DIAC

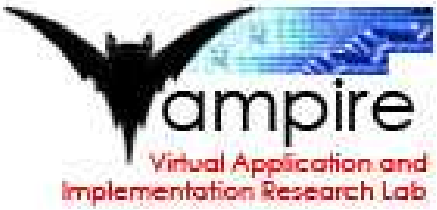
Directions in Authenticated Ciphers

5–6 July 2012

Stockholm, Sweden

Organized by

SymLab



within

ECRYPT II European Network of Excellence in Cryptography

Program committee:

Daniel J. Bernstein, University of Illinois at Chicago, USA
Carlos Cid, Royal Holloway, University of London, UK
Tetsu Iwata, Nagoya University, Japan
Thomas Johansson, Lund University, Sweden
Tanja Lange, Technical University of Eindhoven, The Netherlands
Stefan Lucks, University of Weimar, Germany
Kaisa Nyberg, Aalto University, Finland
Elisabeth Oswald, University of Bristol, UK
Bart Preneel, Katholieke Universiteit Leuven, Belgium
Vincent Rijmen, Katholieke Universiteit Leuven, Belgium
Phillip Rogaway, University of California, Davis, USA
Martijn Stam, University of Bristol, UK
François-Xavier Standaert, Universite catholique de Louvain, Belgium
Ingrid Verbauwhede, Katholieke Universiteit Leuven, Belgium

Subreviewers:

Elena Andreeva, Katholieke Universiteit Leuven, Belgium
Christian Forler, Bauhaus-Universität Weimar, Germany
Kimmo Järvinen, Aalto University, Finland
Jakob Wenzel, Bauhaus-Universität Weimar, Germany

Local organization:

Anita Klooster, Technical University of Eindhoven, The Netherlands
Tanja Lange, Technical University of Eindhoven, The Netherlands

Invited speakers:

Joan Daemen, STMicroelectronics, Belgium
David McGrew, Cisco Systems, USA
Phillip Rogaway, University of California, Davis, USA
Palash Sarkar, Indian Statistical Institute, Kolkata, India

Contributors:

Ignacio Aguilar Sanchez, European Space Agency, The Netherlands
Kazumaro Aoki, NTT Secure Platform Laboratories, Japan
Jean-Philippe Aumasson, NAGRA, Switzerland
Daniel J. Bernstein, University of Illinois at Chicago, USA
Guido Bertoni, STMicroelectronics, Belgium
Andrey Bogdanov, Katholieke Universiteit Leuven, Belgium
Alexandra Boldyreva, Georgia Institute of Technology, USA
Joan Daemen, STMicroelectronics, Belgium
Jean Paul Degabriele, Royal Holloway, University of London, UK
Elena Dubrova, Kungliga Tekniska högskolan, Sweden
Daniel Engels, Revere Security, USA
Daniel Fischer, European Space Agency, Germany
Christian Forler, Bauhaus-Universität Weimar, Germany
Frank K. Gürkaynak, Eidgenössische Technische Hochschule Zürich, Switzerland
Tetsu Iwata, Nagoya University, Japan
Simon Knellwolf, Fachhochschule Nordwestschweiz, Switzerland
Stefan Lucks, Bauhaus-Universität Weimar, Germany
David McGrew, Cisco Systems, USA
Willi Meier, Fachhochschule Nordwestschweiz, Switzerland
Florian Mendel, Katholieke Universiteit Leuven, Belgium
Kazuhiko Minematsu, NEC Corporation, Japan
Hiraku Morita, Nagoya University, Japan
Kenneth G. Paterson, Royal Holloway, University of London, UK
Michaël Peeters, NXP Semiconductors, Belgium
Bart Preneel, Katholieke Universiteit Leuven, Belgium
Francesco Regazzoni, Università della Svizzera italiana, Switzerland
Vincent Rijmen, Katholieke Universiteit Leuven, Belgium
Markku-Juhani O. Saarinen, Revere Security, USA
Shohreh Sharif Mansouri, Kungliga Tekniska högskolan, Sweden
Martijn Stam, University of Bristol, UK
Gilles Van Assche, STMicroelectronics, Belgium
Jakob Wenzel, Bauhaus-Universität Weimar, Germany
Hongjun Wu, Nanyang Technological University, Singapore
Kan Yasuda, NTT Secure Platform Laboratories, Japan

Program and table of contents:

Thursday 5 July

Friday 6 July

The evolution of authenticated encryption

Phillip Rogaway

University of California, Davis
USA

Nowhere has the promise of practice-oriented provable security been more successful than in the area of authenticated encryption. In this talk I will look back and and survey how the provable-security approach has led not just to high-assurance blockcipher-based AE schemes, but, also, how it has led to a blossoming of security notions, attacks, design goals, and designs.

Cryptanalysis of EAX-Prime

Kazuhiko Minematsu[1], Stefan Lucks[2], Hiraku Morita[3], and Tetsu Iwata[4]

[1] NEC Corporation, Japan, k-minematsu@ah.jp.nec.com
[2] Bauhaus-Universität Weimar, Germany, stefan.lucks@uni-weimar.de
[3] Nagoya University, Japan, h_morita@echo.nuee.nagoya-u.ac.jp
[4] Nagoya University, Japan, iwata@cse.nagoya-u.ac.jp

Abstract. EAX′ (EAX-prime) is an authenticated encryption (AE) specified by ANSI C12.22 as a standard security function used for a smart grid. EAX′ is based on EAX, a provably secure AE proposed by Bellare, Rogaway, and Wagner. This paper presents simple and efficient forgery attacks, distinguishers, and message recovery attacks against EAX′ using single-block cleartext and plaintext.

1 Introduction

ANSI C12.22-2008 [1] specifies a blockcipher mode for authenticated encryption (AE) as the standard security function for Smart Grids. It is called EAX′ (or EAX-prime)[5]. As its name suggests, EAX′ is based on EAX proposed by Bellare, Rogaway, and Wagner at FSE 2004 [4]. It was submitted to NIST [7] and NIST is planning to include EAX′ in an additional part of the 800-38 series of Special Publications.

Though EAX′ is similar to EAX, to the best of our knowledge, its formal security analysis is not known to date. In this paper, we investigate the security of EAX′ as a general-purpose authenticated encryption, and provide attacks against EAX′ for both authenticity and confidentiality. Specifically, we present

- *forgeries*, i.e., cleartext/ciphertext pairs with valid authentication tags,
- *chosen plaintext distinguishers*, distinguishing the EAX′ encryption from a random encryption process, and
- *chosen ciphertext message recovery attacks*, decrypting ciphertexts by asking for the decryption of another ciphertext with a valid authentication tag.

Our attacks are simple and efficient as they require only one or two queries. The simplest one even produces a successful forgery without the need to observe any valid plaintext/ciphertext pair at all. Our results imply that, while the original

[5] The authors of [7] exchangeably use the three names, EAX′, EAX', and EAX-prime, to mean their proposal. To avoid any confusion by overlooking the tiny prime symbol or apostrophe, which could be misunderstood as claiming an attack on EAX, we prefer the longer name "EAX-prime" for the title. In the text we prefer the name EAX′.

EAX has a proof of security, the security of EAX′ cannot be proved as a general-purpose authenticated encryption. In Sect. 6 we will show the differences between EAX′ and EAX that make this security separation.

We remark that the above attacks work only if the inputs to EAX′ are quite short[6]. The encryption of EAX′ accepts two elements, N and P, where N is called a "cleartext" (corresponds to a combination of a nonce and a header), and P is called a "plaintext". Our forgery attacks and distinguishers are only applicable when neither the length of N nor the length of P exceeds the block size of the underlying blockcipher (i.e., 128 bits for the AES). Our message recovery attacks are applicable when the length of N does not exceed the block size, even if the length of P exceeds the block size. However, at most the first block of P can be recovered.

Therefore, though we demonstrate the total insecurity of EAX′ when N and P are allowed to be 128 bits or shorter, we do not know if our attacks work to the protocols strictly following ANSI C12.22.

2 Definition of EAX′

First, let $\{0,1\}^*$ be the set of all binary strings, including the null string NULL. The bit length of $x \in \{0,1\}^*$ is denoted by $\mathrm{size}(x)$. Here $\mathrm{size}(\mathrm{NULL}) = 0$. A concatenation of $x, y \in \{0,1\}^*$ is written as $x\|y$. A sequence of a zeros (ones) is denoted by 0^a (1^a). The first i bits of x is written as $\mathrm{MSB}_i(x)$. For $x, y \in \{0,1\}^*$ with $\mathrm{size}(x) \geq \mathrm{size}(y)$, let $x \oplus_{\mathrm{end}} y = x \oplus \big((0^{\mathrm{size}(x)-\mathrm{size}(y)})\|y\big)$.

We describe EAX′. Basically our description is almost the same as the ANSI documents [1, 7], with some minor notational differences. EAX′ is a mode of operation based on an n-bit blockcipher, E_K, where we assume that (n, E_K) is $(128, \text{AES-128})$ as an example, while other choice is possible [7]. Formally, the encryption function of EAX′ accepts a cleartext, $N \in \{0,1\}^*$, a plaintext, $P \in \{0,1\}^*$, and a secret key K to produce the ciphertext, $C \in \{0,1\}^*$, with $\mathrm{size}(C) = \mathrm{size}(P)$ and $T \in \{0,1\}^{32}$. The decryption function, which we also call the verification function, accepts N, C, T, and K to generate the decrypted plaintext $\widetilde{P}$ if (N, C, T) is valid, and the flag `INVALID` if invalid. Cleartext N contains information that needs only authenticity and the ANSI document requires that N must be unique for all encryptions using the same key. Hence N can be seen as a combination of a header and an initial vector (IV) in the standard context for authenticated encryption (e.g., see [4]). In ANSI C12.22, the uniqueness of N is guaranteed by including time information with a specific format. The plaintext P can be the null string NULL, and in this case EAX′ works as a message authentication code for the corresponding N.

We first define the following functions. Here $N, S \in \{0,1\}^*$ and $t, D, Q \in \{0,1\}^n$.

[6] A somewhat similar scenario occurs in TLS, where the attack is possible if the authentication tag is short [8].

Pad function:

function $\textsc{Pad}(S, D, Q)$
  if $\text{size}(S) > 0$ **and** $\text{size}(S) \bmod n = 0$ **then return** $S \oplus_{\text{end}} D$
  else return $(S \| 10^{n-1-(\text{size}(S) \bmod n)}) \oplus_{\text{end}} Q$
  end if
end function

CBC′ function:

function $\text{CBC}'_K(t, S)$
  Let $S_1, \ldots, S_m \leftarrow S$ where $\text{size}(S_i) = n$
  $C_0 \leftarrow t$
  for $i \leftarrow 1$ **to** m **do** $C_i \leftarrow E_K(S_i \oplus C_{i-1})$
  end for
  return C_m
end function

CMAC′ function:

function $\text{CMAC}'_K(t, S, D, Q)$
  return $\text{CBC}'_K(t, \textsc{Pad}(S, D, Q))$
end function

CTR′ function:

function $\text{CTR}'_K(N, S)$
  $m \leftarrow \lceil \text{size}(S)/n \rceil$
  $\text{Ctr} \leftarrow N \wedge (1^{n-32} \| 01^{15} \| 01^{15})$
  $\text{Keystream} \leftarrow E_K(\text{Ctr}) \| E_K(\text{Ctr}+1) \| \cdots \| E_K(\text{Ctr}+m-1)$
  return $S \oplus \text{MSB}_{\text{size}(S)}(\text{Keystream})$
end function

Using these functions, the encryption of EAX′ is defined as follows. Decryption of EAX′ is straightforward and omitted here.

function $\text{EAX}'.\text{encrypt}_K(N, P)$
  $D \leftarrow \text{dbl}(E_K(0^n))$
  $Q \leftarrow \text{dbl}(D)$
  $\underline{N} \leftarrow \text{CMAC}'_K(D, N, D, Q)$
  if $\text{size}(P) = 0$ **then** $C \leftarrow \text{NULL}$
  else $C \leftarrow \text{CTR}'_K(\underline{N}, P)$
  end if
  $T \leftarrow \underline{N} \oplus \text{CMAC}'_K(Q, C, D, Q)$
  $T \leftarrow \text{MSB}_{32}(T)$
  return (C, T)
end function

Here, $\mathrm{dbl}(x)$ with $\mathrm{size}(x) = n$ denotes the "doubling" operation over $\mathrm{GF}(2^n)$, as defined by [2, 7].

In the above definition, we fixed an apparent error in line 72 of the definition of $\mathrm{EAX}'.\mathrm{encrypt}_K$ in [1, 7]. Some editorial errors of [7] were also pointed out by [3].

3 Forgeries

Chosen Message Existential Forgeries

The standard setting for cryptographic forgery attacks is as follows:

- The adversary asks $q \in \{0, 1, 2, \ldots\}$ queries (N_1, P_1), $\ldots$, (N_q, P_q) to the encryption oracle, receiving the corresponding ciphertext/tag pairs (C_i, T_i).
- The adversary queries the verification oracle for some (N, C, T) where no $i \in \{1, \ldots, q\}$ exists with $N = N_i$ and $C = C_i$ and $T = T_i$.
- The adversary succeeds if the (N, C, T) passes the verification.

For $x, y \in \{0,1\}^*$ with $\mathrm{size}(x) \geq \mathrm{size}(y)$, let $x \oplus_{\mathrm{start}} y = x \oplus \big(y \| (0^{\mathrm{size}(x)-\mathrm{size}(y)})\big)$. Then, the definition of CMAC' in the previous section conforms to that

$$\mathrm{CMAC}'_K(t, S, D, Q) = \begin{cases} \mathrm{CMAC}_K(S \oplus_{\mathrm{start}} t) & \text{if } \mathrm{size}(S) > n, \\ E_K(t \oplus Q \oplus (S \| 10^{n-1-\mathrm{size}(S)})) & \text{if } 0 \leq \mathrm{size}(S) < n, \\ E_K(t \oplus D \oplus S) & \text{if } \mathrm{size}(S) = n, \end{cases}$$

where CMAC_K is as defined by [2] (or OMAC1 of [6]) using E_K. Hence, we have

$$\mathrm{CMAC}'_K(D, S, D, Q) = E_K(S) \text{ when } \mathrm{size}(S) = n$$

and

$$\mathrm{CMAC}'_K(Q, S, D, Q) = E_K(S \| 10^{n-1-\mathrm{size}(S)}) \text{ when } 0 \leq \mathrm{size}(S) < n.$$

The above observation immediately implies the following forgery attacks:

Forgery attack 1 ($\mathrm{size}(N) = n$ and $\mathrm{size}(C) < n$)**.**

1. Prepare (N, C) that satisfies $\mathrm{size}(N) = n$ and $\mathrm{size}(C) < n$, such that $C \| 10^{n-1-\mathrm{size}(C)} = N$, and let $T = 0^{32}$.
2. Query (N, C, T) to the verification oracle.

This attack always succeeds as the "valid" tag for (N, C) is

$$\mathrm{MSB}_{32}(E_K(N) \oplus E_K(C \| 10^{n-1-\mathrm{size}(C)})) = 0^{32}.$$

Forgery attack 2 ($\mathrm{size}(N) < n$ and $\mathrm{size}(C) = n$)**.**

1. Prepare (N, C) that satisfies $\mathrm{size}(N) < n$ and $\mathrm{size}(C) = n$, such that $N \| 10^{n-1-\mathrm{size}(N)} = C$, and let $T = 0^{32}$.
2. Query (N, C, T) to the verification oracle.

The attack is again successful as the valid tag for (N, C) is $\mathrm{MSB}_{32}(E_K(D \oplus Q \oplus N \| 10^{n-1-\mathrm{size}(N)}) \oplus E_K(Q \oplus D \oplus C)) = 0^{32}$. These attacks use only one forgery attempt and no encryption query.

By using one encryption query the forgery attack is possible even when $\mathrm{size}(N) = n$ and $\mathrm{size}(C) = n$:

Forgery attack 3.

1. Query (N, P) with $\mathrm{size}(N) = \mathrm{size}(P) = n$ and $N \neq 0^n$ to the encryption oracle[7].
2. Obtain (C, T) (where $\mathrm{size}(C) = n$) from the oracle and see if $C \neq 0^n$ (quit if $C = 0^n$).
3. Query $(\widehat{N}, \widehat{C}, \widehat{T})$ to the verification oracle, where $\mathrm{size}(\widehat{N}) < n$, $\mathrm{size}(\widehat{C}) < n$, $\widehat{N} \| 10^{n-1-\mathrm{size}(\widehat{N})} = C$, $\widehat{C} \| 10^{n-1-\mathrm{size}(\widehat{C})} = N$, and $\widehat{T} = T$.

The above attack is almost always successful; unless $C = 0^n$ we have $T = \mathrm{MSB}_{32}(E_K(N) \oplus E_K(Q \oplus D \oplus C))$ and the valid tag for $(\widehat{N}, \widehat{C})$ is

$$\begin{aligned}&\mathrm{MSB}_{32}(E_K(D \oplus Q \oplus \widehat{N} \| 10^{n-1-\mathrm{size}(\widehat{N})}) \oplus E_K(Q \oplus Q \oplus \widehat{C} \| 10^{n-1-\mathrm{size}(\widehat{C})}))\\ &= \mathrm{MSB}_{32}(E_K(D \oplus Q \oplus C) \oplus E_K(N)),\end{aligned}$$

thus equals to T.

Forgery attack 4.

1. Query (N, P) with $\mathrm{size}(N) < n$ and $\mathrm{size}(P) < n$ to the encryption oracle.
2. Obtain (C, T) (where $\mathrm{size}(C) = \mathrm{size}(P) < n$) from the oracle.
3. Query $(\widehat{N}, \widehat{C}, \widehat{T})$ to the verification oracle, where $\mathrm{size}(\widehat{N}) = \mathrm{size}(\widehat{C}) = n$, $\widehat{N} = C \| 10^{n-1-\mathrm{size}(C)}$, $\widehat{C} = N \| 10^{n-1-\mathrm{size}(N)}$, and $\widehat{T} = T$.

This is the converse of Forgery attack 3; we have $T = \mathrm{MSB}_{32}(E_K(D \oplus Q \oplus N \| 10^{n-1-\mathrm{size}(N)}) \oplus E_K(Q \oplus Q \oplus C \| 10^{n-1-\mathrm{size}(C)}))$ and the valid tag for $(\widehat{N}, \widehat{C})$ is

$$\begin{aligned}&\mathrm{MSB}_{32}(E_K(D \oplus D \oplus \widehat{N}) \oplus E_K(Q \oplus D \oplus \widehat{C}))\\ &= \mathrm{MSB}_{32}(E_K(C \| 10^{n-1-\mathrm{size}(C)}) \oplus E_K(Q \oplus D \oplus N \| 10^{n-1-\mathrm{size}(N)})) = T.\end{aligned}$$

[7] Here we assume the adversary is allowed to query any (N, P) as long as N is unique; that is, nonce (here N)-respecting adversary introduced by Rogaway [9].

From Existential to Partially Selective Forgeries

A forgery is *selective* instead of *existential*, if the adversary can determine the content of the message to be forged.

Since EAX′ provides *authenticated encryption with associated data* (AEAD), the content of the message consists of both the confidential plaintext P and the non-confidential associated data (or "cleartext") N. While the above attacks don't allow to choose the plaintext, the adversary can arbitrarily choose the cleartext N (restricted to $\text{size}(N) \leq n$ and, for $\text{size}(N) = n$, $N \neq 0^n$). In this sense, the forgery attacks above are *partially selective*.

4 Chosen Plaintext Distinguishers

The forgery attacks above were based on the idea of generating cleartexts N and ciphertexts C, such that the authentication tag T is zero, i.e., $T = 0^{32}$. To distinguish EAX′ from a random encryption process one can apply a similar idea: Generate cleartexts N and plaintexts P such that the EAX′ encryption will generate any ciphertext C and a zero authentication tag T.

Distinguishing attack 1.

1. Query (N, P) to the encryption oracle, where $N = 10^{n-1}$ (thus $\text{size}(N) = n$) and $P = \text{NULL}$ (thus $\text{size}(P) = 0$).
2. Obtain T from the oracle.
3. If $T = 0^{32}$ then return 1, otherwise return 0.

As EAX′ returns $T = 0^{32}$ with probability 1 while the same event occurs with probability $1/2^{32}$ with a random encryption process, this enables us to easily distinguish T from random with the distinguishing advantage almost 1, using only one chosen plaintext query.

Distinguishing attack 2.

1. Fix small $1 \leq i \leq n-1$ and any $P \in \{0,1\}^i$, and query (N, P) to the encryption oracle with $N = P \| 10^{n-1-\text{size}(P)}$ (thus $\text{size}(N) = n$).
2. Obtain (C, T) from the oracle.
3. If $C = P$ and $T = 0^{32}$ then return 1, otherwise return 0.

In this case, we have $C = P$ with probability $1/2^i$ for both EAX′ and a random encryption process. Given the event $C = P$, we have

$$T = \text{MSB}_{32}(E_K(N) \oplus E_K(C \| 10^{n-1-\text{size}(C)})) = 0^{32}$$

with probability 1 for EAX′, while $T = 0^{32}$ occurs with probability $1/2^{32}$ for the random encryption process. Thus the distinguisher succeeds with probability $1/2^i$, which is non-negligible when i is small.

5 Chosen-Ciphertext Message Recovery Attacks

Consider a triple (N^*, C^*, T^*) of cleartext N^*, ciphertext C^* and tag T^*. The corresponding plaintext P^* is unknown. The adversary can ask a decryption oracle, for the decryption of any (N, C, T) under its choice, except for $(N, C, T) = (N^*, C^*, T^*)$ (otherwise, finding P^* would be trivial). The adversary receives either an error message (if the tag T doesn't fit), or the decryption P of C. This is the setting in a *chosen ciphertext attack*. Below, we focus on *message recovery attacks*, where the adversary actually finds P^*.

We describe two message recovery attacks: The first for the $\text{size}(N^*) = n$, the second for $\text{size}(N^*) < n$.

Message recovery attack 1 $(\text{size}(N^*) = n)$**.**

1. Require $\text{size}(N^*) = n$
2. Prepare C with $\text{size}(C) < n$ and with $C \| 10^{n-1-\text{size}(C)} = N^*$.
3. Query $(N^*, C, 0^{32})$ to the decryption oracle. Let P be the answer.
4. Compute the keystream $K(N^*) = C \oplus P \in \{0,1\}^{\text{size}(C)}$.

Since the decryption of (N^*, C^*, T^*) uses the same keystream, we now can compute the first $\text{size}(C)$ bits of P^* (or the full P^* if $\text{size}(P^*) \leq \text{size}(C)$). It succeeds for the same reason as Forgery attack 1 (unless $C^* \| 10^{n-1-\text{size}(C^*)} = N^*$ and $T^* = 0^{32}$, in which case the decryption query in Step 3 makes the attack trivial).

Message recovery attack 2 $(\text{size}(N^*) < n)$**.**

1. Require $\text{size}(N^*) < n$
2. Prepare C with $\text{size}(C) = n$ and $N^* \| 10^{n-1-\text{size}(N^*)} = C$.
3. Query $(N^*, C, 0^{32})$ to the decryption oracle. Let P be the answer.
4. The keystream is $K(N^*) = C \oplus P \in \{0,1\}^n$.

Unless $N^* \| 10^{n-1-\text{size}(N^*)} = C^*$ and $T^* = 0^{32}$, the attack succeeds for the same reason as Forgery attack 2.

6 Remarks

As our attacks demonstrate, there is a gap between the security of the original EAX and EAX′, and it is not possible to prove the security of EAX′ as a general-purpose authenticated encryption.

6.1 Differences between EAX′ and the Original EAX

Let $\mathrm{CMAC}_K[V](*)$ be the CMAC using an n-bit blockcipher E_K with initial block mask $V \in \{0,1\}^n$. That is, for $P \in \{0,1\}^*$ we have

$$\mathrm{CMAC}_K[V](P) = \begin{cases} E_K(V \oplus P \| 10^{n-1-\mathrm{size}(P)} \oplus \mathrm{dbl}(\mathrm{dbl}(L))) & \text{if } \mathrm{size}(P) < n \\ \mathrm{CMAC}_K(P \oplus_{\mathrm{start}} V) & \text{if } \mathrm{size}(P) \geq n, \end{cases}$$

where $L = E_K(0^n)$ and CMAC_K is the standard CMAC [2] using E_K. Note that if $V = E_K(t)$ we have $\mathrm{CMAC}_K[V](P) = \mathrm{CMAC}_K(t \| P)$.

The major differences of EAX′ from EAX are summarized as follows[8].

1. Role of N. Inputs to the EAX′ encryption function (except for the key) consist of a "cleartext" N and a plaintext P, whereas those to the original EAX consist of a nonce N, a header H, and a plaintext P. EAX′ requires N to be unique, hence it works as a nonce. EAX′ does not explicitly define a header H: information corresponding to the header is included in the cleartext N.
2. Tweaking method for CMAC. For the original EAX, the tag is (the first τ bits of) the xor of $\mathrm{CMAC}_K[L](N)$ and $\mathrm{CMAC}_K[W](H)$ and $\mathrm{CMAC}_K[Z](C)$ with $L = E_K(0^n)$, $W = E_K(0^{n-1}\|1)$, and $Z = E_K(0^{n-2}\|10)$. Hence it uses three tweaked variants of CMAC.
   EAX′ uses two different CMAC variants. In EAX′ the tag is the first 32 bits of $\mathrm{CMAC}_K[\mathrm{dbl}(L)](N) \oplus \mathrm{CMAC}_K[\mathrm{dbl}(\mathrm{dbl}(L))](C)$, where $L = E_K(0^n)$.
3. Counter mode incrementation. The original EAX uses $\mathrm{CMAC}_K[0^n](N)$ as an initial counter block for CTR mode, while the initial counter block of EAX′ is $\mathrm{CMAC}_K[\mathrm{dbl}(L)](N) \wedge (1^{n-32}\|01^{15}\|01^{15})$.

The third item increases the collision probability of counter blocks, which leads to a small degradation in security (as mentioned by the authors of EAX′ [1,7]).

6.2 Applicability of our Attacks to the Original EAX

We stress that our attacks against EAX′ do not apply to the original EAX.[9]

By the second item in the above list of differences, EAX′ uses the same constants D and Q as tweaks that CMAC uses internally. The original EAX uses two independent constants L and W at that place. This has three consequences:

1. Compared to an efficient implementation of the original EAX, an efficient implementation of EAX′ saves the RAM to store L and W ($2 * 16$ bytes if the blockcipher is the AES).
2. The security proof for EAX does not apply to EAX′ any more. (We believe one could adapt the proof of security for the original EAX to cope with the other differences between EAX and EAX′, with slightly weaker bounds.)
3. Our attacks become possible.

[8] For other minor differences, see Sect. 3 of [7].

[9] How could they? The original EAX has been rigorously proven to resist such attacks.

6.3 Applicability to the ANSI C12.22 Protocol

All our attacks presented here require size(N) $\leq n$. The forgery and distinguishing attacks also require size(P) $\leq n$, and the message recovery attacks actually require at most the first n bits of the ciphertext. In addition, the forgery and message recovery attacks could not be prevented by restricting the input length at encryption: one must implement the input length check at decryption as well.

Specifically, all our attacks would fail if size(N) $> n$ would be guaranteed. We do not know whether this actually holds for the protocol specified by ANSI C12.22 [1]. One can find some ANSI C12.22 communication examples (Annex G of [1]) or test vectors of EAX′ (Section V of [7]). Among them, there is no example[10] satisfying size(N) $\leq n$ ($= 128$), however, some test vectors of [7] satisfy size(C) $= n$.

In [7], "Justification" of Issue 6 (in page 3) states that "The CMAC′ computations here always involve CBC of at least two blocks". This looks odd since P or C can be null (as stated by ANSI) and CMAC′ taking the null string certainly operates on the single-block CBC, but it may be a hint that size(N) $> n$ would hold for any legitimate ANSI C12.22 messages.

6.4 Practical Implications

Attacks as those described in the current paper are often turned down by non-cryptographers as "only theoretical" or "don't apply in practice".

Indeed, none of our attacks is applicable if the cleartext size exceeds 128 bits. But even if the ANSI C12.22 prohibited any cleartexts of size 128 bits or shorter, including EAX′ in the standard would be like an unexploded bomb – waiting to go off any time in the future. Remember that EAX′ is intended for smart grids, i.e., for the use in dedicated industrial systems such as electrical meters, controllers and appliances. It hardly seems reasonable to assume that *every* device will *always* carefully check cleartexts and plaintexts for validity and plausibility. Also, vendors may be tempted to implement their own nonstandard extensions avoiding "unnecessarily long" texts.

For a non-cryptographer, assuming a "decryption oracle" may seem strange – if there were such an oracle, why bother with message recovery attacks at all? However, experience shows that such theoretical attacks are often practically exploitable. For example, some error messages return the input that caused the error: "Syntax error in 'xyzgarble'." Even if the error message does not transmit the entire fake plaintext, any error message telling the attacker whether the fake message followed some syntactic conventions or not is potentially useful for the attacker. See [5] for an early example.

Also note that our forgery attacks allow a malicious attacker to create a large number of messages with given single-block cleartexts and random single-block plaintexts, that appear to come from a trusted source, because the authentication

[10] In the test vectors of [7], there seems an editorial error; the cleartext may mean the plaintext and vice versa.

succeeded. What the actual devices will do when presented with apparently valid random commands is a source of great speculation.

6.5 Provable Security and Open Problem

In modern cryptography the standard approach for the designers of new block-cipher-based schemes is to formally *prove* the security of their schemes, assuming that the underlying blockcipher is secure. The authors of the original EAX did follow that approach.

As the current paper shows, even seemingly minor modifications of a scheme that has been proven secure need attention – the original security proof may not be applicable any more, and the modified scheme may be broken or not. Put simply, whenever you modify an existing (provably secure) scheme, how marginal the modifications appear to be, you must revisit the entire security proof for the existing scheme and check if all security arguments still hold for the modified one. Or else, don't modify existing schemes!

Finally, we remark that it is an open problem to prove or disprove the security of EAX′ if size$(N) > n$.

Acknowledgments. This paper is based on the collaboration started at Dagstuhl Seminar 12031, Symmetric Cryptography. The authors thank participants of the seminar for useful comments, and discussions with Greg Rose were invaluable for writing Sect. 6.4. We also thank Mihir Bellare and Jeffrey Walton for feedback, and thank the reviewers of DIAC for useful comments. The work by Tetsu Iwata was supported by MEXT KAKENHI, Grant-in-Aid for Young Scientists (A), 22680001.

References

1. American National Standards Institute. American National Standard Protocol Specification For Interfacing to Data Communication Networks. ANSI C12.22-2008.
2. National Institute of Standards and Technology. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B, 2005.
3. Toshiba Corporation. Comment for EAX' Cipher Mode. Available from `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/EAX%27/Toshiba_Report2NIST_rev051.pdf`.
4. M. Bellare, P. Rogaway, and D. Wagner. The EAX Mode of Operation. *Fast Software Encryption, FSE 2004*, LNCS 3017, pp. 389–407, Springer, 2004.
5. D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. *CRYPTO '98*, LNCS 1462, pp. 1–12, Springer, 1998.
6. T. Iwata and K. Kurosawa. OMAC: One-Key CBC MAC. *Fast Software Encryption, FSE 2003*, LNCS 2887, pp. 129–153, Springer, 2003.
7. A. Moise, E. Beroset, T. Phinney, and M. Burns. EAX' Cipher Mode (May 2011). NIST Submission. Available from `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/eax-prime/eax-prime-spec.pdf`.

8. K. G. Paterson, T. Ristenpart, and T. Shrimpton. Tag Size Does Matter: Attacks and Proofs for the TLS Record Protocol. *ASIACRYPT 2011*, LNCS 7073, pp. 372–389, Springer, 2011.
9. P. Rogaway. Nonce-Based Symmetric Encryption. *Fast Software Encryption, FSE 2004*, LNCS 3017, pp. 348–359, Springer, 2004.

SipHash: a fast short-input PRF

Jean-Philippe Aumasson[1] and Daniel J. Bernstein[2]

[1] NAGRA
Switzerland
jeanphilippe.aumasson@gmail.com
[2] Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607–7045, USA
djb@cr.yp.to

Abstract. SipHash is a family of pseudorandom functions optimized for short inputs. Target applications include network traffic authentication and hash-table lookups protected against hash-flooding denial-of-service attacks. SipHash is simpler than MACs based on universal hashing, and faster on short inputs. Compared to dedicated designs for hash-table lookup, SipHash has well-defined security goals and competitive performance. For example, SipHash processes a 16-byte input with a fresh key in 140 cycles on an AMD FX-8150 processor, which is much faster than state-of-the-art MACs. We propose that hash tables switch to SipHash as a hash function.

1 Introduction

A message-authentication code (MAC) produces a tag t from a message m and a secret key k. The security goal for a MAC is for an attacker, even after seeing tags for many messages (perhaps selected by the attacker), to be unable to guess tags for any other messages.

Internet traffic is split into short packets that require authentication. A 2000 note by Black, Halevi, Krawczyk, Krovetz, and Rogaway [8] reports that "a fair rule-of-thumb for the distribution on message-sizes on an Internet backbone is that roughly one-third of messages are 43 bytes (TCP ACKs), one-third are about 256 bytes (common PPP dialup MTU), and one-third are 1500 bytes (common Ethernet MTU)."

However, essentially all standardized MACs and state-of-the-art MACs are optimized for long messages, not for short messages. Measuring long-message performance hides the overheads caused by large MAC keys, MAC initialization, large MAC block sizes, and MAC finalization. These overheads are usually quite severe, as illustrated by the examples in the following paragraphs. Applications can compensate for these overheads by authenticating a concatenation of several packets instead of authenticating each packet separately, but then a single forged

This work was supported by the National Science Foundation under grant 1018836. Permanent ID of this document: b9a943a805fbfc6fde808af9fc0ecdfa. Date: 2012.06.20.

packet forces several packets to be retransmitted, increasing the damage caused by denial-of-service attacks.

Our first example is HMAC-SHA-1, where overhead effectively adds between 73 and 136 bytes to the length of a message: for example, HMAC-SHA-1 requires two 64-byte compression-function computations to authenticate a short message. Even for long messages, HMAC-SHA-1 is not particularly fast: for example, the OpenSSL implementation takes 7.8 cycles per byte on Sandy Bridge, and 11.2 cycles per byte on Bulldozer. In general, building a MAC from a general-purpose cryptographic hash function appears to be a highly suboptimal approach: general-purpose cryptographic hash functions perform many extra computations for the goal of collision resistance on public inputs, while MACs have secret keys and do not need collision resistance.

