J. Math. Cryptol. **7** (2013), 253–277
DOI 10.1515/jmc-2013-5005

Indirect message injection for MAC generation

Mufeed Al Mashrafi, Harry Bartlett, Ed Dawson, Leonie Simpson
and Kenneth Koon-Ho Wong

Communicated by Spyros Magliveras

Dedicated to Professor Tran Van Trung on the occasion of his 65th birthday

Abstract. This paper presents a model for the generation of a MAC tag using a stream cipher. The input message is used indirectly to control segments of the keystream that form the MAC tag. Several recent proposals can be considered as instances of this general model, as they all perform message accumulation in this way. However, they use slightly different processes in the message preparation and finalisation phases. We examine the security of this model for different options and against different types of attack, and conclude that the indirect injection model can be used to generate MAC tags securely for certain combinations of options. Careful consideration is required at the design stage to avoid combinations of options that result in susceptibility to forgery attacks. Additionally, some implementations may be vulnerable to side-channel attacks if used in Authenticated Encryption (AE) algorithms. We give design recommendations to provide resistance to these attacks for proposals following this model.

Keywords. MAC, stream ciphers, message injection, collision attacks, forgery attacks, side-channel attacks.

2010 Mathematics Subject Classification. 94A62.

1 Introduction

A Message Authentication Code (MAC) is used to provide assurance of the integrity of messages, and requires use of an algorithm along with knowledge of a secret key. MACs are commonly formed using either keyed hash functions or block ciphers in certain modes of operation, such as CBC mode. More recently, algorithms to construct a MAC using a stream cipher have been proposed. Some proposals which make use of symmetric ciphers claim to provide both confidentiality and integrity assurance simultaneously. Such proposals are referred to as Authenticated Encryption (AE) [5, 6]. The threat model for authenticated encryption may vary, depending on whether the MAC is constructed from plaintext or ciphertext. In [5, 6], it is proved that performing encryption first and then forming a MAC for the resulting ciphertext provides the greatest security for authenticated encryption, provided that the underlying integrity component is strongly unforgeable. In this paper, we investigate the integrity components of several existing authenticated encryption proposals to determine the conditions under which the components are strongly unforgeable. The paper includes research previously presented in [2], along with the results of additional investigations into statistical properties and side channel attacks.

There is extensive research in the existing literature on generating MAC tags using block cipher algorithms, but much less on generating MAC tags using stream ciphers. This may be due to the existence of block cipher standards such as DES and AES with well-known modes of operation. However, stream ciphers have a smaller footprint in hardware (and often software) applications and are generally faster, so a stream-cipher based MAC is worth considering. These may be especially useful for resource constrained applications, where block-cipher based MACs may be unsuitable.

Lai et al. [17] proposed a cryptographic checksum algorithm based on stream ciphers. Golić also describes a mode of operation to generate a MAC using a stream cipher [13]. Of the thirty-four stream cipher proposals submitted to the eSTREAM project [8], seven claimed to provide AE. More recently, the NIST competition to develop a new cryptographic hash algorithm (to be known as SHA-3) received sixty-four submissions; seven of these used stream cipher algorithms to generate a hash value. Although flaws exist in these proposals, resulting in none of them being considered as finalists in the respective competitions, the proposals themselves demonstrate interest in MAC generation using stream ciphers.

Nakano et al. [21] proposed a general model for generating a hash value using a stream cipher by injecting the input message directly into the internal states of the cipher. Their security analysis suggests that the message injection function is critical in achieving collision resistance for MACs which fit their model. In [20], the hash function properties associated with two different methods for direct message injection into the internal states of the ciphers are considered. In [3] this approach is adapted for MAC generation using direct injection into the internal state of a nonlinear filter generator.

In this paper we analyse proposals for MAC generation which use stream ciphers in an entirely different way from the direct injection method of the Nakano model [21]. Rather than direct injection, some existing stream cipher based MAC proposals use the input message to control the compression of a bitstream from a keystream generator to form a MAC tag. We refer to this method as indirect message injection. We present a general model for generating a MAC tag using stream ciphers in this way, and use a matrix representation of the MAC tag generation process in our analysis. We consider certain design options for MAC generation under this model, and analyse the security implications associated with particular option choices with respect to the resistance provided to forgery attacks and side channel attacks. We apply this analysis to three proposals: Sfinks [7], 128-EIA3 [10, 11] and Grain-128a [1].

This paper is organised as follows. Section 2 outlines the phases in generating a MAC tag, and Sections 2.1 and 2.2 describe a method for performing authenticated encryption and a general model for generating a MAC using a stream cipher and indirect message injection. Three specific algorithms that use a stream cipher to generate a MAC tag in this way (Grain-128a, Sfinks and 128-EIA3) are examined in Section 3. Our security analysis for the general model is presented in Section 4, and for these specific MAC algorithms in Section 5. Concluding remarks are contained in Section 6.

2 MAC generation using indirect message injection

A MAC algorithm takes as input an arbitrary length message M of length l bits; a k-bit secret key K; and optionally a v-bit initialisation vector IV. The algorithm output is a MAC tag of fixed length d. Common values for d are 32, 64 or 128 bits. The generation of a MAC tag usually involves three phases, which we refer to as preparation, accumulation and finalisation. The *preparation phase* involves both initialising the internal states of the integrity components of the device, and preparing the input message. Message preparation may require the addition of padding bits to the beginning or end of the message. The *accumulation phase* is where the input message is processed and values are accumulated in the internal states of the integrity components of the device. The *finalisation phase* completes the processing of the MAC tag. This is usually performed by combining the stored value at the end of the accumulation phase with a masking value.

A MAC tag may be computed for either plaintext or ciphertext, so we use the more general term *message* to cover both of these input types. The most common structures for MAC algorithms use the message directly, accumulate bits or words, and then apply a finalisation phase to generate the MAC tag [22]. Some AE stream cipher proposals use a very different authentication strategy, which makes use of a keystream sequence. Rather than using the message (either plaintext or ciphertext) directly in the accumulation phase, these proposals use the message as a means to control the accumulation of a keystream sequence. After the message has been processed, the accumulated value then forms the basis of the MAC tag for the input message. In this section we describe explicitly a model for generating a MAC tag using stream ciphers in this manner.

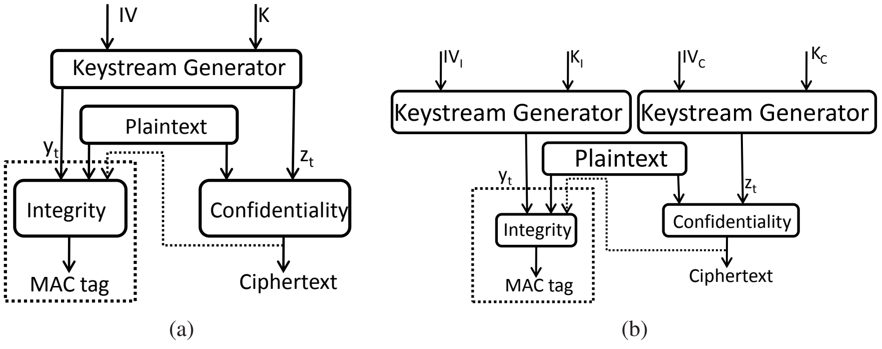

Figure 1. Structures for providing authenticated encryption using stream ciphers.

2.1 General structure

The keystream generator of a stream cipher takes as inputs a secret key K and a public IV, and generates a pseudorandom binary sequence. Usually, these binary sequences are used as keystream for a binary additive stream cipher to provide confidentiality for plaintext messages. However, they can also be used for integrity applications. Where authenticated encryption is required, the bitstream used for the integrity application can be produced by the same generator as the bitstream used for the confidentiality application, but a different keystream generator could also be used.

