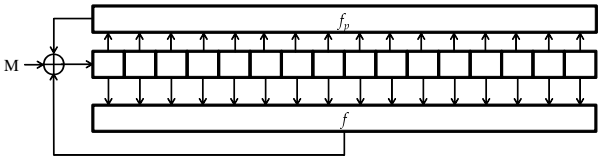

Fig. 3. Inject into Feedback

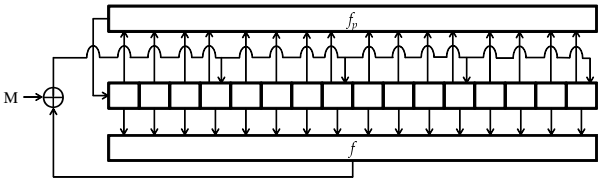

Fig. 4. Inject into the Internal State

4 Security Analysis

In this section, we evaluate the security of SCHs constructed from a simple stream cipher and message injection functions introduced in Section 3 against collision attacks.

4.1 Inject into Feedback

A difference Δ_1 on feedback of the LFSR and/or an output difference Δ_2 of the f function can be easily cancelled out by using a message difference Δm as:

$$\Delta m = \Delta_1 \oplus \Delta_2,$$

and the value in the leftmost register is easily controlled. Assume that the LFSR is clocked n times, where n is large enough when compared to the length of the LFSR. Then the difference in the register is forced out, hence only the difference on the leftmost register should be taken into consideration. Iterating this cancellation of the difference for l times enables an adversary to control the entire internal state of the LFSR. Therefore, the collision is easily generated. This type of message injection cannot provide the collision resistance without the message expansion.

4.2 Inject into the Internal State

Suppose that the same message-dependent data is injected into r positions of the LFSR at regular intervals, then l/r registers can be controlled by the message and a collision of these registers is easily generated. From the fact that the adversary can control l/r bits of the internal state, $l(1-1/r)$ bits of the internal state could have differences. Since feedback has the difference with probability

$1/2$, the filter function must output the difference for $l(1-1/r)/2$ clocks and it must not output the difference for the other $l(1-1/r)/2$ clocks. Suppose that the filter function output the difference with the probability p, then the probability of the collision is given by

$$\Pr[\text{coll}] = [p(1-p)]^{\frac{l(1-1/r)}{2}}.$$

Here we introduce a useful remark for the evaluation of the filter function. Since the filter function is balanced, the difference is output with probability $1/4 + 1/4 = 1/2$.

Remark 2. *If the output of the filter function is balanced, then it propagates the difference with probability* $p = 1/2$.

From Remark 2, we obtain $\Pr[\text{coll}] = 2^{-l(1-1/r)}$. Using the birthday attack, the probability is bounded below by,

$$\Pr[\text{coll}] = 2^{-\frac{l(1-1/r)}{2}}.$$

5 Extension to Two LFSRs

In this section, we consider the extension of the message injection function and its security evaluation to the two-LFSR-based SCH.

5.1 Two-LFSR-Based SCH

The internal state size of LFSR-A and LFSR-B is give by l_a and l_b, respectively. The function f is the same as the stream cipher with one LFSR; the t_f-to-1 balanced filter function. Let f_A and f_B be feedback polynomials of LFSR-A and LFSR-B, and coefficients of each polynomial are given by α_j and β_j. For the sake of the simplicity, we denote $S_t = \bigoplus \alpha_i s_{t,i}$, $U_t = \bigoplus \beta_i u_{t,i}$, and the input to the filter function as S'_t, then the state update of each register can be denoted as

$$s_{t+1,i} = \begin{cases} s_{t,i+1} & (1 \le i \le l_a - 1), \\ f_A(S_t) & (i = l_a), \end{cases}$$

$$u_{t+1,j} = \begin{cases} u_{t,j+1} & (1 \le j \le l_b - 1), \\ f_B(U_t) \oplus f(S'_t) & (j = l_b), \end{cases}$$

We define $\sigma_{A,i}, \sigma_{B,i} \in \{0,1\}$ that determine which registers to be updated by the message. The output of the stream cipher is defined as,

$$z_t = u_{t,1}.$$

Then, we have four choices where to inject the message for this stream cipher as:

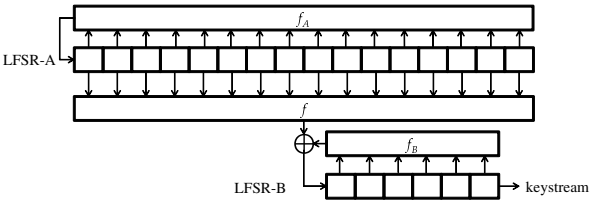

Fig. 5. Two LFSRs and a filter function

$$s_{t+1,i} = \begin{cases} s_{t,i+1} & (1 \le i \le l_a - 1), \\ f_A(S_t) \oplus (z_t \oplus m) & (i = l_a), \end{cases}$$
$$u_{t+1,j} = \begin{cases} u_{t,j+1} & (1 \le j \le l_b - 1), \\ f_B(U_t) \oplus f(S'_t) & (j = l_b), \end{cases} \tag{4}$$

$$s_{t+1,i} = \begin{cases} s_{t,i+1} & (1 \le i \le l_a - 1), \\ f_A(S_t) \oplus (z_t \oplus m) & (i = l_a), \end{cases}$$
$$u_{t+1,j} = \begin{cases} u_{t,j+1} & (1 \le j \le l_b - 1), \\ f_B(U_t) \oplus (z_t \oplus m) \oplus f(S'_t) & (j = l_b), \end{cases} \tag{5}$$

$$s_{t+1,i} = \begin{cases} s_{t,i+1} \oplus \sigma_{A,i}(z_t \oplus m) & (1 \le i \le l_a - 1), \\ f_A(S_t) & (i = l_a), \end{cases}$$
$$u_{t+1,j} = \begin{cases} u_{t,j+1} & (1 \le j \le l_b - 1), \\ f_B(U_t) \oplus f(S'_t) & (j = l_b), \end{cases} \tag{6}$$

$$s_{t+1,i} = \begin{cases} s_{t,i+1} \oplus \sigma_{A,i}(z_t \oplus m) & (1 \le i \le l_a - 1), \\ f_A(S_t) & (i = l_a), \end{cases}$$
$$u_{t+1,j} = \begin{cases} u_{t,j+1} \oplus \sigma_{B,i}(z_t \oplus m) & (1 \le j \le l_b - 1), \\ f_B(U_t) \oplus f(S'_t) & (j = l_b). \end{cases} \tag{7}$$

Two methods described as Eq. (4) and Eq. (5) are essentially the same, the message is XORed with feedback of LFSRs. The internal state is directly updated by the message-dependent data in Eq. (6) and Eq. (7).

5.2 Security Analysis

We first introduce a remark regarding the security evaluation of the LFSR-B. Let Pr[diff. on B cancelled] be the probability that all differences on the LFSR-B are

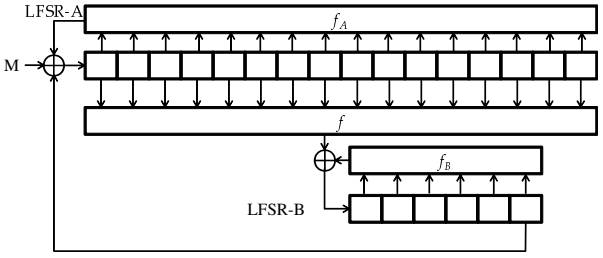

Fig. 6. Inject into Feedback of LFSR-A

cancelled out when no difference is input from the filter function f. We denote the register with the difference as '1' and without as '0'. When the filter function does not output differences, '0's are XORed with feedback of the LFSR-B and the filter function does not influence the LFSR-B, that is, only LFSR-B should be taken into consideration. There exists an integer n for any randomly chosen two internal states U_t and $U_{t'}$ that satisfies $U_t = U_{t'+n}$. The period of the LFSR-B is given by $2^{l_b} - 1$ since its feedback polynomial is primitive. The computational cost for finding such n is given by $2^{l_b/2}$. Hence the probability that difference on the LFSR-B is cancelled is given by $\Pr[\text{diff on B cancelled}] = 2^{-l_b/2}$.

Remark 3. *If the feedback polynomial of the LFSR-B is primitive and the difference is not input into the LFSR-B, then* $\Pr[\textit{diff. on B cancelled}] = 2^{-l_b/2}$.