Much more efficient MACs combine a large-input universal hash function with a short-input encryption function. A universal hash function h maps a long message m to a short hash $h(k_1, m)$ under a key k_1. "Universal" means that any two different messages almost never produce the same output when k_1 is chosen randomly; a typical universal hash function exploits fast 64-bit multipliers to evaluate a polynomial over a prime field. This short hash is then strongly encrypted under a second key k_2 to produce the authentication tag t. The original Wegman–Carter MACs [27] used a one-time pad for encryption, but of course this requires a very long key. Modern proposals such as UMAC version 2 [8], Poly1305-AES [3], and VMAC(AES) [20] [10] replace the one-time pad with outputs of AES-128: i.e., $t = h(k_1, m) \oplus \mathrm{AES}(k_2, n)$ where n is a nonce. UMAC version 1 argued that "using universal hashing to reduce a very long message to a fixed-length one can be complex, require long keys, or reduce the quantitative security" [7, Section 1.2] and instead defined $t = \text{HMAC-SHA-1}(h(k, m), n)$ where $h(k, m)$ is somewhat shorter than m.

All of these MACs are optimized for long-message performance, and suffer severe overheads for short messages. For example, the short-message performance of UMAC version 1 is obviously even worse than the short-message performance of HMAC-SHA-1. All versions of UMAC and VMAC expand k_1 into a very long key (for example, 4160 bytes in one proposal), and are timed under the questionable assumptions that the very long key has been precomputed and preloaded into L1 cache. Poly1305-AES does not expand its key but still requires padding and finalization in h, plus the overhead of an AES call.

(We comment that, even for applications that emphasize long-message performance, the structure of these MACs often significantly complicates deployment. Typical universal MACs have lengthy specifications, are not easy to implement efficiently, and are not self-contained: they rely on extra primitives such as AES. Short nonces typically consume 8 bytes of data with each tag, and force applications to be stateful to ensure uniqueness; longer nonces consume even more space and require either state or random-number generation. There have been proposals of nonceless universal MACs, but those proposals are significantly slower than other universal MACs at the same security level.)

The short-input performance problems of high-security MACs are even more clear in another context. As motivation we point to the recent rediscovery of "hash flooding" denial-of-service attacks on Internet servers that store data in hash tables. These servers normally use public non-cryptographic hash functions, and these attacks exploit multicollisions in the hash functions to enforce worst-case lookup time. See Section 7 of this paper for further discussion.

Replacing the public non-cryptographic hash functions with strong small-output secret-key MACs would solve this problem. However, to compete with existing non-cryptographic hash functions, the MACs must be extremely fast for very short inputs, even shorter than the shortest common Internet packets. For example, Ruby on Rails applications are reported to hash strings shorter than 10 bytes on average. Recent hash-table proposals such as Google's CityHash [13] and Jenkins' SpookyHash [16] provide very fast hashing of short strings, but these functions were designed to have a close-to-uniform distribution, not to meet any particular cryptographic goals. For example, collisions were found in an initial version of CityHash128 [17], and the current version is vulnerable to a practical key-recovery attack when 64-bit keys are used.

This paper introduces the SipHash family of hash functions to address the needs for high-security short-input MACs. SipHash features include:

- **High security.** Our concrete proposal SipHash-2-4 was designed and evaluated to be a cryptographically strong PRF (pseudorandom function), i.e., indistinguishable from a uniform random function. This implies its strength as a MAC.
- **High speed.** SipHash-2-4 is much faster for short inputs than previous strong MACs (and PRFs), and is competitive in speed with popular non-cryptographic hash functions.
- **Key agility.** SipHash uses a 128-bit key. There is no key expansion in setting up a new key or hashing a message, and there is no hidden cost of loading precomputed expanded keys from DRAM into L1 cache.
- **Simplicity.** SipHash iterates a simple round function consisting of four additions, four xors, and six rotations, interleaved with xors of message blocks.
- **Autonomy.** No external primitive is required.
- **Small state.** The SipHash state consists of four 64-bit variables. This small state size allows SipHash to perform well on a wide range of CPUs and to fit into small hardware.
- **No state between messages.** Hashing is deterministic and doesn't use nonces.
- **Minimal overhead.** Authenticated messages are just 8 bytes longer than original messages.

§2 presents a complete definition of SipHash; §3 makes security claims; §4 explains some design choices; §5 reports on our preliminary security analysis; §6 evaluates the efficiency of SipHash in software and hardware; §7 discusses the benefits of switching to SipHash for hash-table lookups.

2 Specification of SipHash

SipHash is a family of PRFs SipHash-c-d where the integer parameters c and d are the number of compression rounds and the number of finalization rounds. A compression round is identical to a finalization round and this round function is called SipRound. Given a 128-bit key k and a (possibly empty) byte string m, SipHash-c-d returns a 64-bit value SipHash-c-$d(k, m)$ computed as follows:

1. **Initialization:** Four 64-bit words of internal state v_0, v_1, v_2, v_3 are initialized as

$$v_0 = k_0 \oplus \texttt{736f6d6570736575}$$
$$v_1 = k_1 \oplus \texttt{646f72616e646f6d}$$
$$v_2 = k_0 \oplus \texttt{6c7967656e657261}$$
$$v_3 = k_1 \oplus \texttt{7465646279746573}$$

   where k_0 and k_1 are the little-endian 64-bit words encoding the key k.
2. **Compression:** SipHash-c-d processes the b-byte string m by parsing it as $w = \lceil (b+1)/8 \rceil > 0$ 64-bit little-endian words $m_0, \ldots, m_{w-1}$ where m_{w-1} includes the last 0 through 7 bytes of m followed by null bytes and ending with a byte encoding the positive integer b mod 256. For example, the one-byte input string $m = \texttt{ab}$ is parsed as $m_0 = \texttt{0100000000000000ab}$. The m_i's are iteratively processed by doing

$$v_3 \oplus = m_i$$

   and then c iterations of SipRound, followed by

$$v_0 \oplus = m_i$$

3. **Finalization:** After all the message words have been processed, SipHash-c-d xors the constant $\texttt{ff}$ to the state:

$$v_2 \oplus = \texttt{ff}$$

   then does d iterations of SipRound, and returns the 64-bit value

$$v_0 \oplus v_1 \oplus v_2 \oplus v_3 \ .$$

Fig. 2.1 shows SipHash-2-4 hashing a 15-byte m.

The function SipRound transforms the internal state as follows (see also Fig.2.2):

$v_0 + = v_1$	$v_2 + = v_3$
$v_1 \lll = 13$	$v_3 \lll = 16$
$v_1 \oplus = v_0$	$v_3 \oplus = v_2$
$v_0 \lll = 32$	
$v_2 + = v_1$	$v_0 + = v_3$
$v_1 \lll = 17$	$v_3 \lll = 21$
$v_1 \oplus = v_2$	$v_3 \oplus = v_0$
$v_2 \lll = 32$	

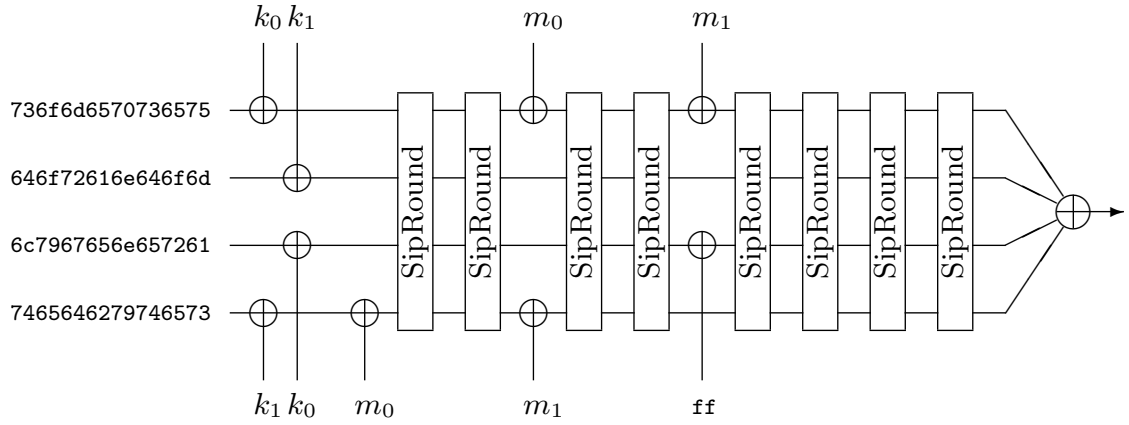


Fig. 2.1. SipHash-2-4 processing a 15-byte message. SipHash-2-4(k, m) is the output from the final $\oplus$ on the right.

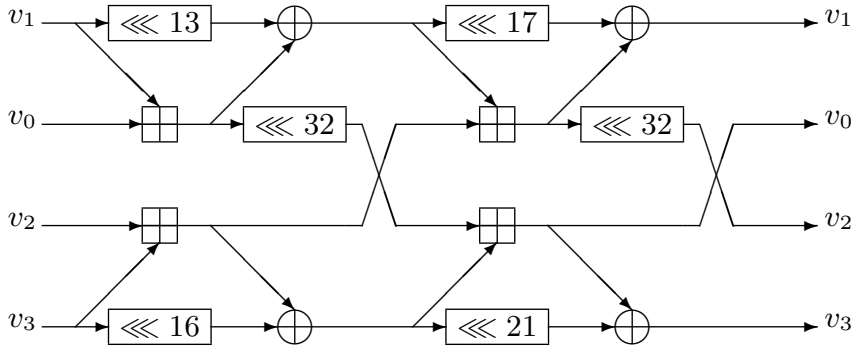


Fig. 2.2. The ARX network of SipRound.

3 Expected strength

SipHash-c-d with $c \geq 2$ and $d \geq 4$ is expected to provide the maximum PRF security possible (and therefore also the maximum MAC security possible) for any function with the same key size and output size. Our fast proposal is thus SipHash-2-4. We define SipHash-c-d for larger c and d to provide a higher security margin: our conservative proposal is SipHash-4-8, which is about half the speed of SipHash-2-4. We define SipHash-c-d for smaller c and d to provide targets for cryptanalysis. Cryptanalysts are thus invited to break

- SipHash-1-0, SipHash-2-0, SipHash-3-0, SipHash-4-0, etc.;
- SipHash-1-1, SipHash-2-1, SipHash-3-1, SipHash-4-1, etc.;
- SipHash-1-2, SipHash-2-2, SipHash-3-2, SipHash-4-2, etc.;

and so on.

Note that the standard PRF and MAC security goals allow the attacker access to the output of SipHash on messages chosen adaptively by the attacker. However, they do not allow access to any "leaked" information such as bits of

the key or the internal state. They also do not allow "related keys", "known keys", "chosen keys", etc.

Of course, security is limited by the key size (128 bits). In particular, attackers searching 2^s keys have chance 2^{s-128} of finding the SipHash key. This search is accelerated in standard ways by speedups in evaluation and partial evaluation of SipHash, for one key or for a batch of keys; by attacks against multiple targets; and by quantum computers.

Security is also limited by the output size (64 bits). In particular, when SipHash is used as a MAC, an attacker who blindly tries 2^s tags will succeed with probability 2^{s-64}.

We comment that SipHash is not meant to be, and (obviously) is not, collision-resistant.

4 Rationale

SipHash is an ARX algorithm, like the SHA-3 finalists BLAKE and Skein. SipHash follows BLAKE's minimalism (small code, small state) but borrows the two-input MIX from Skein, with two extra rotations to improve diffusion. SipHash's input injection is inspired by another SHA-3 finalist, JH.

Choice of constants. The initial state constant corresponds to the ASCII string "somepseudorandomlychosenbytes". There is nothing special about this value; the only requirement was some asymmetry so that the initial v_0 and v_1 differ from v_2 and v_3. This constant may be set to a "personalization string" but we have not evaluated whether it can safely be chosen as a "tweak".

The other constant in SipHash is `ff`, as xored to v_2 in finalization. We could have chosen any other non-zero value. Without this constant, one can reach the internal state after finalization by just absorbing null words. We found no way to exploit this property, but we felt it prudent to avoid it given the low cost of the defense.

Choice of rotation counts. Finding really bad rotation counts for ARX algorithms turns out to be difficult. For example, randomly setting all rotations in BLAKE-512 or Skein to a value in $\{8, 16, 24, \ldots, 56\}$ may allow known attacks to reach slightly more rounds, but no dramatic improvement is expected.

The advantage of choosing such "aligned" rotation counts is that aligned rotation counts are much faster than unaligned rotation counts on many non-64-bit architectures. Many 8-bit microcontrollers have only 1-bit shifts of bytes, so rotation by (e.g.) 3 bits is particularly expensive; implementing a rotation by a mere permutation of bytes greatly speeds up ARX algorithms. Even 64-bit systems can benefit from alignment, when a sequence of shift-shift-xor can be replaced by SSSE3's `pshufb` byte-shuffling instruction. For comparison, implementing BLAKE-256's 16- and 8-bit rotations with `pshufb` led to a 20% speedup on Intel's Nehalem microarchitecture.

For SipHash, the rotation distances were chosen as a tradeoff between security and performance, with emphasis on the latter. We ran an automated search that picks random rotation counts, estimates the number of significant statistical biases on three SipRounds with respect to a specific significance threshold, and finally sorts the sets of rotation counts according to that metric. We then manually shortlisted a few sets, by choosing the ones with rotation counts the closest to multiples of eight. We changed some of those values to the closest multiple of eight and benchmarked them against our original security metric, and repeated this process several times until finding a satisfying set of rotation counts.

We chose counts 13, 16, 17, and 21 for the rotations in the two MIX layers: 13 and 21 are three bits away from a multiple of 8, whereas 17 is just one bit away, and 16 can be realized by byte permutation only. We aggressively set the two "asymmetric" rotation counts to 32 to minimize the performance penalty—it is just a swap of words on 32-bit systems. The 32-bit rotations significantly improve diffusion, and their position on the ARX network allows for an efficient scheduling of instructions.

Choice of injection structure. Like JH, SipHash injects input before and after each block, with the difference that SipHash leaves less freedom to attackers: whereas JH xors the message block to the two halves of the state before and after the permutation, SipHash xors a block to two quarters of the state. Any attack on the SipHash injection structure can be applied to the JH injection structure, so security proofs for the JH injection structure [23] also apply to the SipHash injection structure.

A basic advantage of the JH/SipHash injection structure compared to the sponge/Keccak injection structure is that message blocks of arbitrary length (up to half the state) can be absorbed without reducing preimage security. A disadvantage is that each message block must be retained while the state is being processed, but for SipHash this extra storage is only a quarter of the state.

Choice of padding rule. SipHash's padding appends a byte encoding the message length modulo 256. We could have chosen a slightly simpler padding rule, such as appending a `80` byte followed by zeroes, as in CubeHash. However, our choice forces messages of different lengths modulo 256 to have different last blocks, which may complicate attacks on SipHash; the extra cost is negligible.

5 Preliminary cryptanalysis

We first consider attacks that are independent of the SipRound algorithm, and thus that are independent of the c and d parameters. We then consider attacks on SipRound iterations, with a focus on our proposal SipHash-2-4.

5.1 Generic attacks

Key-recovery. Brute force will recover a key after on average 2^{127} evaluations of SipHash, given two input/output pairs (one being insufficient to uniquely

identify the key). The optimal strategy is to work with 1-word padded messages, so that evaluating SipHash-c-d takes $c + d$ SipRounds.

State-recovery. A simple strategy to attack SipHash is to choose three input strings identical except for their last word, query for their respective SipHash outputs, and then "guess" the state that produced the output $v_0 \oplus v_1 \oplus v_2 \oplus v_3$ for one of the two strings. The attacker checks the 192-bit guessed value against the two other strings, and eventually recovers the key. On average $d2^{191}$ evaluations of SipRound are computed.

Internal collisions. As for any MAC with 256-bit internal state, internal collisions can be exploited to forge valid tags with complexity of the order of 2^{128} queries to SipHash. The padding of the message length forces attackers to search for collisions at the same position modulo 256 bytes.

5.2 Differential cryptanalysis

Truncated differentials. To assess the strength of SipRound, we applied the same techniques that were used [2] to attack Salsa20, namely a search for statistical biases in one or more bits of output given one or more differences in the input. We considered input differences in v_3 and sought biases in $v_0 \oplus v_1 \oplus v_2 \oplus v_3$ after iterating SipRound.

The best results were obtained by setting a 1-bit difference in the most significant bit of v_3. After three iterations of SipRound many biases are found. But after four or more iterations we did not detect any bias after experimenting with sets of 2^{30} samples.

To attempt to distinguish our fast proposal SipHash-2-4 by exploiting such statistical biases, one needs to find a bias on six rounds such that no input difference lies in the most significant byte of the last word (as this encodes the message length).

XOR-linearized characteristics. We considered an attacker who injects a difference in the first message word processed by SipHash-2-4, and then that guesses the difference in v_3 every two SipRounds in order to cancel it with the new message word processed. This ensures that at least a quarter of the internal state is free of difference when entering a new absorption phase. Note that such an omniscient attacker would require the leakage of v_3 every two SipRounds, and thus is not covered by our security claims in §3.

We used Leurent's ARX toolkit [21] to verify that our characteristics contain no obvious contradiction, and to obtain refined probability estimates. Table 5.1 shows the best characteristic we found: after two rounds there are 20 bit differences in the internal state, with differences in all four words. The message injection reduces this to 15 bit differences (with no difference in v_3), and after two more rounds there are 96 bit differences. The probability to follow this differential characteristic is estimated to be 2^{-134}. For comparison, Table 5.2 shows

Round	Differences				Prob.
1	.....	.....	.....	8.....	1 (1)
	.....	.....	8.....	8.....8...	
2	8.....8...	8.....	.....8.....	8.....1...1.8...	13 (14)
	....8.....	.....9...	8.....1.8.1.8...	8.1.....1.....	
3	..1.8.....1.....	8.....11a.1.1...	8.1.1...8.....1.	.....	33 (47)
	a...1...8.1.8.11	8.12b413a2.....	8.1.1...8.....1.	8.1.1...8.....1.	
4	2.1.....1.8..1	6825e.1322.1..35	22....1....2a413	2.....2..82.3	87 (134)
	22118.344835e.13	f4378453.2172d3.	.2....1..2.2261.	.2...21.8..1.61.	
5	a..1..24c834e4.3	fe918.6d5a74e34f	..15.b2.f6378443	.....	145 (279)
	924..74c5e9.8.49	6e9d2b.7.e29f89e	..15.b2.f6378443	..15.b2.f6378443	
6	9255.c6ca8a7.4.a	38863c74.922a1e7	f81e7cdd6e882.27	f64bca9c2.c7.6ab	160 (439)
	a185a5edaad33.18	6d5db13cf5b942fd	.e55b6414e4f268c	c4c9968648e4d.c7	

Table 5.1. For each SipRound, differences in v_0, v_1, v_2, v_3 before each half-round in the xor-linear model. Every two rounds a message word is injected that cancels the difference in v_3; the difference used is then xored to v_0 after the two subsequent rounds. The probability estimate is given for each round, with the cumulative value in parentheses.

the characteristic obtained with the same input difference, but for an attacker who does not guess the difference in v_3: the probability to follow four rounds of the characteristic is estimated to be 2^{-159}.

Better characteristics may exist. However we expect that finding (collections of) characteristics that both have a high probability and are useful to attack SipHash is extremely difficult. SipRound has as many additions as xors, so linearization with respect to integer addition seems unlikely to give much better characteristics than xor-linearization.

Vanishing characteristics. A particularly useful class of differential characteristics is that of vanishing characteristics: those start from a non-zero difference and yield an internal state with no difference, that is, an internal collision. Vanishing characteristics obviously do not exist for any iteration of SipRound; one has to consider characteristics for the function consisting of SipRound iterations followed by $v_0 \oplus = \Delta$, with an input difference Δ in v_3.

No vanishing characteristic exists for one SipRound, as a non-zero difference always propagates to v_2. We ensured that no vanishing xor-linear characteristic exists for iterations of two, three, or four SipRounds, by attempting to solve the corresponding linear system. For sequences of two words, we ensured that no sparse vanishing characteristic exists.

Other attacks. We briefly examine the applicability of other attacks to attack SipHash:

Round	Differences				Prob.
1	.....	.....	.....	8.....	1 (1)
	.....	.....	8.....	8.....8...	
2	8.....8...	8.....	.....8.....	8.....1...1.8...	13 (14)
	....8.....	.....9...	8.....1.8.1.8...	8.1.....1.....	
3	..1.8.....1.....	8.....11a.1.1...	8.1.1...8.....1.	8.1.82.....2..	42 (56)
	a...1...8.1.8.11	8.12b413a2.....	....92..8....21.	82..92..82..82..	
4	22..82...21..211	e835621322.1.235	22...21.8.122613	621.c21.42..42.3	103 (159)
	2.11..24ca35e.13	66778453..57bd22	4.1.c...c212641.	82..82..8.11.6..	
5	a21182244a24e613	2ec144fcb8.115dd	c245d93226674453	e2.18..48a34a6.3	152 (311)
	f225f3ce8cd.c6d8	a44f51d8d.9e5616	2.445936ac53e25.	a.4.d3.2.a5...51	
6	52652.cc868.c689	27baa9d2d.e.fcd8	7ccdb44684.b.8ee	32246acc8cb4ce93	187 (498)
	566.3a5175df891e	2.e5d3.249fb3ea6	4ee9de8a.8bfc67d	2425523ec62cf459	

Table 5.2. For each SipRound, differences in v_0, v_1, v_2, v_3 before each half-round in the xor-linear model. Every two rounds a message with no difference is injected. The probability estimate is given for each half-round, with the cumulative value in parentheses.

- Rotational attacks are differential attacks with respect to the rotation operator; see, e.g., [4, Section 4] and [18]. Due to the asymmetry in the initial state—at most half of the initial state can be rotation-invariant—rotational attacks are ineffective against SipHash.
- Cube attacks exploit a low algebraic degree in the primitive attacked. Due to the rapid growth of the degree in SipHash, as in other ARX primitives, cube attacks are unlikely to succeed.
- Rebound attacks are not known to be relevant for keyed primitives.
- Side-channel attacks cannot exploit any secret-dependent memory address or execution time. The cost of masking SipHash to protect against DPA etc. will be comparable to the cost of masking other ARX primitives.

5.3 Fixed point

Any iteration of SipRound admits a trivial distinguisher: the zero-to-zero fixed-point. This may make theoretical arguments based on the "ideal permutation" assumption irrelevant. But exploiting this property to attack SipHash seems very hard, for

1. Hitting the all-zero state, although easy to verify, is expected to be as hard as hitting any other predefined state;
2. The ability to hit a predefined state implies the ability to recover the key, that is, to completely break SipHash.

That is, the zero-to-zero fixed point cannot be a significant problem for SipHash, for if it were, SipHash would have much bigger problems.

6 Performance

6.1 Lower bounds for a 64-bit implementation

SipRound involves 14 64-bit operations, so SipHash-2-4 involves 30 64-bit operations for each 8 bytes of input, i.e., 3.75 operations per byte. A CPU core with 2 64-bit arithmetic units needs at least 1.875 cycles per byte for SipHash-2-4, and a CPU core with 3 64-bit arithmetic units needs at least 1.25 cycles per byte for SipHash-2-4. A CPU core with 4 64-bit arithmetic units needs at least 1 cycle per byte, since SipRound does not always have 4 operations to perform in parallel.

The cost of finalization cannot be ignored for short messages. For example, for an input of length between 16 and 23 bytes, a CPU core with 3 64-bit arithmetic units needs at least 49 cycles for SipHash-2-4.

6.2 Lower bounds for a 32-bit implementation

32-bit architectures are common in embedded systems, with for example processors of the ARM11 family implementing the ARMv6 architecture. To estimate SipHash's efficiency on ARMv6, we can directly adapt the analysis of Skein's performance by Schwabe, Yang, and Yang [24, §7], which observes that six 32-bit instructions are sufficient to perform a MIX transform. Since SipRound consists of four MIX transforms—the 32-bit rotate is transparent—we obtain 24 instructions per SipRound, that is, a lower bound of $3c$ cycles per byte for SipHash on long messages. This is 6 cycles per byte for SipHash-2-4. An input of length between 16 and 23 bytes needs at least 240 cycles.

6.3 Implementation results

We wrote a portable C implementation of SipHash, and ran preliminary benchmarks on three machines:

- "bulldozer", a Linux desktop equipped with a processor from AMD's last generation (FX-8150, 4×3600 MHz, "Zambezi" core), using `gcc 4.5.2`;
- "ishmael", a Linux laptop equipped with a processor from AMD's previous generation (Athlon II Neo Mobile, 1700 MHz, "Geneva" core), using `gcc 4.6.3`.
- "latour", a Linux desktop equipped with an older Intel processor (Core 2 Quad Q6600, 2394 MHz, "Kentsfield" core), using `gcc 4.4.3`.

We used compiler options `-O3 -fomit-frame-pointer -funroll-loops`. For more comprehensive benchmarks, we will submit implementations of SipHash to SUPERCOP.

On "bulldozer", our C implementation of SipHash-2-4 processes long messages at a speed of 1.96 cycles per byte. On "ishmael", SipHash-2-4 reaches 1.44 cycles per byte; this is due to the Athlon II's K10 microarchitecture having three ALUs, against only two for the more recent Bulldozer. Similar comments apply

Data byte length		8	16	24	32	40	48	56	64
"bulldozer"	Cycles	124	141	156	171	188	203	218	234
	Cycles per byte	15.50	8.81	6.50	5.34	4.70	4.23	3.89	3.66
"ishmael"	Cycles	123	134	145	158	170	182	192	204
	Cycles per byte	15.38	8.38	6.00	4.94	4.25	3.79	3.43	3.19
"latour"	Cycles	135	144	162	171	189	207	216	225
	Cycles per byte	16.88	10.29	6.75	5.34	4.50	4.31	3.86	3.52

Table 6.1. Speed measurements of SipHash-2-4 for short messages.

to "latour". These speeds are close to the lower bounds reported in §6.1, with respective gaps of approximately 0.10 and 0.20 cycles per byte.

Table 6.1 reports speeds on short messages. For comparison, the fastest SHA-3 finalist on "bulldozer" (BLAKE-512) takes approximately 1072 cycles to process 8 bytes, and 1280 cycles to process 64 bytes.

Figures 6.1, 6.2, and 6.3 compare our implementation of SipHash with the optimized C++ and C implementations of CityHash (version CityHash64) and SpookyHash (version ShortHash) on short messages, as well as with OpenSSL's MD5 implementation. Figure 6.4 presents a refined view of relative performance of SipHash-2-4, CityHash, and SpookyHash.

One can see from these tables and diagrams that SipHash-2-4 is extremely fast, and competitive with non-cryptographic hashes. For example, hashing 16 bytes takes 141 Bulldozer cycles with SipHash-2-4, against 82 and 126 for CityHash and SpookyHash, and 600 for MD5. Our conservative proposal SipHash-4-8 is still twice as fast as MD5.

6.4 Hardware efficiency

ASICs can integrate SipHash with various degrees of area/throughput tradeoffs, with the following as extreme choices:

- **Compact architecture** with a circuit for a half-SipRound only, that is, two 64-bit full adders, 128 xors, and two rotation selectors. For SipHash-c-d this corresponds a latency of $c/4$ cycles per byte plus $2d$ cycles for the finalization.
- **High-speed architecture** with a circuit for $e = \max(c, d)$ rounds, that is, $4e$ 64-bit full adders and $256e$ xors. For SipHash-c-d this corresponds to a latency of 1/8 cycle per byte plus one cycle for finalization.

Both architectures require 256 D-flip-flops to store the internal state, plus 64 for the message blocks. For a technology with 8 gate-equivalents (GE) per full adder, 3 per xor, and 7 per D-flip-flop, this is a total of approximately 3700 GE for the compact architecture of SipHash-2-4, and 13500 GE for the high-speed architecture. With the compact architecture a 20-byte message is hashed by SipHash-2-4

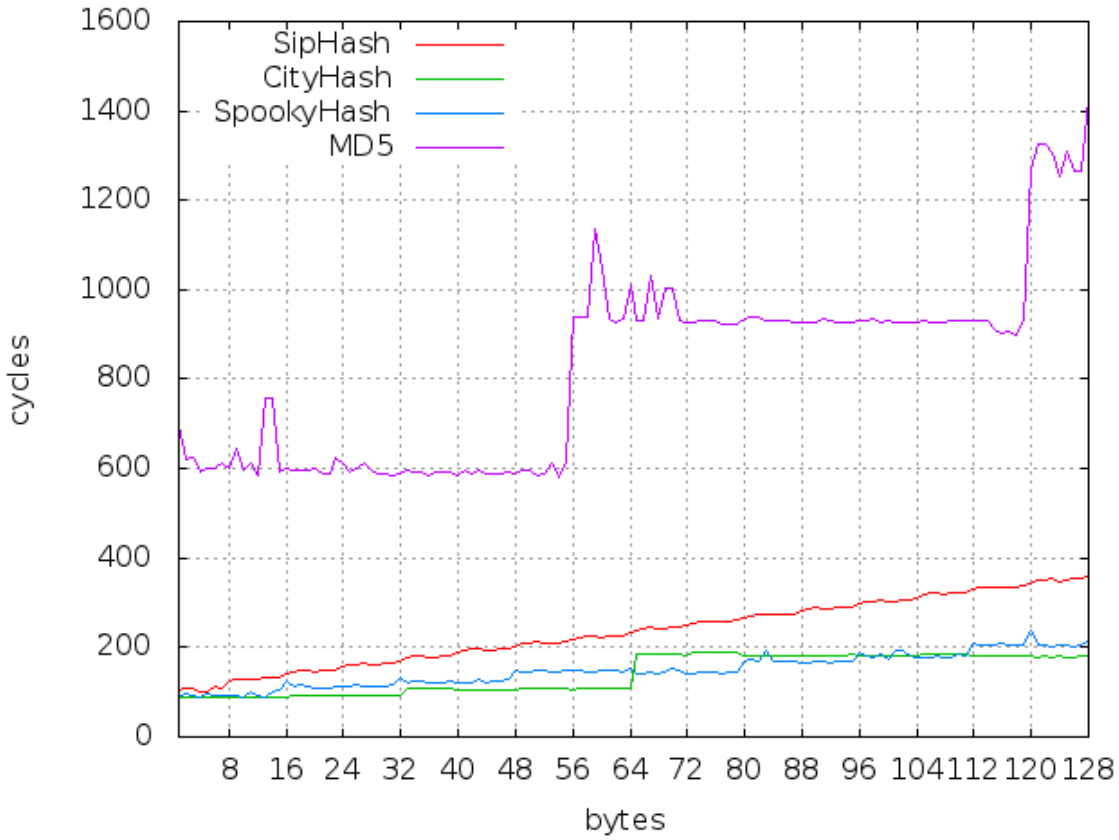


Fig. 6.1. Performance of SipHash-2-4 compared to non-cryptographic hash functions CityHash and SpookyHash and to MD5 on "bulldozer" (AMD FX-8150), for messages of $1, 2, \ldots, 128$ bytes. Curves on the right, from top to bottom, are MD5, SipHash, SpookyHash, and CityHash.

in 20 cycles, against 4 cycles with the high-speed architecture. An architecture implementing $c = 2$ rounds of SipHash-2-4 would take approximately 7900 GE to achieve a latency of 1/8 cycles per byte plus two cycles for finalization, thus 5 cycles to process 20 bytes.

7 Application: defense against hash flooding

We propose that hash tables switch to SipHash as a hash function. On startup a program reads a secret SipHash key from the operating system's cryptographic random-number generator; the program then uses SipHash for all of its hash tables. This section explains the security benefits of SipHash in this context.

The small state of SipHash also allows each hash table to have its own key with negligible space overhead, if that is more convenient. Any attacks must then be carried out separately for each hash table.

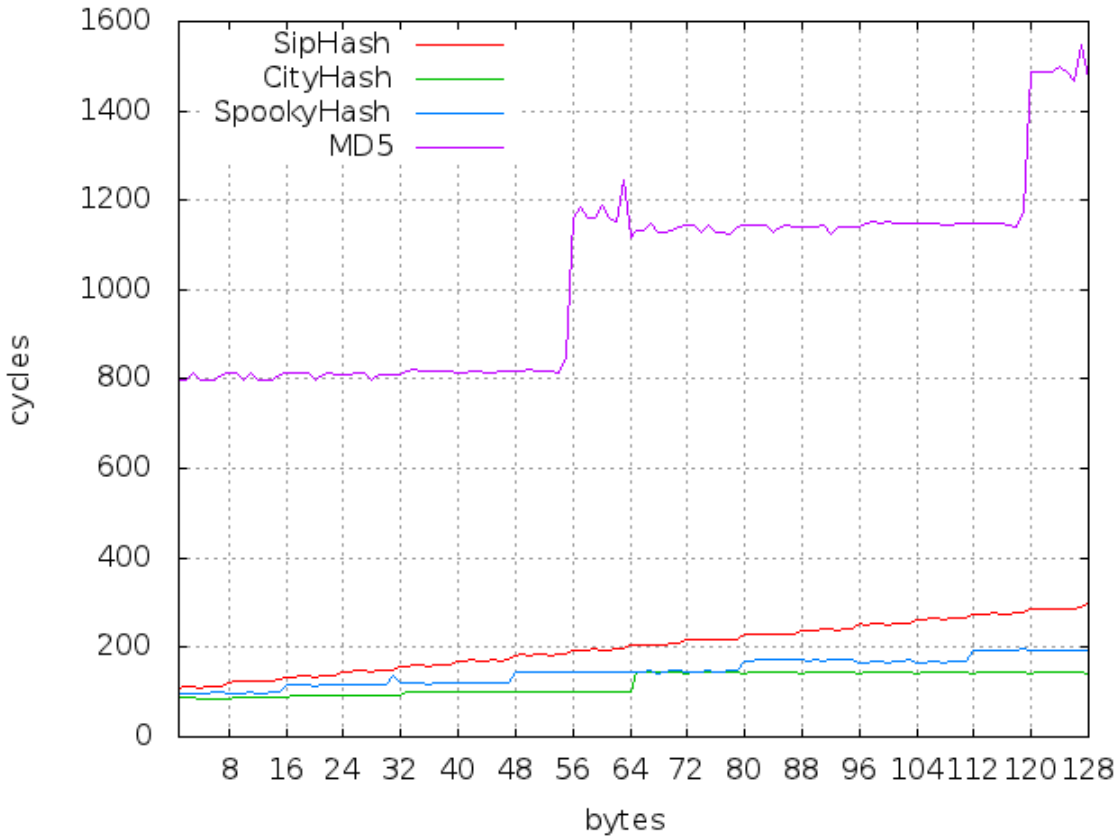


Fig. 6.2. Performance of SipHash-2-4 compared to non-cryptographic hash functions CityHash and SpookyHash and to MD5 on "ishmael" (AMD Athlon II), for messages of $1, 2, \ldots, 128$ bytes. Curves on the right, from top to bottom, are MD5, SipHash, SpookyHash, and CityHash.

