ANALYSIS OF AUTHENTICATED ENCRYPTION STREAM CIPHERS

Sui-Guan Teo, Mufeed Al-Mashrafi, Leonie Simpson, Ed Dawson

Information Security Institute,

Queensland University of Technology,

GPO Box 2434, Brisbane Qld 4001, Australia

{sg.teo, mufeed.almusharafi, lr.simpson, e.dawson}@qut.edu.au

ABSTRACT

Authenticated Encryption (AE) is the cryptographic process of providing simultaneous confidentiality and integrity protection to messages. AE is potentially more efficient than applying a two-step process of providing confidentiality for a message by encrypting the message and in a separate pass, providing integrity protection by generating a Message Authentication Code (MAC) tag. This paper presents results on the analysis of three AE stream ciphers submitted to the recently completed eSTREAM competition. We classify the ciphers based on the methods the ciphers use to provide authenticated encryption and discuss possible methods for mounting attacks on these ciphers.

Key Words: Authenticated Encryption, Message Authentication Codes, Stream Cipher

1. INTRODUCTION

Modern information technology systems use cryptographic mechanisms to provide information security. Aspects of information security include confidentiality and data integrity. Confidentiality is the assurance that information is kept secret from unauthorised people. Where the storage or transmission medium is insecure, confidentiality is provided by using encryption algorithms. Integrity is the assurance that modification of messages will be detected. Integrity protection is achieved by using an authentication algorithm.

Symmetric cryptographic primitives consist of block ciphers, stream ciphers and some Message Authentication Codes (MAC). In symmetric cryptography, both the sender and receiver have in their possession the same secret information (the secret key) and optional public information they use along with an algorithm to encrypt or decrypt data or produce a MAC tag.

AE aims to provide simultaneous confidentiality and integrity for information using symmetric cryptography. The AE mechanism shown in Figure 1, works as follows: for encryption, a plaintext (denoted *P*) is changed to an unreadable format known as ciphertext (denoted *C*) and the MAC tag (denoted *T*) of the message is also calculated. This is performed using an authenticated encryption algorithm, which takes as input the plaintext message, the secret key (denoted *K)* and optionally, some public information (denoted *IV*). The output of the AE encryption consists of the ciphertext and MAC tag. These are sent across the unsecured channel to the receiver.

Upon receiving the ciphertext (*C*') and MAC tag (*T*) the receiver uses the authenticated decryption algorithm to recover the message and check whether it has been modified. This decryption algorithm takes as input the same key and IV used in the authenticated encryption algorithm and the received ciphertext *C*'. The output of the authenticated decryption algorithm is the plaintext *P'* and the MAC tag *T'*. The receiver checks if the value of *T'* is equal to *T*. If it is not the same, the receiver will know that the message has been modified.

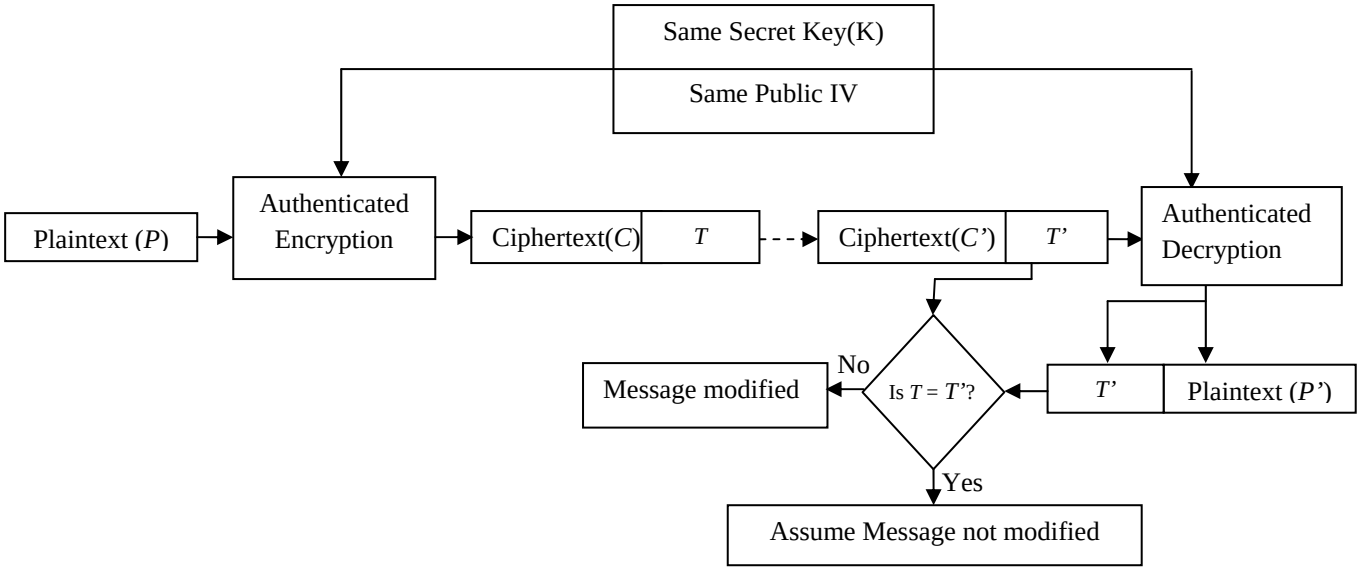

Figure 1. Generic AE Diagram

1.1 Stream Ciphers

Stream ciphers encrypt one character at a time using a time-varying function. Traditionally, the character size is one bit. Modern day stream ciphers, especially those meant for software environments can encrypt more than one bit at a time. These stream ciphers are called word-based stream ciphers. In either case, a critical component of the stream cipher is a keystream generator.

A keystream generator typically consists of a series of storage registers called stages. These stages each contain one-bit for bit-based stream ciphers, or a series of bits for word-based stream ciphers. Typical word sizes are 16-bits and 32-bits. The contents of these stages form the internal state of the keystream generator. The stages are often arranged to form shift registers.

The operation of the keystream generator occurs in two phases. The initialisation phase and the keystream generation phase. In the initialisation phase, the key and IV are loaded into the internal state using an initialisation function. After this is done, the stream cipher transitions to the keystream generation phase. In the keystream generation phase, the internal state gets updated using a state update function. Selected stages are then combined using a carefully chosen output function to produce keystream. Note that this may be produced either one bit or one word at a time.