Figure 1 (a) illustrates the case where a single keystream generator using a key K and an IV produces two different binary sequences, z and y, used for confidentiality and integrity applications, respectively. Considering the order in which the encryption and authentication operations are performed, and using the terminology of [6] this may be described as Encrypt and MAC, if the input to the integrity component is the plaintext; or Encrypt then MAC, if the input to the integrity component is the ciphertext. For the case where one keystream generator produces both sequences z and y, we assume that the two bitstreams used are distinct. Examples of algorithms that operate in this manner are Grain-128a [1] and Sfinks [7].

Alternatively, two bitstreams z and y could be generated by separate keystream generators, as shown in Figure 1 (b). Distinct keys and IVs are used for each generator: K_C and IV$_C$ for the confidentiality component, and K_I and IV$_I$ for the integrity component, respectively. If the MAC is formed from the plaintext, then this is the traditional method of encryption and a separate MAC. If the ciphertext is used as the input to the MAC then, using the terminology of [6], this may be described as Encrypt then MAC. Algorithms operating in this manner are SNOW 3G [9], used in 128-EIA2; and ZUC [10], used in 128-EIA3.

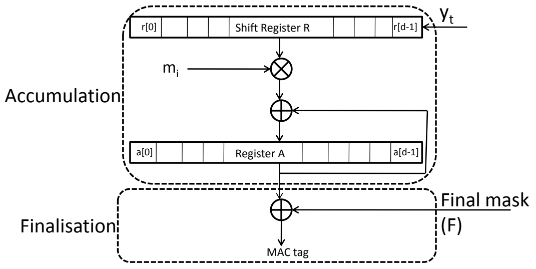

Figure 2. MAC generation using indirect message injection.

In either case, our interest is in the process by which the MAC is generated. That is, our analysis focuses on the integrity component of the designs, which is the part of Figure 1 (in both structures (a) and (b)) inside the dashed line.

2.2 Structure of integrity algorithm

For MAC generation using indirect message injection, the major components used are two registers, each consisting of d binary stages, where d is the length of the MAC tag. The relationship between these two registers, the binary keystream sequence and the input message M is shown in Figure 2.

The first register, denoted R, is a shift register. Let R_t denote the contents of register R at time t, with $R_t = [r_t(0), \ldots, r_t(d-1)]$. The initial contents of R are $R_0 = [r_0(0), \ldots, r_0(d-1)]$. At each time t, for $t > 0$, the contents of R are updated using bit y_{t-1} from the binary sequence y as follows:

$$R_t(i) = \begin{cases} r_{t-1}(i+1) & \text{for } i = 0, \ldots, d-2, \\ y_{t-1} & \text{for } i = d-1. \end{cases} \tag{2.1}$$

Thus the contents of register R can be considered as a d-bit "sliding window" on the sequence $R_0 \| y$, where $\|$ denotes concatenation.

The second register is an accumulation register, denoted A. Let A_t denote the contents of register A at time t, with $A_t = [A_t(0), \ldots, A_t(d-1)]$. The initial contents of A are $A_0 = [a_0(0), \ldots, a_0(d-1)]$. At time t, the contents of register A are updated using the existing contents of A and the contents of register R, conditional on the value of the input message bit M_t. If the message bit M_t at time t is 1, then register A is updated by bitwise XORing with the contents of register R_t; otherwise the contents of A remain unchanged, as given in equation (2.2).

$$A_t = \begin{cases} A_{t-1} \oplus R_{t-1} & \text{if } M_t = 1, \\ A_{t-1} & \text{otherwise.} \end{cases} \tag{2.2}$$

The input message M controls which of the d-bit segments of the sequence $R_0 \| y$ are accumulated into register A. In this paper we use a matrix representation of the accumulation process for our analysis. Consider A_i as an array of d bits, representing the d values in register A when the ith message bit has been processed. The contents of A_i can be expressed as $A_i = A_0 \oplus T_i M_i$, where A_0 is the array containing the initial values of register A, M_i consists of the first i bits of the message M and T_i is a $d \times i$ matrix such that the jth column of T_i contains R_j, for $j = 0, 1, \ldots, i-1$. The matrix representation for A_i as $A_0 \oplus T_i M_i$, is given below.

$$\begin{bmatrix} a_0(0) \\ a_0(1) \\ a_0(2) \\ \vdots \\ a_0(d-1) \end{bmatrix} \tag{2.3}$$
$$\oplus \begin{bmatrix} r_0(0) & r_0(1) & \cdots & r_0(d-1) & y_0 & \cdots & y_{i-d-1} \\ r_0(1) & r_0(2) & \cdots & y_0 & y_1 & \cdots & y_{i-d} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ r_0(d-1) & y_0 & \cdots & y_{d-2} & y_{d-1} & \cdots & y_{i-2} \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ \vdots \\ m_{i-1} \end{bmatrix}.$$

Note that each row of T_i consists of i consecutive bits from the sequence $R_0 \| y$, and is closely related to the rows above and below it (each row is a bit-shifted version of the rows above and below it). The accumulation process begins after the two registers have been initialised and the input message prepared for processing. Many authentication algorithms refer to this message processing phase as a compression function. In this paper we refer to it as the accumulation phase of MAC generation. We consider the message processing phase as a whole, but also as the composition of many iterations of a sub-process. This is necessary in identifying potential forgery attacks.

Once all of the message bits have been processed, the MAC finalisation phase begins. This involves combining the final contents of register A with a masking value. Let $F = [f(0), f(1), \ldots, f(d-1)]$ denote this d-bit final mask. Thus, for a message M of length l the MAC is formed as follows:

$$\mathrm{MAC}(M_l) = A_l \oplus F = A_0 \oplus T_l M_l \oplus F.$$

2.3 Optional processes

For the general model using indirect message injection in the accumulation phase, as shown in Figure 2, we consider several options for the preparation and finalisation phases, respectively. The security implications of these options are discussed in Section 4 of this paper.

In the preparation phase the options relate to the initialisation of the two registers R and A, and to preparation of the message. Either register could be initialised with fixed values, such as all-zeroes; or with key dependent values, such as a segment from the keystream sequence y. The input message could be padded, by appending a specified sequence of bits at either the beginning, the end, or at both places. Alternatively no padding could be applied. If M_l is padded with n bits, we use M_p to denote the padded message, where $p = l + n$.

In the finalisation phase the final contents of accumulation register A are combined with a mask, F, as shown in Figure 2. The mask may be obtained from the sequence y used in the accumulation function, or from another sequence. For AE applications, this may be the sequence z used for the confidentiality component. In some cases, a null mask (all zero values) is used.

3 Recent proposals using this model

Several AE stream cipher proposals use the model presented in this paper, but with slight differences in the choice of options. As each proposal uses the same process for the accumulation phase (described in Section 2.2), we describe only the options taken for the preparation and finalisation phases of the ciphers. In this section we examine the MAC generation processes for Grain128-a, Sfinks and both version 1.4 and version 1.5 of 128-EIA3.

