Security Level of Cryptography —

Security evaluation (especially in mode part) for the stream cipher MULTI-S01

December 2001

Contents

1 Summary

1.1 Description of this document

This document is an evaluation of the MULTI-S01 encryption scheme, as proposed by Hitachi. The document is based on references [DEF, SELF, SLIDES, EVAL]. It was prepared at the request of the Information-Technology Promotion Agency, Japan.

This evaluation focuses on the "mode portion" of MULTI-S01. In particular, MULTI-S01 proposes the use of a particular pseudorandom generator, PANAMA, but the current evaluation does not contain any new cryptanalysis of PANAMA. Instead, we model the underlying pseudorandom generator as ideal and proceed from there.

1.2 Background

THE OBJECT OF STUDY. MULTI-S01 is an authenticated-encryption scheme based on a pseudorandom generator (or, more precisely, based on a infinite-output-length pseudorandom function). In order to encrypt a message M having some positive multiple of 64 bits, one first uses the generator to produce a pad of length $|M| + 256$. The algorithm then specifies how to use the pad to map the plaintext M into a ciphertext C having $|M| + 128$ bits. The sender would then transmit C to the receiver. The receiver recomputes the same pad κ and then uses it to recover from C the ciphertext M. During message recovery, the receiver may obtain a signal INVALID instead of recovering a message M in the message space. Typically, the connection would be dropped in this case. The intent is that any adversarial modification to the ciphertext should result in the receiver recovering INVALID.

Authenticated-encryption methods that correctly intertwine privacy and authenticity in a single, integrated mode only recently emerged, with the work of [IAPM]. The customary alternative—providing for privacy and authenticity by separate primitives and the "gluing them together"—has been called the generic composition paradigm, and it is analyzed in [BeNa]. The main motivation for more tightly comingling privacy and authenticity is a potential savings in speed.

The methods of [IAPM], as well as later work like [OCB], build an authenticated-encryption scheme from a block cipher, not a pseudorandom generator. Motivated by the fact that state-of-the-art pseudorandom generators are faster per byte of output than state-of-the-art block ciphers, the authors of MULTI-S01 set forth to devise an authenticated-encryption scheme built on top of a pseudorandom generator. A good idea, even if the authors did not quite pull it off.

1.3 Principle findings

We summarize our main results and findings:

1. Document [DEF] only looks at how to encrypt a single message M. It is not very clear about how to encrypt a sequence of plaintexts. This was a major hole in the exposition.

2. We needed to fill this hole before we could proceed with any meaningful analysis. Thus the exposition of MULTI-S01 we give here goes beyond that in [DEF] to fully specify what

we guess to be the intended way to use MULTI-S01 to encrypt a sequence of messages. Our exposition includes a formal treatment of *stream ciphers* and *stream-cipher cores*.

3. There is no formal treatment in the submission (or, more broadly, in the cryptographic literature) for the security of a stream-cipher authenticated-encryption scheme. We remedy this, developing formal definitions for authenticity appropriate to the stream-cipher setting. This was essential before we could decide if MULTI-S01 was "right."

4. The formal treatment allows one to reach the (initially non-obvious) conclusion that, assuming privacy, an inability to forge after querying a single message implies an inability to forge after asking a sequence of messages. This result justifies limiting attention to what first looks to be a weak kind of attack described in [SELF].

5. We found the main criticisms of [EVAL] to be without signficant merit. That evaluation complained of a lack of "robustness" because one must use distinct keys for distinct messages. This is an intrinsic aspect of stream ciphers (when using an exposition similar to that of used in [DEF]) and not a relevant criticism to MULTI-S01. Secondly, [EVAL] discounted the claims of integrity, perhaps not understanding them, pointing out the obvious and irrelevant facts that one can forge messages if one knows the key, and that there is some nonzero probability of forgery. In fact, the claims of integrity made in [SELF] are valid and are especially meaningful in light of the results we give here.

6. Despite disagreeing with the thrust of [EVAL], we too did not find major merit in MULTI-S01. The reason is that there are alternative methods, using Carter-Wegman MACs, which are faster, architecturally cleaner, increase ciphertext length by fewer bits, and apply to messages of any bit length.

Finally, we mention that, like all stream ciphers, the effective use of MULTI-S01 requires the receiver to maintain state and that the sender and receiver communicate over a reliable channel. These are serious restrictions and limit the utility of a stream cipher that does not bring with it major compensating advantages. We did not find such compensating advantages in MULTI-S01 and, therefore, would not recommend it for standardization.

2 Stream Ciphers and their Security

It is the stated goal of MULTI-S01 to achieve both privacy and authenticity, but there is no formal definition of either goal in any of [DEF, SELF, EVAL]. More broadly, there are no definitions in the literature for the security of a stream-cipher based authenticated-encryption scheme. We remedy this, giving a model and definitions for the desired goals.

Notions of security for authenticated encryption only recently emerged, and are best described in [BeNa]. However, the stream-cipher setting is different from the setting considered in [BeNa] and related works, necessitating new definitions.

2.1 Syntax of a stream cipher

We begin by describing "syntax" for stream ciphers, divorced from any notion of security. Formally, a *stream-cipher* is a triple of algorithms $\mathcal{SC} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ as follows:

- The key-generator Key is a set endowed with a distribution.
- Encryption algorithm $\mathsf{Enc} : \mathsf{Key} \times \mathsf{Par} \times \mathsf{Msg} \to \{0,1\}^*$ is a stateful, deterministic algorithm. It takes a key $K \in \mathsf{Key}$, a parameter $Q \in \mathsf{Par}$, and a message $M \in \mathsf{Msg}$. It returns a ciphertext $C \in \{0,1\}^*$ of length $|M| + r$. Here $r > 0$ is a number (how much longer a ciphertext is than a plaintext) and Msg is a linear-time testable set (the message space).
- Decryption algorithm $\mathsf{Dec} : \mathsf{Key} \times \mathsf{Par} \times \{0,1\}^* \to \{0,1\}^*$ is a stateful, deterministic algorithm. It takes a key $K \in \mathsf{Key}$, a parameter $Q \in \mathsf{Par}$, and a ciphertext $C \in \{0,1\}^*$. It returns either a plaintext $M \in \mathsf{Msg}$ of length $|C| - r$ or else the distinguished symbol INVALID.

One requires that if $K \in \mathsf{Key}$ and $Q \in \mathsf{Par}$ and a sequence of messages $M^1, \cdots, M^m \in \mathsf{Msg}$ gets encrypted, in order, using Enc_K^Q, resulting in ciphertexts $C^1, \cdots, C^m$, then decrypting these ciphertexts, in order, using Dec_K^Q, results in plaintexts $M^1, \cdots, M^m$.

We say that a ciphertext C is *invalid* (with respect to a given key K and parameter Q) if $\mathcal{D}_K^Q(C) = \text{INVALID}$; otherwise, C is *valid*.