The most common function used as the encryption and decryption function is binary addition modulo 2, also known as the XOR operation. The XOR function is used as it is fast and easy to implement in both hardware and software. Furthermore, due to XOR's commutative properties, the same device can be used to perform both encryption and decryption functions. A stream cipher which uses the XOR function for encryption and decryption is called a binary-additive stream cipher. A diagram of a binary-additive stream cipher is shown in Figure 2.

To encrypt a message, the sender initialises the keystream generator with the secret key and IV and then generates a length of keystream. The keystream and message are then combined using XOR operations to produce ciphertext. To decrypt the message, the receiving end must use the same key and IV to initialise the keystream generator and produce the same

keystream. The ciphertext and keystream are then combined using XOR operations to recover the plaintext.

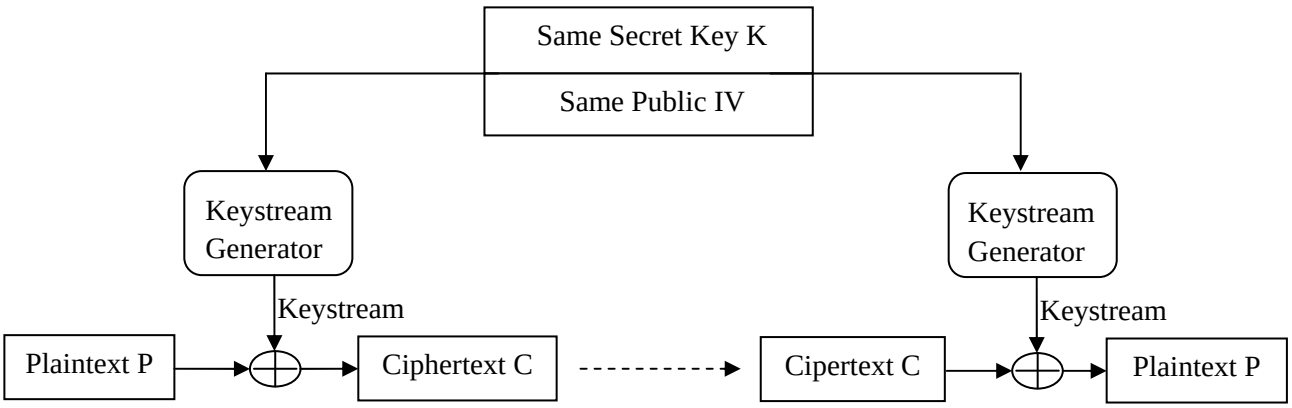

Figure 2. Binary-additive stream cipher.

1.2 Message Authentication Codes

A Message Authentication Code (MAC) is a symmetric cryptographic primitive for providing message integrity protection. MACs take a message of arbitrary length and produce an output of fixed length known as a MAC tag. In addition to the message that is input however, MACs require the use of a secret key for the computation of the MAC tag. The use of the secret key means that even if an attacker has intercepted the message and knows the MAC algorithm that was used, they are unable to make changes to the message and calculate the corresponding MAC tag without knowledge of the key. MACs therefore provide data integrity and some degree of data-origin protection. MACs however, do not provide non-repudiation protection. That is, having verified the MAC tag of a message, we cannot be assured the claimed sender actually sent the message. To provide this non-repudiation, digital signatures are required. These make use of asymmetric cryptographic techniques which are beyond the scope of this paper.

1.3 Authenticated Encryption Stream Ciphers

An Authenticated Encryption (AE) stream cipher combines the mechanisms for providing both confidentiality and integrity protection into a single cryptographic primitive. The primitive has both a confidentiality component and an integrity component. Encryption using an AE cipher occurs in the following phases:

1. Initialisation phase: The secret key and IV are used to initialise the confidentiality and integrity components.
2. Keystream Generation and Message Accumulation phase: The plaintext message is encrypted and ciphertext is generated. Simultaneously, the message (either the plaintext or ciphertext) is used to update the integrity component in a message-dependent way. The updating of the integrity component is referred in this paper the accumulation of the message.
3. MAC Finalisation phase: Once the plaintext has been encrypted, the integrity component undergoes some additional post-encryption operations. At the end of this phase, the MAC tag of the message is produced.

AE decryption occurs in the following phases:

1. Initialisation phase: The secret key and IV are used to initialise the confidentiality and integrity components.
2. Keystream Generation and Message Accumulation phase: The ciphertext message is decrypted and the plaintext is recovered. Simultaneously, the message (either the plaintext or ciphertext) is used to update the integrity component in a message-dependent way.
3. MAC Finalisation phase: Once the plaintext has been decrypted, the integrity component undergoes some additional post-decryption operations. At the end of this phase, the MAC tag of the message is produced.

1.4 Methods for providing Authenticated Encryption

Bellare and Namprempre (Bellare & Namprempre, 2008) investigated the security properties of three methods for providing AE. These are Encrypt-and-MAC (E&M), MAC-then-Encrypt (MTE) and Encrypt-then-MAC (ETM). They refer to these methods collectively as generic compositions. These methods can be described as follows:

1. Encrypt-and-MAC (E&M): The sender first forms a MAC tag of the plaintext, encrypts the plaintext, and then appends a MAC tag of the plaintext to ciphertext. At the receiving end, the receiver checks the message by first decrypting the ciphertext, and then generating his own MAC tag of the decrypted ciphertext and comparing this to the MAC tag that was received along with the message.
2. MAC-then-Encrypt (MTE): The sender first appends a MAC tag of the plaintext to the message and then encrypts the augmented plaintext-MAC message. At the receiving end, the receiver checks the message by first decrypting the message to recover the plaintext and MAC tag, then computes the MAC tag of the received plaintext, and compares it to the MAC that was recovered through decryption.
3. Encrypt-then-MAC (ETM): The sender encrypts the plaintext to get the ciphertext. A MAC tag of the ciphertext is then appended to the message. At the receiving end, the receiver will first check the MAC tag. If the verification check passes, the message is decrypted to recover to get plaintext.