3.1 Grain-128a

Grain-128a [1] is a bit-based cipher from the Grain stream cipher family [16] with an added authentication mechanism. Grain-128a uses a 128-bit secret key and a 96-bit public IV, and generates two different sequences y and z, used for integrity and confidentiality applications, respectively. The MAC tag has length $d = 32$ bits. It is not clear from the cipher specification whether the message used for MAC generation is plaintext or ciphertext. However, the designers claim that the authentication mechanism is similar to ZUC (that is, 128-EIA3), so we will consider this message to be ciphertext. According to the classification of [5, 6], this algorithm therefore uses an encrypt-then-MAC procedure. The two discretionary phases of MAC generation for the Grain-128a design are as follows:

Preparation phase. Both registers R and A are initialised directly from the Grain-128a bitstreams. Let $[y_{-64}, y_{-63}, \ldots, y_{-33}, y_{-32}, \ldots, y_{-1}]$ denote the first 64 bits of the keystream y. Then A_0 and R_0 are as follows:

$$A_0 = \begin{bmatrix} y_{-64} \\ y_{-63} \\ \vdots \\ y_{-33} \end{bmatrix} \quad \text{and} \quad R_0 = \begin{bmatrix} y_{-32} \\ y_{-31} \\ \vdots \\ y_{-1} \end{bmatrix}.$$

The input message is padded with a single bit of value 1 at the end of the message, so $M_p = M_l \| 1 = [m_0, m_1, \ldots, m_{l-1}, 1]$.

Finalisation phase. After all $l + 1$ bits of the padded message have been processed, the contents of register A at time $l + 1$ represents the final MAC tag. That is, the final mask F is a null mask (consists of all zero values). So for Grain-128a, the matrix representation of the MAC tag is given by

$$\text{MAC}(M_p) = \begin{bmatrix} y_{-64} \\ y_{-63} \\ \vdots \\ y_{-34} \\ y_{-33} \end{bmatrix} \oplus \begin{bmatrix} y_{-32} & y_{-31} & \cdots & y_{l-32} \\ y_{-31} & y_{-30} & \cdots & y_{l-31} \\ \vdots & \vdots & & \vdots \\ y_{-2} & y_{-1} & \cdots & y_{l-2} \\ y_{-1} & y_0 & \cdots & y_{l-1} \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ \vdots \\ m_{l-1} \\ 1 \end{bmatrix}.$$

3.2 Sfinks

The Sfinks [7] stream cipher proposal submitted to eSTREAM [8] includes an authentication mechanism. The keystream generator uses an 80-bit secret key and an 80-bit IV to form an initial state for a 256-stage binary linear feedback shift register (LFSR), which is the major component of the keystream generator. Nonlinear filters are applied to the state of the LFSR to produce two different sequences y and z, used for integrity and confidentiality applications, respectively. Two additional 64-bit registers are used in the authentication mechanism in the manner shown in Figure 2. The Sfinks MAC tag has length $d = 64$ bits and is generated from the plaintext; it is then appended to the ciphertext to form the transmitted message. According to the classification of [5, 6], this algorithm therefore uses a MAC-then-encrypt procedure. The two discretionary phases of MAC generation for the Sfinks design are as follows:

Preparation phase. Before processing the message, both registers R and A are set to zero and an initialisation algorithm is used to incorporate the first 128 bits of keystream $[y_{-128}, y_{-127}, \dots, y_{-1}]$ into the initial values of these registers. This algorithm consists of updating the registers R and A 128 times, according to equations (2.1) and (2.2), but with $m_i = 1$ for $-127 \leq i \leq 0$. Note that this process is equivalent to setting both R_0 and A_0 to all zero and padding the input message at the beginning by concatenating with a sequence of 128 ones. That is, $M_p = [1, 1, \dots, 1] \| [m_0, m_1, \dots, m_{l-1}]$.

Finalisation phase. After all of the bits of M_p have been processed, the contents of register A are combined (by XORing) with a final mask that comprises 64 consecutive bits from the confidentiality sequence z, beginning immediately after the segment used to encrypt the input message. For Sfinks, the matrix representation of the MAC tag is given by

$$\mathrm{MAC}(M_p) = \begin{bmatrix} 0 & 0 & \cdots & 0 & y_{-128} & \cdots & y_{l-65} \\ 0 & 0 & \cdots & y_{-128} & y_{-127} & \cdots & y_{l-64} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & y_{-67} & y_{-66} & \cdots & y_{l-3} \\ 0 & y_{-128} & \cdots & y_{-66} & y_{-65} & \cdots & y_{l-2} \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \\ m_0 \\ \vdots \\ m_{l-1} \end{bmatrix} \oplus \begin{bmatrix} z_l \\ z_{l+1} \\ z_{l+2} \\ \vdots \\ z_{l+62} \\ z_{l+63} \end{bmatrix}.$$

3.3 128-EIA3 version 1.4

Version 1.4 of the 128-EIA3 [10] integrity algorithm uses the ZUC stream cipher [11] as a keystream generator. ZUC is a word-based stream cipher operating on 32-bit words. It uses a 128-bit secret key and a 128-bit IV. The 128-EIA3 MAC tag has length $d = 32$ bits and is generated from the ciphertext before the two are concatenated to form the final transmitted message. According to the classification of [5, 6], this algorithm thus uses an encrypt-then-MAC procedure. The two discretionary phases of MAC generation for the 128-EIA3 version 1.4 design are as follows:

Preparation phase. The 128-EIA3 algorithm does not use physical registers for R and A, but instead represents these registers using variables k and T, respectively. In this paper we retain our notation, and use R and A for the two 32-bit intermediate storage variables. In the preparation phase, register A is initialised with all zero values, and register R is initialised with bits from the keystream sequence y. Let $[y_{-32}, y_{-31}, \ldots, y_{-1}]$ denote the first 32 bits of y. Then

$$R_0 = \begin{bmatrix} y_{-32} \\ y_{-31} \\ \vdots \\ y_{-1} \end{bmatrix}.$$

The input message M_l is padded by adding one bit of value 1 at the end of the message, so $M_p = M_l \| 1 = [m_0, m_1, \ldots, m_{l-1}, 1]$.

Finalisation phase. After all $l + 1$ bits of the padded message have been processed, the contents of register A at time $l + 1$ are combined with the final mask. The final mask consists of 32 consecutive bits from the sequence y, beginning at y_{l+32}. The final 128-EIA3 version 1.4 MAC tag is given by

$$\mathrm{MAC}(M_p) = \begin{bmatrix} y_{-32} & y_{-31} & \cdots & y_{-1} & y_0 & \cdots & y_{l-32} \\ y_{-31} & y_{-30} & \cdots & y_0 & y_1 & \cdots & y_{l-31} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ y_{-2} & y_{-1} & \cdots & y_{29} & y_{30} & \cdots & y_{l-2} \\ y_{-1} & y_0 & \cdots & y_{30} & y_{31} & \cdots & y_{l-1} \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ \vdots \\ m_{l-1} \\ 1 \end{bmatrix} \oplus \begin{bmatrix} y_{l+32} \\ y_{l+33} \\ \vdots \\ y_{l+62} \\ y_{l+63} \end{bmatrix}.$$

3.4 128-EIA3 version 1.5

Version 1.5 of 128-EIA3 was proposed in response to a successful forgery attack [12] on version 1.4. We discuss this attack in more detail in Section 4. Version 1.5 is identical to version 1.4 except for the values used for the final mask. Although the final mask is still obtained from the sequence y, instead of taking the 32 consecutive bits starting with y_{l+32}, the final mask in version 1.5 starts at the beginning of the next 32-bit word of keystream. Thus the 128-EIA3 version 1.5 MAC tag is given by

$$\mathrm{MAC}(M_p) =$$

$$\begin{bmatrix} y_{-32} & y_{-31} & \cdots & y_{-1} & y_0 & \cdots & y_{l-32} \\ y_{-31} & y_{-30} & \cdots & y_0 & y_1 & \cdots & y_{l-31} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ y_{-2} & y_{-1} & \cdots & y_{29} & y_{30} & \cdots & y_{l-2} \\ y_{-1} & y_0 & \cdots & y_{30} & y_{31} & \cdots & y_{l-1} \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ \vdots \\ m_{l-1} \\ 1 \end{bmatrix} \oplus \begin{bmatrix} y_{(\lceil l/32 \rceil+1)*32} \\ y_{((\lceil l/32 \rceil+1)*32)+1} \\ \vdots \\ y_{((\lceil l/32 \rceil+1)*32)+30} \\ y_{((\lceil l/32 \rceil+1)*32)+31} \end{bmatrix}.$$

4 Security analysis of MAC algorithms

In this section we consider the security of MAC tags generated using the input message indirectly, as in the general model described above. We consider the security provided by the model with respect to both forgery attacks and side channel attacks. We provide recommendations and design guidelines for providing resistance to these types of attacks. In our analysis we assume that the binary sequences generated by the keystream generators are pseudo-random and cannot be distinguished from a truly random source.