7.1 Review of hash tables

Storing n strings in a linked list usually takes a total of $\Theta(n^2)$ operations, and retrieving one of the n strings usually takes $\Theta(n)$ operations. This can be a crippling performance problem when n is large.

Hash tables are advertised as providing much better performance. The simplest type of hash table contains ℓ separate linked lists $L[0], L[1], \ldots, L[\ell - 1]$ and stores each string m inside the linked list $L[H(m) \bmod \ell]$, where H is a hash function and ℓ is a power of 2. Each linked list then has, on average, only n/ℓ strings. Normally this improves performance by a factor close to n if ℓ is chosen to be on the same scale as n: storing n strings usually takes only $\Theta(n)$ operations and retrieving a string usually takes $\Theta(1)$ operations.

There are other data structures that guarantee, e.g., $O(n \lg n)$ operations to store n strings and $O(\lg n)$ operations to retrieve one string. These data structures avoid all of the security problems discussed below. However, hash tables are perceived as being simpler and faster, and as a result are used pervasively throughout current programming languages, libraries, and applications.

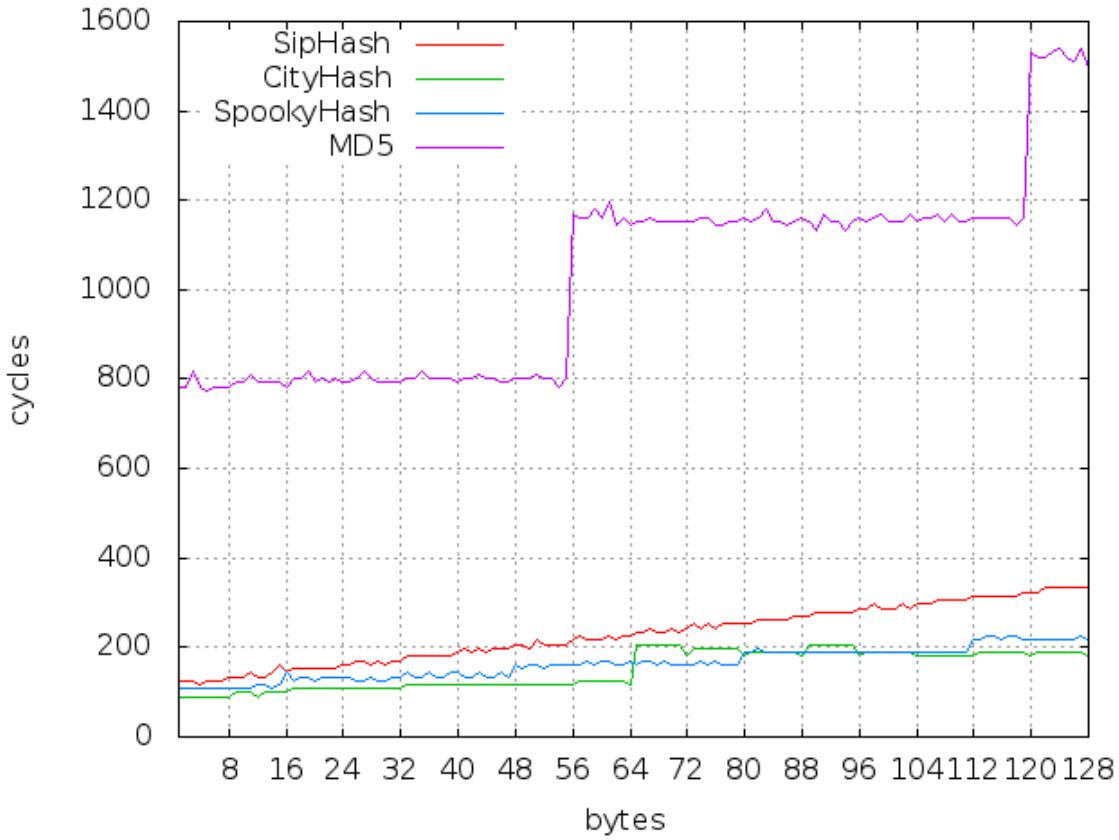


Fig. 6.3. Performance of SipHash-2-4 compared to non-cryptographic hash functions CityHash and SpookyHash and to MD5 on "latour" (Intel Core 2 Quad Q6600), for messages of $1, 2, \ldots, 128$ bytes. Curves on the right, from top to bottom, are MD5, SipHash, SpookyHash, and CityHash.

7.2 Review of hash flooding

Hash flooding is a denial-of-service attack against hash tables. The attacker provides n strings m that have the same hash value $H(m)$, or at least the same $H(m) \bmod \ell$. The hash-table performance then deteriorates to the performance of one linked list.

The name "hash flooding" for this attack appeared in 1999, in the source code for the first release of the `dnscache` software from the second author of this paper:

```
if (++loop > 100) return 0; /* to protect against hash flooding */
```

This line of code protects `dnscache` against the attack by limiting each linked list to 100 entries. However, this is obviously not a general-purpose solution to hash flooding. Caches can afford to throw away unusual types of data, but most applications need to store all incoming data.

Crosby and Wallach reintroduced the same attack in 2003 under the name "algorithmic complexity attack" [9] and explored its applicability to the Squid web cache, the Perl programming language, etc. Hash flooding made headlines

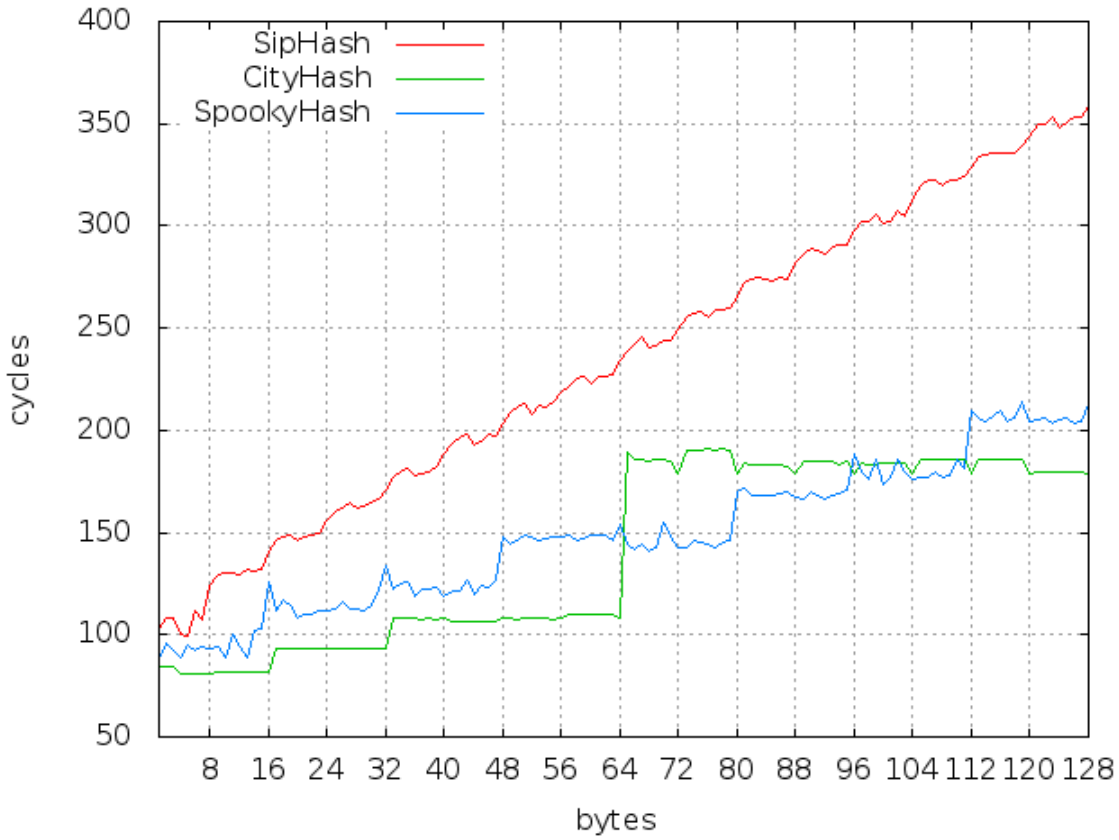


Fig. 6.4. Performance of SipHash-2-4 compared to non-cryptographic hash functions CityHash and SpookyHash on "bulldozer" (AMD FX-8150), for messages of $1, 2, \ldots, 128$ bytes. Curves on the right, from top to bottom, are SipHash, SpookyHash, and CityHash.

again in December 2011, when Klink and Wälde [19] demonstrated its continued applicability to several commonly used web applications. For example, Klink and Wälde reported 500 KB of carefully chosen POST data occupying a PHP5 server for a full minute of CPU time.

7.3 Advanced hash flooding

Crosby and Wallach recommended replacing public functions H with secret functions, specifically universal hash functions, specifically the hash function $H(m_0, m_1, \ldots) = m_0 \cdot k_0 + m_1 \cdot k_1 + \cdots$ using a secret key $(k_0, k_1, \ldots)$. The idea is that an attacker has no way to guess which strings will collide.

We question the security of this approach. Consider, for example, a hash table containing one string m, where m is known to the attacker. Looking up another string m' will, with standard implementations, take longer if $H(m') \equiv H(m) \pmod{\ell}$ than if $H(m') \not\equiv H(m) \pmod{\ell}$. This timing information will often be visible to an attacker, and can be amplified beyond any level of noise if the application allows the attacker to repeatedly query m'. By guessing ℓ choices of strings $m' \neq m$ the attacker finds one with $H(m') \equiv H(m) \pmod{\ell}$.

The linearity of the Crosby–Wallach choice of H then implies that adding any multiple of $m' - m$ to m will produce another colliding string. With twice as many guesses the attacker finds an independent string m'' with $H(m'') \equiv H(m) \pmod{\ell}$; then adding any combination of multiples of $m' - m$ and $m'' - m$ to m will produce even more collisions. With a moderate number of guesses the attacker finds enough information to solve for $(k_0 \bmod \ell, k_1 \bmod \ell, \ldots)$ by Gaussian elimination, and easily computes any number of strings with the same hash value.

One can blame the hash-table implementation for leaking information through timing; but it is not easy to build an efficient constant-time hash table. Even worse, typical languages and libraries allow applications to see all hash-table entries in order of hash value, and applications often expose this information to attackers. One could imagine changing languages and libraries to sort hash-table entries before enumerating them, but this would draw objections from applications that need the beginning of the enumeration to start quickly. One could also imagine changing applications to sort hash-table entries before exposing them to attackers, but ensuring this would require reviewing code in a huge number of applications.

We comment that many of the hash-flooding defenses proposed since December 2011 are vulnerable to the same attack. The most common public hash functions are of the form $m_0 \cdot k_0 + m_1 \cdot k_1 + \cdots$ where $k_0, k_1, \ldots$ are public, and many of the proposed defenses simply add some entropy to $k_0, k_1, \ldots$; but the attack works no matter how $k_0, k_1, \ldots$ are chosen. Many more of the proposed defenses are minor variations of this linear pattern and are broken by easy variants of the same attack.

We do not claim novelty for observing how much damage a single equation $H(m') \equiv H(m) \pmod{\ell}$ does to the unpredictability of this type of hash function; see, e.g., the attacks in [6] and [14] against related MACs. However, the fact that hash tables leak such equations through side channels does not seem to be widely appreciated.

7.4 Stopping advanced hash flooding

The worst possible exposure of hash-table indices would simply show the attacker $H(m) \bmod \ell$ for any attacker-selected string m. We advocate protecting against this maximum possible exposure, so that applications do not have to worry about how much exposure they actually provide. The attacker's goal, given this exposure, is to find many strings m having a single value $H(m) \bmod \ell$.

We propose choosing H to be cryptographically strong: i.e., a PRF. If H is a PRF then the truncation $H \bmod \ell$ is also a PRF (recall that ℓ is a power of 2), and therefore a maximum-security MAC: even after seeing $H(m) \bmod \ell$ for selected strings m, the attacker cannot predict $H(m) \bmod \ell$ for any other string m. The PRF property for H implies the same unpredictability even if the attacker is given hash values $H(m)$, rather than just hash-table indices $H(m) \bmod \ell$. Achieving this level of unpredictability does not appear to be significantly easier than achieving the full PRF property.

Typical hash-table applications hash a large number of short strings, so the performance of H on short inputs is critical. We therefore propose choosing SipHash as H: we believe that SipHash is a PRF, and it provides excellent performance on short inputs. There are previous hash functions with competitive performance, and there are previous functions that have been proposed and evaluated for the same security standards, but as far as we know SipHash is the first function to have both of these features.

Of course, the attacker's inability to predict new hash values does not stop the attacker from exploiting old hash values. No matter how strong H is, the attacker will find two colliding strings after (on average) about $\sqrt{\ell}$ guesses, and then further strings with the same hash value for (on average) ℓ guesses per collision. However, finding n colliding strings in this way requires the attacker to communicate about $n\ell \approx n^2$ strings, so n—the CPU amplification factor of the denial-of-service attack—is limited to the square root of the volume of attacker communication. For comparison, weak secret hash functions and (weak or strong) public hash functions allow n to grow linearly with the volume of attacker communication. A strong secret hash function thus greatly reduces the damage caused by the attack.

References

[1] — (no editor), *20th annual symposium on foundations of computer science*, IEEE Computer Society, New York, 1979. MR 82a:68004. See [26].

[2] Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, Christian Rechberger, *New features of Latin dances: analysis of Salsa, ChaCha, and Rumba*, in FSE 2008 [**22**] (2007), 470–488. URL: http://eprint.iacr.org/2007/472. Citations in this document: §5.2.

[3] Daniel J. Bernstein, *The Poly1305-AES message-authentication code*, in [**12**] (2005), 32–49. URL: http://cr.yp.to/papers.html#poly1305. ID 0018d9551b5546d97c340e0dd8cb5750. Citations in this document: §1.

[4] Daniel J. Bernstein, *Salsa20 security*, in eSTREAM report 2005/025 (2005). URL: http://cr.yp.to/snuffle/security.pdf. Citations in this document: §5.2.

[5] Eli Biham, Amr M. Youssef (editors), *Selected areas in cryptography, 13th international workshop, SAC 2006, Montreal, Canada, August 17–18, 2006, revised selected papers*, Lecture Notes in Computer Science, 4356, Springer, 2007. ISBN 978-3-540-74461-0. See [20].

[6] John Black, Martin Cochran, *MAC reforgeability*, in FSE 2009 [**11**] (2009), 345–362. URL: http://eprint.iacr.org/2006/095. Citations in this document: §7.3.

[7] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, Phillip Rogaway, *UMAC: fast and secure message authentication*, in Crypto '99 [**28**] (1999), 216–233. URL: http://fastcrypto.org/umac/umac_proc.pdf. Citations in this document: §1.

[8] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, Phillip Rogaway, *Update on UMAC fast message authentication* (2000). URL: http://fastcrypto.org/umac/update.pdf. Citations in this document: §1, §1.

[9] Scott A. Crosby, Dan S. Wallach, *Denial of service via algorithmic complexity attacks*, 12th USENIX Security Symposium (2003). URL: http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach_UsenixSec2003.pdf. Citations in this document: §7.2.

[10] Wei Dai, Ted Krovetz, *VHASH security* (2007). URL: http://eprint.iacr.org/2007/338. Citations in this document: §1.

[11] Orr Dunkelman (editor), *Fast software encryption, 16th international workshop, FSE 2009, Leuven, Belgium, February 22–25, 2009, revised selected papers*, Lecture Notes in Computer Science, 5665, Springer, 2009. ISBN 978-3-642-03316-2. See [6].

[12] Henri Gilbert, Helena Handschuh (editors), *Fast software encryption: 12th international workshop, FSE 2005, Paris, France, February 21–23, 2005, revised selected papers*, Lecture Notes in Computer Science, 3557, Springer, 2005. ISBN 3-540-26541-4. See [3].

[13] Google, *The CityHash family of hash functions* (2011). URL: https://code.google.com/p/cityhash/. Citations in this document: §1.

[14] Helena Handschuh, Bart Preneel, *Key-recovery attacks on universal hash function based MAC algorithms*, in CRYPTO 2008 [**25**] (2008), 144–161. URL: http://www.cosic.esat.kuleuven.be/publications/article-1150.pdf. Citations in this document: §7.3.

[15] Seokhi Hong, Tetsu Iwata, *Fast software encryption, 17th international workshop, FSE 2010, Seoul, Korea, February 7–10, 2010, revised selected papers*, Lecture Notes in Computer Science, 6147, Springer, 2010. ISBN 978-3-642-13857-7. See [18].

[16] Bob Jenkins, *SpookyHash: a 128-bit noncryptographic hash* (2010). URL: http://burtleburtle.net/bob/hash/spooky.html. Citations in this document: §1.

[17] Bob Jenkins, *Issue 4: CityHash128 isn't thorough enough* (2011). URL: https://code.google.com/p/cityhash/issues/detail?id=4&can=1. Citations in this document: §1.

[18] Dmitry Khovratovich, Ivica Nikolic, *Rotational cryptanalysis of ARX*, in FSE 2010 [**15**] (2010), 333–346. URL: http://www.skein-hash.info/sites/default/files/axr.pdf. Citations in this document: §5.2.

[19] Alexander Klink, Julian Wälde, *Efficient denial of service attacks on web application platforms* (2011). URL: http://events.ccc.de/congress/2011/Fahrplan/events/4680.en.html. Citations in this document: §7.2.

[20] Ted Krovetz, *Message authentication on 64-bit architectures*, in [**5**] (2007), 327–341. URL: http://eprint.iacr.org/2006/037. Citations in this document: §1.

[21] Gaëtan Leurent, *The ARX toolkit* (2012). URL: http://www.di.ens.fr/~leurent/arxtools.html. Citations in this document: §5.2.

[22] Kaisa Nyberg (editor), *Fast software encryption, 15th international workshop, FSE 2008, Lausanne, Switzerland, February 10–13, 2008, revised selected papers*, Lecture Notes in Computer Science, 5086, Springer, 2008. ISBN 978-3-540-71038-7. See [2].

[23] Souradyuti Paul, *Improved indifferentiability security bound for the JH mode*, Third SHA-3 Conference (2012). URL: http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/March2012/documents/papers/PAUL_paper.pdf. Citations in this document: §4.

[24] Peter Schwabe, Bo-Yin Yang, Shang-Yi Yang, *SHA-3 on ARM11 processors*, Proceedings of Africacrypt 2012, to appear (2012). URL: http://cryptojedi.org/papers/sha3arm-20120422.pdf. Citations in this document: §6.2.

[25] David Wagner (editor), *Advances in cryptology—CRYPTO 2008, 28th annual international cryptology conference, Santa Barbara, CA, USA, August 17–21, 2008, proceedings*, Lecture Notes in Computer Science, 5157, Springer, 2008. ISBN 978-3-540-85173-8. See [14].

[26] Mark N. Wegman, J. Lawrence Carter, *New classes and applications of hash functions*, in [1] (1979), 175–182; see also newer version [27]. URL: http://cr.yp.to/bib/entries.html#1979/wegman.

[27] Mark N. Wegman, J. Lawrence Carter, *New hash functions and their use in authentication and set equality*, Journal of Computer and System Sciences **22** (1981), 265–279; see also older version [26]. ISSN 0022-0000. MR 82i:68017. URL: http://cr.yp.to/bib/entries.html#1981/wegman. Citations in this document: §1.

[28] Michael Wiener (editor), *Advances in cryptology—CRYPTO '99*, Lecture Notes in Computer Science, 1666, Springer, 1999. ISBN 3-5540-66347-9. MR 2000h:94003. See [7].

A Test values

This appendix shows intermediate values of SipHash-2-4 hashing the 15-byte string `000102···0c0d0e` with the 16-byte key `000102···0d0e0f`.

Initialization little-endian reads the key as

$$k_0 = \texttt{0706050403020100}$$
$$k_1 = \texttt{0f0e0d0c0b0a0908}$$

The key is then xored to the four constants to produces the following initial state (v_0 to v_3, left to right):

```
7469686173716475 6b617f6d656e6665 6b7f62616d677361 7b6b696e727e6c7b
```

The first message block `0706050403020100` is xored to v_3 to give

```
7469686173716475 6b617f6d656e6665 6b7f62616d677361 7c6d6c6a717c6d7b
```

and after two SipRounds the internal state is:

```
4d07749cdd0858e0 0d52f6f62a4f59a4 634cb3577b01fd3d a5224d6f55c7d9c8
```

Xoring the first message block to v_0 concludes the compression phase:

```
4a017198de0a59e0 0d52f6f62a4f59a4 634cb3577b01fd3d a5224d6f55c7d9c8
```

The second and last block is the last seven message bytes followed by the message's length, that is, `0f0e0d0c0b0a0908`. After xoring this block to v_3, doing two SipRounds, xoring it to v_0 and xoring `00000000000000ff` to v_2, the internal state is

```
3c85b3ab6f55be51 414fc3fb98efe374 ccf13ea527b9f442 5293f5da84008f82
```

After the four iterations of SipRound, the internal state is

```
f6bcd53893fecff1 54b9964c7ea0d937 1b38329c099bb55a 1814bb89ad7be679
```

and the four words are xored together to return `a129ca6149be45e5`.

How Fast Can a Two-Pass Mode Go? A Parallel Deterministic Authenticated Encryption Mode for AES-NI

(Extended Abstract of Work in Progress)

Kazumaro Aoki*, Tetsu Iwata† and Kan Yasuda*

*NTT Secure Platform Laboratories, Japan
†Nagoya University, Japan
{aoki.kazumaro,yasuda.kan}@lab.ntt.co.jp
iwata@cse.nagoya-u.ac.jp

Abstract. We present a mode of operation for deterministic authenticated encryption. Our design is entirely blockcipher-based, like SIV from Eurocrypt 2006. Unlike SIV, however, our mode is fully parallelizable, both in its authentication and encryption parts. The basic idea is a combination of PMAC for authentication and the CTR mode for encryption. Our implementation yields 2.047 cpb for Intel's Sandy Bridge microarchitecture, which contrasts with 2.900 cpb of the GCM mode in OpenSSL 1.0.1c.

Keywords: DAE, AES-NI, PMAC, CTR, doubling.

1 Introduction

In principle, a two-pass AE (Authenticated Encryption) scheme is slower than a one-pass one. It sounds fundamentally impossible for two-pass schemes to beat the performance of one-pass modes of operation, which is true. However, exactly how much slower are two-pass schemes? One might guess that they are twice as slow. Looking at real-world examples, we soon find that this is not the case—two-pass schemes such as CCM [12] and GCM [9] generally run *more than* twice as slow as one-pass modes such as OCB [8] (depending on the environment, of course).

There are a couple of reasons for this. One is that some of the two-pass schemes contain CBC-like sequential iteration of blockcipher calls. This applies to CCM and SIV [11]. Since OCB is fully parallelizable, CCM and SIV would not be as fast as "twice OCB."

Another reason is that some of the two-pass schemes utilize polynomial hashing. This applies to GCM and BTM [7]. Polynomial hashing is somewhat parallelizable but can be slower than blockcipher calls especially when only small amount of memory is available or when AES-NI (new AES instruction set) is available.

So what happens if we build an entirely blockcipher-based (two-pass) mode that is fully parallelizable both in its authentication and encryption parts? Is it going to be as fast as "twice OCB?"

In this paper, we present a mode of operation for DAE (Deterministic Authenticated Encryption). It works without using nonces, and can also be used as the traditional nonce-based AE (like GCM or OCB) by setting the nonce into a part of the input. The security of nonce-based AE critically relies on the uniqueness of nonces, while the security of DAE is not completely lost even when the uniqueness of the nonce cannot be guaranteed. Our basic idea is the combination PMAC+CTR. We expect PMAC [1] to run as fast as OCB. We also know that CTR runs as fast as, if not faster than, OCB. So there is a good chance that it is going to be as fast as "twice OCB" and hence generally faster than existing two-pass schemes.

In this extended abstract we report our current status and implementation results of our research in this direction.

2 Specification

Figure 1 defines the encipherment algorithm of the PMAC+CTR mode using an n-bit blockcipher $E_K : \{0,1\}^n \to \{0,1\}^n$. Please refer to Fig. 2 for an illustration of the encipherment algorithm. Our primary interest is the case when E is AES, which means $n = 128$. In this case we choose a primitive polynomial $x^{128} + x^7 + x^2 + x + 1 \in \mathrm{GF}(2)[x]$ to represent the field $\mathrm{GF}(2^{128})$. By $2L$ we mean the multiplication of L by '2'$= x$ in the field $\mathrm{GF}(2^{128})$ and by $3L$ the multiplication of L by '3'$= x + 1$.

The operation $\big(A[1], A[2], \ldots, A[a]\big) \xleftarrow{n} A$ is the partitioning operation into n-bit strings; $\big(A[1], A[2], \ldots, A[a]\big) = A$, where $\big|A[i]\big| = n$ for $1 \le i \le a-1$ and $1 \le \big|A[a]\big| \le n$. When A is the null string, the operation $\big(A[1], A[2], \ldots, A[a]\big) \xleftarrow{n} A$ sets $a \leftarrow 1$ and $A[a] = A[1]$ becomes the null string. In such a case we have $\big|A[a]\big| = 0$ and hence $\big|A[a]\big| < n$, which sets $B[a] \leftarrow 2^{a-1}3^2L \oplus 10^{n-1}$. Likewise for M.

The symbol $\boxplus$ represents the Cartesian product of two $n/2$-bit additions, each modulo $2^{n/2}$. That is, for $(a_1, a_2), (b_1, b_2) \in \mathbb{Z}/2^{n/2}\mathbb{Z} \times \mathbb{Z}/2^{n/2}\mathbb{Z}$, we have $(a_1, a_2) \boxplus (b_1, b_2) = (a_1 + b_1, a_2 + b_2) \in \mathbb{Z}/2^{n/2}\mathbb{Z} \times \mathbb{Z}/2^{n/2}\mathbb{Z}$. Here we regard $\mathbb{Z}/2^{n/2}\mathbb{Z} \times \mathbb{Z}/2^{n/2}\mathbb{Z}$ as $\{0,1\}^n$ using the natural conversion. We also assume that n is an even integer. The symbol was used in the design of BTM [7].

The truncation function $\mathrm{trunc}_i(S)$ outputs the leftmost (significant) i bits of the string S. The tag length τ is a positive integer less than or equal to n.

The decipherment algorithm, given A, K, T, C as its inputs, runs in the natural way. It computes a tag T' in the same way and compares it with the given tag value T. Depending on the result of verification, the decipherment algorithm outputs either the reject symbol $\perp$ or the decrypted message M.

Input: associated data $A \in \{0,1\}^*$, a message $M \in \{0,1\}^*$, a key $K \in \{0,1\}^\kappa$
Output: a tag $T \in \{0,1\}^\tau$, a ciphertext $C \in \{0,1\}^*$
$L \leftarrow E_K(0)$; $\big(A[1], A[2], \ldots, A[a]\big) \xleftarrow{n} A$; $\big(M[1], M[2], \ldots, M[m]\big) \xleftarrow{n} M$
for $i = 1$ **to** $a - 1$ **do**
  $B[i] \leftarrow E_K\big(2^{i-1}L \oplus A[i]\big)$
end
if $A[a] = n$ **then** $B[a] \leftarrow 2^{a-1}3L \oplus A[a]$
if $A[a] < n$ **then** $B[a] \leftarrow 2^{a-1}3^2L \oplus A[a]\|10^*$
$W \leftarrow E_K\big(B[1] \oplus B[2] \oplus \cdots \oplus B[a]\big)$
for $i = 1$ **to** $m - 1$ **do**
  $Q[i] \leftarrow E_K\big(2^{i-1}L \oplus M[i]\big)$
end
if $M[m] = n$ **then** $Q[m] \leftarrow 2^{m-1}L \oplus M[m]$
if $M[m] < n$ **then** $Q[m] \leftarrow 2^{m-1}3L \oplus M[m]\|10^*$
$T \leftarrow \mathrm{trunc}_\tau\big(E_K\big(W \oplus Q[1] \oplus Q[2] \oplus \cdots \oplus Q[m]\big)\big)$
for $i = 1$ **to** $m - 1$ **do**
  $C[i] \leftarrow E_K\big(W \boxplus T\|0^* \boxplus i - 1\big) \oplus M[i]$
end
$C[m] \leftarrow \mathrm{trunc}_{|M[m]|}\big(E_K\big(W \boxplus T\|0^* \boxplus m - 1\big)\big) \oplus M[m]$
$C \leftarrow C[1]\|C[2]\|\cdots\|C[m]$
return (T, C)

Fig. 1. The PMAC+CTR Encipherment Algorithm with an n-bit Blockcipher E

3 Design Rationale

3.1 Choice of DAE

We choose DAE (Deterministic Authenticated Encryption) rather than the traditional nonce-based AE. DAE can function as nonce-based AE by inputting nonce values into associated data. DAE is misuse-resistant/fool-proof in the sense that it retains security (in a bit weaker sense) even if the nonce is repeated; this is the characteristic of AE schemes that we consider important in practice (Recall that conventional nonce-based AE schemes would become completely insecure as soon as nonce gets repeated.)

3.2 Choice of PMAC and CTR

We have already mentioned in Sect. 1 that we choose PMAC and CTR for their full parallelizability. Also, it should be noted that PMAC+CTR does not require the blockcipher inverse for decipherment, whereas OCB does.

3.3 Choice of "Doubling"

How to increment an offset is one of the design challenges. According to recent analysis, Gray code seems to perform better than doubling [8]. However, we

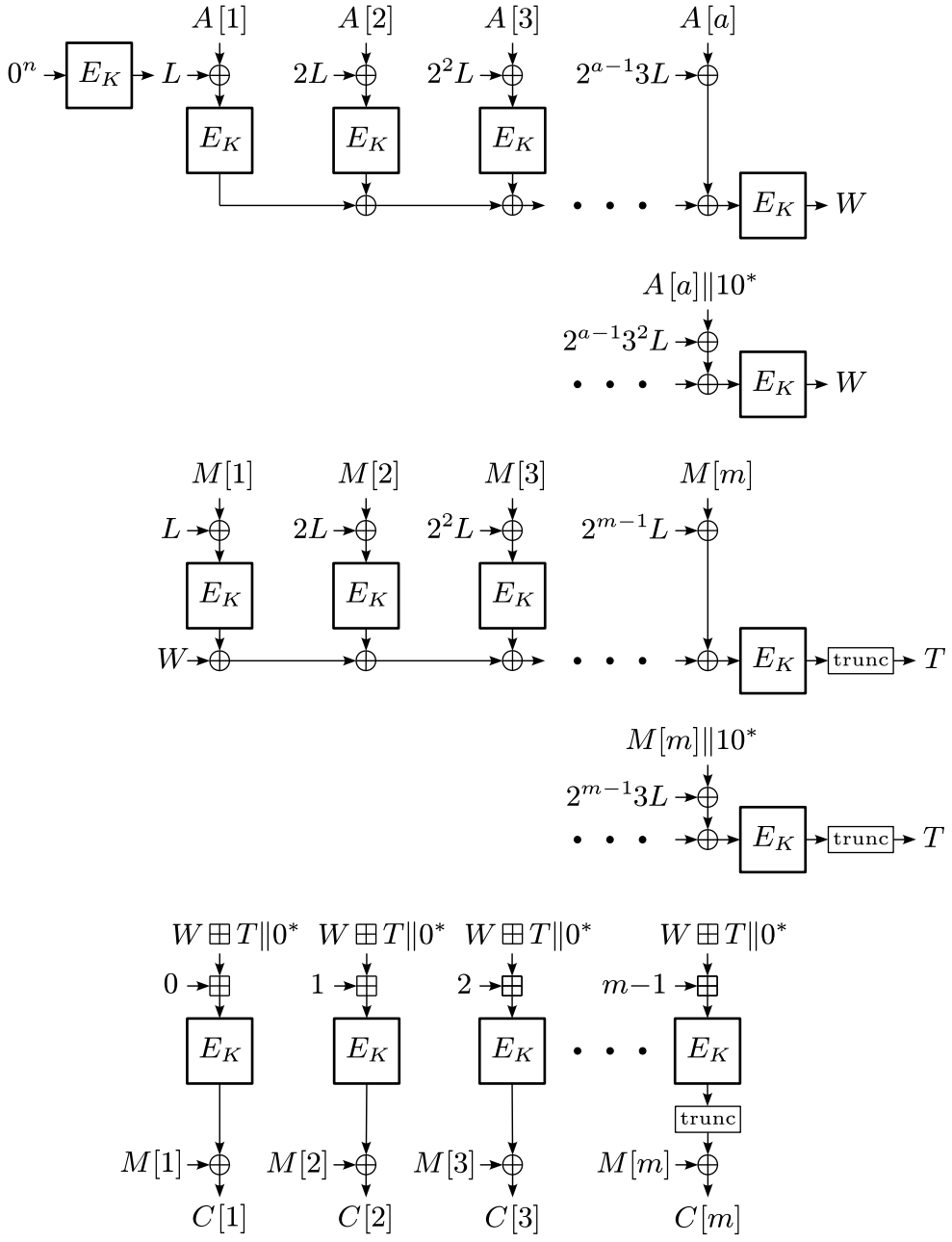


Fig. 2. A Pictorial Representation of the PMAC+CTR Encipherment Algorithm

```
1: add  rax, 1
2: bsf  rbx, rax
3: add  rbx, rbx
4: pxor xmm0, tbl[8 * rbx]
```

Fig. 3. Incrementing offset for Gray code

```
1: movmskpd rax, xmm0
2: add      rax, rax
3: paddq    xmm0, xmm0
4: pxor     xmm0, tbl[8 * rax]
5: vpshufb  xmm2, xmm0, xmm1
```

Fig. 4. Doubling

decided to choose doubling instead of Gray code, for the following reasons. We want to maximize performance on CPUs that have AES-NI. Both doubling and Gray code work good on general purpose (64-bit) registers. AES instructions work on XMM (128-bit) registers. Incrementing an offset requires conversion from general registers to XMM registers. We want to increment an offset stored in an XMM register.