Bellare and Namprempre analyse the security of the three AE methods in terms of the following security properties:

1. Indistinguishability of encryptions under the chosen plaintext attack denoted (IND-CPA).
2. Indistinguishability of encryptions under the chosen ciphertext attack denoted (IND-CCA).
3. Non-malleability under the chosen ciphertext attack denoted (NM-CPA).
4. Integrity of plaintexts denoted (INT-PTXT).
5. Integrity of ciphertexts denoted (INT-CTXT).

In a chosen plaintext attack, the attacker is able to choose the plaintext to be encrypted and get the corresponding ciphertext, while in a chosen ciphertext attack, the attacker can choose ciphertexts to be decrypted and see the plaintext corresponding to that chosen ciphertext. Indistinguishability is the inability of an attacker to learn any information about the plaintext from a challenge ciphertext. Non-malleability is the inability of the attacker, given a challenge ciphertext *C*, to produce a different ciphertext *C'* such that there is no meaningful relationship between the decrypted messages *P* and *P'* corresponding to *C* and *C'* respectively. In their analysis, Bellare and Namprempre prove that if the integrity component of an AE algorithm is

strongly unforgeable, the ETM scheme provides all five of the security properties listed above.

2. The e-STREAM PROJECT

The eSTREAM project (European Network of Excellence for Cryptology, 2008), launched in 2005, was a multi-year project whose goal was to identify new stream ciphers which might be suitable for widespread adoption. Of the 34 stream ciphers that were submitted, seven included an authentication mechanism. In this paper, we analyse three of these ciphers: Sfinks, NLSv2 and Phelix. A summary of the characteristics of these ciphers, with regards to their maximum key sizes, IV sizes, MAC tag size and the maximum amount keystream that can be generated before rekeying needs to be done is given in Table 1.

Table 1. Basic parameters for three selected eSTREAM AE ciphers

Cipher Name	Usage Environment	Max. Key Size in bits	Max. IV Size in bits	MAC tag size in bits	Max. Keystream (in bits) generated from a Key-IV pair
Sfinks	Hardware	80	80	64	2^{40}
Phelix	Software/Hardware	256	128	128	2^{67}
NLSv2	Software	128	128	128	2^{53}

2.1 Description of eSTREAM AE Ciphers

This section of the paper gives a brief description of the three eSTREAM AE ciphers. The description includes the keystream generation, message accumulation and MAC finalization phases. For a more detailed description of the ciphers, the reader is referred to the respective specification papers.

2.1.1 Sfinks

Sfinks (Braeken *et al.*, 2005) is a stream cipher intended for use in hardware environments. The lengths of the secret key and nonce are each 80 bits. Sfinks generates a 64-bit MAC tag for message integrity protection. A diagram showing the relationship between the major components of Sfinks are shown in Figure 3.

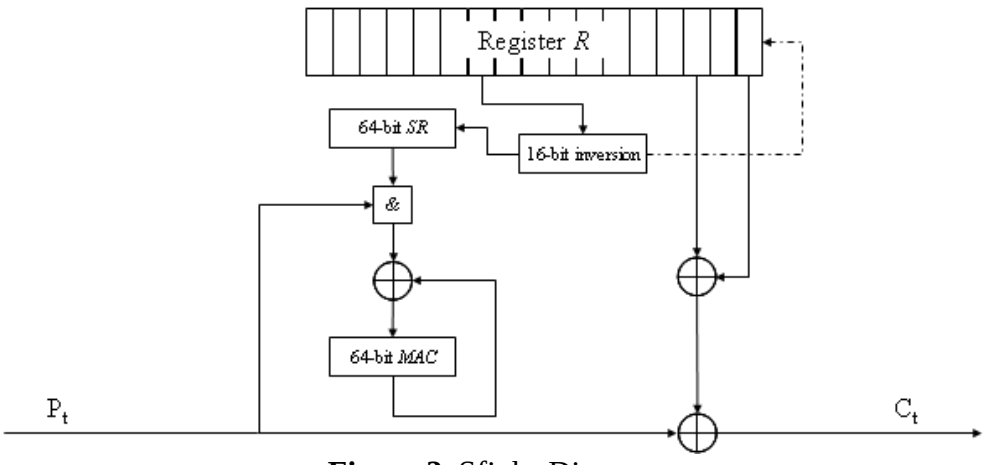

Figure 3. Sfinks Diagram

Sfinks can be divided into two sub-functions, the confidentiality component and the integrity component. Both components make use of the 256-bit regularly clocked LFSR, denoted *R*, and a 16-bit inversion S-Box.

During the keystream generation phase, the contents of selected stages from *R* are fed into the S-Box. One bit from the output of the S-Box is combined with the contents of another stage of *R* to form the keystream output of Sfinks. The plaintext message is then XORed with the keystream to produce ciphertext.

The integrity component of Sfinks makes use of two 64-bit registers. The first register, denoted *SR*, is a pure shift register. During message accumulation, at each time interval the contents of all the stages of *SR* are moved along, with the contents of the last stage being discarded. A single bit output from the S-Box is used to update *SR*. The second 64-bit register is denoted *MAC*. *MAC*'s update function is plaintext dependent. If the current plaintext bit being encrypted is 0, the contents of *MAC* remain unchanged while *SR* is updated. If the current plaintext bit is 1, *SR* is updated and the updated contents of *SR* are XORed with the contents of *MAC*.

Once all the plaintext has been encrypted, Sfinks enters its MAC finalization phase. In this phase, the 64-bit contents of *MAC* are XORed with 64-bits of fresh keystream to generate the 64-bit MAC tag.

2.1.2 Phelix

Phelix (Whiting *et al.*, 2005) is a synchronous cipher intended for use in both software and hardware environments. It uses a secret key, which can be up to 256-bits in size. The IV can be up to 128-bits in size. Phelix generates a 128-bit MAC tag for message integrity protection. Phelix uses one component to provide both confidentiality and integrity protection.

The internal state of Phelix consists of nine 32-bit stages called state words, giving Phelix a total internal state size of 288 bits. These state words are divided into two groups: active state words and old state words. The five active state words are used in the state update function while the four old state words are used in the keystream generation.