4.1 Forgery attacks on MAC algorithms

To be considered secure, a MAC algorithm using a k-bit key, K, and producing a d-bit MAC tag should provide resistance against *forgery attacks*. That is, an attacker who does not know the secret key K but knows other information, such as sets of input messages and the corresponding MACs formed using the secret key (and possibly other public information such as the initialisation vectors), should not be able to produce a valid MAC for any other message with any probability better than guessing [15, 19]. A naive attacker may attempt to guess either the d-bit MAC value, or the secret key, K, which could be used to compute the MAC for any message, requiring at most 2^d and 2^k guesses, respectively. Other attacks can be compared with the effort required for these naive approaches.

Consider the possibility of a forgery attack being conducted as follows. Suppose for a message M a MAC tag $\mathrm{MAC}_{K,\mathrm{IV}}(M)$ is generated using key K and IV. The sender intends to transmit M and the MAC tag $\mathrm{MAC}_{K,\mathrm{IV}}(M)$ to a particular receiver. Assume a man-in-the-middle attacker intercepts the message-MAC tag pair. The attacker tries to modify M and possibly also $\mathrm{MAC}_{K,\mathrm{IV}}(M)$. The attacker then sends the pair $(M', \mathrm{MAC}_{K,\mathrm{IV}}(M'))$ to the intended message recipient. If it is possible to alter M to M' and produce a valid $\mathrm{MAC}_{K,\mathrm{IV}}(M')$ for M' without any knowledge of the keystream sequences used to generate $\mathrm{MAC}_{K,\mathrm{IV}}(M)$, then the forgery attack will be successful. We investigate the possibility of such forgeries for modifications to the message involving flipping, deleting and inserting bits into M.

Our analysis of the resistance to forgery attacks considers the security of the accumulation process, and explores the security implications of particular choices from the various options for the preparation and finalisation phases. Recall from equation (2.3) that $\mathrm{MAC}_{K,\mathrm{IV}}(M_l) = A_0 \oplus T_l M_l \oplus F = (T_l M_l) \oplus (A_0 \oplus F)$. That is, the final MAC tag is simply the linear combination of the separate effects of the accumulation process $T_l M_l$ and the masking vector $A_0 \oplus F$. In many cases, modifying the bits between the ends of a non-zero message changes the segments of the (unknown) pseudorandom sequence accumulated in register A, so does not lead directly to forgery attacks. We consider possible forgeries related to modification of the bits at either of the ends of message M. We first explore the security of the accumulation process, represented in the matrix form as $T_l M_l$, and then explore the security implications of certain choices for the masking vector $A_0 \oplus F$.

4.1.1 Security considerations for the accumulation process

Consider a message M of length l, with no padding. Let $X = [x_0, x_1, \ldots, x_{d-1}]$ denote the output of the accumulation process. We represent this in matrix form as follows:

$$X(M_l) = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{d-1} \end{bmatrix} \tag{4.1}$$

$$= \begin{bmatrix} r_0(0) & r_0(1) & \cdots & r_0(d-1) & y_0 & \cdots & y_{l-d-1} \\ r_0(1) & r_0(2) & \cdots & y_0 & y_1 & \cdots & y_{l-d} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ r_0(d-1) & y_0 & \cdots & y_{d-2} & y_{d-1} & \cdots & y_{l-2} \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ \vdots \\ m_{l-1} \end{bmatrix}.$$

Where a message M is modified to obtain a new message M', we will use the notation $X' = [x'_0, x'_1, \ldots, x'_{d-1}]$ to refer to $X(M')$.

Bit flipping forgeries. Consider firstly the simple case where R is initialised with zero values. That is, $R_0 = [0, 0, \ldots, 0]$. Then all elements in the first column of matrix T_l are zero, and hence m_0, the first bit of M_l, has no effect on the value of $X(M_l)$. Let $\bar{m_i}$ denote the complement of m_i. Consider two messages $M_l = [m_0, m_1, \ldots, m_{l-1}]$ and $M'_l = [\bar{m_0}, m_1, \ldots, m_{l-1}]$. Clearly $X(M_l) = T_l M_l = T_l M'_l = X(M'_l)$. This collision leads directly to a forgery; the attacker can provide a valid X' for M' with probability of 1.

Similarly, since all elements in the second column of matrix T_l are zero, except possibly y_0, clearly m_1, the second bit of M_l, only affects bit x_{d-1} of $X(M_l)$. Modifying M_l so that $M'_l = [m_0, \bar{m_1}, \ldots, m_{l-1}]$ requires the attacker to guess only x'_{d-1} to construct a valid $X(M')$. The probability that $X(M_l) = X(M'_l)$ is therefore 0.5. Similarly, for a message $M''_l = [\bar{m_0}, \bar{m_1}, \ldots, m_{l-1}]$, the probability that $X(M_l) = X(M''_l)$ is 0.5. In general, if we flip bit m_i and any of the bits up to

m_i in the original message, for $0 \le i \le d-1$, then the probability of a collision with $X(M_l)$ is 2^{-i}. Such a forgery attack therefore succeeds with probability 2^{-i}.

Consider now the case where R_0 is known and of the form $[1, 0, 0, \dots, 0]$. Then all elements in the first column of matrix T_l are zero, except $r_0(0)$. Now m_0, the first bit of M_l, affects only the value of bit x_0 of $X(M_l)$. Thus it is possible to modify M_l by flipping m_0 and to provide a valid X' for $M'_l = [\bar{m_0}, m_1, \dots, m_{l-1}]$ by flipping x_0. That is, $X(M') = [\bar{x_0}, x_1, \dots, x_{l-1}]$. Thus an attacker can provide a valid X' for M' with probability of 1.

Similarly, for $R_0 = [1, 1, 0, \dots, 0]$, all elements in the first column of T_l are zero, except $r_0(0)$ and $r_0(1)$, and all elements in the second column of T_l are zero, except $r_0(1)$ and possibly y_0. Thus message bit m_0 affects only the values of bits x_0 and x_1 of $X(M_l)$, while message bit m_1 affects only the values of bits x_0 and possibly x_{d-1}. Modifying M_l by flipping both m_0 and m_1 requires the attacker to flip only x_1 to form x'_1, and to guess x'_{d-1} to provide a valid X'. Therefore the probability that an attacker can construct a valid vector X' for $M'_l = [\bar{m_0}, \bar{m_1}, \dots, m_{l-1}]$ is 0.5.