DISCUSSION. The parameter Q is non-standard for stream ciphers, but it is used in MULTI-S01 and must be directly reflected in the definitions. Think of each $Q \in \mathsf{Par}$ as determining a separate stream with which one can encrypt communications. All of these streams are logically separated. Sender and receiver must "know" Q, but it need not be secret. As a typical example, one might set Q to some first value, Q^1, and then, if some message should get dropped, the receiver could request a "re-synch" and the sender would respond by choosing a new Q-value Q^2 and carrying on, transmitting the new Q-value with the next message.

Making ciphertexts longer than plaintexts by a fixed constants is certainly not necessary, but it shall be convenient later on.

We emphasize that both Enc and Dec are stateful. This is a fundamental characteristic of stream ciphers.

2.2 Privacy of a stream cipher

We give a very strong definition of privacy: indistinguishability from random bits (under an adaptive chosen-message attack). The notion is used, for example, in [OCB].

Fix a stream cipher $\mathcal{SC} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ with parameters r, R. Consider an adversary A with one of two types of oracles:

- A *real* encryption oracle $\mathbf{Enc}_{\mathcal{SC}}$ begins by picking a random value $K \in \mathsf{Key}$. Then, on input of (M, Q), it returns $\mathsf{Enc}_K^Q(M)$.
- A *random* oracle $\mathbf{Ran}_{\mathcal{SC}}$ has the following behavior: on input (M, Q), it returns a string of $|M| + r$ random bits.

An adversary A is *valid* if every query (M, Q) it asks falls in $\mathsf{Msg} \times \mathsf{Par}$. We only consider valid adversaries. (Alternatively, have all oracles respond to invalid queries by returning the emptystring.) We let

$$\mathsf{Adv}_{\mathcal{SC}}^{\text{PRIV}}(A) = \Pr[A^{\mathbf{Enc}_{\mathcal{SC}}} = 1] - \Pr[A^{\mathbf{Ran}_{\mathcal{SC}}} = 1]$$

be the adversary's "advantage" in distinguishing which of the two types of oracles that is has. This is our measure of privacy.

Informally, a scheme is private if the encryptions of plaintexts look like the expected number of random bits. The notion may be stronger than "necessary," but it is achievable and has desirable characteristics.

A stream cipher $\mathcal{SC}$ is *perfectly private* if, for any adversary A, $\mathsf{Adv}^{\mathrm{PRIV}}_{\mathcal{SC}}(A) = 0$.

2.3 Authenticity of a stream cipher

DEFINITION. Fix a stream-cipher $\mathcal{SC} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$. Suppose we run an adversary with two oracles: an encryption oracle **Enc** as described already, and a (stateful) *decryption oracle* **Dec**. The decryption oracle has the following behavior. It is initialized with the same string $K \in \mathsf{Key}$ that initializes the encryption oracle. Then, on input (C, Q), it returns $M = \mathsf{Dec}^Q_K(C)$. Note that M may be INVALID.

Given a stream cipher $\mathcal{SC} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ and an adversary A, run A with encryption oracle **Enc** and decryption oracle **Dec**. If A ever asks a decryption query that results in INVALID being returned, then A fails: it does *not* forge. Otherwise—all responses to **Dec** are valid—focus on the **Enc** and **Dec** queries that use some one particular Q-value. Then the adversary asks **Enc** some sequence of messages $M^1, M^2, \cdots$, getting responses $C^1, C^2, \cdots$. Interleaved with these queries, the adversary asks it decryption oracle **Dec** some sequence of ciphertexts $\tilde{C}^1, \tilde{C}^2, \cdots$, getting back plaintexts $\tilde{M}^1, \tilde{M}^2, \cdots$. We say that the adversary *forges* (during this run) if some $\tilde{C}^s$ is valid (that is, $\tilde{M}^s \in \mathsf{Msg}$) but $\tilde{C}^1, \tilde{C}^2, \cdots, \tilde{C}^s$ is *not* a prefix of $C^1, C^2, \cdots$. Let $\mathsf{Adv}^{\mathrm{AUTH}}_{\mathcal{SC}}(A)$ denote the probability that adversary A forges. Let $\mathsf{Adv}^{\mathrm{AUTH}}_{\mathcal{SC}}(m)$ denote the maximal value of $\mathsf{Adv}^{\mathrm{AUTH}}_{\mathcal{SC}}(A)$ over all adversaries A that make **Enc** queries (M, Q) where $|M| \le m$ and **Dec** queries (C, Q) where $|C| \le m - r$.

DISCUSSION. The authenticity notion can be understood as follows. Imagine a sender and receiver speaking over a channel. An adversary attacking a stream cipher and able to mount a chosen-plaintext attack can ask the sender to encrypt $M^1, \cdots, M^q$, giving rise to ciphertexts $C^1, \cdots, C^q$ that the adversary sees. The adversary knows that if it sends to the receiver $C^1, C^2, \ldots, C^q$ this will give rise to the recovery of valid messages (namely, $M^1, \cdots, M^q$). Any prefix of $M^1, \ldots, M^q$ will also give rise to a sequence of valid messages. So we say that the adversary A wins if it can get anything *else* to be deemed valid.

For example, if the adversary gets a sequence of ciphertexts C^1, C^2, C^3, C^4 for plaintexts M^1, M^2, M^3, M^4 and, using this data, the adversary concocts $\tilde{C}^3$ where $\tilde{C}^3 \neq C^3$ and yet the sequence of ciphertexts $C^1, C^2, \tilde{C}^3$ are all valid, then the adversary has forged. Note that Dec is stateful and so, for example, the third message in the sequence C^1, C^2, C^2 should certainly not be regarded as valid.

While we had the decryption oracle return the actual plaintext, it makes for an equivalent definition for the oracle to return only an indication of valid/invalid. This is because the adversary wins (forges) as soon as it obtains a nontrivial and valid decryption-oracle response. The value of that response is never used.

We have decided to give the adversary no credit—it does not forge—if the decryption oracle returns INVALID. This is meant to capture the expectation that a connection is dropped after an authentication failure. Alternatively, we could allow the adversary to play on, and possibly forge.

3 Stream-Cipher Cores and their Security

A stream-cipher core is a lower-level object than a stream cipher. From a stream-cipher core one can build a stream cipher in a standard way, and yet a stream-cipher core is a conceptually different and simpler object. A stream-cipher core lets you encrypt just *one* message, using a key of the appropriate length for that message. There is no state and no parameter Q for a stream-cipher core, and the security definitions are simpler. And here we shall only consider information-theoretic security definitions for stream-cipher cores.

In this section we describe the syntax for a stream-cipher core and then we give notions of privacy and authenticity for these objects.

3.1 Syntax of a stream-cipher core

A *stream-cipher core* is a triple of algorithms $\mathcal{SCC} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ as follows:

- The key-generator $\mathcal{K}$ is a probabilistic algorithm $\mathcal{K} : \mathsf{N} \to \{0,1\}^*$. It takes a number $m > 0$ (the length of the plaintext that needs to be encrypted) and (depending on internal coin tosses) it returns a string $\mathbf{K} \in \{0,1\}^*$ having length $m + R$, for some constant R.
- Encryption algorithm $\mathcal{E}$ is a deterministic function that takes a message $M \in \mathsf{Msg}$ and a key $\mathbf{K}$ of length $|M|+R$ and returns a string $C = \mathcal{E}_{\mathbf{K}}(K)$ of length $|M|+r$. Here $r > 0$ is a number (how much longer a ciphertext is than a plaintext) and Msg is a a linear-time testable set (the message space).
- Decryption algorithm $\mathcal{D}$ takes a ciphertext $C \in \{0,1\}^*$ and a string $\mathbf{K}$ of length $|C|+R-r$ and returns a value $M = \mathcal{D}_{\mathbf{K}}(M)$ which is either a string in Msg of length $|C| - r$ or else is the distinguished symbol INVALID.