Operations in the state update function of Phelix can be described as being performed as a series of rounds. A round consists of adding (or XORing) one active state word into another active state word, and rotating the former word. An example of a round is shown in Figure 4, where the active state words are represented by a, b, c, d, e. The outputs from a round are a', b', c', d'and e'. Twenty of these rounds make up one Phelix block function. Phelix's block function is actually two applications of the half-block function *H*. The details of *H* are available in the Phelix paper. Phelix uses these block functions to do its encryption, decryption and MAC operations.

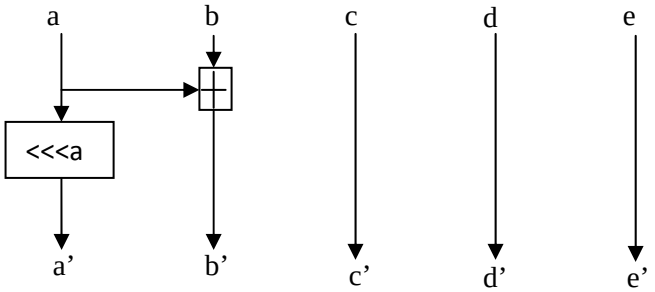

Figure 4. One Phelix Round

During keystream generation and message accumulation, *H* takes in an input key in the first half-block to update the active state words. In the second half-block, the second input key

and a plaintext word are used to compute the keystream word, which is then XORed with the plaintext to form the ciphertext.

After all the plaintext has been encrypted, Phelix enters the MAC finalization phase. In this phase, a state word is modified by XORing with a constant value. After the state word is modified, a post-mixing step is applied. In the post-mixing step, the value *l(p)* mod 4, which is the length of the plaintext in bytes, is treated as plaintext and encrypted eight times. The keystream generated from this process is discarded. After this initial post-mixing step, additional post-mixing is applied. The same value *l(p)* mod 4 is encrypted four more times. The four 32-bit keystream words generated from these four encryption operations form the 128-bit MAC tag for the message.

2.1.3 NLSv2

NLSv2 (Non-Linear Sober Version 2) (Hawkes *et al.*, 2006) is an updated version of NLS. NLS and NLSv2 are synchronous stream ciphers. NLSv2 uses a secret key and nonce, each 128 bits long. NLSv2 generates a 128-bit MAC tag. Relationships between the major components of NLSv2 are shown in Figure 5.

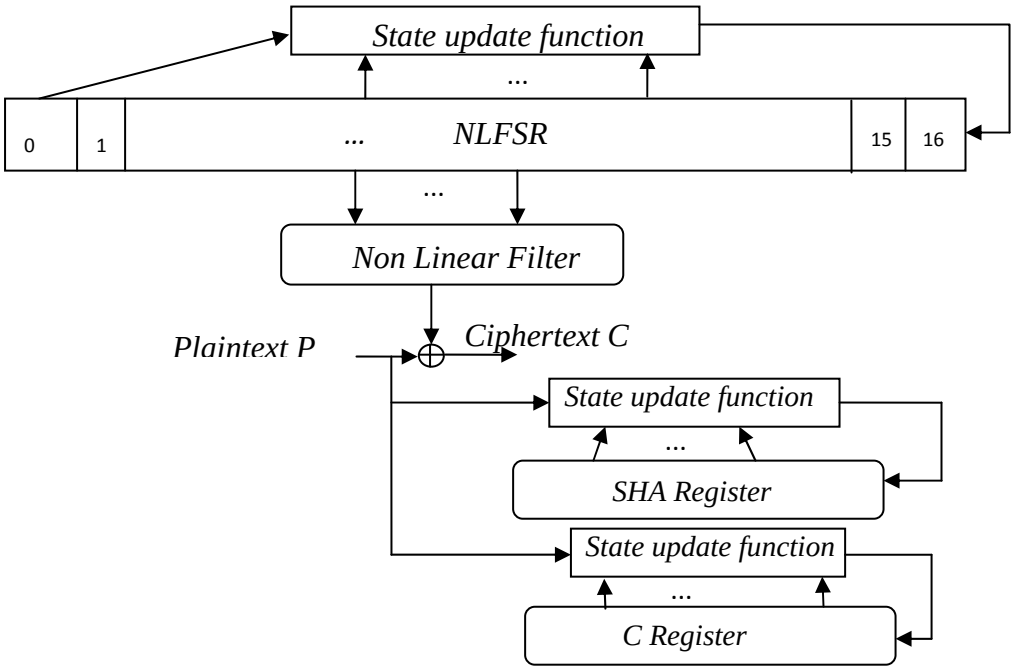

Figure 5. NLSv2 Diagram

NLSv2 can be divided into two sub-functions, the confidentiality component and the integrity component. Both the components make use of a non-linear feedback shift register (NLFSR), denoted *R*, a nonlinear filter and an S-Box denoted *S*. *R* is a shift register consisting of 17 stages, each of size 32 bits, giving *R* a total internal state size of 544 bits. *R* is updated reguarly with a nonlinear state update function. The S-Box *S* is used in NLSv2 to provide the non-linearity.

In the keystream generation phase, the contents of selected stages of *R* are combined with a key-dependent constant, KONST, using a nonlinear filter function. The result of these operations is a 32-bit keystream word. These 32 bits of keystream is XORed with 32 bits of plaintext to form 32 bits of ciphertext.

The integrity component of NLSv2 makes use of a MAC function called Mundja (Hawkes *et al.*, 2007). Mundja consists of two 256-bit registers. One of the registers *SHA* accumulates values using a strengthened version of the SHA-256 hash function, while the other is a 256-bit Cyclic Redundancy Check (CRC) register *C*. During the message accumulation phase, the plaintext message is accumulated in both *SHA* and *C*.

After all the plaintext has been encrypted, NLSv2 enters the MAC finalization phase. This phase consists of three steps. In the first step, a constant word is added into *SHA* and any resultant keystream generated is discarded. In the second step, *C*'s state update function is run eight times with an all-zero plaintext and the content of one of its states is added into *SHA*. *SHA*'s internal state is then updated. In the final step, *C*'s state update function is continually cycled with zero bit words. This time however, after the contents of *C* are added into *SHA*, and *SHA*'s internal state is updated, the contents of one of SHA's stages are used as a 32-bit output for a portion of the MAC tag. This process is repeated another three times to produce the 128-bit MAC tag.