In general, for any known R_0, if we flip bit m_i and any of the bits up to m_i in the original message M_l, for $0 \le i \le d-1$, then we can construct a valid X' for this modified message M'_l by flipping the required bits in X and guessing the final i bits to form X'. Therefore, the probability that an attacker can construct a valid vector X' for M' is 2^{-i}.

Resistance to this type of forgery can be provided in two ways. Firstly, rather than using known values, R can be initialised using key dependent values such as a segment from the keystream sequence y. Secondly, the message M_l may be padded by concatenating with a segment of all ones, so that $M_p = [1, 1, \dots, 1] \| M_l$. The padding should consist of at least d ones, so that all message bits affect all bits in X, and hence all bits in the final MAC tag, in an unpredictable manner. Note that a key dependent initialisation of R may be the more efficient approach, as padding increases both the length of the message and the size of the matrix T, requiring at least $(d + l)$ operations to generate the final MAC tag.

Bit deletion forgeries. Consider modifying message $M_l = [m_0, m_1, \dots, m_{l-1}]$ by deleting the first bit, m_0, to obtain $M'_{l-1} = [m_1, m_2, \dots, m_{l-1}]$. Then the matrix T_{l-1} for M'_{l-1} is just the matrix T_l for M'_l without the last column of T_l. Note in equation (4.1) that row $i+1$ in matrix T is equivalent to row i shifted one position to the left, for $0 \le i \le d-2$, and with a new value as the final element in the row. Applying this to $X = T_l M_l$ and $X' = T_{l-1} M'_{l-1}$, it follows that $x_i = r_0(i) m_0 + x'_{i+1}$, for $0 \le i \le d-2$.

Now consider again the simple case where R is initialised with zero values. As all elements in the first column of matrix T_l are zero, $x_i = x'_{i+1}$ for $0 \le$

$i \leq d-2$. That is, for $M'_{l-1} = [m_1, m_2, \ldots, m_{l-1}]$, it is clear that $X'(M_l) = [\beta, x_0, x_1, \ldots, x_{d-2}]$, for some unknown β. We call this the sliding property of the product $T_l M_l$. An attacker can slide $X(M_l)$ and guess β, and hence provide a valid X' for M' with probability of 0.5.

Similarly, an attacker can form a message $M''_{l-i} = [m_{i-1}, \ldots, m_{l-1}]$ by deleting the first i bits of the message M_l. Now the output of the accumulation phase is

$$X(M''_{l-i}) = [\beta_0, \beta_1, \ldots, \beta_{i-1}, x_0, x_1, \ldots, x_{d-i-1}],$$

where the bits $\beta_0, \beta_1, \ldots, \beta_{i-1}$ must be guessed by the attacker. The attacker can provide a valid X for M''_{l-i} with probability 2^{-i}, for $1 \leq i \leq d$. Note that for $i = d$, this is effectively a brute force attack on X, so this attack is only effective for deletion of up to the first d bits of M_l. This attack can also be adapted for the case where R_0 is non-zero but known; all that is required is to flip the appropriate bits of X, as described for bit flipping forgeries, before shifting X and guessing the bits $\beta_0, \beta_1, \ldots, \beta_{i-1}$.

Suppose now that R_0 is unknown, but that the first j bits of the message are known to be zeroes. These bits do not cause any keystream to be accumulated into register A, so we can again delete the first $i \leq j$ bits of the input message, shift X by i places and guess $\beta_0, \beta_1, \ldots, \beta_{i-1}$ to obtain a valid X' for the modified message M'_{l-i}. The attacker can again provide a valid X' for M' with probability of 2^{-i} for $1 \leq i \leq d$.

Resistance to forgery attacks based on the deletion of bits from the start of the message can be provided in two ways. Firstly, the message can be padded at the start with at least d ones. Alternatively, R can be initialised using key dependent values, and the message can be padded at the start with a single one.

Now suppose we consider deleting bits from the end of the message. Assuming that $l > d$, if we delete the last bit of M_l, then $M'_{l-1} = [m_0, m_1, \ldots, m_{l-2}]$. Then $x_i = x'_i + y_{l-d+i-1} m_{l-1}$, for $0 \leq i \leq d-1$. If $m_l = 1$ the second term is unknown since it involves keystream bits, and the attacker must guess these elements of the keystream. Thus it is not feasible to obtain a forgery in this way. However, if $m_{l-1} = 0$, then $X(M_l) = X(M'_{l-1})$, and the attacker can forge the MAC tag for the new message M'_{l-1} with probability 1. Similarly, if the final j bits of the input message are known to be zero, then deleting these bits will not change the final contents of the accumulation register A, and hence $X = X'$ and a forgery is again possible. Such forgeries can be prevented by padding the message with a final one, since this is equivalent to having a message of length $l+1$ with $m_l = 1$.

Bit insertion forgeries. Consider modifying message $M_l = [m_0, m_1, \dots, m_{l-1}]$ by inserting i zero bits at the end of the message M_l, to obtain $M'_{l+i} = [m_0, m_1, \dots, m_{l-1}, 0, 0, \dots, 0]$. Using our matrix representation, performing the accumulation phase for M'_{l+i} requires adding i columns to the matrix T_l used for M_l. During the accumulation process, regardless of the values in these columns, multiplying by the additional message bits of value zero does not change the value of X. Hence $X(M'_{l+i}) = X(M_l)$, for any $i > 0$. Again, this collision leads to an obvious forgery attack which succeeds with probability 1.

Now consider modifying message $M_l = [m_0, m_1, \dots, m_{l-1}]$ by inserting an additional 1 at the end of message M_l, to obtain $M'_{l+1} = [m_0, m_1, \dots, m_{l-1}, 1]$. In our matrix representation, adding an additional bit to the end of M_l requires the addition of another column to T_l. The additional column of T will be multiplied by the added message bit of value 1, and this may change the values of X'. The probability of obtaining a valid MAC tag X' for this modified message M'_{l+1} is the same as the probability of a brute force attack on the MAC. Therefore, forgery attacks consisting of inserting bits of value zero at the end of the message can be prevented by padding the message with a bit of value 1 at the end. An equivalent solution, which we discuss in the following section, is to use a masking term that depends on the message length.

Now consider modifying message M_l in a different manner; inserting a zero bit at the beginning of message M_l, to obtain $M'_{l+1} = [0, m_0, m_1, \dots, m_{l-1}]$. From the structure of T_l and T_{l+1} it follows that, in a manner similar to that described in the discussion of forgeries associated with bit deletion at the start of the message, $x'_i = r_0(i)0 + x_{i+1} = x_{i+1}$, for $0 \le i \le d-2$. Hence for M'_{l+1} the output of the accumulation phase is $X(M'_{l+1}) = [x_1, x_2, \dots, x_{d-1}, \alpha]$, for some unknown α. An attacker can guess α and hence provide a valid X' for M'_{l+1} with probability equal to 0.5.

Similarly, an attacker can form a message M''_{l+i} by adding i zeroes to the beginning of message M_l. The attacker can form the new X'' for M''_{l+i} as $X'' = [x_i, x_{i+1}, \dots, x_d, \alpha_0, \alpha_1, \dots, \alpha_{i-1}]$, where the bits $\alpha_0, \alpha_1, \dots, \alpha_{i-1}$ must be guessed. Therefore an attacker can provide a valid X'' for M''_{l+i} with probability 2^{-i}, for $1 \le i \le d$. This is the basis of the previously reported attack on 128-EIA3 version 1.4 [12].