Inject into Feedback of LFSR-A. This scheme is shown in Figure 6. The LFSR-A is easily controlled by a message, since the message is just XORed with feedback. Once all differences on LFSR-A are cancelled out, no difference is output from the function f. The LFSR-B still has a difference at this point and this difference has to be cancelled out. Hence the probability which two different messages collide is given by greater of the birthday paradox or the probability of the difference on LFSR-B being cancelled out,

$$\begin{aligned}\Pr[\text{coll}] &= \max(2^{-l_b/2}, \Pr[\text{diff. on B cancelled}]) \\ &= 2^{-l_b/2}.\end{aligned}$$

Inject into Feedback of Both LFSRs. Let us consider the case that messages are injected into feedback of both LFSRs. When the message is injected into both LFSRs, the adversary can control either LFSR-A or LFSR-B. Here, we assume that the size of the LFSR-A is larger than that of the LFSR-B, then it is natural for the adversary to control the LFSR-A. After all differences on LFSR-A are cancelled out, no message difference should be injected. However LFSR-B still has differences to be cancelled out. The probability that those differences are cancelled out is the same as the probability of the collision and given as

$$\begin{aligned}\Pr[\text{coll}] &= \max(2^{-l_b/2}, \Pr[\text{diff. on B cancelled}]) \\ &= 2^{-l_b/2}.\end{aligned}$$

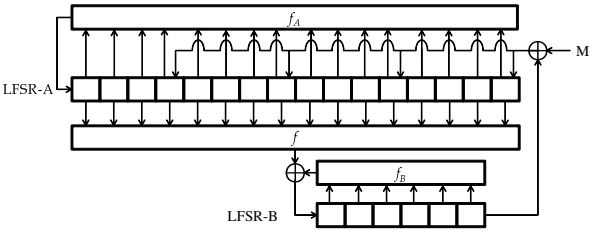

Fig. 7. Inject into the Internal State of LFSR-A

Inject into the Internal State of LFSR-A. Figure 7 shows this scheme. The message is XORed with the output of the LFSR-B and the result is again XORed with the internal state of the LFSR-A. Suppose that r bits of the internal state of LFSR-A are updated by the message-dependent data at regular intervals, then l_a/r bits of the LFSR-A can be controlled. The remaining $l_a(1-1/r)$ bits are unfixed and the probability which the LFSR-A collides is given by

$$\Pr[\text{coll.A}] = 2^{-\frac{l_a(1-1/r)}{2}}.$$

The LFSR-B collides when no difference is input, the difference on LFSR-B is cancelled, or happen to collide. Therefore the LFSR-B collides with the probability,

$$\Pr[\text{coll.B}] = \max((1-p)^{l_b}, \Pr[\text{diff. on B cancelled}], 2^{-l_b/2}) = 2^{-l_b/2}.$$

Hence the probability of the collision is

$$\Pr[\text{coll}] = 2^{-\frac{l_a(1-1/r)+l_b}{2}}.$$

Inject into the Internal State of Both LFSRs. We consider that the message-dependent data is injected into both LFSRs: the message is XORed with r bits of the LFSR-A and q bits of the LFSR-B at regular intervals. The adversary can control part of the internal state of either l_a/r bits of LFSR-A or l_b/r bits of LFSR-B. We assume that l_a/r is larger than l_b/r, then it is natural that the adversary tries to control LFSR-A. The remaining $l_a(1-1/r)$ bits of LFSR-A and the entire internal state of LFSR-B are unfixed. Therefore, the adversary can generate the collision with probability,

$$\Pr[\text{coll}] = 2^{-\frac{l_a(1-1/r)+l_b}{2}}.$$

6 Discussion

In this section, we discuss the security and the efficiency of two types of message injection functions. Table 1 summarizes collision probabilities and the number of required operations for each message injection function.

Table 1. Message Injection Functions

Function	Collision Prob.	No. of Operation
Single LFSR		
Inject into Feedback	1	1 *XOR*
Inject into the Internal State	$2^{-[l(1-1/r)]/2}$	r *XORs*
Two LFSRs		
Inject into the Feedback of LFSR-A	$2^{-l_b/2}$	1 *XOR*
Inject into the Feedback of Both LFSRs	$2^{-l_b/2}$	2 *XORs*
Inject into the Internal State of LFSR-A	$2^{-[l_a(1-1/r)+l_b]/2}$	r *XORs*
Inject into the Internal State of Both LFSRs	$2^{-[l_a(1-1/r)+l_b]/2}$	$(r+q)$ *XORs*

6.1 Single LFSR

Inject into feedback only requires an XOR operation for each cycle of the message injection function. Hence this message injection function is lightweight and we believe SCH constructed with this function achieves as a high performance as its original stream ciphers. However the entire internal state can be controlled by the message and the collision is easily generated without the secure message expansion.