We require that for any m, $\mathbf{K} \in \mathcal{K}(m)$, and $M \in \mathsf{Msg}$, if $C = \mathcal{E}_{\mathbf{K}}(M)$ then $\mathcal{D}_{\mathbf{K}}(C) = M$.

We say that C is *invalid* (with respect to a given key $\mathbf{K}$) if $\mathcal{D}_{\mathbf{K}}(C) = \text{INVALID}$; otherwise, C is *valid*.

OBTAINING KEYS FROM INFINITE STRINGS. We assume that the distribution $\mathcal{K}(m)$ can be obtained as follows: choose an infinite random string κ and process bits of κ left-to-right until one obtains an adequate number of bits—say $\mathit{nbits}(\kappa, m)$ bits. Then output a function of those bits as the key $\mathbf{K} = \mathit{extract}(\kappa, m) \in \{0,1\}^{m+R}$. Functions *nbits* and *extract* are associated to the stream-cipher core $\mathcal{SCC}$.

By the assumption of the last paragraph we may use $\kappa \in \{0,1\}^\omega$ to name a key for $\mathcal{E}$ or $\mathcal{D}$: that is, $\mathcal{E}_\kappa(M)$ or $\mathcal{D}_\kappa(C)$ instead of $\mathcal{E}_{\mathbf{K}}(M)$ or $\mathcal{D}_{\mathbf{K}}(C)$.

DISCUSSION. The restrictions that ciphertexts are a fixed amount longer than plaintexts, r bits, and that keys are a fixed amount longer than plaintexts, R bits, are unnecessary but simplify our later definitions.

3.2 Perfect privacy of a stream-cipher core

A stream-cipher core $\mathcal{SCC} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with parameters r, R and message space Msg is said to be *perfectly private* if, for every $M \in \mathsf{Msg}$, the distribution $[K \xleftarrow{R} \mathcal{K}(|M|) : \mathcal{E}_K(M)]$ is the uniform distribution on $|M| + r$ bits.

3.3 Statistical authenticity of a stream-cipher core

We give a strong notion of authenticity for a stream-cipher core $\mathcal{SCC} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with parameters r, R and message space Msg:

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathcal{SCC}}(m) \quad = \quad \max_{M,C,\tilde{C}} \{\Pr_{\kappa}[\mathcal{D}_\kappa(\tilde{C}) \neq \textsc{invalid} \mid \mathcal{E}_\kappa(M) = C\}$$

The maximum is over all $M, C, \tilde{C}$ such that $C \neq \tilde{C}$ and $|M| \leq m$ and $|C| \leq m + r$ and the conditioning event above is nonempty.

Explanation. The definition above can be understood as follows. An adversary asks for the ciphertext C of a single message M having at most m bits. Based on this, it tries to concoct a forgery $\tilde{C}$ for some (possibly unknown) message $\tilde{M}$ that has at most m bits. The forgery is with respect to the same underlying key that encrypted M. The value $\mathsf{Adv}^{\mathrm{auth}}_{\mathcal{SCC}}(m)$ upperbounds the probability that the adversary will succeed in this game. It is a pessimistic upperbound, insofar as we fix an (M, C) that is most advantageous for the adversary.

4 Definition of MULTI-S01

We will take two different views about what MULTI-S01 *is*: a stream-cipher core and a stream cipher. Only the former object is clearly specified in [DEF], but perhaps the latter may be viewed as implicit.

The description of MULTI-S01 given here does not closely follow that in [DEF]. In particular, we have had to introduce our own abstraction boundaries, both as a way to clarify what is going on in MULTI-S01 and also as a way to facilitate a more scientific exposition of the security properties of MULTI-S01.

4.1 MULTI-S01 as a stream-cipher core

We first describe MULTI-S01 as a stream-cipher core MULTI-S01-SCC $= (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Thus $\mathcal{K}$ shall be a probabilistic function and $\mathcal{E}$ and $\mathcal{D}$ shall be deterministic ones. One samples from $\mathcal{K}(m)$ to determine a key $\mathbf{K}$ to be used to encrypt a message M of length m. To encrypt another message, sample to determine another key.

Illustration. We begin with a picture. Encryption under MULTI-S01 is depicted in Figure 1. Referring to that picture, a message M which is a positive multiple of 64 bits is broken into 64-bit pieces $M_1 \cdots M_m$. A keystream $A\, B_1 \cdots B_m\, S$ is then used to encrypt M, where each A, B_i, and S is 64 bits. The value R is a fixed constant, say $\mathsf{R} = 0^{64}$. Two primitives are employed: xor of 64-bit values (denoted $\oplus$) and GF(2^{64})-multiplication of 64-bit values (denoted $\otimes$). The resulting ciphertext $C = C_1 \cdots C_{m+2}$ is transmitted to the receiver. The receiver recovers M in the natural way, but rejects M if the recovered values S' or R' are different from the anticipated values S and R.

Key generation. The function $\mathcal{K}$ takes a number m (the message length for the message we will subsequently encrypt) and outputs a key $\mathbf{K}$. For MULTI-S01, the key $\mathbf{K}$ output by $\mathcal{K}(m)$ consists of 64 random-but-nonzero bits followed by $m + 192$ random bits.