If R is initialised with zeroes, so that $R_0 = [0, 0, \dots, 0]$, then this attack will work for inserted bits of either value since, using our matrix representation, the first i bits of M''_{l+i} are all multiplied by zeroes in the first $d - i + 1$ rows of T_{l+i}. Therefore, the inserted bits affect only the last $i - 1$ bits of X''_{l+i}, which are bits that must be guessed anyway. Furthermore, this forgery attack can be adapted to the case of known (but not necessarily all zero) R_0. In that case, the effects of any inserted bits of value 1 can be determined and allowed for in applying the attack.

From the discussion above, it is clear that attacks involving the insertion of bits at the start of a message can be prevented by ensuring that R is initialised with unknown values (keystream) and that the start of the message is padded with at least one bit of value one.

4.1.2 Security considerations for the masking vector $A_0 \oplus F$

The forgeries discussed thus far can all be prevented by making suitable choices for R_0 and padding the message M_l appropriately. More specifically, R should be initialised with keystream bits, so that the contents are unknown, and the message M_l should be padded with a bit of value 1 at both ends. The masking vector $A_0 \oplus F$ provides an alternative means of preventing many of these forgeries. We explore the security implications of various options for this term. We begin by noting that if $A_0 \oplus F$ is to contribute to the security of the MAC tag, it is important that the contents are unknown to an attacker. Therefore, at least one of A_0 and F must be sourced from keystream.

If R_0 is known and there is no message padding, then the accumulation term $X = T_l M_l$ is vulnerable to attacks involving the insertion or deletion of zeroes at the end of the message, and to attacks involving the insertion or deletion of bits at the start of the message. If R_0 is unknown then only the attacks involving the insertion or deletion of zeroes apply. Forgeries involving insertions or deletions at the start of the message rely on the sliding property of $T_l M_l$, so that X' is obtained from X by shifting the values in X right or left, respectively, adjusting if necessary and guessing the remaining places. These attacks can be prevented by using an appropriate mask. For the mask to be effective in this respect, it is important that changes in the length of the message do not result in corresponding changes in the position of the mask bits or the sliding property of the accumulation phase will apply to the MAC as a whole. The easiest way to satisfy this requirement is to initialise A with bits from a fixed position, such as the start of the keystream sequence y.

Conversely, forgeries involving the insertion or deletion of zeroes at the end of the message rely on the fact that the additional bits have no effect on the accumulated value X. Such forgeries can be prevented by using an unknown mask that depends on the message length. One example of such a mask is constructed using the values of F obtained as d consecutive keystream bits, starting at a fixed distance from the last bit of the keystream sequence used in the accumulation process. Note that padding the message with a final 1 is equivalent to including $F = [y_{l-d+1}, y_{l-d+2}, \ldots, y_l]$ as the final mask. That is, it is equivalent to starting F at a position $d - 1$ bits before the last bit used in the unpadded case (y_{l-1}).

Together, the choices for A_0 and F discussed above provide an effective alternative to message padding as a means of preventing bit insertion and deletion attacks. Note, however, that the masking term $A_0 \oplus F$ cannot prevent the attacks based on flipping the message bits.

4.2 Side channel attacks

These are attacks that apply to the (software or hardware) implementations of cryptographic systems. Variations in the timing, power consumption or electromagnetic radiation as the message is processed may leak information, such as the secret key used or the plaintext input message to the integrity components. In order to carry out such attacks, the attacker must have the means to access the device and measure the particular characteristic.

In the accumulation phase of the MAC generation process, the input message acts as a control to determine which states of register R are accumulated into A. This control mechanism is represented by the scalar multiplication operation $\otimes$, as shown in Figure 2. An intuitive and effective way to implement the accumulation phase is to use branching processes that are conditional on the input message bit at each time step. If M_t, the input bit at time t, is equal to one; then R_t will be accumulated into register A using the XOR operation; whereas if the input is zero then we do not need to update the value of A.

If this approach is used, there will clearly be differences in the computation time and power consumption for distinct steps in the accumulation process, dependent on the individual bits of the input message. This opens up an avenue for timing attacks, whereby the attacker can learn whether a message bit is 1 or 0 at each clock by monitoring the time taken to complete each step of the accumulation process. The message can then be recovered bit by bit for as long as the cipher is monitored. Simple power analysis is also a possible attack method, since state accumulation at clocks where the input message bit is of value 1 would have a specific power consumption profile.

Side channel attacks associated with MACs represent a threat to the confidentiality of information in situations where authenticated encryption is used and the MAC is computed using the plaintext as the input to the MAC. The ciphertext will be transmitted, and is therefore accessible by an attacker, but the encryption provides confidentiality for the plaintext. However, a side channel attack on the integrity component may reveal this plaintext. Note that if a MAC is computed for the ciphertext, then the side channel attacks do not reveal any additional information, as we assume that an attacker will have access to the ciphertext. Therefore, for authenticated encryption, resistance to side channel attacks can be increased by first encrypting the message and then using the resulting ciphertext as input to the integrity component. Similarly, if a MAC is formed for plaintext that will not be encrypted (that is, we want authentication only, not authenticated encryption) then the side channel attacks do not reveal any additional information.

Side channel attacks against implementations of the authentication mechanism based on the different timing and power consumption during message accumulation can be prevented. This requires a slight modification to the design for the general model given in Figure 2 so that the register A is replaced with two registers: A^0 and A^1. Given these two d-bit registers, the update function given in equation (2.2) is amended so that one register or the other is updated at each time step, conditional on the value of M_t. That is, if $M(t) = 0$, then $A^0_t = A^0_{t-1} \oplus R_{t-1}$ and A^1 is unchanged. However, if $M(t) = 1$, then $A^1_t = A^1_{t-1} \oplus R_{t-1}$ and A^0 is unchanged. Thus the same operation is performed at each step. This modification requires an additional register to implement the design, and additional accumulation operations (XOR) will be performed. On average, the number of additional operations will be around half the message length. In order to implement this modified model, the MAC finalisation phase must be specified. One simple way to complete the MAC process is to use the contents of just one of the registers, say A^0, and treat it as register A in the original model. The final contents of the other register are discarded.

4.3 Statistical analysis

Statistical tests can be used to test whether the values obtained as MAC tags for a particular function are uniformly distributed, that is, whether the probability of obtaining any particular MAC tag value for any input message is equally likely over the set of all possible key–IV pairs. Three different statistical tests have been applied to test for this property in a sample of d-tuples, namely the poker test [4], the universal test [18] and the repetition test [14]. It is shown in [14] that for $d > 20$ the repetition test is the “best” test to use since this test requires significantly fewer samples than the other two tests.

The repetition test counts the number of different d-bit patterns occurring in a set of d-tuples. Based on the birthday paradox, there is an approximately fifty percent chance of finding at least two d-tuples that match in a randomly generated set of size $2^{d/2}$.

To apply the repetition test to a particular stream cipher based MAC, where the length of the MAC tag is d-bits, it is necessary to generate a sample of sufficient size S such that a limited number of repetitions is expected with high probability, under the hypothesis that the sample is selected at random from a set of uniformly distributed d-tuples. Under this hypothesis, the number of repeated patterns in a particular sample of S d-tuples follows a Poisson distribution. The mean, λ, for this distribution can be calculated using the following equation:

$$\lambda = S - N(1 - e^{-S/N}),$$

where $N = 2^d$. If we choose $S = 2^{d/2+3}$, then the value of λ is approximately 32, for $d > 20$. The set of S d-tuples is generated by fixing one of the inputs to the MAC generation algorithm, and varying the other inputs. For example, the MAC tags may be generated for a fixed message, using different secret keys and IVs. Alternatively, the MAC tags may be generated for many different messages, using a fixed secret key and IV.

The process for running the repetition test is as follows:

(i) Generate a sample of $S = 2^{d/2+3}$ d-bit MAC values.