3 ANALYSIS TECHNIQUES

This section briefly describes the major techniques attackers use to attack stream ciphers and MACs.

3.1 Stream Cipher Cryptanalysis

For stream cipher cryptanalysis, it is usually assumed that the attacker has access to the ciphertext, and possibly a certain amount of plaintext corresponding to the known ciphertext, or an amount of keystream. A stream cipher designer needs to take into account all three as possession of any of them makes the cipher vulnerable to certain attacks.

Where the attacker has access to the ciphertext, two attack types should be considered. These are the ciphertext-only (or known-ciphertext) attack and the chosen ciphertext attack. If it is possible to attack a stream cipher using the ciphertext-only attack, without making use of plaintext, this means that the stream cipher's keystream generator is very weak. A chosen ciphertext attack allows the attacker to select which ciphertexts to be decrypted and attempts to recover the key from this ciphertext. This model is not relevant to synchronous stream ciphers since the keystream generated is not ciphertext-dependent.

Where the attacker has possession of some plaintext and the corresponding ciphertext, two attack types can be considered: known-plaintext attack and chosen plaintext attack. For known plaintext attacks, an attacker gains access to an amount of keystream by XORing the plaintext and its corresponding ciphertext. For chosen-plaintext attacks, the attacker chooses the plaintext they want encrypted and obtains the corresponding ciphertext. For most binary-additive stream ciphers, the outcome of the known plaintext and chosen plaintext attacks is the same: a segment of the keystream is also revealed. However, if the keystream is plaintext-dependent, a chosen-plaintext attack may be an effective approach. This may be the case for Phelix.

Distinguishing attacks are a form of attack that allows an attacker to determine if a keystream has been generated from a particular stream cipher. Since the keystream output from a stream cipher is supposed to look random, a successful distinguishing attack on a stream cipher implies that there exist some relationships between the keystream bits that can be used to identify the cipher. Although distinguishing attacks will not directly result in the recovery of either the key or internal state, they reveal a potential weakness. There is also a possibility that the observation may be extended into a key or state recovery attack. In order to curb distinguishing attacks, stream cipher designers often impose limits on how much keystream can be generated using a single key-IV pair before rekeying is required.

An attacker's main goal is to recover the plaintext message without prior knowledge of the key. (In general, the attacker seeks to recover either the master key for the stream cipher or a session key. In some attacks, it may be possible to recover the plaintext without knowledge of a key, for example, if the same keystream is used more than once (Dawson & Nielsen, 1996)).

A master key attack can be performed regardless of whether or not the initialisation function of the stream cipher is one-way. A master key attack means that data from multiple encryption sessions will be rendered insecure. Once the key is recovered, the attacker will initialise the cipher with the recovered key and IV, generate some keystream and recover the plaintext from the ciphertext. The most naive approach to key recovery is to try all possible keys. Stream ciphers are considered weak if an attacker is able to recover the key using less than $O(2^k)$, where k is the size of the key in bits.

Session key attacks are also known as state recovery attacks. In a modern-day stream cipher, the secret master key and the IV are combined together during the initialisation phase using an initialisation function. Recovery of the internal state means that data from that encryption session would be rendered insecure since the attacker who knows the internal state of the cipher would be able to generate keystream to recover the plaintext message. Encrypted data from other sessions, assuming that a different IV was used, might still be secure.

3.2 Message Authentication Code Analysis

For MAC analysis, the attacker is assumed to have complete knowledge of the MAC algorithm and the format of the message. The attacker can either mount a forgery attack or a key recovery attack. Resistance to MAC attacks is measured using smallest number of operations with regards to two variables; the key size of the MAC, denoted *k*, and the size of the MAC tag value, denoted *d*. There are three attack models against which MACs have to provide resistance. These are the known-text attack, the chosen-text attack and the adaptive chosen-text attack.

In MAC forgery attacks, the attacker is able to find a new message whose MAC value has the same value as that of the original message. To avoid MAC forgery the complexity of the attack should be $O(2^{min(k,d)})$ (Hawkes *et al.*, 2006). There are two types of forgery attacks, selective forgeries and existential forgeries:

1. Selective forgery: An attacker is able to produce a new message-MAC pair of their own choosing. In terms of the above-mentioned attack models, the known-text attack implies that the attacker attempts to forge a MAC based on an arbitrary length message. The chosen-text attack allows the attacker to analyse the resultant text-MAC pair (Menzes *et al.*, 1997). In the adaptive chosen-text attack, the attacker is allowed successive queries to the MAC algorithm based on the results of previous queries (Menzes et al., 1997).
2. Existential forgery: An attacker is able to produce a new message-MAC pair, but has no control over the contents of that message.

A key-recovery attack allows the attacker to recover the secret key and enables an attacker to mount selective forgery attacks. A key-recovery attack constitutes a total break on the integrity component of an AE stream cipher. A secure MAC should allow no less than $O(2^k)$ operations to recover the key.

3.3 Comments about AE Cryptanalysis

Key recovery attacks on AE systems are of particular importance. In generic composition constructions, a key recovery attack on either the confidentiality or integrity component does not necessarily affect the security of the other component as separate keys are used for confidentiality and integrity protection. However, for AE systems, the same key is used for

both confidentiality and integrity protection. Thus, a key recovery attack on either component compromises the security provided by the other component.

The interaction between the confidentiality and integrity components of the stream ciphers also needs to be carefully studied. If the AE stream ciphers use some of the same structure for both confidentiality and integrity components, potential information leakage about the internal state of the confidentiality component might be obtained from the MAC tag, or information about the internal state of the MAC seen in the internal state of the integrity components and vice-versa.

4 Analysis of eSTREAM AE Ciphers

In this section, we present our analysis of the three eSTREAM AE ciphers. In Section 4.1, we classify the three using the framework given by Bellare and Namprempre reviewed in Section 1.4. In Section 4.2, we compare the efficiency of the three AE ciphers. In Section 4.3, an analysis of security of the three algorithms is presented.