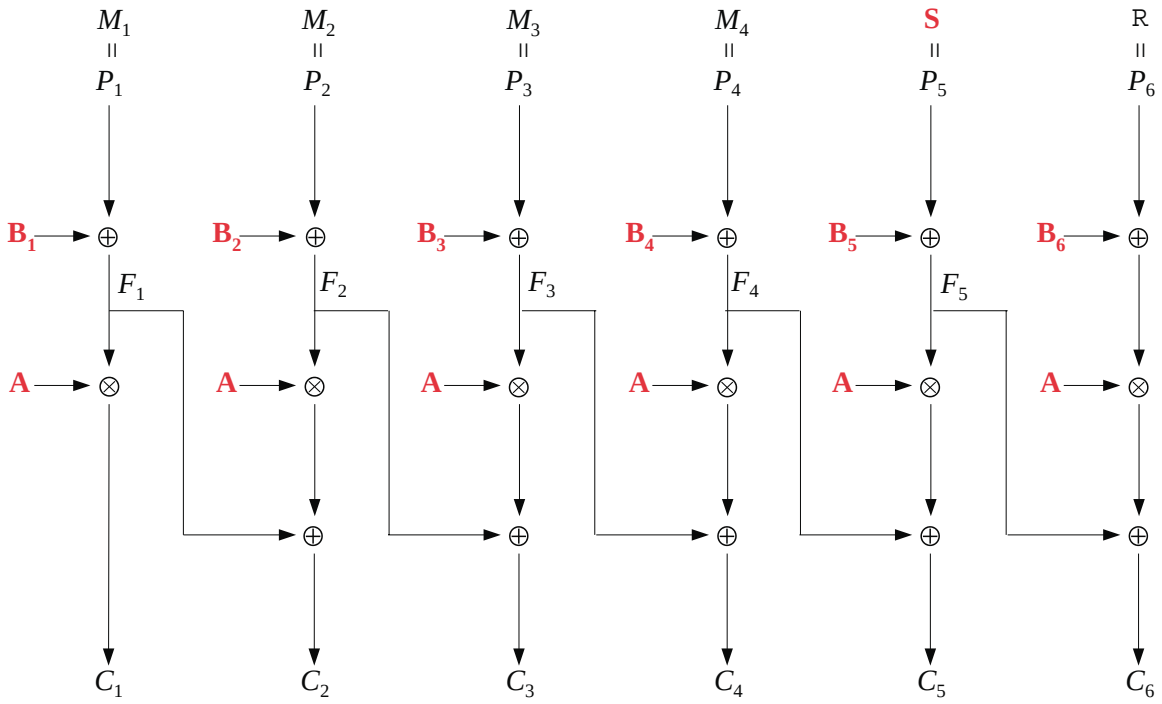

Figure 1: *The encryption function at the core of* MULTI-S01. *All blocks are 64 bits. We are encrypting the 4-block message* $M = M_1M_2M_3M_4$ *into ciphertext* $C = C_1C_2C_3C_4C_5C_6$ *with key* $\mathbf{K} = A\ B_1B_2B_3B_4B_5B_6\ S$ *and "redundancy"* R.

function $\mathcal{K}(m)$
begin
$A \xleftarrow{R} \{0,1\}^{64} \setminus \{0^{64}\}$
$B \xleftarrow{R} \{0,1\}^{m+128}$
$S \xleftarrow{R} \{0,1\}^{64}$
$\mathbf{K} \leftarrow A \,\|\, B \,\|\, S$
return K
end

Encrpytion. First, fix a constant $\mathsf{R} \in \{0,1\}^{64}$ (the "redundancy"). The constant R is used for both encryption and decryption. Let $M \in (\{0,1\}^{64})^+$ be the message we wish to encrypt. That is, M is a positive multiple of the *blocksize*, which is 64 bits. Document [DEF] assumes that $|M| \leq 2^{38}$ (ie., M has at most 2^{32} blocks). So, overall, the message space for MULTI-S01 is $\mathsf{Msg} = \{\{0,1\}^{64}, \{0,1\}^{2\cdot 64}, \ldots, \{0,1\}^{2^{32}\cdot 64}\}$. Let $\mathbf{K}$ be a key determined by running $\mathcal{K}(|M|)$. That is, the key $\mathbf{K}$ used to encrypt M is four blocks longer than M, but the first block is nonzero. We now define $\mathcal{E}_{\mathbf{K}}(M)$ as follows:

```
function E_K(M)
begin
    Parse M into 64-bit blocks M_1 ··· M_m
    Parse K into 64-bit blocks A B_1 ··· B_m B_{m+1} B_{m+2} S
    Let M_{m+1} ← S
    Let M_{m+2} ← R
    Let F_0 ← 0^64
    for i ← 1 to m + 2 do
        F_i ← M_i ⊕ B_i
        C_i ← A ⊗ F_i ⊕ F_{i−1}
    od
    return C ← C_1 ··· C_m C_{m+1} C_{m+2}
end
```

Above, $\oplus$ is used to denote xor of 64-bit strings, while $\otimes$ is used to denote multiplication of points in the finite field $\mathrm{GF}(2^{64})$ using a standard representation of field points as 64-bit strings, as specified in [DEF]. As usual, the multiplication operator is given higher precedence than the addition operator: $A \oplus B \otimes C$ means $A \oplus (B \otimes C)$.

DECRYPTION. The decryption function for MULTI-S01 is defined as follows.

```
function D_K(C)
begin
    If |C| is not divisible by 64 then return INVALID
    Let c ← |C|/64
    If c ≤ 2 or c > 2^32 then return INVALID
    Let m ← c − 2
    Parse C into 64-bit blocks C_1 ··· C_m C_{m+1} C_{m+2}
    Parse K into 64-bit blocks A B_1 ··· B_m B_{m+1} B_{m+2} S
    Let F_0 ← 0^64
    for i ← 1 to m + 2 do
        F_i ← A ⊗ (C_i ⊕ F_{i−1})
        M_i ← F_i ⊕ B_i
    od
    If M_{m+1} ≠ S or M_{m+2} ≠ R then return INVALID
    return M ← M_1 ··· C_m
end
```

Note that decryption may return the distinguished symbols INVALID to indicate that the ciphertext received should be regarded as inauthentic.

4.2 MULTIS-S01 as a stream cipher

One deficiency of document [DEF] is its lack of clarity about how protocol MULTI-S01-SCC = $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ should be used to encrypt a sequence of messages $M^1, M^2, M^3, \cdots$. The document

describes how to construct from an underlying, fixed-length key K (say 256 bits) a keystream $\mathbf{K}$ of the appropriate type and length to encrypt a single message M, but it goes no further. We describe that method and then extend it to solve the "real" problem at hand—the construction of a stream cipher from the MULTI-S01 stream-cipher core.

Note that in this subsection we shall be simultaneously handling two issues: how to use a fixed length key K (say 256 bits) instead of a key $\mathbf{K}$ that depends on the message length; and how to encrpyt a sequence of plaintexts, not just one.

DESCRIPTION OF SINGLE-MESSAGE MODE. The MULTI-S01 specification [DEF] addresses the encryption and decryption of a single message, M, this message having a nonzero multiple of 64 bits, using a fixed-length key K. The method works as follows. The sender and receiver are assumed to share a key K drawn from some set Key. They also share a "deviation parameter" Q, drawn from some set Par. Using a function $\mathrm{P} : \mathsf{Key} \times \mathsf{Par} \to \{0,1\}^\omega$, the sender and receiver map K and Q to a $(|M| + 256)$-bit keystream $\mathbf{K} = \mathrm{P}_K(Q)$. The ciphertext the sender computes and transmits to the receiver is $C = \mathcal{E}_{\mathbf{K}}(M)$. The encryption and decryption methods are now as follows:

function $\mathsf{Enc}_K^Q(M)$	**function** $\mathsf{Dec}_K^Q(M)$
begin	**begin**
$\kappa \leftarrow \mathrm{P}_K(Q)$ (κ an infinite string)	$\kappa \leftarrow \mathrm{P}_K(Q)$
$i \leftarrow 1$	$i \leftarrow 1$
while $\kappa[i..i+63] = 0^{64}$ **do** $i \leftarrow i + 64$	**while** $\kappa[i..i+63] = 0^{64}$ **do** $i \leftarrow i + 64$
$\mathbf{K} \leftarrow \kappa\,[i..\lvert M\rvert + 255]$	$\mathbf{K} \leftarrow \kappa\,[i..\lvert M\rvert + 255]$
return $\mathcal{E}_{\mathbf{K}}(M)$	**return** $\mathcal{D}_{\mathbf{K}}(M)$
end	**end**

The stream cipher is MULTI-S01-SC $= (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$. This should not be confused with the stream-cipher core MULTI-S01-SCC $= (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The former uses a "short" key K while the latter uses a "long" key $\mathbf{K}$.