(ii) Sort S and count the number of repeated values.

(iii) Compare the number of repetitions from the test with the expected number $\lambda = 32$.

(iv) Compare the test statistic to the relevant statistical tables to find the probability of obtaining the observed data under the assumed null hypothesis (the p-value of the test).

The null hypothesis is that the number of repetitions follows a Poisson distribution with mean $\lambda = 32$. The test is applied as a two-tailed test of significance, as either too many or too few repetitions of particular d-tuples may indicate non-randomness. A low p-value provides evidence to reject the null hypothesis. The amount of data required to run this test for the common MAC sizes of 32, 64 and 128 bits is 2^{19}, 2^{35} and 2^{67} MAC tags, respectively. Clearly the sample size required for longer MAC tags makes this test infeasible for 128-bit MACs.

5 Security analysis for existing ciphers

In this section we apply the security analysis from Section 4 to the algorithms described in Section 3. Specifically, we discuss the feasibility of applying forgery attacks, side channel attacks and statistical analysis to each of these algorithms.

5.1 Forgery attacks

5.1.1 128-EIA3 version 1.4

A forgery attack on 128-EIA3 version 1.4 presented by Fuhr et al. [12] describes how a message, M, can be modified by inserting additional bits of value zero at the beginning of M. The attacker makes use of the sliding property between the keystream sequence used in the accumulation process and the final mask to compute a valid MAC for the modified message. We explain their attack using the matrix representation introduced in this paper.

For 128-EIA3, A_0 is initialised with all zeroes, so the masking vector is only F. That is, $\text{MAC}(M_p) = T_p M_p \oplus F$. However, F is formed from y, the same sequence used as input to R. It consists of 32 consecutive bits of y beginning a fixed distance after the last bit of y used for the accumulation process. Any modification to M that increases or decreases the message length causes a corresponding shift in the segment of y used to form F.

Consider inserting a zero at the start of M to form M'. From Section 4 it is clear that X', the output from the accumulation process for M', is related to the value X for the accumulation of the original message M by the sliding relationship, so $X' = [x_1, x_2, \ldots, x_{d-1}, \alpha]$. Since F also slides by one bit when a zero is appended to the message, the entire MAC has this property, as noted in [12]. The attacker need only guess one bit, α, to form a valid MAC for M'. The same process applies for inserting $i < d$ zeroes at the start of the message. The attacker can obtain a valid MAC for such a message with probability 2^{-i}.

We now present another forgery attack on version 1.4 of 128-EIA3; this is a bit deletion attack which can be applied where the message starts with one or more zeroes. If we delete one of these zeroes, then the result X' for the accumulation phase of the modified message M' is related to the value X for the accumulation of the original message M by the sliding relationship, so $X' = [\beta, x_0, x_1, \ldots, x_{d-2}]$. Since F also slides by the same amount the entire MAC has this property. The attacker need only guess one bit, β, to form a valid MAC for M'. The same process applies for deleting $i < d$ zeroes from the start of the message. The attacker can obtain a valid MAC for such a message with probability 2^{-i}.

For other types of forgeries, resistance to bit flipping forgeries is provided by initialising R with keystream, and forgeries involving bit insertion or deletion at the end of the message are prevented because the message has been padded with a single 1 at the end of the message.

5.1.2 128-EIA3 version 1.5

The modification to the starting position of the 32-bit segment of y used to form F, introduced in version 1.5 of 128-EIA3, breaks the sliding property on F and provides effective resistance to the types of forgery discussed above. As was the case for 128-EIA3 version 1.4, resistance to bit flipping forgeries is provided by initialising R with keystream, and forgeries involving bit insertion or deletion at the end of the message are prevented by padding with a single 1 at the end of the message.

5.1.3 Grain-128a

The cipher Grain-128a initialises both registers R and A with keystream. Bit flipping forgery attacks are prevented by this initialisation of R, and forgeries involving insertion or deletion of bits at the beginning of the message are prevented by this initialisation of A, as this breaks the sliding property for the masking value $A_0 \oplus F$. Forgeries involving insertion or deletion of bits at the end of the message are prevented by padding with one bit of value 1 at the end of the message.

5.1.4 Sfinks

The cipher Sfinks uses message padding, by prepending $2d$ ones to M_l to ensure both registers R and A are initialised with keystream before the actual message M_l is input to the accumulation phase. Bit flipping forgery attacks and forgeries involving insertion or deletion of bits at the beginning of the message are prevented by this initialisation of R and A. Additionally, the final mask F is constructed from a 64 bit segment of z (the sequence used to encrypt M_l for confidentiality) beginning at z_l. Forgeries involving insertion or deletion of bits at the end of the message are prevented by the use of this segment that is related to the length of the message.

5.2 Side channel attacks

Reference implementations are publicly available for each of 128-EIA3 version 1.4 (for example, the C code implementation included in [10, Annex 2]), 128-EIA3 version 1.5 [11] and Sfinks [7]. In each case, these algorithms are implemented with branching statements in the accumulation phase, and are therefore susceptible to the side channel attacks discussed in Section 4. This is a security concern if these implementations are used in actual applications.

We have been unable to locate any publicly available reference implementation for Grain-128a. However, the considerations discussed above obviously apply in developing any practical implementations of this cipher.

5.3 Statistical analysis

The repetition test described in Section 4.3 was applied to sets of MAC tags generated using each of three algorithms: the Grain-128a algorithm and versions 1.4 and 1.5 of the 128-EIA3 algorithm. Each of these produces a MAC tag of length 32 bits.

In each case, the test was applied to seven different sets of data, each consisting of 2^{19} MAC tags. Four of the data sets were formed using random messages and fixed secret keys, two of the data sets were formed using fixed messages and ran-

		128-EIA3 v1.4		128-EIA3 v1.5		Grain-128a	
Message	Key	NR	p-value	NR	p-value	NR	p-value
Random 1	Fixed	38	0.2888	34	0.7237	27	0.3768
Random 2	Fixed	27	0.3768	40	0.1573	31	0.8597
Random 3	Fixed	24	0.1573	25	0.2159	35	0.5959
Random 4	Fixed	35	0.5959	25	0.2159	24	0.1573
Fixed	Random 1	25	0.2159	31	0.8597	26	0.2888
Fixed	Random 2	26	0.2888	35	0.5959	38	0.2888
Random 4	Random 3	31	0.8597	31	0.8597	31	0.8597

Table 1. Results of repetition test for 128-EIA3 and Grain-128a algorithms.

domly generated secret keys, while the seventh data set used random messages and random secret keys. The number of repetitions observed (NR) and the associated p-values are given for each set of tags and each algorithm in Table 1.

As shown in Table 1, the minimum p-value for each of the algorithms tested was over 15%. Thus, there is insufficient evidence to reject the null hypothesis in any of the cases tested, and the results support the conclusion that the MAC tags for each of these algorithms are uniformly distributed.

Similar tests can be carried out on sets of MAC tags from the Sfinks algorithm, but the longer tag length (64 bits) will require larger storage (2^{35} tags per test) and a longer processing time.

6 Conclusion

In this paper we describe a general model for using a stream cipher to generate a MAC tag, where the input message is indirectly injected into the integrity device in the accumulation phase. The input message could be plaintext or ciphertext. We outline the options available for the preparation and finalisation phases in this model, and relate these options to the security provided against forgery attacks. We consider forgeries where a message is modified either by flipping, inserting or deleting message bits from either the start or the end of the message. In addition, we consider the application of certain forms of side-channel attack. We examine several authentication proposals that can be described by this model: Grain-128a, Sfinks and 128-EIA3 (both versions 1.4 and 1.5).