For the case of Gray code, offset incrementing can be realized as straightforward way using `bsf` instruction which computes the number of trailing '0's. For the case of doubling, the lack of the shift instruction for full bits of XMM registers is the big problem. How to solve the problem is as follows. The doubling operation for L can mathematically be computed as $(L^{\ll 1}) \oplus (\mathrm{msb}(L) \cdot \texttt{0x87})$, where msb extracts the most significant bit. We can only shift left for internal two 64-bit words using `paddq` instruction or `psllq` instruction. Fortunately, we can extract the most significant bits of the two internal registers to a general purpose register using `movmskpd` instruction, which is introduced in SSE2. Thus, we can efficiently compute doubling by preparing the table $[0, 1^{\ll 64}, \texttt{0x87}, 1^{\ll 64} \oplus \texttt{0x87}]$.

Figures 3 and 4 show the standard and our implementation of incrementing Gray code and doubling, respectively. In Figure 3, registers `rax`, `rbx`, and `xmm0` are used for the counter, a temporary register, and the mask, respectively. In Figure 4, registers `rax`, `xmm0`, `xmm1`, and `xmm2` are used for a temporary register, the mask, the constant for the endian conversion, and the mask to be XORed, respectively. `tbl` in Figures 3 and 4 are the precomputed tables.

We compare the efficiency characteristics of both figures on Sandy Bridge. The total number of μ-operations, the total latency, and the critical path are 5, 10 clock cycles, 10 clock cycles for Figure 3, and 7, 10 clock cycles, 9 clock cycles for Figure 4, respectively. These numbers are obtained from [6, 2]. The precomputed table for Figure 3 depends on the secret key, but the precomputed table for Figure 4 does not depend on the secret key and it can be computed in advance and constant size. As we described above, we choose the conservative doubling, since there is not so big difference between Gray code and doubling.

4 Characteristics

4.1 Implementation Aspects

In the encipherment algorithm, we need a result of PMAC part to execute CTR part. Thus, we first execute PMAC part followed by CTR part. PMAC and CTR parts are both parallelizable. In the decipherment algorithm, to verify the correctness of the tag, we need the corresponding plaintext. Thus, we first compute the first part of PMAC to derive the intermediate value W, and recover the plaintext M using W and the given tag. We then compute the last part of PMAC and compare the result with the given tag to verify its the correctness. Each part, the first part of PMAC, CTR decryption, and the last part of PMAC, are paralellizable, but we need to compute the decipherment of PMAC+CTR in this order. The computational order for the decipherment is different from

one for the encipherment, though the performance of both the encipherment and decipherment is similar, since the main computation and parallelizability are the same for both the encipherment and decipherment.

4.2 Security

The PMAC+CTR mode is provably secure. Unfortunately in this manuscript we are unable to provide full security proofs. Instead, we state our dae-security result of PMAC+CTR for the case $\tau = n$.

We first review the security notion. Let $\mathcal{E}_K$ and $\mathcal{D}_K$ be the encipherment and decipherment algorithms, respectively, of PMAC+CTR. The encipherment algorithm takes associated data A and a message M, and outputs a pair of a tag T and a ciphertext C, where $|T| = \tau$ and $|C| = |M|$. The decipherment algorithm takes (A, C, T), where C is a ciphertext, and outputs either the corresponding plaintext M or the symbol $\perp$. The goal of an adversary $\mathcal{A}$ attacking dae-security of PMAC+CTR is to distinguish between the pair $(\mathcal{E}_K, \mathcal{D}_K)$ and the pair $(\mathcal{R}, \perp)$, where $\mathcal{R}$ is an oracle that returns a random string of $\tau + |M|$ bits for a query (A, M), and $\perp$ is an oracle that always returns the $\perp$ symbol. The success probability of $\mathcal{A}$ is measured by

$$\mathrm{Adv}^{\mathrm{dae}}_{\mathrm{PMAC+CTR}}(\mathcal{A}) := \Pr\left[\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K} = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{R}_K, \perp} = 1\right],$$

where trivial queries are excluded.

For a blockcipher E_K, we consider the security as a PRP (Pseudo-Random Permutation). The goal of an adversary $\mathcal{A}$ attacking the prp-security of E is to distinguish the blockcipher E_K from a truly random permutation P. The success probability of $\mathcal{A}$ is measured by

$$\mathrm{Adv}^{\mathrm{prp}}_{E}(\mathcal{A}) := \Pr\left[\mathcal{A}^{E_K} = 1\right] - \Pr\left[\mathcal{A}^{P} = 1\right],$$

where trivial queries are excluded.

We have the following theorem.

Theorem 1. *The PMAC+CTR mode with $\tau = n$ is secure as a DAE scheme based on the assumption that the underlying blockcipher E is a secure PRP. Specifically, for an adversary $\mathcal{A}$ attacking dae-security of PMAC+CTR and making queries of σ blocks in total, there exists an adversary $\mathcal{A}'$ that attacks prp-security of the underlying blockcipher E so that*

$$\mathrm{Adv}^{\mathrm{dae}}_{\mathrm{PMAC+CTR}}(\mathcal{A}) \le \mathrm{Adv}^{\mathrm{prp}}_{E}(\mathcal{A}') + (\textit{some constant}) \cdot \frac{\sigma^2}{2^n},$$

where the resources of $\mathcal{A}'$ are comparable to those of $\mathcal{A}$.

The above theorem suggests that PMAC+CTR has the standard birthday bound security. The proof of the above theorem combines techniques from [10] and those from [7], and we expect "some constant" to be small, like 10 or 20. In addition to the techniques from [10], we shall need to extend the framework

Table 1. Performance Comparison

Mode	cycles per byte Nehalem	Sandy Bridge	Source	Message Length
ECB	1.28	—	[4]	1KB
	1.26	0.851	OpenSSL 1.0.1c	8KB
	1.28	0.81	[3]	1KB
	—	0.705	Ours (intrinsic)	8KB
	—	0.702	Ours (assembly)	8KB
CTR	1.38	—	[4]	1KB
	1.27	—	[8]	4KB
	1.26	0.916	OpenSSL 1.0.1c	8KB
	1.30	0.83	[3]	1KB
	—	0.787	Ours (intrinsic)	8KB
CCM	4.06	—	OpenSSL 1.0.1c	8KB
GCM	3.73	—	[8]	4KB
	3.54	—	[5]	16KB
	—	2.900	OpenSSL 1.0.1c	8KB
	—	2.59	[3]	16KB
	3.90	2.67	[3]	4KB
OCB3	1.48	—	[8]	4KB
XTS	1.87	1.18	[3]	1KB
	1.33	0.978	OpenSSL 1.0.1c	8KB
PMAC+CTR	—	2.047	Ours (intrinsic)	8KB

Refer to the main text for how we have obtained the figures in the table.

of tweakable ciphers; we shall need to extend the set of masks from $\{2^i 3^j L\}$ to masks generated from L and W, in such a way as the generation of W depends on the value L. Moreover, to handle queries made to the decipherment oracle, we also need to utilize the "auxiliary" oracle introduced in [7]. Thus, unfortunately the proof would not be a simple one, but after all all the bad events can be bounded by (some constant times) $\sigma^2/2^n$.

The PMAC+CTR mode also achieves other security notions. For example, it achieves the usual security notion of nonce-based AE when associated data A is nonce. The authentication part can be used as standalone to provide a secure MAC (message authentication code).

5 Experimental Results

We implement PMAC+CTR for the case of null associated data on Intel's Sandy Bridge CPU with AES-NI. Table 1 summarizes the performance results of modes with AES-128 encryption using AES-NI. All results are measured in one core even if the target CPU has many cores. All Nehalem results are reprinted from the corresponding source. Sandy Bridge results for our implementation and OpenSSL 1.0.1c are measured on FreeBSD/amd64 9.0-STABLE

(June 1, 2012) with Intel Xeon CPU E3-1225 (3.10GHz). We turn off the Turbo Boost technology, and we do not use `powerd` daemon to keep the regular operating frequency. Nehalem numbers for OpenSSL 1.0.1c are summarized from the comments in `openssl-1.0.1c/crypto/aes/asm/aesni-x86_64.pl`. Sandy Bridge numbers for OpenSSL 1.0.1c are measured by "`openssl speed -evp aes-128-{ecb,ctr,gcm,xts}`", and converted them to the unit of cycle per byte. The comments in `aesni-x86_64.pl` also describe that AES-128-ECB and AES-128-XTS runs in 0.84 and 0.97 cycles per byte, respectively on Sandy Bridge, which are slightly faster than our measurement. All of our codes are compiled using `ports/lang/gcc48` (gcc 4.8.0 20120603) with "`-O3 -funroll-loops -maes -mavx`" options. "intrinsic" means the use of intrinsic functions such as `_mm_aesenc_si128`, and "assembly" means the use of inline assembly codes. All functions of our implementation require the pointers to ciphertext, plaintext, subkey, and the length of message in bytes. The pointers of the messages are assumed to be aligned by 4KB boundary, and the length of the message is assumed to be the multiple of 8 blocks (128 bytes). The clock cycles are obtained by the standard sequence of `cpuid`, `rdtsc`, `/*` *call the function to be measured* `*/`, `rdtscp`. We subtract the overhead of the timing measurement code from the obtained cycles, but include the overhead of function calls. The clock cycles are chosen from the minimum of 2^{16} trials to avoid machine interrupts, cache misses, branch mispredictions, and other lags to make the clock cycles stable.

6 Conclusion

We have presented and implemented a deterministic authenticated encryption mode, PMAC+CTR, which is basically a combination of PMAC and CTR. The mode requires two-pass message processing, being entirely blockcipher-based. GCM is one of the most widely used authenticated encryption modes and is standardized by NIST as SP800-38D. It uses a polynomial hash, and its implementation is believed to be fast on most platforms. On the contrary, our experiments show that AES-128-PMAC+CTR achieves 2.047 cycles per byte on Sandy Bridge, and it is about 20% faster than OpenSSL implementation of AES-128-GCM on the same platform. This is because AES can be implemented quite fast in a parallelizable way with AES-NI which is included in Sandy Bridge, Nehalem, and Bulldozer microarchitectures. Compared with the overhead of a blockcipher call, the performance overhead of a mode is not negligible, especially with recent CPUs that implement efficient AES-NI. To get better performance in the software environment, we need to carefully choose operations that are used in modes of operation.

Acknowledgments. The work by Tetsu Iwata was supported in part by MEXT KAKENHI, Grant-in-Aid for Young Scientists (A), 22680001.

References

1. Black, J., Rogaway, P.: A block-cipher mode of operation for parallelizable message authentication. In: Knudsen, L.R. (ed.) Advances in Cryptology — EUROCRYPT 2002. Lecture Notes in Computer Science, vol. 2332, pp. 384–397. Springer-Verlag, Berlin, Heidelberg, New York (2002)
2. Fog, A.: Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs (2012), (`http://www.agner.org/optimize/#manuals`)
3. Gueron, S.: Software optimizations for cryptographic primitives on general purpose x86_64 platforms (2011), IndoCrypt 2011 — Tutorial (`http://2011.indocrypt.org/schedule.html`)
4. Intel Corporation: Intel Advanced Encryption Standard (AES) Instruction Set — Rev 3 (2010), `http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set/`
5. Intel Corporation: Intel Carry-Less Multiplication Instruction and its Usage for Computing the GCM Mode — Rev 2 (2010), `http://software.intel.com/en-us/articles/intel-carry-less-multiplication-instruction-and-its-usage-for-computing-the-gcm-mode/`
6. Intel Corporation: Intel 64 and IA-32 Architectures Optimization Reference Manual (2012), order Number: 248966-026 (`http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html`)
7. Iwata, T., Yasuda, K.: BTM: A single-key, inverse-cipher-free mode for deterministic authenticated encryption. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) Selected Areas in Cryptography, SAC 2009. Lecture Notes in Computer Science, vol. 5867, pp. 313–330. Springer-Verlag, Berlin, Heidelberg, New York (2009)
8. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) Fast Software Encryption — 18th International Workshop, FSE 2011. Lecture Notes in Computer Science, vol. 6733, pp. 306–327. Springer-Verlag, Berlin, Heidelberg, New York (2011)
9. McGrew, D.A., Viega, J.: The security and performance of the Galois/Counter Mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) Progress in Cryptology — INDOCRYPT 2004. Lecture Notes in Computer Science, vol. 3348, pp. 343–355. Springer-Verlag, Berlin, Heidelberg, New York (2004)
10. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. Lecture Notes in Computer Science, vol. 3329, pp. 16–31. Springer (2004)
11. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) Advances in Cryptology — EUROCRYPT 2006. Lecture Notes in Computer Science, vol. 4004, pp. 373–390. Springer-Verlag, Berlin, Heidelberg, New York (2006)
12. Whiting, D., Housley, R., Ferguson, N.: AES encryption & authentication using CTR mode & CBC-MAC. IEEE P802.11 doc 02/001r2 (2002)

AEGIS: A Fast Authenticated Encryption Algorithm

Hongjun Wu[1], Bart Preneel[2]

[1] School of Physical and Mathematical Sciences
Nanyang Technological University
wuhj@ntu.edu.sg
[2] Dept. Elektrotechniek-ESAT/COSIC
Katholieke Universiteit Leuven
bart.preneel@esat.kuleuven.be

Abstract. This paper introduces a dedicated authenticated encryption algorithm AEGIS, which is efficient for protecting internet packets. AEGIS-128 uses five AES round functions to process a 16-byte message block (one step); AES-256 uses six AES round functions. On the Intel Sandy Bridge Core i5 processor, the speeds of AEGIS-128, AEGIS-256 are around 0.64 cycles/byte (cpb) and 0.71 cpb, respectively. Our analysis shows that AEGIS offers a very high security level.

Key words: Authenticated encryption, AEGIS, AES-NI

1 Introduction

The protection of a message often involves the protection of confidentiality and authenticity. There are two main approaches to authenticate and encrypt a message. One approach is to treat the encryption and authentication separately. A block cipher or stream cipher is used to encrypt the plaintext, and a message authentication code is used to authenticate the ciphertext. For example, we may apply AES [13] in CBC mode [14] to the plaintext, then apply AES-CMAC [18] (or Pelican MAC [4] or HMAC [15]) to the ciphertext to generate authentication tag. This approach is relatively easy to analyze since the security of authentication and encryption can be analyzed almost separately. Bellare and Namprempre have performed detailed analysis of this type of authenticated encryption [3]. Another approach is to apply an integrated authenticated encryption algorithm to the message. This approach is expected to be more efficient since authentication and encryption can share part of the computation.

There are three possible approaches to design an integrated authenticated encryption algorithm. The first approach is to use a block cipher in special modes (the block cipher is treated as a black box). The research on this approach starts about ten years ago [6, 8, 9]. There are now two NIST recommended modes of operation for authenticated encryption, namely, CCM [16] and GCM [17]. OCB [20, 21, 10] is a widely known authenticated encryption mode, and OCB2 is an ISO standard. The second approach is to use a stream cipher (the stream cipher

is treated as a black box). The keystream is divided into two parts: one part for encryption and another part for authentication. There are two typical examples of this approach, namely, 3GPP algorithm 128-EIA3 (using stream cipher ZUC) [1] and Grain-128a [2]. The third approach is to design dedicated authenticated encryption algorithms. In this approach, a message is used to update the state of the cipher, and the message authentication can be achieved almost for free. Two examples of this approach are Helix [5] and Phelix [22]. The attack against Phelix [23] shows that highly unlikely this type of authenticated encryption algorithm can withstand the nonce-reuse attacks if the algorithm requires much less computation than block cipher.

In this paper, we propose a dedicated authenticated encryption algorithm AEGIS that is constructed from the AES encryption round function (not the last round). AEGIS-128 process a 16-byte message block with 5 AES round functions, and AEGIS-256 uses 6 AES round functions. The computational cost of AEGIS is about half of AES. AEGIS is very fast. On the Intel Sandy Bridge processor Core-i5, the speeds of AEGIS-128 and AEGIS-256 are about 0.64 cpb and 0.71 cpb, respectively. The speed is close to that of AES in counter (CTR) mode, and is about 8 times that of AES encryption in CBC mode. AEGIS offers a very high security. As long as the nonce is not reused, it is impossible to recover the AEGIS state and key faster than exhaustive key search (suppose that 128-bit authentication tag is used, and the forgery attack is not allowed to be repeated for the same key for more than 2^{128} times).

This paper is organized as follows. The operations, variables and functions are introduced in Sect. 2. The specifications of AEGIS-128 and AEGIS-256 are given in Sect. 3 and Sect. 4, respectively. Section 5 gives the security analysis of AEGIS-128 and AEGIS-256. Section 6 illustrates the stream ciphers and message authentication codes derived from AEGIS. The software performance of AEGIS is given in Sect. 7. The design rationale is given in Sect. 8. Section 9 concludes this paper.

2 Operations, Variables and Functions

2.1 Operations

The following operations are used in AEGIS:

- $\oplus$: bit-wise exclusive OR
- $\&$: bit-wise AND
- $\|$: concatenation
- $\lceil x \rceil$: ceiling operation, $\lceil x \rceil$ is the smallest integer not less than x

2.2 Variables and constants

The following variables and constants are used in AEGIS:

C	: The ciphertext.
C_i	: A 16-byte ciphertext block (the last block may be a partial block).
$const$	: A 32-byte constant in the hexadecimal format. $const = 00 \parallel 01 \parallel 01 \parallel 02 \parallel 03 \parallel 05 \parallel 08 \parallel 0d \parallel 15 \parallel 22 \parallel 37 \parallel 59 \parallel 90 \parallel e9 \parallel 79 \parallel 62 \parallel db \parallel 3d \parallel 18 \parallel 55 \parallel 6d \parallel c2 \parallel 2f \parallel f1 \parallel 20 \parallel 11 \parallel 31 \parallel 42 \parallel 73 \parallel b5 \parallel 28 \parallel dd$. It is the Fibonacci sequence modulo 256.
$const_0$	: The first 16 bytes of $const$.
$const_1$	: The second 16 bytes of $const$.
IV_{128}	: The 128-bit initialization vector of AEGIS-128.
IV_{256}	: The 256-bit initialization vector of AEGIS-256.
$IV_{256,0}$	: The first half of IV_{256}.
$IV_{256,1}$	: The second half of IV_{256}.
K_{128}	: The 128-bit key of AEGIS-128.
K_{256}	: The 256-bit key of AEGIS-256.
$K_{256,0}$	: The first half of K_{256}.
$K_{256,1}$	: The second half of K_{256}.
$msglen$	: The bit length of plaintext or ciphertext.
m_i	: A 16-byte data block.
P	: The plaintext.
P_i	: A 16-byte plaintext block (the last block may be a partial block).
S_i	: The state at the beginning of the ith step.
$S_{i,j}$	: The j-th 16-byte element of the state S_i. For AEGIS-128, $0 \leq j \leq 4$; for AEGIS-256, $0 \leq j \leq 5$.
T	: The authentication tag.
t	: The bit length of the tag. $64 \leq t \leq 128$.

2.3 Functions

The AES encryption round function (not the last round) is used in AEGIS:

$AESRound(A, B)$: A is the 16-byte state, B is the 16-byte round key. It can be implemented using the AES instruction `__m128_aesenc_si128(A, B)`, where A and B are two 128-bit integers `__m128i`.

3 AEGIS-128

In this section, we describe AEGIS-128. With a 128-bit key and a 128-bit initialization vector, it encrypts and authenticates a message with length up to 2^{64} bits. The authentication tag length is less than or equal to 128 bits. We recommend the use of a 128-bit tag.

3.1 The state update function of AEGIS-128

The state update function updates the 80-byte state S_i with a 16-byte message block m_i. $S_{i+1} = \texttt{StateUpdate128}(S_i, m_i)$ is given as follows:

$$
\begin{aligned}
S_{i+1,0} &= AESRound(S_{i,4}, S_{i,0} \oplus m_i);\\
S_{i+1,1} &= AESRound(S_{i,0} \oplus m_i, S_{i,1});\\
S_{i+1,2} &= AESRound(S_{i,1}, S_{i,2});\\
S_{i+1,3} &= AESRound(S_{i,2}, S_{i,3});\\
S_{i+1,4} &= AESRound(S_{i,3}, S_{i,4});
\end{aligned}
$$

The state update function is shown in Fig. 1 :

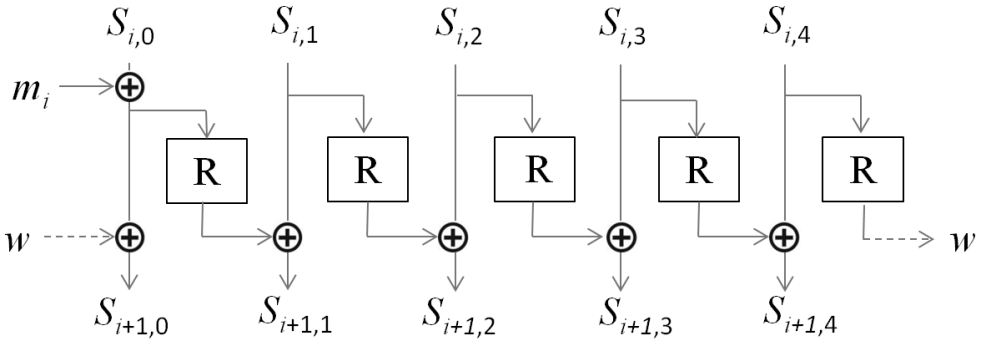


Fig. 1. The state update function of AEGIS-128. R indicates the AES encryption round function without XORing with the round key. w is a temporary 16-byte word.

3.2 The initialization of AEGIS-128

The initialization of AEGIS-128 consists of loading the key and IV into the state, and running the cipher for 10 steps with the key and IV being used as message.

1. Load the key and IV into the state as follows:

$$
\begin{aligned}
S_{-10,0} &= IV_{128};\\
S_{-10,1} &= const_1;\\
S_{-10,2} &= const_0;\\
S_{-10,3} &= K_{128} \oplus const_0;\\
S_{-10,4} &= K_{128} \oplus const_1;
\end{aligned}
$$

2. For $i = -5$ to -1, $m_{2i} = K_{128}$, $m_{2i+1} = K_{128} \oplus IV_{128}$

3. For $i = -10$ to -1, $S_{i+1} = \texttt{StateUpdate128}(S_i, m_i)$.

3.3 The encryption of AEGIS-128

After the initialization, at each step of the encryption, a 16-byte plaintext block P_i is used to update the state, and P_i is encrypted as C_i. If the size of the last message block is less than 128 bits, it is padded with 0 bits to a full block, and the padded full block is used to update the state.

1. If the last plaintext block is not a full block, use 0 bits to pad it to 128 bits.

2. For $i = 0$ to $\lceil \frac{msglen}{128} \rceil - 1$, we update the state and perform encryption:

$$\begin{aligned} C_i &= P_i \oplus S_{i,4} \oplus S_{i,3} \oplus S_{i,2} \oplus S_{i,1} \oplus S_{i,0} \ ; \\ S_{i+1} &= \texttt{StateUpdate128}(S_i, P_i) \ ; \end{aligned}$$

3. If the last plaintext block is not a full block, the last ciphertext block needs to be truncated accordingly so that the length of plaintext and ciphertext are the same.

3.4 The finalization of AEGIS-128

After encrypting all the plaintext blocks, we generate the authentication tag using six more steps. The message being used at this stage is part of the state at the end of the encryption, together with the tag length t and the length of the last message block (as pointed out by an anonymous reviewer, the message length can be used here. The reason that we use the length of the last message block is to eliminate the cost of calculating the length of message, although the lenght of message is normally available in most of the applications).

1. Let $tmp = t \parallel (msglen \ \& \ 127) \parallel 0^{112}$, where the tag length t is one byte, $(msglen \ \& \ 127)$ is one byte (the length of the last block. If the last block is a full block, its value is 0), 0^{112} represents 112 bits of 0.

2. For $i = \lceil \frac{msglen}{128} \rceil$ to $\lceil \frac{msglen}{128} \rceil + 5$, $m_i = S_{\lceil \frac{msglen}{128} \rceil, 3} \oplus tmp$;

3. For $i = \lceil \frac{msglen}{128} \rceil$ to $\lceil \frac{msglen}{128} \rceil + 5$, we update the state:
$S_{i+1} = \texttt{StateUpdate128}(S_i, m_i)$.

4. We generate the authentication tag from the state $S_{\lceil \frac{msglen}{128} \rceil + 6}$ as follows:
$T' = \bigoplus_{i=0}^{4} S_{\lceil \frac{msglen}{128} \rceil + 6, i}$.
The authentication tag T is the leftmost t bits of T' .

3.5 The decryption and verification of AEGIS-128

The decryption is similar to the encryption process. It is described below:

1. If the last ciphertext block is not a full block, use 0 bits to pad it to 128 bits.

2. For $i = 0$ to $\lceil \frac{msglen}{128} \rceil - 1$, we update the state and perform encryption:

$$P_i = C_i \oplus S_{i,4} \oplus S_{i,3} \oplus S_{i,2} \oplus S_{i,1} \oplus S_{i,0} \ ;$$
$$S_{i+1} = \texttt{StateUpdate128}(S_i, P_i) \ ;$$

3. If the last plaintext block is not a full block, the last ciphertext block needs to be truncated accordingly so that the length of plaintext and ciphertext are the same.

The finalization in the decryption process is the same as that in the encryption process. We emphasize that if the verification fails, then the ciphertext and the newly generated authentication tag should not be given as output; otherwise, the state of AEGIS-128 is vulnerable to the known-plaintext or chosen ciphertext attack (using a fixed IV). This requirement also applies to AEGIS-256.

4 AEGIS-256

In this section, we describe AEGIS-256. With a 256-bit key and a 256-bit initialization vector, it encrypts and authenticates a message with length up to 2^{64} bits. The authentication tag length is less than or equal to 128 bits. We recommend the use of a 128-bit tag.

4.1 The state update function of AEGIS-256

The state update function updates the 96-byte state S_i with a 16-byte message block m_i. $S_{i+1} = \texttt{StateUpdate256}(S_i, m_i)$ is illustrated as follows:

$$S_{i+1,0} = AESRound(S_{i,5}, S_{i,0} \oplus m_i);$$
$$S_{i+1,1} = AESRound(S_{i,0} \oplus m_i, S_{i,1});$$
$$S_{i+1,2} = AESRound(S_{i,1}, S_{i,2});$$
$$S_{i+1,3} = AESRound(S_{i,2}, S_{i,3});$$
$$S_{i+1,4} = AESRound(S_{i,3}, S_{i,4});$$
$$S_{i+1,5} = AESRound(S_{i,4}, S_{i,5});$$

4.2 The initialization of AEGIS-256

The initialization of AEGIS-256 consists of loading the key and IV into the state, and running the cipher for 16 steps with the key and IV being used as message.

1. Load the key and IV into the state as follows:

$$S_{-16,0} = IV_{256,0};$$
$$S_{-16,1} = IV_{256,1};$$
$$S_{-16,2} = const_1;$$
$$S_{-16,3} = const_0;$$
$$S_{-16,4} = K_{256,0} \oplus const_0;$$
$$S_{-16,5} = K_{256,1} \oplus const_1;$$

2. For $i = -4$ to -1,
   $m_{4i} = K_{256,0}$;
   $m_{4i+1} = K_{256,1}$;
   $m_{4i+2} = K_{256,0} \oplus IV_{256,0}$;
   $m_{4i+3} = K_{256,1} \oplus IV_{256,1}$.
3. For $i = -16$ to -1, $S_{i+1} = \texttt{StateUpdate256}(S_i, m_i)$.

4.3 The encryption of AEGIS-256

After the initialization, at each step of the encryption, a 16-byte plaintext block P_i is used to update the state, and P_i is encrypted as C_i. If the size of the last message block is less than 128 bits, it is padded with 0 bits to a full block, and the padded full block is used to update the state.

1. If the last plaintext block is not a full block, use 0 bits to pad it to 128 bits.
2. For $i = 0$ to $\lceil \frac{msglen}{128} \rceil - 1$, we update the state and perform encryption:

   $C_i = P_i \oplus S_{i,5} \oplus S_{i,4} \oplus S_{i,3} \oplus S_{i,2} \oplus S_{i,1} \oplus S_{i,0}$;
   $S_{i+1} = \texttt{StateUpdate256}(S_i, P_i)$;
3. If the last plaintext block is not a full block, the last ciphertext block needs to be truncated accordingly so that the length of plaintext and ciphertext are the same.

4.4 The finalization of AEGIS-256

After encrypting all the plaintext blocks, we generate the authentication tag using seven more steps. The message being used at this stage is part of the state at the end of the encryption, together with the tag length t and the length of the last message block.

1. Let $tmp = t \parallel (msglen \ \& \ 127) \parallel 0^{112}$, where the tag length t is one byte, $(msglen \ \& \ 127)$ is one byte (the length of the last block. If the last block is full block, the value is 0), 0^{112} represents 112 bits of 0.
2. For $i = \lceil \frac{msglen}{128} \rceil$ to $\lceil \frac{msglen}{128} \rceil + 5$, $m_i = S_{\lceil \frac{msglen}{128} \rceil, 3} \oplus tmp$;
3. For $i = \lceil \frac{msglen}{128} \rceil$ to $\lceil \frac{msglen}{128} \rceil + 5$, we update the state:
   $S_{i+1} = \texttt{StateUpdate256}(S_i, m_i)$.
4. We generate the authentication tag from the state $S_{\lceil \frac{msglen}{128} \rceil + 6}$ as follows:
   $T' = \bigoplus_{i=0}^{5} S_{\lceil \frac{msglen}{128} \rceil + 6, i}$.
   The authentication tag T is the leftmost t bits of T' .

5 The Security of AEGIS

The following requirements should be satisfied in order to use AEGIS securely.

1. Each key should be randomly generated.
2. Each key and IV pair should not be used to encrypt and authenticate more than one message; and each key and IV pair should not be used with two different tag lengths.
3. If the verification fails, the decrypted plaintext and the wrong authentication tag should not be given as output.

If the above requirements are satisfied, we have the following security claims:

Claim 1. The key cannot be recovered faster than exhaustive key search.

Claim 2. The success rate of a forgery attack is 2^{-t}. If the forgery attack is repeated n times, then the success rate of a forgery attack is about $n \times 2^{-t}$.

Claim 3. The state cannot be recovered faster than exhaustive key search if the forgery attack is not allowed to be repeated for the same key and IV pair (i.e., the ciphertext and authentication tag are dropped if a key and IV pair is reused).

Claim 4. The state can be recovered faster than exhaustive key search if the forgery attack is repeated more than 2^t times for the same key and IV pair. We recommend the use of 128-bit tag length for AEGIS. (Note that with 128-bit tag, the state of AEGIS-256 can be recovered faster than exhaustive key search if an attacker is allowed to modify the message and/or the tag for more than 2^{128} times for the same key and IV pair.)

5.1 The security of the initialization

The difference in IV is the main threat to the security of the initialization of AEGIS. The difference in IV would eventually propagate into the ciphertexts, and thus it is possible to apply a differential attack against AEGIS. In AEGIS-128, there are 50 AES round functions (10 steps) in the initialization. If there is a difference in IV, the difference would pass through more than 10 AES round functions. In AEGIS-256, there are 96 AES round functions (16 steps) in the initialization. If there is a difference in IV, the difference would pass through more than 16 AES round functions. Furthermore, in order to prevent the difference in the state being eliminated completely in the middle of the initialization, we inject the IV difference repeatedly into the state (5 and 8 times into the state of AEGIS-128 and AEGIS-256, respectively). We expect that the differential attack against the initialization would be more expensive than exhaustive key search.

5.2 The security of encryption

The state update function involves five AES round functions in AEGIS-128, and six AES round functions in AEGIS-256. We should ensure that IV is not reused for the same key; otherwise, the states of AEGIS can be recovered easily with either known-plaintext attacks or chosen plaintext attacks. For example, if we re-use the IV and inject a difference in P_i, the difference would propagate into C_{i+1}, and part of the state can be attacked by analyzing the difference pair $(\Delta P_i, \Delta C_{i+1})$. If an authenticated encryption algorithm remains secure for re-used IVs, we expect that such algorithm can only be as fast as a block cipher. The reason is that: once the IV is re-used, the attacks against block cipher can be applied to attack the state.

If IV is used only once for each key, it is impossible to apply a differential attack to the encryption process. And it is extremely difficult to apply the linear attack (or correlation attack) to recover the secret state since the state of AEGIS is updated in a nonlinear way. In general, it would be difficult to apply any statistical attack to recover the secret state due to the nonlinear state update function (the statistical correlation between any two states vanishes quickly as the distance between them increases).

When the nonlinear state update functions are used in the stream cipher design, the main threat against the state update function is the guess-and-determine attack or solving the nonlinear equations involving several plaintext-ciphertext blocks. In AEGIS, each plaintext-ciphertxt block leaks 16-byte state information. For AEGIS-128, it requires at least five plaintext-ciphertext blocks to recover the secret state, and it involves at least 20 AES round functions in the equations. For AEGIS-256, it requires at least six plaintext-ciphertext blocks to recover the secret state, and it involves at least 30 AES round functions. In AEGIS, large state is used and the output function is carefully chosen so that each keystream block would involve all the state elements, we expect that it is difficult to launch the guess-and-determine attack. It would be very difficult to solve these equations since we are dealing with a very limited number of nonlinear equations. (If we introduce more nonlinear equations, we are also increasing the complexity of the equations. This is different from the algebraic attack against block cipher in which there are heavily overdefined equations for a fixed function.) Due to the large number of AES round functions being involved, we expect that the state cannot be recovered faster than exhaustive key search.

5.3 The security of message authentication

There are two main approaches to attack a MAC algorithm. One approach is to recover the secret key or secret state, another approach is to introduce or detect an internal state collision. Besides these two approaches, when we analyze the security of message authentication, we need to consider that the encryption in AEGIS would affect the security of message authentication.