In single-message mode, the deviation parameter Q should be used for at most one message: with every message one must choose a new deviation parameter Q. This will be relaxed shortly.

PANAMA, AND WHAT IS PANAMA. The definition of MULTI-S01 [DEF] suggests that P be the function PANAMA described in [PANA]. The inventors refer to this function as a pseudorandom generator. We comment that the name *pseudorandom generator* is not quite correct—neither for what PANAMA is nor for the object that MULTI-S01 requires. Namely, a pseudrandom generator is an algorithm $\mathrm{P} : \mathsf{Key} \to \{0,1\}^\omega$ but MULTI-S01 needs an algorithm $\mathrm{P} : \mathsf{Key} \times \mathsf{Par} \to \{0,1\}^\omega$. (Here $\{0,1\}^\omega$ means an "infinite" sequence of random bits. In actuality, one is only using some finite prefix of this infinite string.) Such a map is more properly called an *infinite-output-length pseudorandom function*. PANAMA is an infinite-output-length pseudorandom function. It has a signature $\text{PANAMA} : \{0,1\}^{256} \times \{0,1\}^{256} \to \{0,1\}^\omega$, meaning that the key length and the deviation-parameter length are each 256 bits. Therefore, using PANAMA as above, MULTI-S01 allows one to encrypt a message M (having a positive multiple of 64 bits) using a 256-bit key K and a 256-bit deviation parameter Q into a ciphertext $C = \mathsf{Enc}_K^Q(M)$.

STREAM-PER-MESSAGE MODE. The definition of MULTI-S01 given in [DEF] is not specific about how one should encrypt a sequence of messages $M^1, M^2, \cdots$. One possiblity is what we call *stream-per-message mode*: for each message M^i the sender chooses a new deviation parameter Q^i and then transmit $C^i = \mathsf{Enc}_K^{Q^i}(M^i)$ along with adequate information for the receiver to recover Q^i. As an exmple, one could encrypt the i-th message as i and $C^i = \mathsf{Enc}_K^i(M^i)$, where i, as a superscript to $\mathcal{E}$, indicates the string in Par encoding the number i in the usual way (recall $Q = \{0,1\}^{256}$ when using PANAMA). If the sender and receiver are communicating over a reliable channel then i would not need to be explicitly transmitted, since the receiver can maintain the matching value himself. If the sender and receiver are communicating over an unreliable channel then i could be transmitted in the clear, as one does not need for Q^i to be secret. It is essential that the values $Q^1, Q^2, Q^3, \cdots$ used for $M^1, M^2, M^3, \cdots$ are all distinct.

MULTI-MESSAGE MODE: CONSTRUCTING A PROPER STREAM CIPHER. We expect that the inventors of MULTI-S01 didn't really envision the use of stream-per-message mode. Besides the inefficiency associated to re-keying the pseudorandom function, stream-per-message mode goes against the basic idea of how stream ciphers are supposed to be used. That idea entails that the sender and receiver share an effectively infinite string κ and use its pieces for each of the different messages that get sent. Namely, the sender wants to transmit some sequence of messages $M^1, M^2, \cdots$. To accomplish this, the sender will partition the string κ into pieces $K^1, K^2, \cdots$, as needed, and the sender will use K^i to encrypt M^i into ciphertext C^i.

The fact that MULTI-S01 chose to use a pseudorandom function instead of a pseudorandom generator suggests that the inventors have in mind still more flexibility: allowing the sender to "re-synch" at will—choose a new value Q and then transmit a message sequence $M^1, M^2, \cdots$ using the keystream κ determined by K and Q. The mode, which we call *multi-message mode*, is MULTI-S01-SC = $(\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$, defined as follows:

function $\mathsf{Enc}_K^Q(M)$	**function** $\mathsf{Dec}_K^Q(M)$
begin	**begin**
static $\mathrm{in}(Q) \leftarrow 1$ for all $Q \in \mathsf{Par}$	**static** $\mathrm{in}(Q) \leftarrow 1$ for all $Q \in \mathsf{Par}$
$\kappa \leftarrow \mathrm{P}_K(Q)$	$\kappa \leftarrow \mathrm{P}_K(Q)$
while $\kappa[\mathrm{in}(Q)..\mathrm{in}(Q)+63] = 0^{64}$	**while** $\kappa[\mathrm{in}(Q)..\mathrm{in}(Q)+63] = 0^{64}$
do $\mathrm{in}(Q) \leftarrow \mathrm{in}(Q) + 64$	**do** $\mathrm{in}(Q) \leftarrow \mathrm{in}(Q) + 64$
$\mathbf{K} \leftarrow \kappa\,[\mathrm{in}(Q)..\lvert M\rvert + 255]$	$\mathbf{K} \leftarrow \kappa\,[\mathrm{in}(Q)..\lvert M\rvert + 255]$
$\mathrm{in}(Q) \leftarrow \mathrm{in}(Q) + \lvert M\rvert + 256$	$\mathrm{in}(Q) \leftarrow \mathrm{in}(Q) + \lvert M\rvert + 256$
return $\mathcal{E}_{\mathbf{K}}(M)$	**return** $\mathcal{D}_{\mathbf{K}}(M)$
end	**end**

Note that our treatment of multi-message mode is a proper extension of single-message mode: the two modes agree when presented a sequence of distinct Q^i-values.

At this point we have a stream-cipher core, MULTI-S01-SCC, and a stream cipher proper, MULTI-S01-SC.

5 Security of MULTI-S01-SCC

Though the language of [EVAL] is quite different from that introduced here, [EVAL] does, in effect, give valid proofs for the privacy and authenticity of MULTI-S01-SCC. We now state the requisite theorems, as adapted from [EVAL].

5.1 Perfect privacy of MULTI-S01-SCC

For the MULTI-S01 stream-cipher core, privacy is perfect. This is quite natural. Referring to Figure 1, note that B_6 is uniformly distributed and A is non-zero so $A \otimes B_6$ is uniformly distributed. This value is independent of the value $A \otimes P_6 \oplus F_5$ it gets xor'ed with to form C_6, and so C_6 is uniformly distributed. The same reasoning now applies to $C_5, C_4, \cdots, C_1$, and we can conclude the following.

Theorem 1 MULTI-S01-SCC *is perfectly private.*

5.2 Statistical authenticity of MULTI-S01-SCC

Possibly the most interesting part of [SELF] was an argument along the following lines. Suppose that an adversary sees a MULTI-S01-SCC ciphertext C for a plaintext M produced under a random key $\mathbf{K} = ABS$. Information theoretically, the ciphertext tells the adversary nothing about A. In the absence of any information about A (even with complete information about B, S), a candidate ciphertext $\tilde{C}$ different from C will be valid with a probablity easily bounded using some case analysis and the fundamental theorem of algebra. In our language, the result from [SELF] is as follows.