The security analysis with respect to forgeries of the general model reveals that a man-in-the-middle attacker could intercept a message M and MAC $\mathrm{MAC}_{K,\mathrm{IV}}(M_l)$ and modify the message and possibly also the MAC in order to provide a valid MAC for the modified message. The probability that the attacker constructs a valid

MAC for their modified message is better than guessing, for many combinations of options. In order to prevent bit flipping forgeries, the register R should be initialised with keystream. This also reduces the scope of forgeries involving the insertion or deletion of bits at the start of the message. Prepadding the message with at least d ones is a feasible but arguably less efficient alternative. To prevent the remaining bit insertion and deletion forgeries, both of the following practices should also be adopted:

(i) Pad the message with a 1 at the start and/or initialise register A with keystream from a fixed location in the sequence.

(ii) Pad the message with a 1 at the end and/or construct the final mask F so that the contents of F depend on the length of the message.

Implementing the accumulation phase of the authentication mechanism using a branching process conditional on the input message bit at each time step is an intuitive and effective approach. However, this approach may also contribute to the mechanism being vulnerable to certain forms of side channel attack, such as a timing attack or simple power analysis. An implementation that consumes the same amount of time and power for each accumulation step, regardless of the value of the input message bit, will prevent this type of side channel attack. Such an implementation may be achieved at the cost of an additional storage register and a slight reduction in throughput.

As mentioned in Section 1, it was shown in [5, 6] that the AE method offering the greatest security is provided by an algorithm which first performs encryption and then uses the ciphertext to form a MAC, provided that the underlying integrity component is strongly unforgeable. Of the AE algorithms described in Section 3, the ones which use the encrypt-then-MAC procedure and are strongly unforgeable (based on the results of Section 4.1) are 128-EIA3 version 1.5 and Grain-128a. Both of these algorithms also passed the statistical test of Section 4.3, so there is no evidence of non-uniformity in the distributions of these MAC tags. The only concern from our analysis of these two algorithms is the need to be careful in the actual implementation to avoid possible side channel attacks.

Acknowledgments. This research is partially supported by the Omani government, through the provision of a doctoral scholarship for Mufeed Al Mashrafi.

Bibliography

[1] M. Ågren, M. Hell, T. Johansson and W. Meier, Grain-128a: A new version of Grain-128 with optional authentication, *Int. J. Wireless Mobile Comput.* **5** (2011), no. 1, 48–59.

[2] M. AlMashrafi, H. Bartlett, L. Simpson, E. Dawson and K. Wong, Analysis of indirect message injection for MAC generation using stream ciphers, in: *Proceedings of the 17th Australasian Conference on Information Security and Privacy: ACISP 2012*, Lecture Notes in Comput. Sci. 7372, Springer (2012), 138–151.

[3] H. Bartlett, M. AlMashrafi, L. Simpson, E. Dawson and K. Wong, A general model for MAC generation using direct injection, in: *Information Security and Cryptology* (Inscrypt 2012), Lecture Notes in Comput. Sci. 7763, Springer (2013), 198–215.

[4] H. Beker and F. Piper, *Cipher Systems: The Protection of Communications*, Wiley, 1982.

[5] M. Bellare and C. Namprempre, Authenticated encryption: Relations among notions and analysis of the generic composition paradigm, in: *Advances in Cryptology – Asiacrypt 2000*, Lecture Notes in Comput. Sci. 1976, Springer (2000), 531–545.

[6] M. Bellare and C. Namprempre, Authenticated encryption: Relations among notions and analysis of the generic composition paradigm, *J. Cryptol.* **21** (2008), no. 4, 469–491.

[7] A. Braeken, J. Lano, N. Mentens, B. Preneel and I. Verbauwhede, SFINKS: A synchronous stream cipher for restricted hardware environments, eSTREAM, ECRYPT Stream Cipher Project, Report 2005/026, `www.ecrypt.eu.org/stream/sfinks.html`.

[8] eSTREAM, The ECRYPT Stream Cipher Project, `www.ecrypt.eu.org/stream`.

[9] ETSI/SAGE, Specification of the 3GPP confidentiality and integrity algorithms UEA2 & UIA2, Document 2: SNOW 3G Specification, 2006.

[10] ETSI/SAGE, Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3, Document 1: 128-EEA3 and 128-EIA3 Specification, Version 1.5, 4th January, 2011.

[11] ETSI/SAGE, Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3, Document 2: ZUC Specification, Version 1.5, 4th January, 2011.

[12] T. Fuhr, H. Gilbert, J. Reinhard and M. Videau, Analysis of the initial and modified versions of the candidate 3GPP integrity algorithm 128-EIA3, in: *Selected Areas in Cryptography 2011*, Lecture Notes in Comput. Sci. 7118, Springer (2012), 230–242.

[13] J. Golić, Modes of operation of stream ciphers, in: *Selected Areas in Cryptography: SAC2000*, Lecture Notes in Comput. Sci. 2012, Springer (2001), 233–247.

[14] H. M. Gustafson, *Statistical analysis of symmetric ciphers*, PhD thesis, Queensland University of Technology, 1996.

[15] P. Hawkes, M. Paddon, G. Rose and M. Wiggers de Vries, Primitive specification for NLSv2, eSTREAM, ECRYPT Stream Cipher Project, Report 2006/036, `www.ecrypt.eu.org/stream/nlsp3.html`.

[16] M. Hell, T. Johansson and W. Meier, Grain: A stream cipher for constrained environments, *Int. J. Wireless Mobile Comput.* **2** (2007), no. 1, 86–93.

[17] X. Lai, R. Rueppel and J. Woollven, A fast cryptographic checksum algorithm based on stream ciphers, in: *Advances in Cryptology – AUSCRYPT'92*, Lecture Notes in Comput. Sci. 718, Springer (1993), 339–348.

[18] U. M. Maurer, A universal statistical test for random bit generators, *J. Cryptol.* **5** (1992), 89–105.

[19] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[20] Y. Nakano, C. Cid, K. Fukushima and S. Kiyomoto, Analysis of message injection in stream cipher-based hash functions, in: *Applied Cryptography and Network Security*, Lecture Notes in Comput. Sci. 6715, Springer (2011), 498–513.

[21] Y. Nakano, J. Kurihara, S. Kiyomoto and T. Tanaka, On a construction of stream-cipher-based hash functions, in: *Security and Cryptography (SECRYPT 2010)*, SciTePress (2010), 334–343.

[22] G. J. Simmons, A survey of information authentication, *Proc. IEEE* **76** (1988), no. 5, 603–620.

Received January 23, 2013; accepted August 1, 2013.

Author information

Mufeed Al Mashrafi, Institute for Future Environments, Science and Engineering Faculty, Queensland University of Technology, Brisbane, Australia.
E-mail: mufeed03@hotmail.com

Harry Bartlett, Institute for Future Environments, Science and Engineering Faculty, Queensland University of Technology, Brisbane, Australia.
E-mail: h.bartlett@qut.edu.au

Ed Dawson, Institute for Future Environments, Science and Engineering Faculty, Queensland University of Technology, Brisbane, Australia.
E-mail: e.dawson@qut.edu.au

Leonie Simpson, Institute for Future Environments, Science and Engineering Faculty, Queensland University of Technology, Brisbane, Australia.
E-mail: lr.simpson@qut.edu.au

Kenneth Koon-Ho Wong, Institute for Future Environments, Science and Engineering Faculty, Queensland University of Technology, Brisbane, Australia.
E-mail: kk.wong@qut.edu.au