4.1 Classification based on Bellare and Namprempre generic composition scheme.

The classification under Bellare and Namprempre's generic composition scheme is shown in Table 2. It should be noted that although the methods used in Bellare and Namprempre's analysis were based on a two-pass method, this can be mapped to a one-pass method based on how the cipher encrypts the plaintext message and calculates a MAC.

Table 2. Classification of eSTREAM authenticated encryption ciphers using Bellare and Namprempre's scheme.

Cipher	E&M	MTE	ETM
Sfinks		✓	
Phelix	✓		
NLSv2	✓		

Discussion of classification results

NLSv2 would be classified as an E&M scheme. In NLSv2, the plaintext is accumulated in two registers, the SHA register and *C*. Although a content of *R* is used in the state update function for the SHA registers, NLSv2 are ciphertext independent. Furthermore, the MAC for NLSv2 is not a ciphertext output. Instead, the MAC is generated using the contents of one the SHA registers.

Phelix would be classified as using an E&M scheme. Phelix uses one register for performing encryption, decryption and MAC generation operations. After all the plaintext has been encrypted, Phelix runs additional operations on the registers and generates the MAC using the keystream word output of these additional operations.

Sfinks would be classified as a MTE scheme. During the encryption or decryption process, the plaintext is accumulated in *SR* and *MAC*. During the MAC finalization process, the contents of the plaintext-dependent *MAC* are encrypted with 64-bits of fresh keystream. This is similar to the MTE scheme, whereby the MAC of the message is appended to the encrypted message and encrypted.

We can also make two observations about the MTE scheme requires two passes over the same message. The first pass calculates the MAC of the plaintext message and the second pass encrypts the appended plaintext and MAC. Although Sfinks uses the MTE method for providing authenticated encryption, it might not be the most efficient way of designing an AE stream cipher.

The second observation is that while Bellare and Namprempre prove that the ETM scheme provides the most security, none of the three ciphers use it. This could be because the E&M scheme potentially provides the most efficient way of encrypting the plaintext message and the MAC. With the E&M scheme, the cipher would be able to perform operations for obtaining the ciphertext and MAC of the plaintext simultaneously, while for the ETM scheme, there is the requirement that the ciphertext would need to be available before any MAC calculations can be performed on it.

4.2 Efficiency Analysis

One of the key reasons for using stream ciphers is that they offer superior performance at encrypting large amounts of data when compared to the other class of symmetric confidentiality algorithms namely, block ciphers.

In analysing the efficiency of a stream cipher, one important measure is cycles-per-byte (CPB). This measures how much CPU cycles it takes to encrypt a single byte. The lower the CPB, the more efficient the stream cipher is. Another CPB measurement we can measure is how much CPU cycles the ciphers require for initialising and rekeying. This is an important measure as the longer it takes, the longer the sender needs to wait before generation of fresh keystream begins. This in turn, leads to a delay in sending encrypted data, a delay which might not be acceptable in real-time communications. CPB cycles for initialisation and rekeying are not listed in this paper.

The other important measure is throughput. This is typically measured in Gigabits per second (Gbps). A larger Gbps value means that the cipher can encrypt a larger amount of data per second.

One of the key criteria of the eSTREAM project was that the ciphers submitted had to be faster than the AES block cipher in counter-mode (AES-CTR) (Dworkin, 2001). Note that we were unable to find any publicly available literature which compares the efficiency of all three stream ciphers when implemented on a single machine. Neither was there any comparison which compared all seven AE stream ciphers performing both confidentiality and integrity operations. One possible reason for this is that the some of the efficiency analysis on the stream ciphers were done during the later stages of the project, when insecure ciphers were dropped from the competition. In this case, it wouldn't make sense to do an efficiency analysis on ciphers which have been deemed insecure by the cryptographic community.

Table 3 lists the timings of NLSv2 and Phelix. For the NLSv2 cipher, we have included two versions. NLSv2 (submitted version) is the version which was submitted by the authors. NLS with poly1305 is a modified design by Bernstein. In his design, Bernstein used the NLSv2 confidentiality component together with his own MAC algorithm, poly1305. The idea of including NLS with poly1305 is to show how efficient the original integrity component is compared with an integrity component added by others. Column 1 lists timings by Bernstein's timing comparisons of NLS and Phelix. Column 2 lists the timings for AES-OCB (Rogaway, 2007). AES-OCB is one of the most efficient AE block cipher designs today. For a fair comparison, AE stream cipher timings should be compared with AE block cipher timings. AES-CTR mode only provides confidentiality protection. The extra computational operations needed to provide integrity protection should be taken into account when analysing the efficiency of AE stream ciphers.

Table 3. Efficiency Comparison of ciphers. This table's timings are obtained combination from Bernstein (Bernstein, 2007) and Lipmma (Lipmma, 2008)**.**

Cipher	**CPU Type/Speed**	**1**	**2**
		Cycles per byte (encryption). (Bernstein, 2007)	**Cycles per byte (encryption) for AES-OCB (Lipmma, 2008).**
Phelix	Pentium 4 3400 MHz	10.91	16.60
NLSv2 (submitted version)	Pentium 4 3400 MHz	18.44	
NLS with poly1305 (Bernstein's version)	Pentium 4 3400 MHz	13.24	

Discussion of efficiency results

There are various variables which could affect the timing results. Firstly, the hardware the ciphers are run on will affect the results. For example, a hardware-based cipher run on software might not give the best performance results. The same code run on two different processors could give different efficient results as well.

Secondly, we need to consider if the code run on the machines are optimised versions of the algorithm. If the code for the algorithm is optimised for the CPU the code is being run on, it is possible to achieve very good speeds. For example, the fastest known AES-CTR code was implemented on an Intel Core 2 Quad 64-bit processor and achieved a CPB of 7.6 (Käsper & Schwabe, 2009), a three-fold increase over the official benchmarks used in eSTREAM. The programming language is also an important factor. Implementations written in assembly are usually faster than implementations written in other programming languages.