Recovering key or state. From Sect. 5.1, we expect that the secret key cannot be recovered faster than exhaustive search by attacking the initialization. From Sect. 5.2, we expect that the state cannot be recovered faster than exhaustive search by attacking the encryption process if the IV is used only once. Similarly, we expect that the state cannot be recovered faster than exhaustive search by attacking the tag generation process if IV is not reused.

An attacker can still inject a difference into the state in the tag verification process if the forgery is allowed to be repeated for multiple times for the same key and IV pair. As stated in the Security Claim 3, if the ciphertext and authentication tag are dropped when the same key and IV pair is reused, the differential attack against the state cannot be performed since the attacker can try only once for each key and IV pair. However, if the forgery attack can be performed for more than 2^t times for the same key and IV pair, the decrypted plaintext would be known to the attacker with probability 2^{-t} (if the verification is successful). It means that the same key and IV are used for more than once in the decryption of messages, and the decrypted messages may be known to the attacker. It thus becomes possible to recover the state. We thus recommend the use of 128-bit tag length in AEGIS, so that recovering the state requires at least 2^{128} forgery attempts.

We notice that for AEGIS-256, the security of the state cannot reach the theoretical maximum level (2^{256}). However, we believe that in practice, trying the verification process for more than 2^{128} times in order to recover a state is not practical.

Internal collision. A powerful attack against MAC is to introduce and detect internal collision. A general approach based on birthday attack was given by Preneel and van Oorschot [19]: an internal collision can be detected after a key is used to generate the authentication tags of about $2^{n/2}$ chosen messages, where n is the state size and tag length in bits. The internal collision can be exploited to forge the tags of new messages. The birthday attack was later further applied to other MACs [24]. Another approach to create internal collision is through differential cryptanalysis. Suppose that the difference cancellation in the state occurs with probability 2^{-a}, then we can detect an internal collision after a secret key is used to generate the tags of those 2^a message pairs. The resulting internal collision can be used to forge the tags of new messages.

Note that in AEGIS, a unique key and IV pair is used to protect each message, so the states for all the messages are completely different, no matter whether the messages are the same or not. Since the state size is sufficiently large (640-bit state in AEGIS-128 and 768-bit state in AEGIS-256), we expect that the state collision occurs with negligible probability.

An attacker can still inject a difference into the state in the decryption and tag verification process by modifying the ciphertext or tag length. We notice that the first difference being injected into ciphertext would pass through five round functions without being affected by another ciphertext difference in AEGIS-128. There are more than 26 active Sboxes being involved and we expect that

the difference would become sufficiently random after passing through five AES round functions. Furthermore, when a difference passes through five AES round functions, the difference would be injected into each 16-byte elements in the state. And the difference cancellation would involve at least 52 active Sboxes (26 active Sboxes for generating the difference patterns, and 26 active Sboxes for difference cancellation), and with probability less than $2^{-6\times 52} = 2^{-312}$. We notice that if we ensure that if the verification fails, the authentication tag is not given as output: detecting a state collision in the verification process requires at least 2^{312} modifications to the ciphertext. Attacking AEGIS-256 is more difficult since it involves larger state and more AES round functions..

We now analyze whether the noninvertible AEGIS state update function affects the security of the authentication of AEGIS. In AEGIS, a difference in the state would be eliminated even if there is no difference being introduced to cancel it. However, it would only happen if the difference in every 16-byte element is able to eliminate the difference in the next element after passing through an AES round function. It means that at least 26 active Sboxes are involved in this difference elimination process in AEGIS-128, and generating these particular differences in the state involves more than 26 additional active Sboxes. We consider that this type of weak state difference has negligible effect on the security of the authentication of AEGIS.

Exploiting the ciphertext for forgery. At the finalization stage, we use part of the state at the end of the encryption as message in order to prevent the ciphertext blocks from being used as authentication tags in the forgery attack.

The analysis given above shows that the authentication of AEGIS is very strong.

5.4 Other Attacks

There are weak states in AEGIS since no pseudorandom constants are introduced into the state in the encryption process (if there is little change to plaintext blocks). In a first type of weak states all the 16-byte elements in a state are equal: then all the 16-byte elements in the next state are equal (if the message block is set to 0). However, there are only 2^{128} such states, so this type of weak state appears with probabilities 2^{-512} and 2^{-640} for AEGIS-128 and AEGIS-256, respectively. In a second type of weak state the four columns in each 16-byte element are equal and every 16-byte element has this property: this property would then remain in the next state (if the message block also has such property). However, there are only $2^{32\times 5} = 2^{160}$ such states in AEGIS-128 and $2^{32\times 6} = 2^{192}$ such states in AEGIS-256, so we expect that this type of weak state appears with probabilities 2^{-480} and 2^{-608}. At the initialization stage, we introduce a 16-byte constant so that these weak states would not be generated from weak keys.

We believe that related-key attack is not a threat to this type of authenticated encryption algorithm since a user can always use different IVs with the same key. Anyway, we think that AEGIS is strong enough against the related-key attack

since the key is used in every step in the initialization to update the state, and there are 10 and 16 steps in the initialization of AEGIS-128 and AEGIS-256, respectively. Furthermore, the constants being used to initialize the state would be helpful to resist the slide attack.

6 The Applications of AEGIS

AEGIS is useful for the protection of data transmission and data storage. The amount of computation in AEGIS is about half of AES for long messages. For network communication, the packet header should not be encrypted but should be authenticated. It is called the Authenticated-Encryption with Associated-Data (AEAD). AEGIS is suitable for the AEAD application since it is straightforward to authenticate the whole message while leaving part of the message unencrypted.

It is straightforward to use AEGIS as an asynchronous stream cipher since we can simply discard the generation of the message authentication tag. AEGIS can be converted to a synchronous stream cipher by generating keystream without the plaintext being injected into the state. We expect that the security of this synchronous stream cipher is as strong as the security of the AEGIS encryption.

6.1 Message authentication codes

AEGIS can be used as a message authentication code. There are two types of message authentication codes: MAC with nonce (named as MAC-wn) and MAC without nonce (named as MAC-won). The design and analysis of MAC-wn and MAC-won are different. In general, MAC-wn is more efficient than MAC-won for the same security level since the nonce being used in MAC-wn provides completely different initial states for different messages. However, the security of MAC-wn is sensitive to the uniqueness of nonce.

Using AEGIS as MAC-wn is straightforward since we can simply remove the encryption part. Furthermore, if we use AEGIS as MAC, the amount of operations at each step can be reduced since there is no ciphertext information available to the attacker. The state in AEGIS can be reduced to four 128-bit elements, then it requires only 4 AES round functions to process a 16-byte message block. If we use AEGIS with eight 128-bit elements, and use two 16-byte message blocks to update the first and fifth elements in the state at each step, we can obtain a MAC that can utilize the 8-stage pipeline of AES-NI in the Intel Sandy Bridge microprocessor. Using AEGIS as MAC-won requires the redesign of the algorithm.

In the following, we present the design of several MACs using the AEGIS design approach. The names of MACs are given in the following format: AEGIS-XXX-Y-MAC-ZZZ, where XXX indicates the key size, Y indicates the number of 128-bit elements in state, ZZZ indicates whether it is WN or WON type of MAC. We mainly illustrate the MACs with 128-bit key (the MACs with 256-bit key will be proposed after performing rigorous security analysis).

AEGIS-128-5-MAC-wn. The design is almost the same as AEGIS-128, except that there is no encryption.

AEGIS-256-6-MAC-wn. The design is almost the same as AEGIS-256, except that there is no encryption.

AEGIS-128-4-MAC-wn. In this design, we use 64-byte state. The design is given in Appendix A.

AEGIS-128-4-MAC-won. In this design, there is no nonce. We set the nonce in AEGIS-128-4-MAC-wn to 0, and increase the step number in the finalization in AEGIS-128-4-MAC-wn from 6 to 10.

AEGIS-128-8-MAC-wn. In this design, we use 128-byte state, and at each step, we process two 16-byte message blocks. The design is given in Appendix B. The idea of using large state size so as to fully utilize the Sandy Bridge processor was given by anonymous reviewer. We did not use this idea in the design of authenticated encryption (due to security concern), but we found that this idea is useful for designing a MAC algorithm that is very efficient for long messages.

AEGIS-128-8-MAC-won. In this design, there is no nonce. We set the nonce in AEGIS-128-8-MAC-wn to 0, and increase the step number in the finalization in AEGIS-128-8-MAC-wn from 6 to 10.

7 The Performance of AEGIS

To process a 16-byte message block, AEGIS-128 and AEGIS-256 use five and six AES round functions, respectively. The computational cost of AEGIS is about half that of AES. AEGIS is very efficient when it is implemented using the AES new instruction (AES-NI). With parallel AES functions at each step, AEGIS can fully utilize the 3-stage pipeline in AES-NI in Intel Westmere processor, and can utilize most of th 8-state pipeline in the AES-NI in the Intel Sandy Bridge processor.

We implemented AEGIS in C code using the AES instruction. We tested the speed on the Intel Core i5-2540M 2.6GHz processor (Sandy Bridge) running 64-bit Ubuntu 11.04. The turbo boost is turned off, so the CPU runs at 2.6GHz in the experiment. The compiler being used is gcc 4.5.2, and the options "-O3 -msse2 -maes -mavx" are used. The test is performed by processing a message repeatedly, and printing out the final message. To ensure that the tag generation is not removed in the compiler optimization process, we use the tag as the IV for processing the next message. To ensure that the tag verification is not removed during the compiler optimization process, we use the tag computed from the

ciphertext as the IV for processing the next message, and we sum up the number of failed verifications and print out the final result.

The performance is given in Table 1. For 65536-byte messages, the speed of AEGIS-128 and AEGIS-256 is about 0.64 cpb and 0.71 cpb, respectively. It takes about 175 and 238 clock cycles for AEGIS-128 and AEGIS-256 to process a one-byte message. Note that in our test, we use the fixed 128-bit tag length, so as to save the cost of tag generation and verification. More speed comparison will be provided later.

Table 1. The speed comparison (in cycles per byte) for different message length

	1B	16B	64B	512B	1024B	4096B	65536B	IPI[a]
AEGIS-128(EA[b])	149	9.20	2.74	0.91	0.79	0.67	0.64	1.03
AEGIS-128(DV[c])	175	9.30	2.85	0.99	0.83	0.69	0.64	1.08
AEGIS-256(EA)	228	12.54	3.64	1.08	0.88	0.71	0.66	1.19
AEGIS-256(DV)	238	12.88	4.00	1.16	0.93	0.76	0.71	1.27
AEGIS-128-5-MAC-wn	148	9.19	2.67	0.82	0.69	0.59	0.56	0.95
AEGIS-128-4-MAC-wn	149	9.33	2.65	0.82	0.70	0.59	0.56	0.95
AEGIS-128-4-MAC-won	182	11.41	3.17	0.88	0.72	0.60	0.56	1.00
AEGIS-128-8-MAC-wn	197	12.2	2.88	0.64	0.48	0.37	0.32	0.77
AEGIS-128-8-MAC-won	234	14.56	3.56	0.73	0.52	0.38	0.33	0.85
AEGIS-256-6-MAC-wn	228	12.40	3.48	0.98	0.77	0.62	0.56	1.08

[a] Internet Performance Index (IPI) is a weighted average of timings for messages of 44 bytes (5%), 552 bytes (15%), 576 bytes (20%) and 1500 bytes (60%) [12]. However, this definition may not be the exact contemporary, real-word distribution of packet lengths [10].
[b] EA: Encryption-Authentication
[c] DV: Decryption-Verification

We need to mention that when the last message block is a partial block, some extra operation is needed to process it, so the speed for 16-byte messages is not simply 1/16 of the speed for 1-byte messages. And the cost of processing a partial block in decryption is slightly more expensive than that in encryption (it is related to the access of XMM registers when we use AES-NI).

In Table 1, the speed of AEGIS-128 is only slightly slower than AEGIS-256 for long messages, although the computational cost of AEGIS-256 is about 20% more than that of AEGIS-128. The reason is that on the Sandy Bridge microprocessor, AES-NI is implemented with eight-stage pipeline, and both AEGIS-128 and AEGIS-256 do not fully utilize the pipeline, so the performance of AEGIS-128 is close to that of AEGIS-256. On the Intel Westmere microprocessors, we expect that the speed of AEGIS-256 is about 20% slower than that of AEGIS-128.

8 Design Rationale

AEGIS is designed to achieve high performance and strong security. To achieve high performance, we use the AES round function. The AES round functions are now implemented in the latest Intel and AMD microprocessors as Intel AES New Instructions (AES-NI). AES-NI is a very efficient method for achieving diffusion and confusion on a modern microprocessor. In our design of AEGIS, we will use several parallel AES round functions in each step in order to use most of the stages in AES instruction. The AES instructions are implemented on Intel Westmere (06_25H, 06_2CH, 06_2FH) microprocessors with three-stage pipeline (6 clock cycles), and are implemented on Intel Sandy Bridge (06_2AH) microprocessors with eight-stage pipeline (8 clock cycles) [7]. We are not sure how the AES-NI would be implemented on the future microprocessors, but having several parallel AES round functions in AEGIS would significantly improve its performance.

To achieve strong encryption security, we try to ensure that the IV difference is randomized at the initialization stage, and the state cannot be recovered from the ciphertext. There are 10 steps and 16 steps in the initialization of AEGIS-128 and AEGIS-256, so we expect that the initialization of AEGIS is strong. To ensure that the state cannot be recovered from the ciphertext faster than brute force key search, we use large state in the design, so as to ensure that at least 20 and 30 AES round functions are involved in the state recovery attack against AEGIS-128 and AEGIS-256.

To achieve strong authentication security, we try to ensure that any difference being introduced into the state would result in a particular difference with sufficiently small probability, so that it is difficult to launch a forgery attack. Our design is partly motivated by the design of Pelican MAC [4]. In Pelican MAC, a difference would pass through 4 AES round functions before meeting with another difference, so at least 25 active Sboxes are involved. The security proof against differential forgery attack is very simple for Pelican MAC (however, there is a birthday type attack against Pelican MAC due to its 128-bit size [24]). In AEGIS, the first difference in the state would pass through at least 5 AES round functions before being affected by another difference. In addition, when a difference passes through AES round functions, the difference would be injected into every element in the state, it becomes thus more difficult to eliminate it.

9 Conclusion

In this paper, we introduced a new dedicated authenticated encryption algorithm AEGIS. It requires about half of the computation of AES. AEGIS is fast for both short and long messages, and it is extremely fast on the microprocessors with the AES instruction set.

We have performed a security analysis of the encryption and authentication of AEGIS. Our analysis shows that the encryption and authentication of AEGIS are very secure. However, the security of any new cipher requires the research

from the cryptographic community. We welcome the security analysis of this new authenticated encryption algorithm.

Acknowledgement. We would like to thank the anonymous reviewers for their helpful comments, especially the idea of fully utilizing the 8-stage pipeline of AES-NI on the Sandy Bridge processor by increasing the state size (we used this idea in the MAC design, but not in the encryption design due to security concern).

References

1. 3GPP. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3, Document 1 128-EEA3 and 128-EIA3 specification, The 3rd Generation Partnership Project (3GPP). Available online at: http://www.gsmworld.com/our-work/programmes-andinitiatives/fraud-and-security/gsm_security_algorithms.htm
2. M. Agren, M. Hell, T. Johansson, W. Meier. Grain-128a: A New Version of Grain-128 with Optional Authentication. *International Journal of Wireless and Mobile Computing 2011*, Vol. 5, No.1 pp. 48-59.
3. M. Bellare and C. Namprempre. Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm. *Advances in Cryptology - Asiacrypt 2000*, pp. 531-545.
4. J. Daemen, V. Rijmen. The Pelican MAC Function. IACR Cryptology ePrint Archive 2005: 88 (2005).
5. N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks and T. Kohno. Helix, Fast Encryption and Authentication in a Single Cryptographic Primitive. *Fast Software Encryption–FSE 2003*, LNCS 2887, pp. 330-346.
6. V. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. *Fast Software Encryption – FSE 2001*, LNCS vol. 2355, pp. 92–108, 2001.
7. Intel. Intel 64 and IA-32 Architectures Optimization Reference Manual. Available at http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf
8. C. Jutla, Encryption modes with almost free message integrity. *Advances in Cryptology – EUROCRYPT 2001*, LNCS vol. 2045, Springer, pp. 529–544, 2001.
9. J. Katz and M. Yung. Unforgeable encryption and adaptively secure modes of operation. *Fast Software Encryption–FSE 2000*, LNCS vol. 1978, 2001.
10. T. Krovetz, P. Rogaway. The Software Performance of Authenticated-Encryption Modes. *Fast Software Encryption – FSE 2011*, pp. 306-327.
11. T. Krovetz, P. Rogaway. Authenticated-Encryption Software Performance: Comparison of CCM, GCM, and OCB. Available at http://www.cs.ucdavis.edu/ rogaway/ocb/performance/
12. D. McGrew and J. Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. INDOCRYPT 2004, LNCS vol. 3348, Springer, pp. 343-355, 2004.
13. National Institute of Standards and Technology. Advanced Encryption Standard. FIPS 197.
14. National Institute of Standards and Technology. Recommendation for Block Cipher Modes of Operation. NIST special publication 800-38A, 2001 Edition.

15. National Institute of Standards and Technology. The Keyed-Hash Message Authentication Code (HMAC). FIPS PUB 198.
16. National Institute of Standards and Technology. Recommendations for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. NIST special publication 800-38C, May 2004.
17. National Institute of Standards and Technology. Recommendations for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST special publication 800-38D, November 2007.
18. National Institute of Standards and Technology. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST special publication 800-38B.
19. B. Preneel, P. C. van Oorschot. On the Security of Iterated Message Authentication Codes. *IEEE Transactions on Information Theory 45(1)*, 188–199 (1999).
20. P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. ACM Trans. on Information and System Security, 6(3), pp. 365–403, 2003. Earlier version, with T. Krovetz, in CCS 2001.
21. P. Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. *ASIACRYPT 2004*, LNCS vol. 3329, Springer, pp. 16–31, 2004.
22. D. Whiting, B. Schneier, S. Lucks and F. Muller. Phelix: Fast Encryption and Authentication in a Single Cryptographic Primitive. eSTREAM, ECRYPT Stream Cipher Project Report 2005/027.
23. H. Wu, B. Preneel. Differential-Linear Attacks Against the Stream Cipher Phelix. *FSE 2007*, pp. 87–100.
24. Z. Yuan, W. Wang, K. Jia, G. Xu, X. Wang. New Birthday Attacks on Some MACs Based on Block Ciphers. *CRYPTO 2009*, pp. 209–230.

A AEGIS-128-4-MAC-wn

In this section, we describe AEGIS-128-4-MAC-wn. With a 128-bit key and a 128-bit initialization vector, it authenticates a message with length up to 2^{64} bits. The authentication tag length is less than or equal to 128 bits. We recommend the use of 128-bit tag.

A.1 The state update function of AEGIS-128-4-MAC-wn

The state update function updates the 64-byte state S_i with a 16-byte message block m_i. $S_{i+1} = \texttt{StateUpdate1284}(S_i, m_i)$ is given as follows:

$$
\begin{aligned}
S_{i+1,0} &= AESRound(S_{i,3}, S_{i,0} \oplus m_i);\\
S_{i+1,1} &= AESRound(S_{i,0} \oplus m_i, S_{i,1});\\
S_{i+1,2} &= AESRound(S_{i,1}, S_{i,2});\\
S_{i+1,3} &= AESRound(S_{i,2}, S_{i,3});
\end{aligned}
$$

A.2 The initialization of AEGIS-128-4-MAC-wn

The initialization of AEGIS-128-4-MAC-wn consists of loading the key K, IV into the state, and running the cipher for 10 steps with the key and IV being used as message.

1. Load the key and IV into the state as follows:

$$S_{-10,0} = IV_{128};$$
$$S_{-10,1} = const_1;$$
$$S_{-10,2} = const_0;$$
$$S_{-10,3} = K_{128} \oplus const_0;$$

2. For $i = -5$ to -1, $m_{2i} = K_{128}$, $m_{2i+1} = K_{128} \oplus IV_{128}$

3. For $i = -10$ to -1, $S_{i+1} = \texttt{StateUpdate1284}(S_i, m_i)$.

A.3 Processing the message

After the initialization, at each step, a 16-byte message block m_i is used to update the state. If the size of the last message block is less than 128 bits, it is padded with 0 bits to a full block, and the padded full block is used to update the state.

1. If the last plaintext block is not a full block, use 0 bits to pad it to 128 bits.

2. For $i = 0$ to $\lceil \frac{msglen}{128} \rceil - 1$, we update the state and perform encryption:

$$S_{i+1} = \texttt{StateUpdate1284}(S_i, P_i) \ ;$$

A.4 The finalization of AEGIS-128-4-MAC-wn

After encrypting all the plaintext blocks, we generate the authentication tag using six more steps. The message being used at this stage is part of the state at the end of the encryption, together with the tag length t and the length of the last message block.

1. Let $tmp = t \parallel (msglen \ \& \ 127) \parallel 0^{112}$, where the tag length t is one byte, $(msglen \ \& \ 127)$ is one byte (the length of the last block. If the last block is a full block, its value is 0), 0^{112} represents 112 bits of 0.

2. For $i = \lceil \frac{msglen}{128} \rceil$ to $\lceil \frac{msglen}{128} \rceil + 5$, $m_i = S_{\lceil \frac{msglen}{128} \rceil, 2} \oplus tmp$;

3. For $i = \lceil \frac{msglen}{128} \rceil$ to $\lceil \frac{msglen}{128} \rceil + 5$, we update the state:
$S_{i+1} = \texttt{StateUpdate1284}(S_i, m_i)$.

4. We generate the authentication tag from the state $S_{\lceil \frac{msglen}{128} \rceil + 6}$ as follows:
$T' = \bigoplus_{i=0}^{3} S_{\lceil \frac{msglen}{128} \rceil + 6, i}$.
The authentication tag T is the leftmost t bits of T' .

B AEGIS-128-8-MAC-wn

In this section, we describe AEGIS-128-8-MAC-wn. With a 128-bit key and a 128-bit initialization vector, it authenticates a message with length up to 2^{64} bits. The authentication tag length is less than or equal to 128 bits. We recommend the use of 128-bit tag.

B.1 The state update function of AEGIS-128-8-MAC-wn

The state update function updates the 128-byte state S_i with two 16-byte message blocks m_{2i} and m_{2i+1}. $S_{i+1} = \texttt{StateUpdate1288}(S_i, m_{2i}, m_{2i+1})$ is given as follows:

$$
\begin{aligned}
S_{i+1,0} &= AESRound(S_{i,7}, S_{i,0} \oplus m_{2i});\\
S_{i+1,1} &= AESRound(S_{i,0} \oplus m_{2i}, S_{i,1});\\
S_{i+1,2} &= AESRound(S_{i,1}, S_{i,2});\\
S_{i+1,3} &= AESRound(S_{i,2}, S_{i,3});\\
S_{i+1,4} &= AESRound(S_{i,3}, S_{i,4} \oplus m_{2i+1});\\
S_{i+1,5} &= AESRound(S_{i,4} \oplus m_{2i+1}, S_{i,5});\\
S_{i+1,6} &= AESRound(S_{i,5}, S_{i,6});\\
S_{i+1,7} &= AESRound(S_{i,6}, S_{i,7});
\end{aligned}
$$

B.2 The initialization of AEGIS-128-8-MAC-wn

The initialization of AEGIS-128-8-MAC-wn consists of loading the key K, IV into the state, and running the cipher for 10 steps with the key and IV being used as message.

1. Load the key and IV into the state as follows:

$$
\begin{aligned}
S_{-10,0} &= IV_{128};\\
S_{-10,1} &= const_1;\\
S_{-10,2} &= const_0;\\
S_{-10,3} &= K_{128} \oplus const_0;\\
S_{-10,4} &= IV_{128};\\
S_{-10,5} &= const_0;\\
S_{-10,6} &= const_1;\\
S_{-10,7} &= K_{128} \oplus const_1;
\end{aligned}
$$

2. For $i = -20$ to -1, $m_{2i} = K_{128}$, $m_{2i+1} = K_{128} \oplus IV_{128}$

3. For $i = -10$ to -1, $S_{i+1} = \texttt{StateUpdate1288}(S_i, m_{2i}, m_{2i+1})$.

B.3 Processing the message

After the initialization, at each step, two 16-byte message blocks m_{2i} and m_{2i+1} are used to update the state.

1. If the size of the message is not the multiple of 256 bits, it is padded with 0 bits to the multiple of 256 bits.

2. For $i = 0$ to $\lceil \frac{msglen}{256} \rceil - 1$, we update the state and perform encryption:

$$S_{i+1} = \texttt{StateUpdate1288}(S_i, m_{2i}, m_{2i+1}) \ ;$$

B.4 The finalization of AEGIS-128-8-MAC-wn

After encrypting all the plaintext blocks, we generate the authentication tag using six more steps. The message being used at this stage is part of the state at the end of the encryption, together with the tag length t and the length of the last message block.

1. Let $tmp = t \parallel (msglen \ \& \ 255) \parallel 0^{112}$, where the tag length t is one byte, $(msglen \ \& \ 255)$ is one byte (the length of the last two blocks. If the last blocks are full blocks, its value is 0), 0^{112} represents 112 bits of 0.

2. For $i = \lceil \frac{msglen}{256} \rceil$ to $\lceil \frac{msglen}{256} \rceil + 5$, $m_{2i} = S_{\lceil \frac{msglen}{256} \rceil, 2} \oplus tmp$, $m_{2i+1} = S_{\lceil \frac{msglen}{256} \rceil, 6} \oplus tmp$.

3. For $i = \lceil \frac{msglen}{256} \rceil$ to $\lceil \frac{msglen}{256} \rceil + 5$, we update the state:
$S_{i+1} = \texttt{StateUpdate1288}(S_i, m_{2i}, m_{2i+1})$.

4. We generate the authentication tag from the state $S_{\lceil \frac{msglen}{256} \rceil + 6}$ as follows:
$T' = \bigoplus_{i=0}^{7} S_{\lceil \frac{msglen}{256} \rceil + 6, i}$.
The authentication tag T is the leftmost t bits of T' .

Hash-CFB

Authenticated Encryption Without a Block Cipher – Extended Abstract –

Christian Forler[1], David McGrew[2], Stefan Lucks[1], and Jakob Wenzel[1]

[1] Bauhaus-University Weimar, Germany, [2] Cisco Systems, USA
{Christian.Forler, Stefan.Lucks, Jakob.Wenzel}@uni-weimar.de
mcgrew@cisco.com

Abstract. This paper presents a mode of operation for authenticated encryption – with the very unusual property of employing a hash function as the underlying primitive, rather than a block cipher, used in so many authenticated encryption modes. This research has been motivated by the challenge to fit secure cryptography into constrained devices – some of these devices have to use a hash function, anyway, and the challenge is to avoid the usage of an additional block cipher. Beyond that, our mode has some unique security features, namely some form of resistance against side-channel attacks, and a misuse-resistant and failure-friendly authentication.

Keywords: authenticated encryption, hash function, provable security, internet of things

1 Introduction

There are plenty of block cipher modes of operations for authenticated encrcyption mentioned in [2] and [4]. So what is the additional value our mode provides over all the other modes that are already there?

One thing is – our mode does not employ a block cipher – a hash function can do as well. A literature search seems to indicate that this property is unique, since essentially all the authors of authenticated encryption modes talk about using a "block cipher". Note that [1] employs the compression function *inside* the hash function, not the raw hash function itself, and is only applicable to a very specific hash function construction.

However, several modes for authenticated encryption employ a block cipher in a way that even authenticated decryption does not need the block cipher's decryption operation. Thus, for a secret key, the block cipher can be seen as a random function $\{0,1\}^\nu \to \{0,1\}^\nu$, rather than a

random permutation over ν-bit blocks. In principle, this allows the ν-bit block cipher to be replaced by a ν-bit hash function, restricted to $\nu+k$ bit inputs, to hash a secret k-bit key and a ν-bit "plaintext block" into a ν-bit "ciphertext block". Under reasonable assumptions on the hash function, the proof of security for the block cipher mode can be maintained.

1.1 Security Properties

We preferred to define a hash function mode from the scratch. As it turned out, this allowed us to achieve and prove some rather unique security features for our proposed mode:

1. *Misuse-resistant authenticity:* It is standard for an authenticated encryption scheme to claim (and prove) security against *nonce-respecting* adversaries. Almost always, security breaks apart if nonces are ever reused [2]. While the privacy of Hash-CFB only holds for nonce-respecting adversaries (unlike the scheme presented in [2]), its authenticity does not depend on unique nonces.
2. *Failure-friendly authenticity:* The security proofs of most modes of operation assume the underlying block cipher to behave like a random permutation or a random function. If this fails, the mode is likely to be insecure. While the privacy of Hash-CFB requires the hash function under a secret key to behave like a random function, authenticity can be established even for failing pseudorandomness. We aim to prove authenticity under a much weaker unpredictability assumption.
3. *Resistance against side-channel attacks:* Resistance against side-channel attacks is a matter of the implementation of a cryptosystem. However, the design of a cryptosystem can contribute to ease with which its implementations can defend against side-channel attacks. Hash-CFB does so, by using the secret key in a single hash operation, once for the encryption or decryption of a message of arbitrary length. See Section 3.3 for further elaboration on side-channel resistance.

1.2 The Core Construction

Let Hash : $\{0,1\}^* \rightarrow \{0,1\}^\nu$ be a ν-bit hash function. Given some *associated data* D, we compute

$$T[0] := \mathrm{Hash}(D).$$

Given a long-term *key* K and a *nonce* N, we compute a short-term key

$$S := Hash(K \mid\mid N \mid\mid 0),$$

where "||" denotes the concatenation of strings. Given a message $M = M[1], \ldots, M[n]$ of ν-bit blocks, we compute some "chaining value"

$$T[i] := \text{Hash}(S \mid\mid T[i-1] \mid\mid C[i-1] \mid\mid 1)$$

and the ciphertext

$$C[i] := T[i] \oplus M[i]$$

for $i \in \{1, \ldots, n\}$.

The *authentication tag* is computed as

$$T := \text{Hash}(S \mid\mid T[n] \mid\mid C[n] \mid\mid 2).$$

Observe the "domain separation trick": when computing the authentication tag T, our hash input ends with a "2", while the hash input for the chained encryption operations ends with a "1". Furthermore, when computing the short-term key S, our hash input ends with a "0".

This leaves open many details, such as

- How can we deal with messages M whose length is not a multiple of the hash size ν?
- How large should K and N be?
- Can their sizes be variable, and, if yes, how can we defend against attacks shifting the "border" between K and N?
- How can we use the properties of the underlying hash function most efficiently?

The last point becomes clear when one begins studying the details of practical hash functions: the longer the input, the more often the internal compression function has to be evaluated. If we use a hash function taking message blocks of more than 3ν bits for each call of the internal compression function, we can essentially apply the core construction, filling in the details we left open.

In practice, few hash functions have such a large native message block size, so we will have to tweak our construction, in order to reduce the input size:

1. Replace "$S \mid\mid T[i-1]$" by "$S \oplus T[i-1]$", thus reducing the input size by ν bit.
2. Use message blocks $M[i]$ of less than ν bits.

1.3 Motivation: Constrained Devices

An important trend in information technology is the connection of physical objects to large scale networks using standard communication protocols, often called the Internet of Things (IoT) [5]. For instance, an energy controller can communicate with an electrical utility and with controllable appliances and lights to better manage power use, especially during times of increased demand. Many such IoT applications need to be cryptographically secured, to prevent (real-world) damage, theft, and other malfeasance.

It is a challenge to fit cryptography into a computationally constrained environment, such as a small circuit or an inexpensive microprocessor, of the sort used in many IoT devices. These devices themselves are often inexpensive, which limits the computing resources that they can incorporate. To enable a device to communicate securely requires the implementation of algorithms that provide data encryption and authentication, key derivation, and preferably, key establishment and digital signatures, as well. Recent studies have shown that it is possible to use current cryptographic algorithms and protocols in many of these devices, but that it is highly desirable to keep the size of the code and/or circuit to a minimum [6, 7]. Thus, it is a worthwhile goal to minimize the number of cryptographic primitives that need to be implemented, to keep the size of the cryptographic footprint small. By using a cryptographic hash for data encryption and authentication, we avoid the need to include a block cipher. This strategy of abandoning a block cipher is more appealing than that of abandoning a hash function, since the use of a hash function is a strict requirement for digital signatures, and its use is desirable for other uses such as deriving keys from a Diffie-Hellman shared secret.The hash-abandonment strategy has been pursued [8], but it is not clear that the forfeiture of cryptographic hashing and digital signatures will prove to be a good tradeoff.

Hash-CFB can use any cryptographic hash function, and can replace the symmetric encryption that has typically been performed using a block cipher. Of course, it can be part of a minimal cryptographic suite that includes hashing and digital signatures. Because it is an authenticated encryption with associated data algorithm, it could be used in the AEAD interface of the Datagram TLS security protocol [9]. That protocol has been identified by the IETF Constrained Application working group as suitable for IoT applications. Hash-CFB seems well suited to these constrained applications.

1.4 Alternatives

There are several alternatives to our approach:

1. Using a block cipher for both authenticated encryption and hashing.
2. Using a hash-function based MAC (e.g., HMAC) for authentication and run the hash function in an appropriate mode (e.g., counter, output feed-back, ...) for encryption and generically combine both.
3. Using the internal block cipher or compression function inside any practical hash function.

For many cryptographers, the *first alternative*, using a block cipher for encryption and a block cipher based hash function for hashing, would appear as the most natural one – there is a huge amount of research papers on block cipher based hashing and, in recent years, especially on double-block-length hash functions. But there is a lack of an official "standard" for a, say, AES-based hash function. Furthermore, if an existing protocol requires the usage of a certain standard hash function and one is extending the protocol to support authenticated encryption, the hash function already has been fixed, and a block cipher based hash function is no alternative.

The *second option* – the generic composition of a hash-based MAC and a hash-based encryption scheme – would require to maintain two different and independent keys (perhaps pseudorandomly generated from one single master key) and two different internal states, one for authentication and one for encryption, thus doubling memory requirements. On many constrained devices, memory is a very precious resource.