Theorem 2 $\mathsf{Adv}^{\mathrm{auth}}_{\text{MULTI-S01-SCC}}(m) \leq (m/64 + 2)/(2^{64} - 1)$.

(The division by 64 is simply to convert to blocks, the addition of 2 accounts for the lengthening of the ciphertext, and the subtraction by 1 is because $A \neq 0^{64}$.) Though we found the exposition of this result in [SELF] somewhat lacking, the result and proof approach are correct, so we do not repeat that work here.

6 From Stream-Cipher Core to Stream Cipher

As we have just explained, [SELF] contains what amounts to a proof of perfect privacy and statistical authenticity for the MULTI-S01-SCC stream-cipher core. At first glance, this falls well short of proving what is actually desired: proving that the stream cipher MULTI-S01-SC achieves privacy and authenticity. We now argue that this is not problem: under our definitions, the manner in which MULTI-S01-SCC is promoted from a stream-cipher core to a stream cipher automatically promotes privacy and authenticity, too.

6.1 Converting a stream-cipher core to a stream cipher

Let $\mathcal{SCC} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a stream-cipher core with parameters r, R and associated functions $\mathit{nbits}(\cdot,\cdot)$ and $\mathit{extract}(\cdot,\cdot)$. Let $\mathrm{P} : \mathsf{Key} \times \mathsf{Par} \to \{0,1\}^\omega$ be a function. Consider the following way to extend stream-cipher core $\mathcal{SCC} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ to a stream cipher $\mathcal{SC} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$:

```
function Enc_K^Q(M)
begin
    static in(Q) ← 1 for all Q ∈ Par
    κ ← P_K(Q)
    m ← |M|
    K ← extract(κ[in(Q)..], m)
    in(Q) ← in(Q) + nbits(κ[in(Q)..], m)
    return E_K(M)
end
```

```
function Dec_K^Q(M)
begin
    static in(Q) ← 1 for all Q ∈ Par
    κ ← P_K(Q)
    m ← |M|
    K ← extract(κ[in(Q)..], m)
    in(Q) ← in(Q) + nbits(κ[in(Q)..], m)
    return D_K(M)
end
```

The above extension agrees with how we have extended MULTI-S01 from a stream-cipher core to a stream cipher. In particular, our earlier exposition coincides with the above where *extract* and *nbits* are defined by

```
function nbits(κ, m)
begin
    i ← 0
    while κ[i + 1..i + 64] = 0^64 do i ← i + 64
    return i + m + 256
end

function extract(κ, m)
begin
    i ← 0
    while κ[i + 1..i + 64] = 0^64 do i ← i + 64
    return κ[i + 1..m + 256]
end
```

The information-theoretic extension of a stream-cipher core. One can think of the function $\mathrm{P} : \mathsf{Key} \times Q \to \{0,1\}^\omega$ in a couple of ways: as a typical pseudorandom function, such as Panama, where $\mathsf{Key} = \{0,1\}^\kappa$ is a finite set; or as the function $\mathrm{P}(K, Q) = K(Q)$, where Key is a map from Par to $\{0,1\}^\omega$. The latter gives an information-theoretic extension of a stream-cipher core to a stream cipher; the former, a complexity-theoretic one.

6.2 Theorems that justify a focus on stream-cipher cores

The following results effectively justify the security treamement of [SELF]. In that work, the authors argue that MULTI-S01-SCC achieves perfect privacy and that MULTI-S01-SCC achieves the expected degredation in authenticity. By looking at security only with respect to the stream-cipher core, the analysis effectively pays attention only to what happens if one encrpyts a single message and tries to forge a single ciphertext after having seen a single plaintext/ciphertext pair. It turns out that this is enough. Our first theorem speaks to privacy.

Theorem 3 *Let $\mathcal{SCC} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a stream-cipher core and let $\mathcal{SC} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ be its information-theoretic extension. Suppose $\mathcal{SCC}$ is perfectly private. Then $\mathcal{SC}$ is perfectly private, too.*

Theorem 3 seems reasonably obvious and so we omit its proof.

Our next theorem speaks to the authenticity of the information-theoretic extension of a stream-cipher core. Note that privacy is necessary for this second result.

Theorem 4 *Let $\mathcal{SCC} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a stream-cipher core and let $\mathcal{SC} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ be its information-theoretic extension. Assume that $\mathcal{SCC}$ is perfectly private. Then, for any m, $\mathsf{Adv}^{\mathrm{AUTH}}_{\mathcal{SC}}(m) \le \mathsf{Adv}^{\mathrm{auth}}_{\mathcal{SCC}}(m)$.*

Proof sketch. Let A be an adversary that attacks the stream cipher $\mathcal{SC} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ obtained from stream-cipher core $\mathcal{SCC} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Without loss of generality (by the standard averaging argument), assume that A is deterministic. We construct a tuple $(M, C, \tilde{C})$ where $|M| \le m$, $|C| = |M| + r$, $|\tilde{C}| \le m + r$, and such that $\Pr_\kappa[\mathcal{D}_\kappa(\tilde{C}) \neq \textsc{invalid} \mid \mathcal{E}_\kappa(M) = C] \ge \mathsf{Adv}^{\mathrm{AUTH}}_{\mathcal{SC}}(A)$.

To construct $(M, C, \tilde{C})$, start off by running adversary A. This adversary has two oracles: $\mathbf{Enc}_{\mathcal{SC}}$ and $\mathbf{Dec}_{\mathcal{SC}}$. When A makes a query (M, Q) to its $\mathbf{Enc}_{\mathcal{SC}}$ oracle, answer with $|M| + r$ random bits, and record the query and its answer. Let $(M^{1,Q}, \ldots, M^{a(Q),Q})$ record the sequence of queries asked by A using parameter Q, and let $(C^{1,Q}, \ldots, C^{a(Q),Q})$ record the corresponding answers. When A makes a query (C, Q) to its $\mathbf{Dec}_{\mathcal{SC}}$ oracle, let $(\tilde{C}^{1,Q}, \ldots, C^{b(Q),Q})$ record the sequence of queries, to date, having parameter Q. Then if $(\tilde{C}^{1,Q}, \ldots, C^{b(Q),Q})$ is a prefix to $(C^{1,Q}, \ldots, C^{a(Q),Q})$, return $C^{a(Q),Q}$. Otherwise, consider $(M, C, \tilde{C})$ to be a "candidate triple."

Note that we have simulated the "correct" behavior for $\mathbf{Enc}_{\mathcal{SC}}$ because of the perfect privacy of $\mathcal{SC}$.

By our definitions of stream-cipher authenticity, if $\tilde{C}$ is not a valid forgery then the adversay A has not and will never forge on this run. On the other hand, if $\tilde{C}$ is a valid forgery then the adversary A has succeeded on this run.