In his efficiency analysis, Bernstein compared various stream ciphers combined with his own MAC, poly1305. He implemented two versions of NLSv2; one of them used the default authentication method given in the specification while the other combined the confidentiality component with poly1305. Bernstein did not implement Phelix with poly1305 as Phelix already uses the same register for providing confidentiality and integrity protection. As Column 1 shows, the original NLSv2 CPB value is the least efficient of the ciphers Bernstein tested, even when compared to his own design. It is because of NLSv2's slow integrity performance that the integrity component was dropped in the final phase of the project.

Phelix exhibited the best results of all three ciphers. It is even faster than the timings AES-OCB obtained in Colum 3. This makes Phelix an ideal candidate for AE stream ciphers, since it meets the original goal of having an AE stream cipher whose performance is faster than an AE block cipher.

The efficiency analysis of a hardware-based stream cipher is different from a software-based stream cipher. Hardware-based stream ciphers are usually implemented in small, low-powered devices with a small amount of memory. This means that components which can be implemented efficiently in software, like S-Boxes, might not be as efficient in hardware. One method of measuring the hardware cost of a stream cipher is through NAND gates (Braeken *et al.*, 2005). The smaller the number of NAND gates, the smaller the hardware the cipher is implemented on could be. If the number of NAND gates is too large, the chances of the stream cipher being implemented on hardware is lower (Good *et al.*, 2006). Another concern of hardware stream ciphers is power usage. It is essential that if the card is to be implemented

on, for example, a contactless smart card, the peak power consumption should be as low as possible. If the device is battery-powered, the cipher would need to be as energy-efficient as possible (Good *et al.*, 2006).

4.3 Cryptanalysis on selected e-STREAM ciphers

This section describes the current attacks which have been applied to three eSTREAM ciphers. It presents the current publicly known cryptanalysis on the stream ciphers and presents some ideas on new attacks.

4.3.1 Sfinks Analysis

Current Cryptanalysis of Sfinks

The only cryptanalysis on Sfinks in the public literature is a paper by Courtois (Courtois, 2005). Courtois used a cryptanalyical technique based on the known plaintext attack called a fast algebraic attack. Courtois used the technique to attack the confidentiality component of Sfinks, recovering the internal state of Sfinks. Algebraic attacks on LFSR-based stream ciphers work because an attacker is able to construct a series of linear combinations which relate to keystream bits to all the bits of internal state. Therefore, if an attacker has enough keystream bit outputs, he can construct a series of multivariate equations and may be possible to solve them. Courtois showed that it may be possible to recover the internal state of Sfinks using $2^{38.5}$ keystream bits. While the amount of keystream falls within the limits set by Sfinks's designers (see Table 1), it could only be a theoretical break since total operations required in the other phases of the attack surpass the security bounds of Sfinks.

New Analysis of Sfinks

One possible attack which could be applied to Sfinks is the MAC forgery attack. Since the common component of both the confidentiality and integrity component is the Boolean function that is what we focus on in this paper. Preliminary investigations of the output of a scaled-down, 64-bit Boolean function behind a 17-bit LFSR have revealed that the n-bit tuple output is not uniform. In this case, the n-bit tuple output refers to an n consecutive bit output from a Boolean function. Furthermore, if the n-bit tuple output is large enough, there are some outputs which will not occur at all. Both these situations mean that there may be some internal state of *SR* and some 64 bit keystream which could occur more frequently than others. Recalling the description of Sfinks from Section 2.1.1, the state update function of *MAC* is plaintext-dependent. This potentially leaves Sfinks vulnerable to a chosen-plaintext attack. In this attack, the attacker could control when the contents of *SR* gets XORed to *MAC*. Thus, the final contents of *MAC* before MAC finalization occurs could be biased. If the 64-bit of fresh keystream generated during MAC finalization is also biased, the final MAC value would be biased.

4.3.2 Analysis of Phelix

Current Cryptanalysis of Phelix

Wu and Preneel (Wu & Preneel, 2006) used cryptanalyical technique based on the chosen-plaintext attack, called the differential-linear attack, to mount a key recovery attack on Phelix's confidentiality component. In this technique, carefully chosen plaintexts are encrypted and the output as the plaintexts are passed through a selected round is observed. However, for this attack to succeed, repeated IVs had to be used. For the attack to succeed, a total of 2^{24} chosen IVs and 2^{42} chosen plaintext bits had to be used. The total number of operations used in the attack was $2^{41.5}$ operations. Although the amount of chosen plaintext

and operations needed in the attack is below the bounds imposed by the designers, this attack required the use of repeated IVs, which violates the basic requirements needed for proper use of a stream cipher. This use calls for a nonce to be only used once, not twice per key, as was the case of Wu & Preneel's attack.

New Analysis of Phelix

Recall the description of Phelix in Section 2.1.2. The plaintext message is used in the generation of fresh keystream. This leaves it open to chosen plaintext attacks, specifically attacks with regards to IND-CPA, which was described in Section 1. If an attacker wants to, for example, forge a MAC, they would need to be able to also choose carefully selected plaintext. However, unlike the differential-linear attack which Wu and Preneel employed, where they used the differences observed after the plaintext is passed through the particular round, the attacker who hopes to mount a successful MAC forgery would need to ensure that differences in the plaintext would cancel out after being passed through all 20 encryption rounds and the 12 additional rounds it goes through before the final MAC is produced.

4.3.3 Analysis of NLSv2

Current Cryptanalysis of NLSv2

Cho and Pieprzyk (Cho & Pieprzyk, 2006) used a technique based on the known plaintext attack called a distinguishing attack on NLSv2's confidentiality component. This attack investigates high correlation between two neighbouring bits of the cipher. In their analysis, they noted that bit number 29 and 30 of the output of S-Box had a high correlation. So the attack exploits this correlation to make a linear approximation on NFSR and on NLF. As a result of that, a distinguisher of NLSv2 has a bias of 2^{-37} and so the attacker required only 2^{79} bits of keystream to distinguish the keystream of NLSv2 from random. Cho and Pieprzyk claim that this falls within the amount of keystream generated (2^{80} bits) before rekeying needs to be done. However, from Table 1, a new session key should be generated after at most 2^{53} bits of plaintext are encrypted. This means that the distinguishing attack of Cho and Pieprzyk has no practical significance.