Consider *alternative three*, using the internal block cipher or compression function. Technically, this may be easy, since most constrained devices are programmed in some low-level language, so one would just have to jump to the right address in the hash function implementation. Furthermore, this approach would allow to write some more efficient authenticated encryption scheme, compared to our proposal. At the beginning of this research, we seriously considered this approach.

On the other hand, while cryptographers know what is meant by "the internal compression function", typical standards, such as the SHA-2 standard, don't formally define it. So without the explicit specification of a "new" cryptographic primitive, engineers (non-cryptographers) would not be likely to properly implement the authenticated hash function. Also, there may be constrained devices where calling the internal compression function is much more complicated than just "jumping" to the address.

If one does not need to maximize performance, our approach of directly using the hash function is much simpler and more portable, which we decided to prefer it.

2 The Hash-CFB Mode for Authenticated Encryption

Hash Functions. The underlying primitive for our mode of operation is a plain hash function, not a block cipher, and not even the compression function that the hash function is made of. A ν-bit hash function is a function

$$H' : \{0,1\}^* \rightarrow \{0,1\}^\nu$$

that takes any number of input bits and produces a fixed number of ν output bits. We also consider the restriction H of H' to a fixed input size

$$H : \{0,1\}^{2m+d+e} \rightarrow \{0,1\}^{m+d},$$

where m is the number of bits of one message block, d is the number of additional state bits, i.e., $d := \nu - m$ (we could also allow $d < \nu - m$, truncating the output of H' to $m + d$ bit), and e is chosen such that hashing $2m + d + e$ bits is an efficient operation.

While the core ideas for Hash-CFB are applicable to any secure hash function, we will focus on n H'=SHA-224 and fix m, d, and e accordingly.

Relevant Properties of SHA-224. The SHA-224 compression function takes a 64-byte message block as its input, so it would seem useful to allow H to accept a 64-byte input as well. We do not have access to the compression function, however – and the padding rules require SHA-224 to always append at least 65 bits to the message. So hashing a message as small as 56 bytes would actually require SHA-224 to call its internal compression function *twice*, while 55 bytes plus 7 bit would be the largest input size that could be hashed by calling the internal compression function only *once*. For simplicity, we prefer an integral number of 55 bytes (440 bit) as the input size for the hash function.

Parameter Choice. Given this constraint, we choose our parameters m, d, and e as follows: We set m to 192 bit (24 bytes). This defines $d = 224 - m = 32$ bit (4 bytes) and $e = 440 - 2 * m - d = 24$ bit (3 bytes). We *could* theoretically improve the performance by slightly increasing m, but setting m to any value less than 220 bit – but we prefer m to be a multiple of 32, which simplifies implementations on many (32-bit oriented) machines, and 192 is the largest multiple of 32.

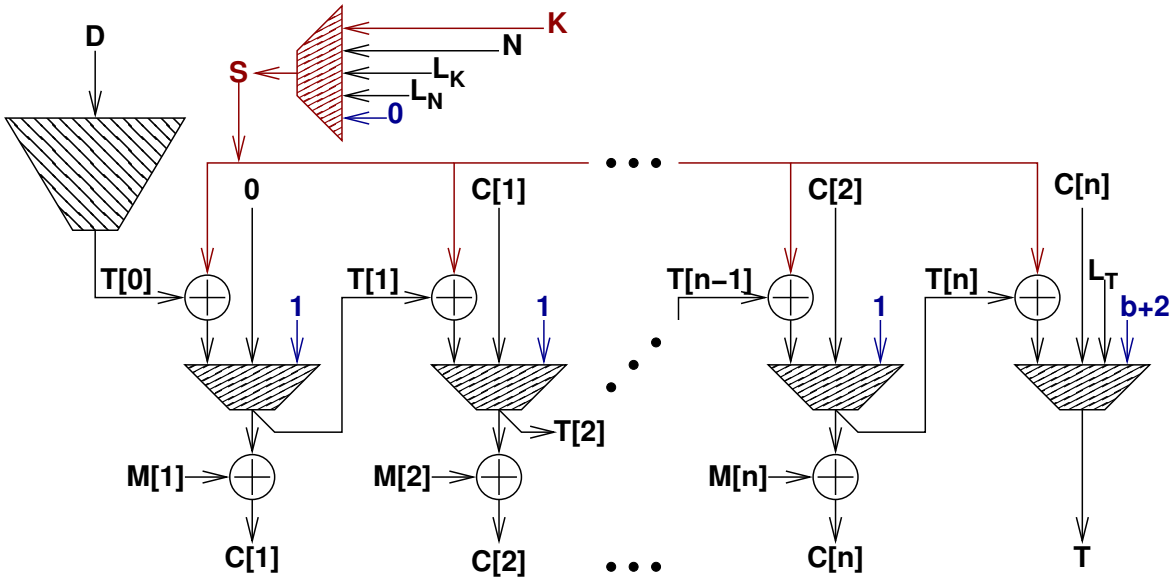


Fig. 1. Illustration of the encryption and authentication process of Hash-CFB.

Why do we prefer SHA-224 over SHA-256? SHA-224 and SHA-256 use the same internal compression function and the same padding rule, but SHA-256 produces four more bytes of output. Instead of using SHA-224, one can just as well, use SHA-256 and truncate away four output bytes. The only reason to prefer SHA-224 ofer SHA-256 is the 224-bit hash output of SHA-224 precisely represents the claimed security level of our proposed scheme.

Notation. We write "||" for the concatenation of strings of bytes. For any integer $x \in \{0, \ldots, 255\}$, we write $[x]_1$ for the representation of x as a single byte, $[0]_y$ for a chain of y zero-bytes, and $[0]_{*55*}$ for a string of zero or more zero-bytes to fill a given input for H up to exactly 55 bytes. If X is a 28-byte value and $y \leq 28$, we write $[X]_y$ for the y least significant bytes of X.

Encryption and Decryption. In the next part of this section we introduce the encryption and decryption process of Hash-CFB (cf. Figure 1).

The inputs for encryption:

- A confidential message M of 24-byte blocks $M[1], M[2], \ldots, M[n-1]$ and a final message block $M[n]$ of $b \leq 24$ bytes.
- Some non-confidential associated data D of any size, that only need to be authenticated.
- A nonce N of L_N bytes.
- A key K of L_K bytes, with the constraint $L_K + L_N \leq 52$. A good default would be 14-byte keys and 24-byte nonces.

The outputs from encryption:

- A ciphertext C of 24-byte blocks $C[1], C[2], \ldots, C[n-1]$, and a final b-byte block $C[n]$.
- An authentication tag T of $L_T \leq 28$ bytes.

The encryption operation: (given $D, N, M[1], \ldots, M[n], K, L_T$)**:**

1. $T[0] := H'(D)$
2. $S := H(K \mid\mid N \mid\mid [0]_{*55*} \mid\mid [L_K]_1 \mid\mid [L_N]_1 \mid\mid [0]_1)$
3. $C[0] := [0]_{24}$
4. **for** i **in** $\{1..n-1\}$ **loop**
   $T[i] := H((S \oplus T[i-1]) \mid\mid C[i-1] \mid\mid [0]_{*55*} \mid\mid [1]_1)$
   $C[i] := [T[i]] \oplus_{24} M[i]$
5. $T[n] := H((S \oplus T[n-1]) \mid\mid C[n-1] \mid\mid [0]_2 \mid\mid [1]_1)$
   $C[n] := [T[n]]_b \oplus M[n]$ $(*$ recall $b = \text{Length}(M[n])$ in bytes $*)$
6. $T := [(H((S \oplus T[n]) \mid\mid C[n] \mid\mid [0]_{*55*} \mid\mid [L_T]_1 \mid\mid [b+2]_1))]_{L_T}$
   return $C[1], \ldots, C[n], T$

The decryption operation (given $D, N, C[1], \ldots, C[n], K, T$)**:**

1. $T[0] := H'(D)$
2. $S := H(K \mid\mid N \mid\mid [0]_{*55*} \mid\mid [L_K]_1 \mid\mid [L_N]_1 \mid\mid [0]_1)$
3. $C[0] := [0]_{24}$
4. **for** i **in** $\{1..n-1\}$ **loop**
   $T[i] := H((S \oplus T[i-1]) \mid\mid C[i-1] \mid\mid [0]_{*55*} \mid\mid [1]_1)$
   $M[i] := [T[i]] \oplus_{24} C[i]$
5. $T[n] := H((S \oplus T[n-1]) \mid\mid C[n-1] \mid\mid [0]_2 \mid\mid [1]_1)$
   $M[n] := [T[n]]_b \oplus C[n]$ $(*$ recall $b = \text{Length}(M[n])$ in bytes $*)$
6. $T := [(H((S \oplus T[n]) \mid\mid C[n] \mid\mid [0]_{*55*} \mid\mid [L_T]_1 \mid\mid [b+2]_1))]_{L_T}$
   if $T = T'$, **then return** $M[1], \ldots, M[n]$
   else return ERROR

3 A Preliminary Security Analysis

In this section, we will provide a rather sketchy security analysis. In the full paper, we will provide a more thorough security analysis and claim concrete results.

3.1 Security Assumption

We make the following security assumptions:

1. *Pseudorandomness* (of $f_K(\ldots)$):
   For any random key $K \in \{0,1\}^k$, the function

$$f_K(X) \;=\; H(K \mid\mid X)$$

with $X \in \{0,1\}^{2m+d+e-k}$ is indistinguishable from a random function. Consider an adversary asking $q \ll 2^{(m+d)/2}$ adaptive queries $X_1, \ldots, X_q$, with $X_i \neq X_j$ for $i \neq j$. Either all queries are answered by independent random values F_i, or in real by $F_i = F_K(X_i)$. Pseudorandomness means that this adversary cannot feasibly distinguish the random from the real case.
Pseudorandomness of $g_K(\ldots) = H(f_K(\ldots) \oplus \ldots \mid\mid \ldots)$:
For a random key K, the function

$$g_K(X, Y, Z) \;=\; H((f_K(X) \oplus Y) \mid\mid Z)$$

with X as above, $Y \in \{0,1\}^{m+d}$, and $Z \in \{0,1\}^{m+e}$ is indistinguishable from a random function.

2. *Unpredictability:* For secret random $K \in \{0,1\}^m$ and $K' \in \{0,1\}^d$, the function g_K with X, Y, Z as above is unpredictable – and even its truncation to L_T bytes is.
   For an adversary asking $q \ll 2^{(m+d)/2}$ adaptive queries (X_1, Y_1, Z_1), $\ldots$, (X_q, Y_q, Z_q), each answered by $[g_{K,K'}(X_1, Y_1, Z_1)]_{L_T}$, it is infeasible to find (X, Y, Z, G) with $(X, Y, Z) \notin \{X_1, Y_1, Z_1), \ldots, (X_q, Y_q, Z_q)\}$ and $G = [g_K(X, Y, Z)]_{L_T}$.
3. *Collision resistance:* The adversary does not find a collision for H'.

Some remarks on these assumptions and the relationship between them:

- *The concrete security of f_K as a PRF:* If we model H as a random oracle, the best attack is tantamount to exhaustively searching for K, i.e., to making about 2^k oracle calls. So the effective key length of f_K is k bit.
- *The concrete security of g_K as a PRF:* In general, one could prove the pseudorandomness of g_K, based on the assumption that f_K is pseudorandom. When considering the concrete security, this would not quite work. While f_K could be made arbitrarily strong by increasing the key length k, at least in a formal model, the concrete security of g_K would not exceed $(m + d)/2$ bit.
  An adversary asking $2^{(m+d)/2}$ queries $g_K(X, Y_i, Z)$ with fixed X and Z is able to find the value $f_K(X)$ by trying out an expected number of $2^{(m+d)/2}$ candidates. For our case of employing SHA-224, the effective key length of g_K is $\min\{k, 112\}$.
- Pseudorandomness of $g_K \Rightarrow$ Unpredictability of g_K: If one can predict the output of g_K, one can distinguish g_K from a random function.

3.2 Formal Security Claims and Proof Sketches

Based on these assumptions, we claim the following security properties for Hash-CFB.

1. *Chosen-plaintext privacy against nonce-respecting adversaries.*
   Scenario:
   Claim: Under the pseudorandomness assumption, Hash-CFB is secure against chosen-plaintext distinguishers.
   Core idea for the proof: Consider a nonce-respecting chosen-plaintext adversary, asking for the encryption of $q \ll 2^{(m+n)/2}$ messages. By the pseudorandomness assumption, the nonce-respecting behavior of the adversary, and since the number of messages encrypted is $q \ll 2^{(m+n)}$, we expect that no two queries will use the same short-term key K. By the pseudorandomness assumption on g_K, all ciphertexts visible to the adversary are indistinguishable from the same number of independent random bits.
2. *Authenticity against any adversary.*
   Scenario: We consider a chosen-message existential-forgery scenario. The adversary adaptively chooses triples (D_i, M_i, N_i) of associated data D_i, message M_i, and nonce N_i. For each such query, the adversary receives the corresponding pair (C_i, T_i) of ciphertext C_i and authentication tag T_i. After making q such queries the adversary presents an attempted forgery, namely a quadruple (D, N, C, T), with no $i \in \{1, \ldots, q\}$ such that $(D, N, C) = (D_i, N_i, C_i)$. The forgery is valid, if its attempted decryption actually returns a message M. It is invalid, if the attempted decryption returns an ERROR.
   Claim: Assuming the unpredictability of g_K and the collision resistance of H', finding a valid forgery with non-negligible probability is infeasible.
   Core idea for the proof: Consider Step 6 of the encryption algorithm, computing

$$T := [(H(\overbrace{(S \oplus T[n]) \;||\; C[n] \;||\; [0]_{*55*} \;||\; [L_T]_1 \;||\; [b+2]_1}^{Z}))]_{L_T}.$$

   We argue that the value Z, used in a valid forgery, is different from all the other inputs to H used before, so finding T would be the same as forging g_K.
   First, observe our "domain separation": the 55-th byte of the message is the value $[b+2]_1$, and all other queries to H use either $[1]_1$

or $[0]_1$ at that position. So we will only have to consider the values $Z_1, \ldots, Z_1$ that had been used in the final step of encrypting and authenticating the previous message.
Second, comparing the decryption and tag computation of (D, N, C) and any of the (D_i, N_i, C_i). If $D \neq D_j$ then $T[0] \neq T_i[0]$. Similarly our assumptions give $T[1] \neq T_i[1]$ if $D = D_j$ and then $N \neq N_i$, and $T[j] \neq T[j]$ if $D = D_j$. Now assume $D = D_j$, $N = N_j$, and consider all but the last and possibly incomplete block of C and C_j. If a j exists with $C[j-1] \neq C_i[j-1]$, then $T[j] \neq T_i[j]$ by our assumptions. Once $T[j] \neq T_i[j]$, our assumptions ensure that eventually $Z \neq Z_i$ holds.
What if $D = D_j$, $N = N_j$ and no such j exists. If either C or C_i is at least one block longer than the other, we get $Z \neq Z_i$ again. If they are of the same length in blocks, but not exactly the same lengths in bytes, then the input $[b+2]_1$ of Z, which gives the length of the last block (plus 2), must be different from its counterpart in Z_i. If C and C_i are of exactly the same length (in bytes), then they must be different and thus $Z \neq Z_i$, since otherwise we had $(D, N, C) = (D_i, N_i, C_i)$.
Thus, all these cases give us $Z \neq Z_i$.

3. *Chosen-ciphertext privacy against nonce-respecting adversaries.* This follows from a standard argument used for authenticated encryption in general: if a scheme provides privacy in the chosen-plaintext sense and also provides authenticity, the adversary gains nothing from asking chosen-ciphertext queries.

3.3 Informal Security Claim: Resistance against side-channel attacks

The authors of block cipher modes take care not to change the key, to avoid a heavy performance penalty for re-running the key schedule. If an n-block message is to be encrypted, the key is used at least n times. This is helpful for side-channel attacks, who typically depend on noisy side-channel information about the secret key. Because of the noise, a typical side-channel adversary needs many key-dependent operations to eventually determine the key.

As we are using the raw hash function, all the inputs are part of the message to be hashed, including the key. So we are free to change the key any time. Actually each time a message of any length is to be encrypted or decrypted, we derive some *short term key* S from the original key and

the nonce and then use S for all further operations, thus, preventing the side-channel adversary from gathering good statistical information about the real key.

References

1. Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche: Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. *Selected Areas in Cryptography* 2011: 320-337.
2. Ewan Fleischmann, Christian Forler, Stefan Lucks: McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. *Fast Software Encryption Workshop*, 2012.
3. Stefan Lucks, David McGrew, Doug Whiting: Batteries Included – Features and Modes for Next Generation Hash Functions. *3rdSHA-3 Candidate Conference*, 2012. ⟨`http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/March2012/documents/papers/LUCKS_paper.pdf`⟩
4. Ted Krovetz, Phillip Rogaway: The Software Performance of Authenticated-Encryption Modes. *Fast Software Encryption Workshop* 2011: 306–327.
5. Internet of Things Architecture Consortium, `http://www.iot-a.eu/public`.
6. Tschofenig H., et. al. Report from the Smart Object Security Workshop. 23rd March 2012, Paris. `http://tools.ietf.org/agenda/83/slides/slides-83-saag-3.pdf`.
7. Workshop on Smart Object Security, in conjunction with IETF83, Paris 23 March 2012.
8. R. Moskowitz HIP Diet EXchange (DEX), Work in Progress, *IETF Internet Draft* draft-moskowitz-hip-rg-dex-05, March 14, 2011.
9. E. Rescorla, N. Modadugu, Datagram Transport Layer Security Version 1.2, *IETF Request for Comments* (RFC) 6347, January, 2012.

A Do-It-All-Cipher for RFID: Design Requirements (Extended Abstract)

Markku-Juhani O. Saarinen and Daniel Engels

REVERE SECURITY
4500 Westgrove Drive, Suite 300, Addison, TX 75001, USA.
mjos@reveresecurity.com

Abstract. Recent years have seen significant progress in the development of lightweight symmetric cryptoprimitives. The main concern of the designers of these primitives has been to minimize the number of gate equivalents (GEs) of the hardware implementation. However, there are numerous additional requirements that are present in real-life RFID systems. We give an overview of requirements emerging or already present in the widely deployed EPCGlobal Gen2 and ISO / IEC 18000-63 passive UHF RFID air interface standards. Lightweight stateful authenticated encryption algorithms seem to offer the most complete set of features for this purpose. In this work we give a Gen2-focused "lessons learned" overview of the challenges and related developments in RFID cryptography and propose what we see as appropriate design criteria for a cipher (dubbed "Do-It-All-Cipher" or DIAC) for the Internet of Things. We also comment on the applicability of NSA's new SIMON and SPECK proposals for this purpose.

Keywords: RFID Security, Authenticated Encryption, Lightweight Cryptography, Gen2, DIAC, SIMON, SPECK.

1 Introduction: Supply Chain RFID

Passive RFID (Radio Frequency IDentification) technologies have found use in a vast array of important application areas from animal microchip implants to electronic passports and payment systems. However, in economic terms the highest single application area is in supply chain management. During the 2000s, much of the retail supply chain world converged to use the EPCGlobal UHF Class 1 Generation 2 ("Gen2") and ISO/IEC 18000-63 "Type C" passive UHF RFID air interface [34]. The latter standard is otherwise equivalent to the first but also includes sensor and battery assisted passive specifications.

These standards were designed with a minimal set of security mechanisms as they were initially designed to operate primarily as barcode replacements in a physically secure supply chain. However, the success of this RFID technology has led to its use within a broad range of supply chain and consumer applications and industry verticals that would benefit from a higher level of security than provided by the existing standards.

The real-life risks and services provided by RFID systems are best protected by cryptographically secure RFID tags. The main risks and services addressed by the deployment of cryptographically secure RFID tags (and which are used to justify their additional cost) include the following:

1. **Counterfeit goods.** Cryptography is used to make RFID tags difficult to clone or modify. We have seen entire counterfeit aircraft engines; the risks and liability issues involved are difficult to even measure.
2. **Secure logging.** Tamper resistant recording of environmental information such the temperature is vital in supply chain management of products such as fresh goods and medical supplies.
3. **Privacy protection.** The Electronic Product Code (EPC) used in Gen2 differs from product bar codes in that it is indeed unique (basically a serial number). It may be used to track an individual tag. This raises serious privacy issues if such tags are attached to personal items. Therefore the tag must also identify the reader as trusted before divulging traceable information.
4. **Antitheft.** Data may be written to the tag to indicate to an exit portal whether or not that item has been sold. Persistent memory write and locking operations must be protected to prevent theft.
5. **Returns.** When a tag is returned to a store or manufacturer an authenticated reset/write mechanism allows it to be reused. The tags maintain some amount of persistent memory; read, write and lock operations to this memory must be authenticated to prevent tamper and unauthorized modification. Authenticated reads allow data to be visible only for the tags owner.

Many academic cryptographers working in the field of lightweight cryptography tend to concentrate only on a cryptoprimitive (such as a hash function) or a single function (such as authentication). In real life secure RFID actually requires a wide palette of security functionality and a multi-faceted, yet light-weight, protocol.

2 Developments in Lightweight Cryptography

The basic symmetric cryptographic one-way authentication mechanism consists of an Interrogator (RFID Reader) Alice sending a (random) challenge C to the Responder (RFID Tag) Bob. The Bob-Tag then uses a shared secret S and a cryptographic function f to compute and transmit the response $R = f(S, C)$ to Interrogator-Alice. Alice can then verify knowledge of the secret S by verifying whether the response indeed matches with the shared secret. The main cryptographic requirement is that there should be no computationally efficient way of deriving the secret S (or other way of computing the matching response R) based on observations of C and R alone. There are many variations and subtleties to this scheme and practical RFID systems outside simple access control tend to need more security services than just authentication. However, for the basic authentication scheme the fucntion f can be a hash function, a block cipher, a stream cipher – essentially any keyed pseudorandom permutation or function.

A decade ago, Sarma, Weis and Engels were one of the first to draw attention to lightweight cryptographic challenges emerging in the RFID field [53, 54, 58]. Their

work defined the challenge and called for hardware-efficient cryptographic hash functions and symmetric encryption schemes. The requirements can be boiled down to four points:

- The system (including the cryptographic primitives) should be realizable with a circuit of between 3000-4000 Gate Equivalents (GEs) or less.
- The cryptographic implementation and the authentication protocol should be able to meet the stringent timing requirements of ISO 18000-63.
- A passive device should not consume more than 30–50 μW of energy. Note that energy consumption peaks effectively limit the range of the device, hence they should be avoided.
- The cryptographic protocols should have as few round-trip communications as possible. Also, the number of bits transmitted should be minimized, while maintaining a tolerably low probability of a false authentication.

Established standards such as the Advanced Encryption Standard AES [41] and Secure Hash Algorithm [43] have widely been deemed too complex and energy-hungry for the RFID environment, although advances have been made in small-scale implementations of both AES [26, 31] and SHA-1 and the SHA-2 family [25, 44].

Vajda and Buttyán [57] responded to this call but some of their protocols were cryptanalyzed and found to be weak [18]. During subsequent years literally hundreds of authentication protocols for RFID systems have been proposed to counter various attacks scenarios, including:

- Tag cloning (unauthorized copying) and other impersonation and relay attacks.
- Privacy violations, information leakage, illicit tracking and monitoring (anonymity).
- Denial of service, availability.
- Forward security – it should be infeasible to compromise past sessions.

We note that a vast majority if these protocol and design proposals have either failed in their goals or have been simply ignored by the RFID industry. However, the "RFID Challenge" has also generated a lot of fresh and innovative research, some of which will be highlighted here as prior art.

Block ciphers and hash functions. Since 2007 we have seen a significant progress in the development of lightweight ciphers and hash functions, illustrated by the development of block ciphers DESL [38], PRESENT [15, 50], KATAN/KTANTAN [16], and LED [30], hash functions QUARK [1], PHOTON [29] and SPONGENT [14], and many others. These proposals have been designed to meet the low gate count requirements and usually exhibit appropriate cryptographic strength. However, these are general purpose cryptographic primitives and not specifically crafted for use with RFID.

Stream ciphers. Light-weight stream ciphers (especially those equipped with an Initialization Vector that can be quickly set up) may also be used for authentication. We are aware of a proprietary RFID solution based on Grain-128 [28, 32]. There has been recent progress in cryptanalysis of Grain-128 [19, 20]. Grain-128 is also used as a building block of the SQUASH-128 MAC function [55].

Dedicated RFID protocol designs. The field of (ultra-) lightweight RFID authentication protocols has proceeded quite independently from the ciphers and hash functions, starting with the UMAP family [45–47], which was swiftly broken in a series of publications [2, 3, 33, 39, 40]. These protocols are often designed to resist only passive attacks and are constructed from simple bit-wise operations. Notable later examples include Chien's SASI [17, 56] and the Gossamer protocol [13, 48].

Combined cipher / protocol proposals. Just as Hummingbirds have co-evolved in the nature with (ornithophilous) flowers, the Hummingbird-1 authenticated encryption algorithm [22, 23] was proposed *together* with a protocol: Hummingbird is a protocol-cipher pair. After cryptographic vulnerabilities were discovered in the cipher [51], it was amended to produce Hummingbird-2 [24].

3 An RFID Security Suite and Its Design Criteria

Many cryptographers tend to see the RFID authentication challenge from an overly abstract viewpoint, ignoring the underlying communication channel details, such as how the radio channel actually operates. Work on light-weight cryptography such as [1, 14–16, 28–30, 32, 38, 50] has almost exclusively concentrated on minimizing the number of Gate Equivalents (GEs) of the light-weight primitive, while generally ignoring the inherent timing constraints (number of clocks) required to generate a response. The following discussion is partially from [24].

The HB2-128 cipher and protocol is a cryptography suite for use in the security standards EPCglobal Gen2 version 2.0 and the ISO/IEC JTC1 SC31 18000-63 passive UHF RFID air protocols [21]. The basic security services provided include:

- Interrogator and/or tag authentication.
- An authenticated and/or confidential communication channel.
- Message Authentication Code (MAC) for message integrity.

The proposed secure HB2-128 Gen2 protocol utilizes five basic commands to provide security services: *Authenticate*, *Challenge*, *KeyUpdate*, *AuthComm*, and *SecureComm*. A finite state machine defines the operation of the security functions including the authentication protocol.

The primary functionality of the Gen2 protocol is for fast and efficient tag identification across a range of operating environments. Consequently, Gen2 supports a range of data rates at which the interrogator and the tag may communicate. Communications are controlled by the interrogator in a Reader Talks First communication scheme. The interrogator begins a command by issuing a preamble that defines the length of the logic 0 and logic 1 symbols. The length of the logic 0 symbol is referred to as Tari, which is a fundamental timing parameter for communications.

For all commands except the Write command, the Tari value determines the amount of time the tag has to begin its response to the reader after the reader has completed the last symbol in its command to the tag. This time is referred to as T1 time. Table 1 shows the T1 timing for the minimum Tari value of 6.25 μs, the maximum Tari value of 25 μs,

Table 1. T1 Timing Values and Available Clock Cycles.

Tari (μs)	T1 Time (μs)	Cycles 1.5 MHz	Cycles 2 MHz	Cycles 2.5 MHz
6.25	39.06	58	78	97
12.5	78.125	117	156	195
25	187.5	281	375	468

and a commonly used Tari value of 12.5 μs. A Gen2 tag typically has an on-chip clock that operates netween 1.5 MHz and 2.5 MHz.

In addition to the T1 timings, Table 1 shows the number of full clock cycles available for computation within T1 for three on tag clock frequencies around 2 MHz, a common on-tag clock frequency. We see that hashes and ciphers with a latency exceeding 58 cycles are already unworkable in some allowable communication rates. This means that otherwise attractive but high-latency proposals such as U-Quark [1] (544 cycles with 1379 GE, 68 cycles with 2392 GE), SPONGENT-128 [14] (2380 cycles with 1060 GE, 70 cycles with 1687 GE), and PHOTON-128/16/16 [29] (996 cycles with 1122 GE, 156 cycles with 1708 GE) are largely inappropriate for Gen2 RFID authentication.

4 Power and Clock Frequency

Power is the product of current and voltage ($W = A \times I$), and for most CMOS circuits the current draw is close to linearly dependent upon the clock frequency for a fixed voltage. All of these factors affect the all-important range of an RFID tag. Implementors also try to make the power usage profile of cryptoprimitives independent of the secret keying material as a countermeasure against DPA (Differential Power Analysis) [36].

Microcontrollers. Software designers think in terms of Watts/Hertz since the only thing they can really change is the clock frequency of the device. Voltage is fixed and current is basically linearly dependent upon clock frequency. Therefore the algorithm implementation is optimized for throughput in order to consume fewer cycles.

Hardware implementations. Circuit designers in in the passive UHF space tend to have a different mindset. Max instantaneous power is a important regardless of what clock frequency is used on chip since more instantaneous power means shorter range. A power spike that drops the chip voltage below the threshold ($1.5V$ typical) causes a tag to reset itself. This is because it's the instantaneous power that can perform a quick power draw from the storage cap that drops its voltage below the minimum required to operate which then shuts down or reboots the tag. Hence our $3\mu W$ to $30\mu W$ limit (Section 8) is really an instantaneous power limit not dependent upon clock frequency.

Tag power usage. Tags typically *do not* perform operations unrelated to communication during the communication phase due to the lack of power harvesting; thus, all computations, including cryptographic computations, related to the command execution should be done before communication starts.

For Gen2 tags, the voltage varies from about $1.8V$ to $2.2V$ (typical values) with most devices trying for the 2V average. Most of these CMOS circuits are able to operate at about 1.5V and the voltage limiters cap voltage at about $3V$. This is a design feature that is trying to provide protection against sudden voltage spikes which, in practice, tends to limit the upper voltage range. Although this level is rarely achieved in practice. 2.5V seems to be the maximum level reached in realistic laboratory testing.

A tag will often operate over the entire range of usable voltage levels during a single communication sequence since the power cap goes from zero to 1.5 V at which point the chip begins operating then up to a maximum voltage, usually $2 - 2.5V$ depending on power radiated by reader, then back down to 1.8V (or lower) as the chip receives a command and performs its operations and communicates back to the reader. At long range, the peak voltage is often close to 1.8V. A tag typically does not harvest much if any power as it is communicating, and it is during this time that the tag's voltage drops to its lowest point.

5 Case study: A Do-It-All-Cipher Proposal

Hummingbird-2 [24] as an authenticated encryption primitive does not fall directly into the block cipher / stream cipher / hash function classifications, but operates in similar fashion to the HELIX [27], PHELIX [59] and Duplex Sponge [10] constructions. A HB2 engine suitable for RFID authentication can be realized with as little as 1250 GE.

In addition to standard security requirements – such as resistance against chosen plaintext- and ciphertext attacks – HB2 was designed to have the following features that are desirable in RFID cryptography:

1. **Nonce / IV / Tweak** which allows rapid re-initialization of the cipher without re-keying the cipher. This is essential for communications protocols over a radio channel and also for challenge-response authentication mechanisms.
2. **Secure with repeated IVs.** The primitive should be resistant to repeated IV attacks such as [51]. Note that many (if not all) authenticated single block cipher encryption modes such as GCM [42] and OCB [37, 49] are insecure when IV is repeated [35].
3. **Message Authentication code (MAC).** The primitive should be capable of producing a secure MAC for the processed data. We would also like the algorithm to produce a **secure hash** when a fixed IV is used.
4. **Authenticated Associated Data (AAD).** The primitive should be able to authenticate both encrypted data and unencrypted data without re-initialization. This is extremely useful for explicit plaintext data (message headers such as a destination) or implicit data (untransmitted information such as message sequence numbers).
5. **Small block size.** The primitive should be able to process data in 8- or 16- bit increments, or even smaller (HB2 can transmit authenticated data in under 8-bit increments, even though full 16-bit processing is required in such cases). 128 bits is clearly too much.

Comparison to block ciphers. Block ciphers, by themselves, are stateless. Hence an external state must be maintained (as part of a "mode of operation") to counter birthday paradox attacks in encryption/decryption use. Typically, additional state information is required to achieve authenticated encryption.

Comparison to hash functions. In a multi-round authentication protocol, a hash function would have to be "re-initialized" with various transmitted or shared secrets to generate output each transmitted message, whereas HB2 maintains all of that communicated information in its 128-bit state. Furthermore, confidential communications channels cannot be established in a straight-forward fashion with a hash functions alone.

HB2 Criticism. The main drawback of a design such as HB2 [24] is the same as that of the AES candidate Serpent [11, 12]; encryption and decryption operations do not fully use the same components since the inverses of S-boxes and linear transforms must be implemented separately. In fact, an implementation with both encrypt and decrypt functionality tends to be about 70 % larger than one that only implements encryption (2150 GE vs 1250 GE for a 16-cycle / word implementation). However, we note that only encryption functionality is sufficient for tag authentication for all of the security functionality in the HB2-128 Gen2 proposal.

Another issue is that the "critical path" of the encryption function is rather long; each 16-bit word requires invocation of 16 S-Box layers. However, since each set of 4 layers of S-Boxes is isolated into its own keyed WD16 function, 4-cycle / word implementations can be constructed. Furthermore, the low clock rate on tags allows for long critical paths without compromising tag performance.

6 NSA's Proposal: SIMON and SPECK

The U.S. National Security Agency has recently published performance and implementation footprint numbers for their in-house developed lightweight block cipher families SIMON and SPECK [4]. NSA has considerable institutional expertise and taxpayer funded resources in cryptography and we would welcome their help in civilian anti-counterfeit and other infrastructure protection (RFID, remote keys and locks, etc). However, in order to fully benefit commerce, industry, and the general public, the algorithm details must also be released. This will make the algorithm standardizable as most international bodies are reluctant to blindly trust technology that has its origins within the security apparatus of any one nation. ECRYPT and the international cryptologic community can analyze the ciphers and may choose to modify and perhaps even to improve upon the given design for standardization.

SIMON and SPECK are block cipher families with block size of 32, 48, 64, 96, and 128 bits, with up to three key sizes for each block size. Hardware implementations are very impressive in size. A target latency of 58 cycles for a full block operation is not reached by the very smallest implementations, but due to it's highly tunable nature, implementations meeting the Gen2 passive RFID criteria are also reported.

However, a question remains on how to perform authenticated encryption on short messages such as those required for "Internet of Things". We know that a standard authenticated mode such as GCM is insecure with small block sizes [52]. One may wish to construct a mode with a much larger state from SIMON / SPECK. Such a construction could be modeled as a larger Sponge PRP (see Section 7). This would also make the cost of decryption functionality basically free. We note that authors of [4] only report the cost of the block cipher encryption function but claim that there is some "symmetry" and a degree of reusable components between encryption and decryption.