When A forges it produces a candidate triple $(M, C, \tilde{C})$. The value $\mathsf{Adv}^{\mathrm{AUTH}}_{\mathcal{SC}}(A)$ can be regarded as a a weighted sum $\sum \lambda_{M,C,\tilde{C}} \Pr[A \text{ forges} \mid \text{candidate triple } (M, C, \tilde{C})]$. Thus there is a tuple $(M, C, \tilde{C})$ such that $\Pr[A \text{ forges} \mid \text{candidate triple } (M, C, \tilde{C})] \ge \mathsf{Adv}^{\mathrm{AUTH}}_{\mathcal{SC}}(A)$. ∎

6.3 Requisite security property for function P

None of [DEF, SELF, EVAL] discuss the properties that P needs to have in order for MULTI-S01 to be secure. But a necessary and sufficient property for P is that it be secure in the usual sense of a pseudorandom function. We now review that notion.

Let $\mathrm{P} : \mathsf{Key} \times \mathsf{Par} \to \{0,1\}^\omega$. Consider an adversary A having one of two oracles: a "real" oracle or a "random" oracle. A real oracle behaves as follows. It begins by choosing a random key $K \xleftarrow{R} \mathsf{Key}$. Then, to any query (X, ℓ), the oracle returns the first ℓ bits of $\mathrm{P}_K(X)$. A random oracle behaves as follows. It begins by associating to every string X an infinite string of random bits $\rho(X)$. Then, to any query (X, ℓ), the oracle returns the first ℓ bits of $\rho(X)$. The *advantage* that A gets in attacking the pseudorandom function P is the real number

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathrm{P}}(A) = \Pr[A^{\mathrm{Real}} = 1] - \Pr[A^{\mathrm{Rand}} = 1]$$

where the oracles "Real" and "Random" act in the way we have just described. Informally, a "good" pseudorandom function is a map P for which $\mathsf{Adv}^{\mathrm{prf}}_{\mathrm{P}}(A)$ is small (close to 0) whenever A is computationally reasonable algorithm (it runs in a reasonable amount of time and obtains from its oracle a reasonable number of bits).

7 Criticisms and Suggestions

Much of this report has focussed on verifying that, fundamentally, MULTI-S01 is *correct*: properly used, MULTI-S01 *does* provide privacy and authenticity, with the expected bounds, assuming PANAMA (say) is secure in the sense of an infinite-output-length pseudorandom function. Thus we find no basis for the main criticisms of [EVAL] that vaguely questioned the privacy and authenticity of MULTI-S01. All the same, we are not very supportive of the MULTI-S01 technique. Let us explain the reasons.

7.1 Length restrictions

MULTI-S01 requires that messages to be encrypted have a length which is a positive multiple of 64 bits. While mandatory padding techniques can of course be used to ensure this, a modern and refined encryption algorithm can and should do better, saving on communication bits. One would prefer a solution for which ciphertext lengths will not be increased unnecessarily, and it is not hard to construct such techniques. Generic composition of a Vernam cipher and a MAC is one such a technique.

7.2 Ciphertext length

The length of a MULTI-S01 ciphertext C for a plaintext M is $|C| = |M| + 128$. (If mandatory padding is added so as to address the issue of the last subsection, the length of a ciphertext for a plaintext M becomes longer still.) This would be fine if the forgery probability were approximately $\|M\| \cdot 2^{-128}$, say. (Here $\|M\|$ means $|M|/64$.) But it is not; the forgery probability is approximately $\|M\| \cdot 2^{-64}$. For such a forgery probability one should not be spending 128 extra bits, but no more than 64. Indeed we do not understand why the inventors choose to expend *two* blocks in order to get authenticity in MULTI-S01 when it would seem to be simple to make do with one.

7.3 Forging probability

The forging probability of approximately $\|M\| \cdot 2^{-64}$ may be larger than is acceptable for some applications. We would suggest that a new authenticated-encryption scheme should be able to transmit a message in a way so as to have forging probability more like $\|M\| \cdot 2^{-128}$. Ideally, one should be able to truncate a 128-bit tag so that if τ bits are transmitted then one gets a forging probability of approximately $2^{-\tau}$. Even if the tag were extended to 128 bits, results like this may not be true with the method of MULTI-S01 as the security impact of truncation on the tag is far from clear.

7.4 Software speed

The main motivation for integrating privacy and authenticity is to obtain speed benefits unobtainable (or difficult to obtain) through generic composition. But data presented in [SELF, SLIDES], as well as back-of-envelope calculations, suggest that MULTI-S01/PANAMA does not obtain speeds superior to those obtainable by generic composition. Consider the Pentium III speeds reported in [SLIDES] (which appear to be the fastest figures reported): 17.6 cycles/byte for 256 KByte messages. This is inferior with speeds reported by Lipmaa for OCB-AES128 encryption: 16.9 cycles/byte for 1 KByte messages.

The authors of [SELF] suggest that performance suffers on a 32-bit processor like a Pentium III because of the lack of available 64-bit instructions (discounting MMX instruction). But the inventors' figures for a 64-bit platform are not particularly better: [SLIDES] reports 17.7 cycles/byte for a 4 KByte message.

Key setup is quite slow for MULTI-S01/PANAMA; the authors of [SELF] report key setup times of about 32,000 cycles. Compared to AES key-setup, this is enormous, and it largely limits MULTI-S01 applicability to contexts in which a long-lived session is being initiated.

In view of the added difficulty of using a stream cipher compared to an authenticated-encryption scheme based on a block cipher, we feel that the target speed for a stream-cipher-based authenticated-encryption scheme needs to be a substantial amount faster than the speed obtained by a block-cipher-based authenticated-encryption scheme like IAPM or OCB or generic composition with with a software-optimized MAC. Thus a stream-cipher-based authenticated-encryption scheme ought to reach for speeds of 10 cycles/byte, or faster, until the performance-advantage starts to become attractive. If one chooses an infinite-output-length pseudorandom function that does pad-generation in roughly 5 cycles/byte, as reported for SEAL, or 7 cycles/byte, as reported for PANAMA, then one has roughly 5 cycles/byte left to spend on authenticity and still out-perform an AES-based solution. This would seem to be quite doable using an aggressively optimized MAC (recall that UMAC performs at about 1 cycle/byte).

Fundamentally, the "overhead" of MULTI-S01 (meaning the work done beyond PANAMA) is more than desired because every 64 bits of message results in multiplying two GF(2^{64})-values, neither of which are fixed beyond the scope of a single message. A GF(2^{64})-multiplication requires several cycles. To get impressive performance one will have to base the authenticity-enabling calculations on something faster than a GF(2^{64})-multiplication of frequently-changing values. We given an example of such a technique in the next section.

7.5 Generic composition as a better alternative

The alternative to *any* tightly-integrated authenticated-encryption scheme is generic composition. The generic composition paradigm has been studied in [BeNa]. Though that work did not directly address stream ciphers, stream ciphers are no less appropriate as starting points for generic composition.

There are three natural approaches to generic composition: encrypt-and-mac, encrypt-then-mac, and mac-then-encrypt. The analysis in [BeNa] shows that encrypt-then-mac has the most desirable security characteristics. Though the results of that paper do not directly apply to stream ciphers, we believe that the results in [BeNa] can, all the same, be lifted to

deal with stream ciphers and that, once one does so, the overall conclusion will remain the same: encrypt-then-mac will have the best security properties.