New Analysis of NLSv2

This analysis focuses on the integrity component of NLSv2. From the description in 2.1.3, we can make an observation about the integrity component of NLSv2. The CRC register *C* is updated in a linear manner and it takes as input, the plaintext word that is going to be encrypted. This may leave it vulnerable to a chosen plaintext attack. In this attack, the attacker mount a chosen-plaintext attack and select carefully selected plaintext that would yield the same final state before the final MAC processing phase starts. Assuming the same key and IV are used, any further updates from this point onwards is insignificant, as the final MAC processing consists of a standard series of steps. The values generated from this are the same if the same key and IV are used. However, even if the attacker is able to generate collisions for the CRC registers, they will not be able to forge a MAC. This is because the integrity component of NLSv2 consists of another register, the SHA registers. Therefore, even with collisions in the CRC registers, with plaintext feeding into the SHA registers as well, the MAC values generated in the end will be different.

5 CONCLUSION

In this paper, we have classified the three AE ciphers according to Bellare and Namprempre based on the three common methods of providing authenticated encryption. The majority of the ciphers used the E&M scheme, due to the potential parallelisability of the confidentiality

and integrity components. However, more detailed analysis of other AE ciphers need to be done in order to establish if the E&M scheme of providing AE is the "best" choice.

Secondly, a comparison of the efficiency of two ciphers was done. It is shown that NLSv2's original authenticator is slower than AES-OCB. This was the main reason why NLSv2 was removed from the eSTREAM project. One of the key reasons for choosing AE stream ciphers over AE block ciphers is their high efficiency and speed. If the stream cipher is slower than an AE block cipher, it is unlikely to be adopted for use. Thus, an important design criterion for AE stream ciphers is that they should be faster than their block cipher counterparts.

A review of the current attacks on the three ciphers shows that none of these attacks fall within the security bounds set by the designers and are of no practical significance. However, these attacks reveal weaknesses in the AE stream cipher design and warrant further investigation to see if more efficient attacks can be found.

Finally, the integrity components of all three ciphers were analysed. The biased n-bit tuple output of a nonlinear Boolean function needs further investigation to establish the factors which could contribute to the bias and how this could be applied to a MAC forgery attack on Sfinks. We described a possible MAC forgery attack on Phelix assuming the attacker is able to find a pair of plaintext which can produce the same output after going through the keystream generation and MAC finalisation phases. Analysis of NLSv2 integrity component reveals it could be possible to mount a chosen-plaintext attack on the *C* registers if the attacker is able to choose a pair of plaintext that will give similar *C* final states before MAC finalisation occurs. The possibility of extending this to a MAC forgery attack needs further investigation.

REFERENCES

Bellare, M., & Namprempre, C. (2008). Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *Journal of Cryptology*, ***21*****(4)**, 469-491.

Bernstein, D. J. (2007). Cycle counts for authenticated encryption. Retrieved 22 July 2009, from http://www.ecrypt.eu.org/stream/papersdir/2007/015.pdf

Braeken, A., Lano, J., Mentens, N., Preneel, B., & Verbauwhede, I. (2005). SFINKS : A Synchronous Stream Cipher for Restricted Hardware Environments. Retrieved 22 July 2009, from http://www.ecrypt.eu.org/stream/ciphers/sfinks/sfinks.ps

Cho, J. Y., & Pieprzyk, J. (2006). Crossword Puzzle Attack on NLSv2. In J. A. Garay, A. K. Lenstra, M. Mambo & R. Peralta (Eds.), *10th International Conference on Information Security, ISC2007* (Vol. 4779 of Lecture Notes in Computer Science, pp. 230-248): Springer.

Courtois, N. (2005). Cryptanalysis of Sfinks. In D. Won & S. Kim (Eds.), *Information Security and Cryptology - ICISC 2005* (Vol. 3935 of Lecture Notes in Computer Science, pp. 261-269): Springer.

Dawson, E., & Nielsen, L. (1996). Automated cryptanalysis of XOR plaintext strings. *Cryptologia*, ***20*****(2)**, 165-181.

Dworkin, M. (2001). Recommendation for Block Cipher Modes of Operation: Methods and Techniques. Retrieved 26 July 2009, from http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

European Network of Excellence for Cryptology (2008). *The eSTREAM Project.* Retrieved 20 July 2009, from http://www.ecrypt.eu.org/stream/index.html

Good, T., Chelton, W., & Benaissa, M. (2006). Review of stream cipher candidates from a low resource hardware perspective. Retrieved 26 July 2009, from http://www.ecrypt.eu.org/stream/papersdir/2006/016.pdf

Hawkes, P., Paddon, M., & Rose, G. G. (2007). The Mundja Streaming MAC. Retrieved 26 July 2009, from http://eprint.iacr.org/2004/271.pdf

Hawkes, P., Paddon, M., Rose, G. G., & de Vries, M. W. (2006). Primitive Specification for NLSv2. Retrieved 22 July 2009, from http://www.ecrypt.eu.org/stream/p3ciphers/nls/nls_p3.pdf

Käsper, E., & Schwabe, P. (2009). Faster and Timing-Attack Resistant AES-GCM. Retrieved 23 July 2009, from http://homes.esat.kuleuven.be/~ekasper/papers/fast_aes.pdf

Lipmma, H. (2008). *Fast Implementations of AES and IDEA for Pentium 3 and 4.* Retrieved July 23 2009, from http://research.cyber.ee/~lipmaa/implementations/

Menzes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (1997). *Handbook of Applied Cryptography*. Florida: CRC Press.

Rogaway, P. (2007). *OCB - An Authenticated Encryption Scheme.* Retrieved July 23 2009, from http://www.cs.ucdavis.edu/~rogaway/ocb/index.html

Whiting, D., Schneier, B., Lucks, S., & Muller, F. (2005). Phelix - Fast Encryption and Authentication in a Single Cryptographic Primitive. Retrieved 22 July 2009, from http://www.ecrypt.eu.org/stream/ciphers/phelix/phelix.pdf

Wu, H., & Preneel, B. (2007). Differential-Linear Attacks against the Stream Cipher Phelix. In A. Biryukov (Ed.), *Fast Software Encryption (FSE 2007)* (Vol. 4593 of Lecture Notes in Computer Science, pp. 87-100): Springer.