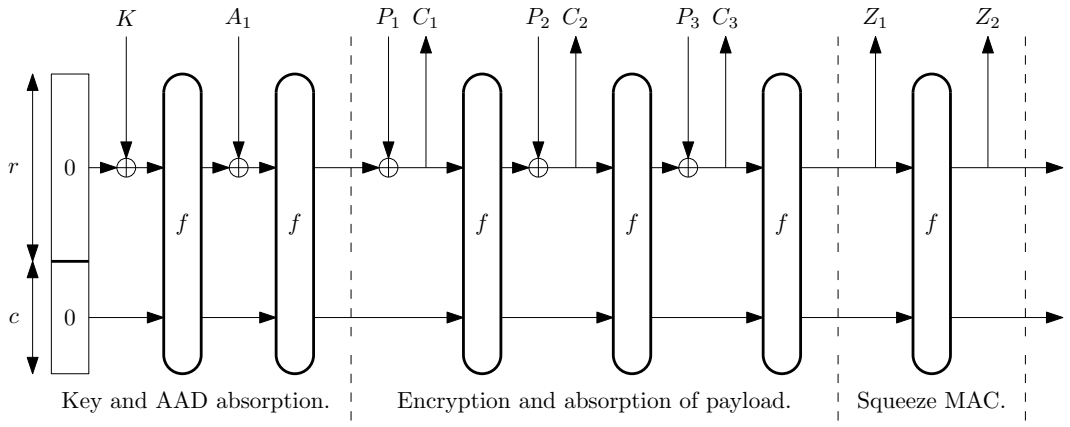


Fig. 1. A simplified view of how a Sponge Function f with a rate of r bits and state $r + c$ bits may be used to process a secret key / tweak K, Associated Authenticated Data (AAD) block A_1 and plaintext blocks $P_1 \ldots P_3$ to produce ciphertext blocks $C_1 \ldots C_3$ and the message authentication blocks $Z_1 \ldots Z_2$.

7 A Duplex Sponge Can (almost) Do It All

As shown by Bertoni et al in their Sponge function work, s fixed-length PRP (pseudo-random permutation) can be used to "do it all"; it can meet all of the requirements set forth in Section 5 [5, 6, 8–10]. The only major thing that seems to be missing from the proposal of [10] is Authenticated Associated Data (AAD). A padding mechanism to achieve this would seem to be reasonably easy to achieve.

Figure 1 illustrates how a Sponge Function f can be used to process keying, nonce and AAD data, then encrypt a message and finally to squeeze out a message authentication code.

One additional feature that we would like to see is "MAC without reset" where one would be able to authenticate each message while also authenticating all previously sent messages (and perhaps even received messages). Here the state is not reset after the MAC or hash has been "squeezed" out of the state, but instead the state is used for the next message. Thereby the system would not be dependent on the uniqueness of the message IV to protect the confidentiality of the first block of data, among other advantageous features.

One notable advantage of block cipher authenticated modes such as GCM [42] and OCB [37, 49] is that they can handle *any* bandwidth; one just has to implement a sufficient number of parallel AES cores. This shortcoming of the Sponge function construction may be addressed with something akin to Tree Hashing [7].

One of the major advantages of the Sponge construction in RFID is that there is no key to protect after initialization, just the present state. We may construct a PRF g from PRP f by using a construction such as $g(x) = x \oplus f(x)$. If we always store the $g(x)$ value in persistent storage between subsequent authentications or message exchanges, we obtain a degree of forward security; in a key-search based authentication [58] the previous sessions remain anonymous and secure even after a state compromise. This is naturally just optional as persistent storage is not available on all RFID platforms and the special protocols must be used for such "rolling code" applications.

8 Conclusions

We conclude with a set of design goals for a RFID or lightweight sensor network "Do-It-All-Cipher" (DIAC) for Internet of Things.

- The primitive must be able to operate without significant modification as a single-pass tweakable authenticated encryption & decryption algorithm and as a cryptographically secure hash.
- The padding and operation rules for various inputs, including IVs, Authenticated Associated Data (AAD) must be unambiguously defined. The data rate should be small to avoid message expansion.
- Initialization vector (IV) does not have to be a nonce (secure in a repeated chosen-IV attack). The security level must be consistent with key and state size under all reasonable attack models.
- Hardware implementation must be under 2000 GE, including state memory and both encryption and decryption functionality. Software implementation speed and size across all MCU and CPU platforms should also be a design consideration.
- The primitive must have a latency of under 50 cycles and encryption/decryption throughput of more than 1 bit / cycle.
- The power usage should be less than $1 \ldots 10 \frac{\mu W}{MHz}$ average with peaks below $3\mu W$ and $30\mu W$ respectively. Thus, at $2MHz$, peaks should be below $1.5 \frac{\mu W}{MHz}$ to $15 \frac{\mu W}{MHz}$.

We hope to see proposals meeting not just a few of these goals but to simultaneously meet *all* of these criteria.

References

1. AUMASSON, J.-P., HENZEN, L., MEIER, W., AND NAYA-PLASENCIA, M. Quark: A lightweight hash. In *CHES 2010* (2010), S. Mangard and F.-X. Standaert, Eds., vol. 6225 of *LNCS*, Springer, pp. 1–15.
2. BÁRÁSZ, M., BOROS, B., LIGETI, P., LÓJA, K., AND NAGY, D. A. Breaking LMAP. In *RFIDSec '07* (July 2007).
3. BÁRÁSZ, M., BOROS, B., LIGETI, P., LÓJA, K., AND NAGY, D. A. Passive attack against the M^2AP mutual authentication protocol for RFID tags. In *EURASIP RFID 2007* (September 2007).
4. BEAULIEU, R., SHORS, D., SMITH, J., TREATMAN-CLARK, S., WEEKS, B., AND WINGERS, L. Performance of the SIMON and SPECK families of lightweight block ciphers. Tech. rep., National Security Agency, May 2012.
5. BERTONI, G., DAEMEN, J., PEETERS, M., AND ASSCHE, G. V. Sponge functions. In *Ecrypt Hash Workshop 2007* (May 2007).
6. BERTONI, G., DAEMEN, J., PEETERS, M., AND ASSCHE, G. V. On the indifferentiability of the sponge construction. In *EUROCRYPT 2008* (2008), N. P. Smart, Ed., vol. 4965 of *LNCS*, Springer, pp. 181–197.
7. BERTONI, G., DAEMEN, J., PEETERS, M., AND ASSCHE, G. V. Sufficient conditions for sound tree and sequential hashing modes. IACR ePrint 2009/210, `http://eprint.iacr.org/2008/210`, December 2009.

8. BERTONI, G., DAEMEN, J., PEETERS, M., AND ASSCHE, G. V. Sponge-based pseudo-random number generators. In *CHES 2010* (2010), S. Mangard and F.-X. Standaert, Eds., vol. 6225 of *LNCS*, Springer, pp. 33–47.
9. BERTONI, G., DAEMEN, J., PEETERS, M., AND ASSCHE, G. V. Cryptographic sponge functions, version 0.1. `http://sponge.noekeon.org/`, STMicroelectronics and NXP Semiconductors, January 2011.
10. BERTONI, G., DAEMEN, J., PEETERS, M., AND ASSCHE, G. V. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *SAC 2011* (2011), A. Miri and S. Vaudenay, Eds., vol. 7118 of *LNCS*, Springer, pp. 320–337.
11. BIHAM, E., ANDERSON, R. J., AND KNUDSEN, L. R. Serpent: A new block cipher proposal. In *FSE 98* (1998), S. Vaudenay, Ed., vol. 1372 of *LNCS*, Springer, pp. 222–238.
12. BIHAM, E., ANDERSON, R. J., AND KNUDSEN, L. R. Serpent: A proposal for the advanced encryption standard. AES Proposal to NIST. `http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf`, 1999.
13. BILAL, Z., MASOOD, A., AND KAUSAR, F. Security analysis of ultra-lightweight cryptographic protocol for low-cost RFID tags: Gossamer protocol. In *Proc. NBIS '09* (2007), IEEE Computer Society, pp. 260–267.
14. BOGDANOV, A., KNEZEVIC, M., LEANDER, G., TOZ, D., VARICI, K., AND VERBAUWHEDE, I. Spongent: A lightweight hash function. In *CHES 2011* (2011), B. Preneel and T. Takagi, Eds., vol. 6917 of *LNCS*, Springer, pp. 312–325.
15. BOGDANOV, A., KNUDSEN, L. R., LEANDER, G., PAAR, C., POSCHMANN, A., ROBSHAW, M. J. B., SEURIN, Y., AND VIKKELSOE, C. PRESENT: An ultra-lightweight block cipher. In *CHES 2007* (2007), P. Paillier and I. Verbauwhede, Eds., vol. 4727 of *LNCS*, Springer, pp. 450–466.
16. CANNIÈRE, C. D., O.DUNKELMAN, AND ZEVIĆ., M. K. KATAN & KTANTAN – a family of small and efficient hardware-oriented block ciphers. In *CHES 2009* (2009), C. Clavier and K. Gaj, Eds., vol. 5747 of *LNCS*, Springer, pp. 272–288.
17. CHIEN, H.-Y. SASI: A new ultralightweight RFID authentication protocol providing strong authentication and strong integrity. *IEEE Trans. in Dependable and Secure Computing 4*, 4 (2007), 337–340.
18. DEFEND, B., FU, K., AND JUELS, A. Cryptanalysis of two lightweight RFID authentication schemes. In *Proc. Fifth IEEE International Conference on Pervasive Computing and Communications Workshops* (2007), PERCOMW '07, IEEE Computer Society, pp. 211–216.
19. DINUR, I., GÜNEYSU, T., PAAR, C., SHAMIR, A., AND ZIMMERMANN, R. An experimentally verified attack on full Grain-128 using dedicated reconfigurable hardware. In *ASIACRYPT 2011* (2011), vol. 7073 of *LNCS*, Springer, pp. 327–343.
20. DINUR, I., AND SHAMIR, A. Breaking Grain-128 with dynamic cube attacks. In *FSE 2011* (2011), A. Joux, Ed., vol. 6733 of *LNCS*, Springer, pp. 167–187.
21. ENGELS, D. HB2-128 crypto-suite proposal. Tech. rep., Revere Security, December 2011. Version 1.1.
22. ENGELS, D., FAN, X., GONG, G., HU, H., AND SMITH, E. M. Ultra-lightweight cryptography for low-cost RFID tags: Hummingbird algorithm and protocol. Tech. Rep. CACR-2009-29, Centre for Applied Cryptographic Research (CACR), University of Waterloo, 2009. `http://www.cacr.math.uwaterloo.ca/techreports/2009/cacr2009-29.pdf`.
23. ENGELS, D., FAN, X., GONG, G., HU, H., AND SMITH, E. M. Hummingbird: Ultra-lightweight cryptography for resource-constrained devices. In *Financial Cryptography and Data Security, FC 2010 Workshops* (2010), R. Sion, R. Curtmola, S. Dietrich, A. Kiayias, J. M. Miret, K. Sako, and F. Sebé, Eds., vol. 6054 of *LNCS*, Springer, pp. 3–18.

24. ENGELS, D., SAARINEN, M.-J. O., SCHWEITZER, P., AND SMITH, E. M. The Hummingbird-2 lightweight authenticated encryption algorithm. In *RFIDSec '11* (2011), A. Juels and C. Paar, Eds., vol. 7055 of *LNCS*, Springer, pp. 19–31.
25. FELDHOFER, M., AND RECHBERGER, C. A case against currently used hash functions in RFID protocols. In *OTM Workshops (I)* (2006), R. Meersman, Z. Tari, and P. Herrero, Eds., vol. 4288 of *LNCS*, Springer, pp. 372–381.
26. FELDHOFER, M., WOLKERSTORFER, J., AND RIJMEN, V. AES implementation on a grain of sand. *IEE Proc. Inf. Sec 153*, 1 (October 2005), 13–20.
27. FERGUSON, N., WHITING, D., SCHNEIER, B., KELSEY, J., LUCKS, S., AND KOHNO, T. Helix: Fast encryption and authentication in a single cryptographic primitive. In *FSE 2003* (2003), T. Johansson, Ed., vol. 2887 of *LNCS*, Springer, pp. 330–346.
28. GREN, M. A., HELL, M., JOHANSSON, T., AND MEIER, W. Grain-128a: a new version of Grain-128 with optional authentication. *International Journal of Wireless and Mobile Computing 5*, 1 (2011), 48–59.
29. GUO, J., PEYRIN, T., AND POSCHMANN, A. The PHOTON family of lightweight hash functions. In *CRYPTO 2011* (2011), P. Rogaway, Ed., vol. 6841 of *LNCS*, Springer, pp. 222–239.
30. GUO, J., PEYRIN, T., POSCHMANN, A., AND ROBSHAW, M. J. B. The LED block cipher. In *CHES 2011* (2011), B. Preneel and T. Takagi, Eds., vol. 6917 of *LNCS*, Springer, pp. 326–341.
31. HÄMÄLÄINEN, P., ALHO, T., HÄNNIKÄINEN, M., AND HÄMÄLÄINEN, T. Design and implementation of low-area and low-power AES encryption hardware core. In *Ninth Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD 2006)* (2006), IEEE Computer Society, pp. 577–583.
32. HELL, M., JOHANSSON, T., AND MEIER, W. Grain: a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing 2*, 1 (2007), 86–93.
33. HUNG-YU, C., AND CHEN-WEI, H. Security of ultra-lightweight RFID authentication protocols and its improvements. *ACM SIGOPS Operating Systems Review 41*, 4 (July 2007), 83–86.
34. INTERNATIONAL STANDARDIZATION ORGANIZATION. *ISO/IEC 18000-63. Information technology – Radio frequency identification for item management – Part 6: Parameters for air interface communications at 860 MHz to 960 MHz Type C*, 2012.
35. JOUX, A. Authentication failures in NIST version of GCM. NIST Standardization Comment, 2006.
36. KOCHER, P., JAFFE, J., AND JUN, B. Differential power analysis. In *Crypto '99* (1999), M. J. Wiener, Ed., vol. 1666 of *LNCS*, Springer, pp. 104–114.
37. KROVETZ, T., AND ROGAWAY, P. The software performance of authenticated-encryption modes. In *FSE 2011* (2011), A. Joyx, Ed., vol. 6733 of *LNCS*, Springer, pp. 306–327.
38. LEANDER, G., PAAR, C., POSCHMANN, A., AND SCHRAMM, K. New lightweight DES variants. In *FSE 2007* (2007), A. Biryukov, Ed., vol. 4593 of *LNCS*, Springer, pp. 196–210.
39. LI, T., AND DENG, R. Vulnerability analysis of EMAP - an efficient RFID mutual authentication protocol. In *Proc. ARES 2007* (2007), IEEE Computer Society, pp. 238–245.
40. LI, T., AND WANG, G. Security analysis of two ultra-lightweight RFID authentication protocols. In *IFIP-SEC 2007* (2007), H. S. Venter, M. M. Eloff, L. Labuschagne, J. H. P. Eloff, and R. von Solms, Eds., no. 232 in IFIP, Springer, pp. 109–120.
41. NIST. Advanced Encryption standard (AES). Federal Information Processing Standards 197, 2001.
42. NIST. Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. NIST Special Publication 800-38D, 2007.
43. NIST. Secure Hash Standard (SHS). Federal Information Processing Standards 180-3, 2008.

44. O'NEILL, M. Low-cost SHA-1 hash function architecture for RFID tags. In *RFIDSec '08* (2008).
45. PERIS-LOPEZ, P., HERNANDEZ-CASTRO, J. C., ESTEVEZ-TAPIADOR, J. M., AND RIBAGORDA, A. EMAP: An efficient mutual-authentication protocol for low-cost RFID tags. In *OTM 2006 Workshops* (2006), R. Meersman, Z. Tari, and P. Herrero, Eds., vol. 4277 of *LNCS*, Springer, pp. 352–361.
46. PERIS-LOPEZ, P., HERNANDEZ-CASTRO, J. C., ESTEVEZ-TAPIADOR, J. M., AND RIBAGORDA, A. LMAP: A real lightweight mutual authentication protocol for low-cost RFID tags. In *RFIDSec '06* (2006).
47. PERIS-LOPEZ, P., HERNANDEZ-CASTRO, J. C., ESTEVEZ-TAPIADOR, J. M., AND RIBAGORDA, A. M^2AP: A minimalist mutual-authentication protocol for low-cost RFID tags. In *UIC 2006* (2006), J. Ma, H. Jin, L. T. Yang, and J. J.-P. Tsai, Eds., vol. 4159 of *LNCS*, Springer, pp. 912–923.
48. PERIS-LOPEZ, P., HERNANDEZ-CASTRO, J. C., TAPIADOR, J. M., AND RIBAGORDA, A. Advances in ultralightweight cryptography for low-cost RFID tags: Gossamer protocol. In *WISA 2008* (2009), K. Chung, K. Sohn, and M. Yung, Eds., vol. 5379 of *LNCS*, Springer, pp. 56–68.
49. ROGAWAY, P., BELLARE, M., AND BLACK, J. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security (TISSEC) 6*, 3 (August 2003), 365–403.
50. ROLFES, C., POSCHMANN, A., LEANDER, G., AND PAAR, C. Ultra-lightweight implementations for smart devices - security for 1000 gate equivalents. In *CARDIS 2008* (2008), G. Grimaud and F.-X. Standaert, Eds., vol. 5189 of *LNCS*, Springer, pp. 89–103.
51. SAARINEN, M.-J. O. Cryptanalysis of Hummingbird-1. In *FSE 2011* (2011), A. Joux, Ed., vol. 6733 of *LNCS*, Springer, pp. 328–341.
52. SAARINEN, M.-J. O. Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. In *FSE 2012: 19th International Workshop on Fast Software Encryption. 19-21 March 2012, Washington DC, USA. To appear* (2012).
53. SARMA, S. E., WEIS, S. A., AND ENGELS, D. W. Radio-frequency identification: Security risks and challenges. *CryptoBytes 6*, 1 (2003), 2–9.
54. SARMA, S. E., WEIS, S. A., AND ENGELS, D. W. RFID systems and security and privacy implications. In *CHES 2002* (2003), B. S. K. Jr., Ç. K. Koç, and C. Paar, Eds., vol. 2523 of *LNCS*, Springer, pp. 454–469.
55. SHAMIR, A. SQUASH - a new MAC with provable security properties for highly constrained devices such as RFID tags. In *FSE 2008* (2008), K. Nyberg, Ed., vol. 5086 of *LNCS*, Springer, pp. 144–157.
56. SUN, H.-M., TING, W.-C., AND WANG, K.-H. On the security of chien's ultralightweight RFID authentication protocol. IACR ePrint 2008/083, `http://eprint.iacr.org/2008/083`, February 2008.
57. VAJDA, I., AND BUTTYÁN, L. Lightweight authentication protocols for low-cost RFID tags. In *2nd Workshop on Security in Ubiquitous Computing, in conjunction with UbiComp 2003* (October 2003). *Note: This paper does not appear in the Springer UbiComp 2003 proceedings*.
58. WEIS, S. A., SARMA, S. E., RIVEST, R. L., , AND ENGELS, D. W. Security and privacy aspects of low-cost radio frequency identification systems. In *Security in Pervasive Computing 2003* (2004), D. Hutter, G. Müller, W. Stephan, and M. Ullmann, Eds., vol. 2802 of *LNCS*, Springer, pp. 201–212.
59. WHITING, D., SCHNEIER, B., LUCKS, S., AND MULLER, F. Phelix – fast encryption and authentication in a single cryptographic primitive. ECRYPT Stream Cipher Project Report 2005/027, 2005. `http://www.schneier.com/paper-phelix.html`.

Authenticated encryption in civilian space missions: context and requirements

Ignacio Aguilar Sanchez
European Space Agency
Keplerlaan 1
2200 AG Noordwijk
The Netherlands
+31-71-565-5695
ignacio.aguilar.sanchez@esa.int

Daniel Fischer
European Space Agency
Robert-Bosch-Str. 5
D-64293 Darmstadt
Germany
+49-6151-90-2718
daniel.fischer@esa.int

1 Introduction

Civilian space agencies have undertaken a joint long-term effort to engineer and standardise security concepts and solutions for the particular environment of civilian space missions development and operations. The motivation is mainly to benefit from acquired ad-hoc project experience, share knowledge, minimize the continuation of ad-hoc project engineering in this difficult domain and ease the adoption and implementation of security solutions for most future civilian space missions.

Space agencies work together under the aegis of the Consultative Committee for Space Data Systems (CCSDS). The European Space Agency (ESA), like the United States (US) National Aeronautics and Space Administration (NASA), is a founding member of the CCSDS. Within the Systems Engineering Area (SEA) of CCSDS, the Security Working Group has been tasked with addressing standardization in the area of civilian space mission security and supporting other working groups on security matters. This effort, in cooperation with other areas of CCSDS like the Space Link Services (SLS), has resulted so far in a number of publications (mainly reports, labelled Green Books, and standards, labelled Blue Books, whereas the latter ones are International Standards Organization (ISO) standards as well) addressing the protection of the space missions and its space links from malicious attempts to eavesdrop on sensitive data or violate the integrity of sensitive data.

As part of those standards, the CCSDS has developed the so-called Space Data Link Security (SDLS) protocol [1]-[2]. This security protocol offers security services to the three Space Data Link protocols [3]-[4]-[5] previously standardized by CCSDS: telecommand (TC), telemetry(TM) and Advanced Orbiting Systems (AOS). The protocol allows for the provision of the following recommended security services: authentication, encryption and authenticated encryption. The protocol offers flexibility in the selection of services and cryptographic algorithms. However, CCSDS promotes the adoption of certain services and algorithms [6]-[7] with the so-called 'Baseline modes' in SDLS and their companion cryptographic algorithms as recommend in another key CCSDS standard on security: Cryptographic Algorithms [8].

It is important for the cryptographic research community to understand that civilian space agencies have taken the practical approach of adopting where possible standardized civilian cryptographic algorithms issued by credible standard organizations like the ISO or the US National Institute of Standards and Technology (NIST). Those standardization efforts have been strongly supported by the cryptographic research community with competitions like the one resulting in the Advanced Encryption Standard (AES). However, space agencies have studied the

practical problems related to their use in the space environment and issues relevant for the space mission context, which are perceived to be unique in some cases, and not necessarily taken into account by those standardization efforts.

The currently SDLS-recommended authenticated encryption service is based on the well-known AES Galois Counter Mode (AES-GCM)[9]. The fact that a single cryptographic key and single algorithm can provide both authentication and confidentiality without the need for data padding or an initialization vector is an extremely attractive proposition for secure space mission operation.

However, it is understood that the fact that the Message Authentication Code (MAC) can reach a maximum of 128 bits, limits the security strength of the authentication to 64 bits due to birthday attack [10]. Or in other words, to exploit the full 256 bits of cryptographic key possible with an AES cipher, a (cipher-based) MAC as large as 512 bits should be required. No civilian standard for authenticated encryption attaining this level of security is known to civilian space agencies.

Space agencies are aware that an alternative to the use of Authenticated Encryption is the combination of separate Authentication and Encryption algorithms with their corresponding keys. While there is research literature on this topic, space agencies are not in a position to determine ground rules and/or preferable combinations of such algorithms. Although space agencies have initiated contacts with the cryptographic research community for their education and help in dealing with cryptographic problems relevant to space missions, space agencies are not equipped to perform cryptographic algorithm research. However, they have identified the potential need for a stronger authenticated encryption service for future missions as a key cryptographic research objective.

Therefore, the purpose of this paper is to illustrate some of the particular issues, concerns and requirements that a future authenticated encryption concept, or an equivalent alternative, should have to be beneficial for future civilian space missions. Before addressing them, it could be useful to recall the approach taken by CCSDS space agencies and in particular ESA to secure space missions.

2 Securing Space Missions

Two security problems are identified and differentiated when considering how to secure a space mission. The first one concerns the protection of the space mission assets and their infrastructure, e.g., the satellite or the constellation when more than one satellite is involved, the ground stations, the operations control centre(s), the mission control centre(s), the networks that interconnect them and the interface with the user(s). The second security problem corresponds to the protection of the mission products, that is, the signals and/or data produced by the spacecraft.

2.1 Space asset protection

Of particular relevance for most missions is the protection of space assets. Confidentiality, integrity and availability are the main TC and TM end-to-end data system features that could be subject to threats and, therefore, harm the regular conduct of mission operations. Threats to the spacecraft end-to-end data system include among others unauthorized access to spacecraft control (spoofing), denial-of-service on the command link, and traffic analysis. Depending on the mission criticality, which is determined by a contextual mission risk assessment at an early planning stage, measures may have to be implemented to deter these malicious attacks.

It is expected that as a minimum most space missions will have to avoid unauthorized control of spacecraft. Typically protection against this threat is countered with a Command Authentication protocol. Further protection can be achieved by ciphering commands and even telemetry. Finally, when utmost protection is required the command link availability can be improved by means of spread spectrum modulation using cryptographic codes, antenna null-steering, ground station diversity together with contact time diversity, improved spacecraft autonomy or a combination of them all.

2.2 Mission Products Protection

The second major security problem lies on the protection of the generated mission products. These mission products typically encompass satellite instrument raw data, ground-processed data for Earth observation or scientific missions but also ranging signals like the Public Regulate Service (PRS) signal for missions like GALILEO. This protection, once more depending on the space mission risk assessment, may involve measures like payload data down-link authentication and encryption or cryptographic spread spectrum (GALILEO) for the avoidance of undetected data manipulation and the enforcement of mission data products access control policy.

For some Earth observation missions like the future Global Monitoring for the Environment and Security (GMES) Sentinels, the security problem may be mostly confined to the interaction between the user and the payload data ground segment. Raw instrument data is acquired continuously by satellite instruments and downloaded during contacts with ground stations.

The two problems, inherently different, happen to share some common resources for the implementation of their design solutions. Thus, Telemetry, Tracking and Command (TT&C) networks not only have to implement security features to protect satellites but also have to implement specific features to support security management functions installed on satellite payloads like key management, relevant to mission products protection.

The reference system architecture in Fig. 1 illustrates the overall communications and data processing architecture supporting a space mission. Radio and data links inside the Space Link and Space Network boxes are the main concern for this paper. The radio links are implemented with specific space communications protocols. The data links inside spacecraft implement either space or non-space link or bus data communication protocols. In contrast to those, the links within the Ground Network box can much more easily benefit from commercial products available to secure terrestrial links and networks.

In securing the space communications links, the current approach followed by space agencies for missions having a simple network topology consists on implementing end-to-end security with a combination of a novel frame security protocol specified by CCSDS and called Space Data Link Security (SDLS) and the so-called space link extension (SLE) service as illustrated by Fig. 2. The SLE services extend the space data link protocol link, formerly terminating at the ground station site, up to the mission control centre and thus creating a direct point-to-point connection. The SLE services are based on ground network and transport protocols. The reader is referred to [11] for a detailed overview of SLE.

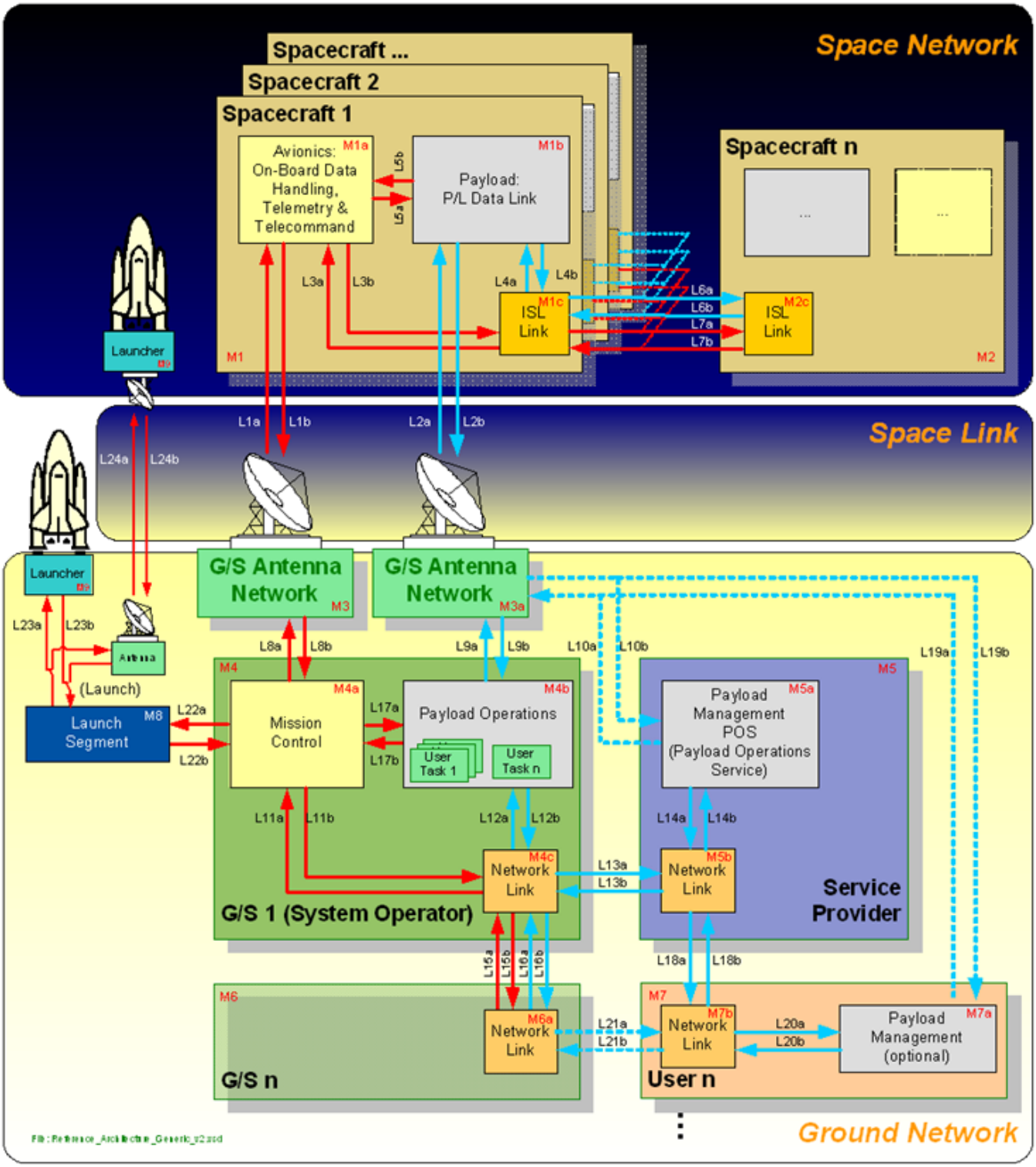


Figure 1 Reference System Architecture

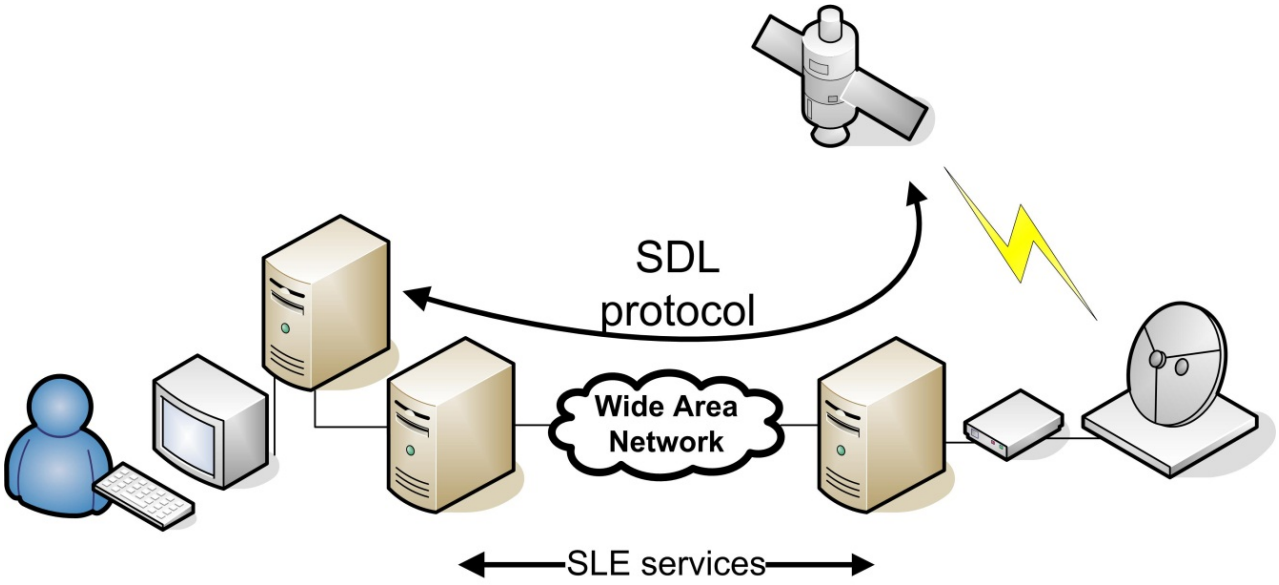


Figure 2 End-to-end Security Concept

In this simple topology payload and housekeeping telemetry are multiplexed on the same space link. In many missions however those telemetry data flows are separated, thus employing two space downlinks (see Fig. 3). In some cases, the payload telemetry downlink may not even flow to the mission control centre, but directly to the customer or payload processing centre.

Securing end-to-end communications between the mission control centre and the spacecraft is divided into two tasks. The first task is protecting the SLE services and ground network interconnecting the mission control centre and ground station(s).