To use encrypt-then-mac for a stream cipher takes a bit of care. To start with, assume a message authentication code $\text{MAC} : \mathcal{K}_{\text{mac}} \times \{0,1\}^* \to \{0,1\}^\tau$. Assume that the sender and receiver share an encryption key K_{enc} drawn from some set $\mathcal{K}_{\text{enc}}$, and assume that they share a MAC key K_{mac} drawn from some set $\mathcal{K}_{\text{mac}}$. Assume one has a privacy-only stream cipher in which, to encrypt a message M, the sender computes a ciphertext $C = \mathcal{E}_{K_{\text{enc}}}(M) = \mathit{MakePad}(\mathit{state}_{\text{sender}}) \oplus M$. The functions $\mathcal{E}$ and *MakePad* depend on, and mutate, the internal variable $\mathit{state}_{\text{sender}}$. Likewise, to decrypt a ciphertext C, the receiver computes $M = \mathcal{D}_{K_{\text{enc}}}(C) = \mathit{MakePad}(\mathit{state}_{\text{receiver}}) \oplus C$. These functions depends on, and mutate, the internal variable $\mathit{state}_{\text{receiver}}$.

To modify the privacy-only Vernam cipher to provide for authenticity too, have the sender send, instead of C, the ciphertext $C' = \langle C,\ \text{MAC}_{K_{\text{mac}}}(\langle \mathit{state}_{\text{sender}},\ C\rangle)\rangle$. Have the receiver, on receipt of $\langle C^*,\ T^*\rangle$, compute the expected tag $T' = \text{MAC}_{K_{\text{mac}}}(\langle \mathit{state}_{\text{receiver}},\ C^*\rangle)$. If $T' = T^*$ then the receiver recovers the plaintext M by $M = \mathcal{D}_{K_{\text{enc}}}(C)$. Otherwise, the decryption gives INVALID. Note that MAC generation and MAC verification does not mutate any internal state, and note that the scope of the MAC includes (and must include) the state which the sender and receiver maintain when using a stream cipher. Typically, this state is a counter of just a few bytes. Assume in what follows that it is 16 bytes.

The efficiency of the approach above depends, almost entirely, on the efficiency of the MAC: the computational overhead for authenticity is, very nearly, the computational overhead of MACing a message having the same length as the plaintext. The question is, therefore, if one can MAC using an overhead less than the overhead of MULTI-S01 (meaning the computational work done by MULTI-S01 above and beyond computing the keystream).

The answer to this would seem to be *yes*. As an example, the UMAC message authentication code [UMAC] authenticates messages with approximately 1 cycle/byte. But one can argue that it is much more complex and less suited for hardware than MULTI-S01. Therefore we sketch an example message authentication code (a standard Carter-Wegman MAC) more in the spirit of MULTI-S01.

Let C be the message we wish to MAC. Let $\mathit{paddedC} = C \parallel 1 \parallel 0^i$, where i is the smallest value such that $|\mathit{paddedC}|$ is divisible by 64. Partition *paddedC* into 64-bit blocks $C_{m-1} \cdots C_1 C_0$. Let the MAC key be $K_{\text{mac}} = \alpha \parallel \beta$ where $|\alpha| = |\beta| = 64$. To MAC C, compute $\text{MAC}_{\alpha\beta}(C) = \text{AES}_\beta(\mathit{state}_{\text{sender}}) \oplus H_\alpha(\mathit{paddedC})$ where

$$H_\alpha(\mathit{paddedC}) \;=\; \alpha^m \;\oplus\; C_{m-1} \otimes \alpha^{m-1} \;\oplus\; C_{m-2} \otimes \alpha^{m-2} \;\oplus\; \cdots \;\oplus\; C_1 \otimes \alpha \;\oplus\; C_0$$

As before, $\oplus$ and $\otimes$ refer to addition and multiplication in $\text{GF}(2^{64})$. Correctness of this method as a MAC is standard; it depends on H being a good xor-universal hash-function family.

To efficiently compute $H_\alpha(\mathit{paddedC})$, start a 64-bit register A with the value $0^{127}1$. Repeatedly take the value of A, multiply it by α, then add in (xor) the next value C_i (as one reads left-to-right down $C_{m-1} \cdots C_1 C_0$). The computational efficiency of this method is better, per byte, than multiplying two points in $\text{GF}(2^{64})$ since one of the two points, being fixed for as long as one uses the underlying authenticated-encryption key, can be preprocessed into tables to permit more the efficient computation, if desired. Indeed one could move to using $\text{GF}(2^{128})$ and still preserve efficiency. All in all, the generic composition approach, as sketched above, would seem to be faster and more flexible than the approach take by MULTI-S01.

References

[BeNa] **M. Bellare** and **C. Namprempre**. Authenticated encryption: Relations among notions and an analysis of the generic composition paradigm. *Advances in Cryptology – ASIACRYPT '00*. Lecture Notes in Computer Science, vol. 1976, 2000.

[DEF] **Hitachi, Ltd.** A Symmetric Key Encryption Algorithm: MULTI-S01 — An Integrity-Aware Block Encryption Based on Cryptographic Pseudorandom Number Generator. Undated manuscript with copyright 2000, 2001. Note: Hitachi maintains website `http://www.sdl.hitachi.co.jp/crypto/s01/index.html` for information about MULTI-S01.

[EVAL] **Anonymous.** Evaluation of MULTI-S01. Unpublished and confidential manuscript. January 17, 2001.

[IAPM] **C. Jutla.** Encryption modes with almost free message integrity. *Advances in Cryptology – EUROCRYPT 2001.* Lecture Notes in Computer Science, vol. 2045, Springer-Verlag, 2001.

[OCB] **P. Rogaway**, **M. Bellare**, **J. Black**, and **T. Krovetz**. OCB: A block-cipher mode of operation for Efficient Authenticated Encryption. *Proceedings of the 8th ACM Conference on Computer and Communications Security* (CCS-8), ACM Press, 2001.

[PANA] **J. Daemen** and **C. Clapp**. *Fast Software Encryption, 5th International Workshop* (FSE '98), Lecture Notes in Computer Science, vol. 1372, Springer-Verlag, 1998.

[SELF] **Hitachi, Ltd.** Self-Evaluation Report MULTI-S01 (revised for 2001 submittion[sic]). Undated manuscript with copyright 2000, 2001.

[SLIDES] **S. Furuya** (Hitachi Ltd.) A stream cipher for Cryptrec: MULTI-S01. Transparencies from an undated talk, copyright 2001.

[UMAC] **J. Black**, **S. Halevi**, **H. Krawczyk**, **T. Krovetz**, and **P. Rogaway**. UMAC: Fast and secure message authentication. *Advances in Cryptology — CRYPTO '99.* Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, 1999.

[XCBC] **V. Gligor** and **P. Donescu.** Fast encryption and authentication: XCBC encryption and XECB authentication modes. *Fast Software Encryption*, Lecture Notes in Computer Science, Springer-Verlag, 2001.