Inject into the internal state requires r XOR operations for a cycle of the message injection function. It also requires the selector, which selects registers to be updated with the message. This selector is fixed by the specification of the algorithm, hence does not affect the performance of the SCH. This scheme can be collision resistant by choosing r and l adequately without the message expansion.

6.2 Two LFSRs

Inject into feedback of LFSR-A only requires an XOR operation and inject into feedback of both LFSRs requires two XOR operations. These two injections are lightweight and achieve as a high performance as their original stream ciphers. However, the entire internal state of LFSR-A can be fixed and the computational cost for the collision is reduced from $2^{(l_a+l_b)/2}$ to $2^{l_b/2}$. By making the LFSR-B larger than the size of hash value, the collision resistance can be ensured. The computational cost is not affected if the message is injected into feedback of LFSR-A or both LFSRs.

Inject into the internal state of LFSR-A and both LFSRs requires r and $(r+q)$ operations of XOR, respectively. Similar to the case of single LFSR, the selector has to be introduced to these schemes. These two scheme is the most secure way among six since the adversary can only control the part of the LFSR-A. The adequate three parameters l_a, l_b, and r can ensure the collision resistance. Although inject into both LFSRs require additional q operations of

XOR compared to inject into LFSR-A, probabilities of collisions are the same in two cases.

7 Comparison to Real Algorithms

In this section, we apply our security analysis to real algorithms, which are Abacus, MCSSHA-3, and Boole. Note that we cannot directly compare our evaluation to above algorithms since these algorithms use word-oriented shift registers. However, computational costs shown in attacks on these algorithms meet our estimated ones.

7.1 Abacus

The message injection of Abacus is inject into feedback. Abacus can be considered as the extended version of two-LFSR-based SCH, since Abacus has four registers. The adversary cannot control the entire state, but the largest register can be fixed to zero. This is exploited in the second pre-image attack and collision attack in [9] and [18].

Four registers of Abacus can be classified into registers updated by the message and the others. Suppose the byte-oriented shift register is the extended version of the bit-oriented LFSR, then security analysis of Section 5 is approximately applicable to Abacus. The register where the message is injected corresponds to LFSR-A of Section 5 and the others correspond to LFSR-B. Therefore, the size of the LFSR-B is given by $l_b = 344$ and the lower-bound of the probability for the collision is estimated at 2^{-172} from table 1. This probability is pretty close to the probability of the collision attack given in [18].

7.2 MCSSHA-3

MCSSHA-3 also uses the byte-oriented feedback shift register, and as the same reason of Abacus, our security evaluation is applicable to it. The message injection of MCSSHA-3 is also inject into feedback. Since the message injection of MCSSHA-3 is done every four clocks, the FSR of MCSSHA-3 can be divided into two categories; registers where the message is injected and the others. Hence we can divide the FSR of MCSSHA-3 into two FSRs and our security evaluation for two LFSRs is applicable. From table 1, the lower bound of the collision probability for two LFSRs with inject into feedback is given by $2^{-l_b/2}$. In MCSSHA-3, the size of the registers is given by $l_b = 192$, the probability for collision is estimated at 2^{-96}. This probability is pretty close to the probability of the collision attack given in [1].

7.3 Boole

Boole applies the inject into the internal state. In the security evaluation, we assumed that the message-dependent data is injected into r registers at regular intervals. We cannot directly compare our security analysis and that of Boole

since the message injection of Boole does not inject at regular intervals. The FSR of Boole corresponds to the LFSR-A in our evaluation and three accumulators corresponds to the LFSR-B. The message is injected into two registers of the LFSR-A and three registers of the LFSR-B. From our evaluation, the probability of collision is estimated to be $2^{-[512(1-1/2)+96]/2} = 2^{-176}$ since $l_a = 512$, $l_b = 96$, and $r = 2$.

We assume that the same message-dependent data is injected into multiple registers in our evaluation, however, different message-dependent data is injected into each register in Boole. This is the first reason why the probability of the collision derived from our evaluation and shown in [6] has a gap.

In [6], they control the message difference that leads to collision, and the computational cost for collision attack is dramatically reduced to be 2^{33}. This is the second reason of the gap of the probability between ours and that of [6]. Boolean functions used in Boole have a vulnerability and collisions on Boolean functions are easily generated. By exploiting this vulnerability, differences in the accumulators can be cancelled. The difference injected into the register can be also cancelled by the difference which is output from accumulators. As the result, the difference is cancelled efficiently and the computational cost for the collision attack is dramatically reduced to 2^{33}.

8 Conclusion

In this paper, we evaluated the security (mainly collision resistance) and the efficiency of SCHs based on the classic filter generator. We considered a stream cipher consisting of a single LFSR, and then extended it to the case with two LFSRs. We defined two message injection functions for each stream cipher: inject into feedback and inject into the internal state. We constructed six SCHs by appending these message injection functions to the stream ciphers and derived probabilities of collision.

Inject into feedback is lightweight and can achieve high performance, however it cannot be secure without a message expansion. On the other hand, inject into the internal state has the potential to realize the collision resistance without a message expansion. As discussed in Section 4, the security of the resulting SCHs can be increased by making r large. The computational cost of calculating hash values can be optimized to use appropriate r. The construction that injects the message into the internal state is adjustable not only for systems that require a high security level but also for small devices whose resources are highly constrained. Our analysis suggests criteria for parameters, length of FSRs, and the number of message-injecting registers in the design of SCHs. We also compared our security analysis with real algorithms and confirmed that our evaluation appears to be reasonably accurate, especially for inject into feedback function.

In our opinion, there remains many aspects to be researched for achieving a secure design of stream cipher-based stream ciphers; these include analysis of pre-image and second pre-image resistance, extension to other hash function constructions, such as ones based on sponge functions, among others. We see however this paper as an initial step in the more formal study of the security

of hash function constructions based on stream ciphers, and hope it will spur further research in this topic.

References

1. Aumasson, J.-P., Naya-Plasencia, M.: Cryptanalysis of the MCSSHA Hash Functions (2009), Presented at WEWoRC 2009, http://131002.net/data/papers/AN09.pdf
2. Black, J., Rogaway, P., Shrimpton, T.: Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 320–335. Springer, Heidelberg (2002)
3. Golić, J.D.: Modes of Operation of Stream Ciphers. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 233–247. Springer, Heidelberg (2001)
4. Jonsson, J., Widmayer, C., Kelsey, J.: Public Comments on the Draft Federal Information Processing Standard (FIPS) Draft FIPS 180-2, Secure Hash Standard, SHS (2001), http://www.cs.utsa.edu/~wagner/CS4363/SHS/dfips-180-2-comments1.pdf
5. Maslennikov, M.: Secure hash algorithm MCSSHA-3. Submission to NIST (2008), http://registercsp.nets.co.kr/MCSSHA/MCSSHA-3.pdf
6. Mendel, F., Nad, T., Schläffer, M.: Collision Attack on Boole. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 369–381. Springer, Heidelberg (2009)
7. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
8. Nakano, Y., Kurihara, J., Kiyomoto, S., Tanaka, T.: On a Construction of Stream-cipher-based Hash Functions. In: SECRYPT, pp. 334–343 (2010)
9. Nikolić, I., Khovratovich, D.: Second preimage attack on Abacus (2008), http://lj.streamclub.ru/papers/hash/abacus.pdf
10. NIST. Secure hash standard. FIPS180-1 (1995)
11. Preneel, B., Govaerts, R., Vandewalle, J.: Hash Functions Based on Block Ciphers: A Synthetic Approach. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 368–378. Springer, Heidelberg (1994)
12. Rivest, R.: The MD5 message digest algorithm. RFC1321 (1992)
13. Rose, G.G.: Design and primitive specification for Boole. submission to NIST (2008), http://seer-grog.net/BoolePaper.pdf
14. Sholer, N.: Abacus a candidate for SHA-3. submission to NIST (2008), http://ehash.iaik.tugraz.at/uploads/b/be/Abacus.pdf
15. Stam, M.: Blockcipher-Based Hashing Revisited. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 67–83. Springer, Heidelberg (2009)
16. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
17. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
18. Wilson, D.: A second-preimage and collision attack on Abacus (2008), http://web.mit.edu/dwilson/www/hash/abacus_attack.pdf