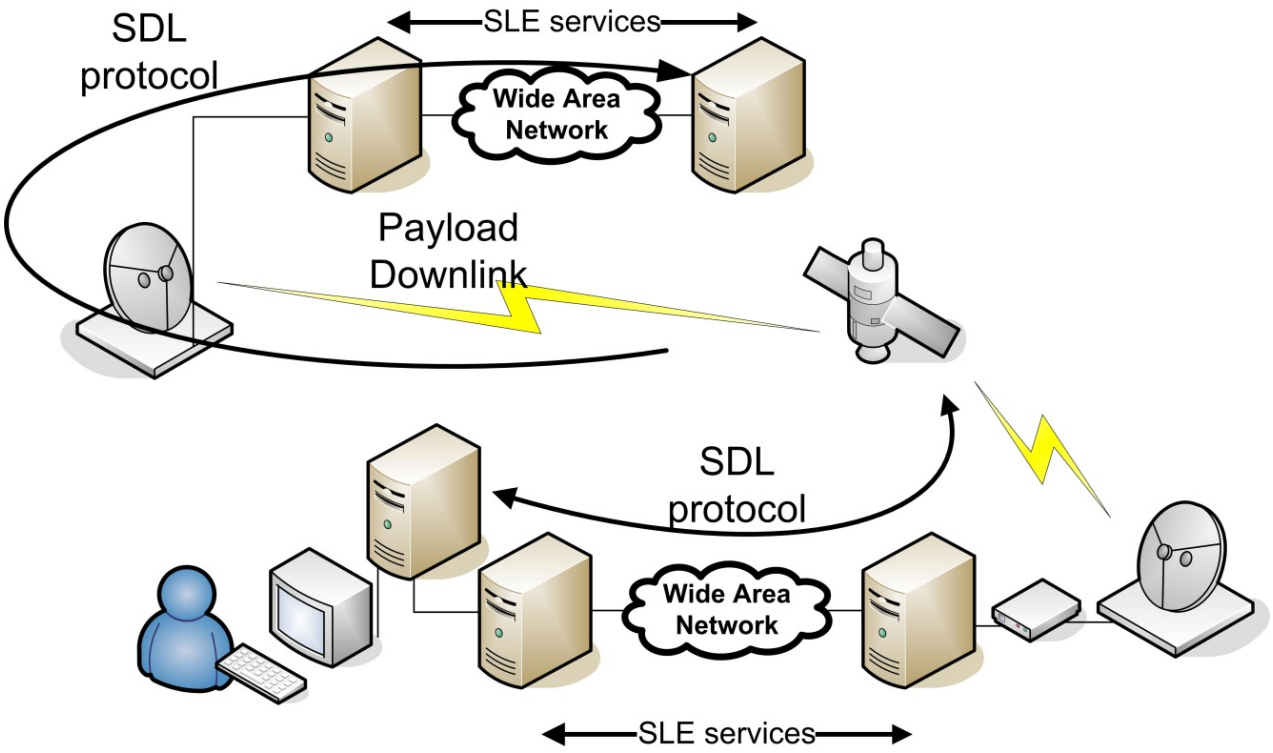


Figure 3 End-to-end Security with Payload Downlinks

The second task, the subject of the SDLS protocol, involves protecting the end-to-end space data link protocols now closed at the mission control system on the ground side.

This communications and data systems architecture provides interoperability between space agencies with SLE services and end-to-end security between a space agency and its spacecraft. A service level agreement is required to ensure secure cross-support with SLE services between the agency or operator owning the mission control centre and the agency owning or operating the ground station. It has to be noted again, that the link security is completely transparent to the third-party ground station, i.e. the owner of the ground station does not need to have access to any security critical information such as cryptographic keys. If needed, protection against denial of service on the TC space link can be achieved by using transmission security (TRANSEC) techniques in the physical layer (e.g., spread spectrum), which are perfectly compatible with SDLS protocol.

Data links between ground stations and mission control centres are protected with SLE services and corresponding protocols.

These simple network topologies are found on many Earth Observation and Science missions. Those topologies can also be applicable to Geostationary Telecommunication missions. Classical telecommunication missions rely on a single Telemetry, Command and Ranging (TCR) link. Some advanced telecommunication payloads have segregated direct payload control and configuration (PCC) links that could benefit from the second topology with the addition of an up-link.

2.3 Spacecraft Security Services Implementation

The security services like authentication, encryption or authenticated encryption as well as their supporting key management functions are typically implemented based on Hardware (HW) on the spacecraft side, while they are

implemented in SW on the ground side. Thus, it is evident that the spacecraft security implementation has the bigger impact in terms of cost and complexity while the ground side can easily adapt to the security approach chosen for the spacecraft. The choice of HW technology is driven by complexity, quality assurance (e.g., reliability, resistance to radiation), non-recurrent cost and security evaluation needs.

In some missions, the security function is integrated with other communications protocol and data handling functions inside an Application Specific Integrated Circuit (ASIC). As is well known, ASICs allow for higher density and complexity than Field Programmable Gate Arrays (FPGAs) in microelectronics designs. While they incur in higher non-recurrent cost, they might be preferred to satisfy the particular quality assurance requirements of that mission.

If the space mission implementing security functions is subject to security evaluation, a segregated implementation of the security functions with a so-called 'security module' on-board the spacecraft is preferred. The security module is supposed to define the boundaries for the security perimeter. Depending of the complexity of the module, one or more ASICs and/or FPGAs may be involved. The evaluation of the spacecraft security is thus limited to the evaluation of the security module. Evaluation authority can be provided with the module and associated documentation for security evaluation and testing. Furthermore, the development, integration and testing of the complete spacecraft can proceed in general without the need for the security function to be involved and avoid additional security-related constraints for those activities. Final integrated tests may require the inclusion of a security module, although equipped with 'test' keys (not the 'flight' keys). Flight keys may be injected shortly before launch, if the launch facility environment allows for such operational procedure in a secure manner.

However, in some future applications implementation by Software could substitute Hardware. For low data rate applications and in view of a better utilization of spacecraft data processing resources, the concept of 'secure partitioning' is appealing. This is a concept inherited from the aeronautics industry and adapted to security applications. The idea is to build isolated virtual machines running on a common processor. One of those virtual machines could process security. Critical for the viability of this technology is the assurance that such virtual machine would not 'leak' and/or be affected by performance of other virtual machines running on the same processor.

3 Issues, concerns, constraints and requirements

The following sections briefly introduce some of the main issues, concerns, constraints and requirements currently identified as being relevant for space mission security and cryptography.

3.1 Payload vs. Platform Space Links

Typically spacecraft include two types of space data links: platform and payload links.

Platform links are used to support spacecraft operation control and monitoring. Telecommand and Housekeeping telemetry are typically channelled through a particular radio link. The data rate requirements are generally modest, with telecommand of the order of a few kbps and telemetry slightly higher to tens or few hundreds of kbps.

Payload links are used to download instrument data. For very large volumes of data, as generated by optical instruments or microwave radars, a separate radio link is typically used. Data rates can range to several hundreds of Mbps and are expected to reach close to 2 Gbps and possibly up to 6 Gbps in future space missions. The volume of data to be secured is potentially much, much larger than for spacecraft platform data links.

Thus, it is expected that two separate security functions with different operational concepts in terms of cryptographic key management will continue to be required. Note that the platform security function provides a secure channel for the control and monitoring of the payload security function. Among other things this channel supports cryptographic key management.

3.2 Telecommand vs. Telemetry Links

The telecommand link communicates the commands to the spacecraft. It is the most critical space link in what concerns the spacecraft protection as an asset. Thus, telecommand security is a first priority in securing space missions. In particular the authentication service applied to spacecraft telecommand frames provides the assurance that the spacecraft (space asset) can only be controlled/ commanded by an authorised control centre. In other words, the spacecraft would reject commands that cannot be validated to originate from a genuine source.

In addition, certain telecommands will require that the data they contain is encrypted. In order to support over-the-air-rekeying (OTAR), telecommands with encrypted packet data field will be used to upload new session (or traffic) keys.

Beyond these two basic complementary security services, that we consider the bare minimum space missions should implement, we anticipate that some missions may not want that their telecommand data is readable by unauthorized parties (i.e. protection against telecommand eavesdropping). In such case all the telecommands will be encrypted at frame level.

In contrast to telecommanding, telemetry links are most concerned by the second security problem: protecting the data generated by the spacecraft. The priority will be to make sure that only those entities which are authorized to read the encrypted data can actually do this. The confidentiality service, based on encryption, coupled with a proper encryption/decryption key management/distribution concept will provide such assurance. It has to be noted that, depending on the telemetry distribution mechanism of the particular mission, key management may pose a complex problem.

From our understanding of cryptography, and in order to protect against undetected data manipulation, we recommend for space missions the use of the authenticated confidentiality service. This service employs an authenticated encryption algorithm.

3.3 Limited or absent in-flight maintenance capability

Once spacecraft data processing functions are implemented, validated, tested, and the spacecraft is launched, it is not possible to perform maintenance tasks in flight. This of course excludes any on-board software, which could be updated during flight. Thus, the security function, which is a data processing function, has to be conceived to be robust and reliable with no allowance for failure or degradation of the security level provided.

Since it is well known that cryptographic techniques and attacks progress with time and technology improvements in the ‘ground’ (e.g., computing power), cryptographic solutions planned to be embarked in long-term space missions require to be effective until the end of life of the space mission or beyond. Usually mission lifetime is extended beyond the initially planned duration.

To illustrate the example note that an advanced telecommunications satellite may remain operational up to 15 years in orbit. Depending on the maturity of the technology, 4 or 5 years before the launch the security function design may have been established. The manufacturer will plan to use such product for a number of years, maybe 5 to 10 years. One can see that space missions can easily take ‘security’ commitments that could last 20 to 30 years.

We currently address this issue with substantial 'margins', if one can say so in the security domain, in defining the security functions. Longer MACs and longer cryptographic keys than are actually required as a result of the attack models are employed. More keys than initially anticipated are embarked and planned for. The view is that one can adjust (reduce) the cryptographic period of critical keys according to time (date). However, it is clear that the protection resulting from this approach is limited to evolution of computational capacities as described by Moore's Law. Of course, margins cannot protect against a total breach of the cryptographic algorithm, i.e. by discovering a flaw in one of its processing functions or if it is discovered that certain keys are insecure.

3.4 Limited scope for side channels

As long as the security function design, development, qualification and testing is ensured in a controlled manner, there is no need for side channel attack protection for the implemented spacecraft security functions. Human beings cannot (within effort that could be considered feasible) access flying spacecraft (astronauts are an exception!). Flight cryptographic keys are injected shortly before launch under a secure procedure so their exposure before flight is very limited. However, note that the same cannot be said for the corresponding implemented security function in the ground segment in charge of the mentioned spacecraft.

Thus, while not a priority for flight units, it might still be interesting to protect ground cryptographic units against side-channel attacks. Note, however, that access to cryptographic units will be generally controlled and protected. Having said that, the scope for insider attack is always a possibility. And defence-in-depth philosophy is very much liked in space system security engineering.

3.5 Unique crypto-period

Spacecraft operations are complex, delicate and rely on a long tradition of proven approaches and concepts. The arrival of security functions and cryptography to civilian space missions has been met with reluctance by stakeholders, developers and operators. As any novel 'thing' in space mission engineering it is perceived as a potential 'troublemaker'.

Therefore, it is crucial that proposed security functions and concepts are easy to operate, reliable and predictable.

Maintaining effective security and in particular valid cryptographic keys is a new operational burden for spacecraft operators although some telecommunication satellite operators have already some experience. The simpler it is, the better for its acceptability.

In considering the combination of authentication and encryption, an ideal design should provide similar crypto-period for both authentication and encryption. One could even explore the possibility to share the length of the cryptographic material between authentication and encryption such that this is achieved. Imagine for instance having 512 bits of an 'equivalent' authenticated encryption key to be shared. Perhaps one could find that say 128 bits for encryption and 394 bits for authentication achieve that result. Flexibility in apportioning these values considering the specifics of a particular mission context could be valuable.

3.6 Flexible MAC length

In applications like telemetry protection, the additional overhead incurred by including a MAC in a frame is acceptable given the generally long length of the frames. Hence, exploiting the theoretically possible longest MAC given a cryptographic key length is possible.

In contrast, in applications like telecommand protection where the message to be protected, a TC transfer frame, has variable length ranging from 64 bits up to 1024 octets, the pressure to reduce overhead to the strictly minimal

to guarantee security may win the day. Short telecommand frames, also called High-Priority Commands (HPCs) are essential in off-nominal conditions to recover space missions. For example the spacecraft could be tumbling, making difficult the acquisition, tracking and re-acquisition of the received TC signal both at radio and data level. Likewise, the radiated TM signal may be difficult to be received by the Ground Station. In fact, the Mission Control may be operating without TM, just hoping to send TCs to recover the situation ('blind commanding'). Although the coverage provided by the TC and TM radio link antennas is close to omnidirectional, there is typically a region of interferometers produced by the combination of the radio signal received by two hemispherical antennas resulting in large signal amplitude variation at the input of the receiver. The individual antenna patterns are distorted by electromagnetic interaction with the surrounding spacecraft structures. Furthermore, certain structures like the solar panel can block the radio signal. Moreover, the tumbling rate introduces an additional frequency variation. Similar phenomena affect the radiated TM signal. Thus, in such conditions short messages have higher chances to be received than long messages. These messages could contain just a single HPC. Short MACs and frequent rekeying may be preferred.

However, given the long duration of space mission development and operational lifetime, a provision to increase such MAC length with time could be attractive to some extent. For instance a mission may initially fly with a 128 bit MAC and evolve with time to longer value like 196 bit or longer.

3.7 Intermittent contact and latency

In some space missions radio contact can only be established for short period of times. Typical example is the so-called Low Earth Orbit (LEO) Earth Observation missions where once per orbit a period of about six to twelve minutes is available to exchange data with a ground station. The particular contact times/duration depend very much on the orbital parameters of the mission, the available ground stations to communicate with the spacecraft and the spacecraft –ground station geometry for a particular pass.

However, note that in other cases such as missions with geostationary orbits, the contact time may be continuous. This offers more scope for attacks directly to the spacecraft but also more monitoring capability by the legal space mission operator.

For deep space missions, the data latency is substantial. This is particularly relevant for communication protocols that include re-transmission loops. However, note that deep space missions are so far not particularly troubled by security issues, maybe at their own peril! After all, as any other space mission deep space missions include the so-called launch and early orbit operation (LEOP), which follows the launch phase and is particularly critical for all space missions.

3.8 Anti-replay

Protection against replay attacks is a mandatory requirement for space mission telecommand when implementing authentication or authenticated encryption. A replay attack consists on recording a previously sent telecommand and up-link it with the intention of having it executed. The usual protection approach is to include a time-dependent information element, generally a counter value, as part of the message to be authenticated.

In contrast, anti-replay is not perceived as critical for telemetry in civilian space missions. The generation of rogue telemetry and reception by a ground station without raising suspicion to the operators is a very difficult proposition. The TM *spoofer* would need to make sure to replicate certain analogue signal performances like the Doppler and amplitude profile, typical of a pass, experienced by the ground station receivers. Those receivers are coupled to very directive antennas, whose maximum gain is limited to a very small solid angle of their radiation pattern and whose pointing is often controlled by the received RF signal (auto-tracking) or by a program based on

the predicted azimuth and elevation parameters for the pass. For certain very high data rate TM downlinks, due to both the very high data rate and antenna directivity, it becomes even harder.

3.9 Operation both in connection and connectionless protocols

In designing security for space link protocols a central consideration has been the compatibility with communication scenarios where there is no guarantee of repetition, omission or sequence in a group of messages. Unfortunately, in off-nominal conditions telecommand to a spacecraft may be carried out in a 'blind' mode, that is, without telemetry to supervise spacecraft operation and in particular re-transmission mechanisms inherent to the sequence-controlled service offered in nominal condition.

Because of these two scenarios the adopted approach has been to rely on sequence counters with some flexibility in the values that can be accepted by the receiving processor.

3.10 Graceful transition from secure to clear mode

Given the prevalence of safety over security in civilian space missions, the demand of a so-called CLEAR mode, which implies bypassing the security functions, is a must. Passing from authenticated encryption to authentication only to clear mode could be a new transition model between SECURE and CLEAR modes. It is unclear, though, to what extent such property could be attractive. Certainly being able to maintain always authentication in applications like telecommand could provide an advantage in avoiding periods in which a spacecraft is vulnerable to unauthorized telecommand.

3.11 Forensic analysis of secure data

It is anticipated the possible need to be able to decrypt telemetry frames totally or partially even if they cannot be authenticated. Unfortunately, failure analysis of major space mission catastrophes like the last US Space Shuttle Columbia required investigators to be able to extract meaningful data from telemetry frames initially rejected by frame processors given they contained too many symbol errors but partially recovered afterwards.

Certain cryptographic algorithms/modes limit the error propagation caused by communication channel errors. For instance in the Counter Mode a single channel bit error propagates to a single bit decryption error. To maintain such limited error propagation will be desirable for future cryptographic algorithms/modes used in space communications.

Nevertheless, the ability to extract some meaningful data from noisy data may be at odds with attempting to secure the data in the first place. Unfortunately, the trade-off between safety and security is recurrent in many a space mission design, development and operations concept. No valid answer or general recipe has been found so far. What experience has shown is that civilian space missions tend to privilege safety over security.

3.12 High number of invocations with a given key

In space missions, cryptographic keys are limited resources. As a consequence cryptographic keys should be used in the most efficient manner. Current cryptographic key management concepts for simple space mission network topology rely on Symmetric Key Infrastructures (SKIs). A limited number of static keys, so called Master keys, are pre-loaded with PROM/EEPROM before a spacecraft is launched. Those Master keys can be used as Key Encryption Keys (KEKs) to support the secure transfer from ground to space of Session keys (or Traffic keys), which are actually used to secure the data transmitted on the space links.

Agencies like ESA are researching symmetric key infrastructures for space missions with particular research action [12], aimed to optimize the number of master and session keys considering the particular mission constraints.

3.13 Arrangement of encryption and authentication operations

From our understanding, there are several ways in which encryption and authentication operations can be combined: authenticate-then-encrypt, authenticate-and-encrypt, encrypt-then-authenticate.

Our preference, in line with the conclusions highlighted by cryptographic research [13] has been encrypt-then-mac. AES GCM [9] fulfils that order of operations. AES CCM [14] does not. Mainly for this reason, AES GCM has been considered more secure and, therefore, preferred for CCSDS standardization.

4 Conclusion

Space agencies rely on the civilian cryptography research community and standardization bodies to develop advanced cryptographic algorithms for their future civilian space missions. This white paper has briefly addressed a number of topics considered relevant for the evolution of cryptographic algorithms, in particular authenticated encryption, used to implement security services on space communications links supporting space mission operations. It is hoped that such paper will provide further input to stimulate cryptographic research.

5 References

[1] I. Aguilar Sanchez, C. Biggerstaff, D. Fischer, G. Moury, B. Saba, H. Weiss, *"Towards Completion of the CCSDS Space Data Link Security Protocol"*, proc. IEEE Aerospace Conference 2012, Big Sky, Montana, USA, 2012.

[2] "Space Data Link Security Protocol", Draft Recommended Standard CCSDS 355.0-R-2, Red Book, February 2012.

[3] "TC Space Data Link Protocol", Recommended Standard CCSDS 232.0-B-1, Blue Book, September 2003.

[4] "TM Space Data Link Protocol", Recommended Standard CCSDS 132.0-B-1, Blue Book, September 2003.

[5] "AOS Space Data Link Protocol", Recommended Standard CCSDS 732.0-B-2, Blue Book, July 2003.

[6] "Authentication/Integrity – Algorithms Issues Survey", Informational Report CCSDS 350.3-G-1, Green Book, March 2008.

[7] "Encryption Algorithm Trade Survey", Informational Report CCSDS 350.2-G-1, Green Book, March 2008

[8] "Cryptographic Algorithms", Draft Recommended Standard CCSDS 352.0-R-1, Red Book, February 2012.

[9] "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, November, 2007.

[10] A. Menezes, P. van Oorschot, and S. Vanstone, "Handbook of Applied Cryptography", Chapter 9, par. 9.33, p.366, CRC Press, 1996.

[11] "Space Link Extension Services – Executive Summary", Informational Report CCSDS 910.0-G-2, Green Book, March 2006.

[12] M. Juliato, I. Aguilar Sanchez, C. Gebotys, "Cryptographic Key Infrastructure for Security Services Protecting TT&C and Payload Links on Space Missions", poster presented at Technology Innovation Days, ESTEC, Noordwijk, The Netherlands, 2011.

[13] M. Bellare, N. Namprere, "*Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm*", Advances in Cryptology | ASIACRYPT 2000, Volume 1976 of Lecture Notes in Computer Science, pages 531-545, Kyoto, Japan, December 3-7, 2000.

[14] "Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality", NIST Special Publication 800-38C, May, 2004.

Stronger Security Guarantees for Authenticated Encryption Schemes

Alexandra Boldyreva[1], Jean Paul Degabriele[2], Kenneth G. Paterson[2], and Martijn Stam[3]

[1] Georgia Institute of Technology
[2] Royal Holloway, University of London
[3] University of Bristol

Abstract. We here refer to recent and upcoming research that is relevant to Authenticated Encryption (AE) schemes as used in practice. This research leads directly to novel design criteria for authenticated encryption schemes. The first piece of research considers the *security of symmetric encryption in the presence of ciphertext fragmentation*, which is a setting that commonly arises in practice. Recent attacks have shown that provably secure AE schemes may become insecure when used in this setting. We propose a theoretical framework for the analysis of AE schemes in the presence of ciphertext fragmentation. This framework enables us to study the relationship between confidentiality, the ability to hide boundaries between ciphertexts, and the resistance of schemes to Denial of Service attacks. It also leads to new designs for AE schemes. The second piece of research relates to *distinguishable decryption failures*. Most security proofs implicitly assume that an adversary is unable to distinguish distinct failure events that occur during the decryption of a ciphertext. Practice has shown that this assumption is not well justified: implementations often leak information that allows an adversary to distinguish among decryption failures, either through error messages or timing, thereby opening up avenues of attack. Our upcoming work analyses the security of AE schemes in the setting where failure events are distinguishable, providing new models and security relations for this setting. From this work, we can extract design criteria which protect implementations against such error-based side channel attacks.

1 Ciphertext Fragmentation

A major application of authenticated encryption (AE) schemes is to protect communication over the internet. The classical examples are SSL/TLS, IPsec, and SSH. These protocols traditionally employed AE schemes obtained through generic compositions of MAC algorithms and stream ciphers or modes of operation of block ciphers, but were later extended to support more efficient AEAD schemes. Recently SSH and MAC-then-encrypt configurations of IPsec have fallen foul of attacks that exploit ciphertext fragmentation [2, 7]. Fragmentation is a necessary feature of the TCP/IP protocol stack that allows it to abstract

away the low-level details of the distinct physical networks that constitute the Internet. While details vary depending on the implementation, the interface seen by an application communicating over TCP/IP is often that of a *fragmented channel*. In a fragmented channel, messages may be arbitrarily split into smaller chunks (hereafter referred to as fragments) while in transit. Each message fragment is then delivered to the receiver independently. At the receiver side the application needs to somehow take this sequence of message fragments and recover the original individual messages. The use of encryption over a fragmented channel therefore introduces *ciphertext fragmentation*. This feature imposes new requirements on the decryption algorithm of an AE scheme. In particular, it must maintain correctness and security when receiving a sequence of fragments of ciphertexts, even with individual fragments possibly spanning ciphertext boundaries. As the SSH and IPsec attacks demonstrate, security (privacy) in the standard model for AE does not necessarily imply security in the fragmented setting.

In our recent work [4] we provided a formal treatment of security of AE in the ciphertext fragmentation setting. Our analysis there brings to the fore the fact that attempts to support ciphertext fragmentation often introduce new security vulnerabilities relating to Denial of Service (DoS) and Traffic Analysis. In [4], we formalise certain types of Denial of Service and Traffic Analysis attacks and construct an AE scheme, InterMAC, demonstrating that it is possible to efficiently and simultaneously achieve all three of our security notions: confidentiality in the fragmented, chosen ciphertext setting, anti-DoS, and hiding of ciphertext boundaries. On the other hand we acknowledge that the InterMAC scheme does not match the efficiency of state-of-the-art AE and AEAD schemes like OCB, and we encourage further research to find more efficient schemes that meet our security notions for the fragmented setting.

In view of the known attacks and the wide application of schemes in the fragmented setting, we argue that a good AE scheme should be able to operate over fragmented channels in a secure manner if it is intended to be truly versatile. In this sense, our work can be seen as providing new design criteria for AE schemes.

2 Distinguishable Decryption Failures

This work is motivated by the recurring question of whether IND-CCA is the right privacy notion for AE. It is well understood by the cryptographic community that in many practical settings IND-CPA security is not enough. A number of attacks have demonstrated that partial information about the plaintext may leak from the receiver's side during decryption. An adversary could then exploit this in an iterative manner by forging related ciphertexts to recover further information about some target ciphertext. Intuitively, the IND-CCA security notion should prevent this kind of attack, as it takes the conservative approach where the adversary is given access to a decryption oracle to which it can adaptively query any sequence of ciphertexts (excluding the challenge ciphertext). Thus an IND-CCA-secure scheme ensures that the only information an adversary can

gather from a ciphertext is the information leaked by the implementation (if any) during the decryption of that ciphertext.

In [8] Krawczyk showed that MAC-then-encrypt with CBC mode encryption yields a secure channel. By combining results from [5] and [3], one can extend Krawczyk's proof to show that this configuration also yields an IND-CCA secure scheme. Nevertheless attacks against TLS [6], DTLS [1], and IPsec [7] have successfully managed to break this MAC-then-encrypt construction through attacks of the above nature. The explanation for this apparent contradiction resides in the way schemes are formalised, rather than IND-CCA being an inadequate notion for privacy. In practice, a variety of checks may be carried out during the decryption of any given ciphertext, with the ciphertext being deemed invalid if any one of these checks should fail. Now if an adversary can distinguish among the different decryption failure events, this may provide him with sufficient partial information to enable him to mount chosen-ciphertext attacks – in essence, invalid ciphertexts may leak information too. In fact, the above-mentioned attacks can recover the full plaintext without ever querying the decryption oracle with a valid ciphertext. However the standard formalism employed to study the security of AE schemes implicitly assumes that decryption failures are indistinguishable, and hence instances where the type of the decryption failure leaks information are not accounted for in this formalism.

Recent work [9, 10] has considered the security of block cipher modes and AE schemes under one specific class of attacks involving distinguishable error events, namely padding oracle attacks [13]. In upcoming work we analyse in general the setting where decryption failures are distinguishable. We extend established security notions and relations for AE to this setting, and study the security of generic compositions. Our approach enables the more accurate modelling of AE schemes as they are used in practice, and highlights the importance to practice of taking great care in handling errors when implementing AE.

Authenticated encryption was first analysed by Bellare and Namprempre in [5]. Their work contributed the following two useful relations:

$$\text{IND-CPA} \wedge \text{INT-CTXT} \Rightarrow \text{IND-CCA}, \text{ and INT-CTXT} \Rightarrow \text{INT-PTXT}.$$

For a symmetric scheme it is therefore sufficient to prove semantic security together with ciphertext integrity, and it will automatically guarantee chosen-ciphertext security and plaintext integrity. Consequently the combination of semantic security and ciphertext integrity has become the generally accepted security notion for authenticated encryption. These have been elegantly combined into an equivalent notion described by a single game in [12] and [11]. In our upcoming work however, we show (by means of a separation) that for symmetric schemes with multiple error symbols, semantic security and ciphertext integrity no longer imply IND-CCA security. This illustrates that in general this security notion no longer guarantees confidentiality against active adversaries when decryption failures are distinguishable. We fill this gap by deriving similar relations for proving chosen-ciphertext security in the multiple error setting. In addition, an ambiguity arises as to how ciphertext integrity should be defined

in the multiple error setting. We address this technicality by presenting a non-trivial separation showing that our choice for this definition is indeed necessary.

Again, from our work, one can extract new design criteria for AE schemes. Firstly, they should be formally proven to resist attacks based on distinguishable decryption failures (in the case where the scheme cannot be implemented without introducing such distinguishability). Secondly, and more informally, schemes should be "implementation-proof," meaning that it should be difficult to inadvertently introduce additional sources of distinguishability of errors during implementation.

References

1. N.J. AlFardan and K.G. Paterson. Plaintext-Recovery Attacks Against Datagram TLS. In *Network and Distributed System Security Symposium (NDSS 2012)*.
2. M.R. Albrecht, K.G. Paterson, and G.J. Watson. Plaintext recovery attacks against SSH. In *IEEE Symposium on Security and Privacy*, pages 16–26. IEEE Computer Society, 2009.
3. M. Bellare, O. Goldreich, A. Mityagin. The Power of Verification Queries in Message Authentication and Authenticated Encryption. *IACR Cryptology ePrint Archive*, http://eprint.iacr.org/2004/309.
4. A. Boldyreva, J.P. Degabriele, K.G. Paterson, and M. Stam. Security of Symmetric Encryption in the Presence of Ciphertext Fragmentation. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 682–699, 2012.
5. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto (ed.), *ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.
6. B. Canvel, A.P. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password interception in a SSL/TLS channel. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 583–599, 2003.
7. J.P. Degabriele and K.G. Paterson. On the (in)security of IPsec in MAC-then-Encrypt configurations. In E. Al-Shaer, A.D. Keromytis and V. Shmatikov (eds.), Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS 2010), pp. 493-504, ACM, 2010.
8. H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer, 2001.
9. K.G. Paterson and G.J. Watson. Immunising CBC Mode Against Padding Oracle Attacks: A Formal Security Treatment. In R. Ostrovsky, R. De Prisco and I. Visconti, editors, *SCN 2008*, volume 5229 of *Lecture Notes in Computer Science*, pages 340–357, Springer 2008.
10. K.G. Paterson and G.J. Watson. Authenticated-Encryption with Padding: A Formal Security Treatment. In D. Naccache, editor, *Cryptography and Security: From Theory to Applications – Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, volume 6805 of *Lecture Notes in Computer Science*, pages. 83–107, Springer 2011.

11. P. Rogaway, and T. Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390, 2006.
12. T. Shrimpton. A Characterization of Authenticated-Encryption as a Form of Chosen-Ciphertext Security. *IACR Cryptology ePrint Archive*, http://eprint.iacr.org/2004/272.
13. S. Vaudenay. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS .... In L.R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546. Springer 2002.

Authenticated Encryption in Practice

David McGrew

Cisco Systems
USA

Over the last decade, authenticated encryption has become accepted as the appropriate way to provide symmetric confidentiality services. It has been adopted into many standards, though in many cases it is currently an option that is not typically exercised. This talk reviews issues that have arisen in the use of authenticated encryption in practice, and considers the role of authenticated encryption in cryptographic architectures.

On Some Constructions for Authenticated Encryption with Associated Data

Palash Sarkar
Indian Statistical Institute

Abstract

Two approaches to achieving authenticated encryption with associated data (AEAD) will be considered.

The first approach is to use certain block cipher modes of operations which make a single pass over the message. For this approach, both tweakable block cipher based constructions and direct constructions will be described. Both types of constructions yield families of AEAD functionalities which offer high efficiency along with other desirable features such as easy reconfigurability.

In the second approach, a stream cipher supporing an initialisation vector is used in conjunction with a hash function with provably low differential probabilities. Several possibilities will be outlined, both with and without the requirement of authenticating associated data. This approach gives efficient solutions for stream ciphers which are either very fast in software or have very low hardware footprint.

Security of the constructions are derived using the 'provable security' methodology for the single-user setting. Issues related to security in the multi-user setting will be briefly mentioned.

The discussion will primarily touch upon the material in the following papers.

References

[1] D. Chakraborty and P. Sarkar. A General Construction of Tweakable Block Ciphers and Different Modes of Operations. *IEEE Transactions on Information Theory*, 54(5):1991–2006, 2008.

[2] S. Chatterjee, A. Menezes and P. Sarkar. Another Look at Tightness. In *Proceedings of Selected Areas in Cryptography, 2011*, Lecture Notes in Computer Science, volume 7118, pages 293–319.

[3] P. Sarkar. Pseudo-Random Functions and Parallelizable Modes of Operations of a Block Cipher. *IEEE Transactions on Information Theory*, 56(8):4025–4037, 2010.

[4] P. Sarkar. A Simple and Generic Construction of Authenticated Encryption with Associated Data. *ACM Transactions on Information and Systems Security*, 13(4):33:1–33:16, 2010.

[5] P. Sarkar. On Authenticated Encryption Using Stream Ciphers Supporting an Initialisation Vector. *Cryptology ePrint Archive, Report 2011/299*, `http://eprint.iacr.org/2011/299`.

Efficient Lightweight AES-Based Authenticated Encryption

Andrey Bogdanov[1], Florian Mendel[1], Francesco Regazzoni[2], and Vincent Rijmen[1]

[1] KU Leuven, ESAT/COSIC and IBBT, Belgium
[2] ALaRI - USI, Switzerland

Abstract. In this paper, we propose an *A*uthenticated *L*ightweight *E*ncryption algorithm coined *ALE*. The basic operation of ALE is the AES round transformation and the AES-128 key schedule. ALE is an on-line single-pass authenticated encryption algorithm that supports optional associated data. Its security relies on using nonces.
We provide an optimized low-area implementation of ALE in ASIC hardware and demonstrate that its area is about 2.5 kGE which is almost two times smaller than that of the lightweight implementations for AES-OCB and ASC-1 using the same lightweight AES engine. At the same time, it is at least 2.5 times more performant than the alternatives by requiring only about 4 AES rounds to both encrypt and authenticate a 128-bit data block.
Keywords: authenticated encryption, lightweight cryptography, AES

1 Introduction

Motivation. As essential security applications go ubiquitous, the demand for cryptographic protection in low-cost embedded systems (such as RFID and sensor networks) is drastically growing. This necessitates secure yet efficiently implementable cryptographic schemes. In such use cases, the area and power consumptions of a primitive in hardware are usually of paramount importance and standard solutions are often prohibitively costly to deploy.

Once this problem was recognized, the cryptographic community was fast to address it by proposing a great deal of specialized lightweight cryptographic algorithms, which include stream ciphers like Trivium [10], Grain [21], and Mickey [4], block ciphers like SEA [31], DESL, DESXL [27], HIGHT [22], mCrypton [23], KATAN/KTANTAN [9], and PRESENT [6], and hash functions like Quark [3], Photon [20], and Spongent [7] — to mention only some fraction of them.

However, when it comes to *authenticated encryption* – the fundamental security functionality in most real-world security systems – one has to establish that, rather surprisingly, only a few lightweight schemes have been proposed so far, examples are Grain-128a [2] and Hummingbird-2 [18]. At the same time, message secrecy – as provided by plain encryption – is often of limited value in practice if not accompanied by message authentication. This stipulates the acute need for authenticated encryption in the field which is reflected in NIST [17] and ISO/IEC [1] documents on modes of operation for block ciphers.

In the context of lightweight cryptography though, these standard modes of operation have significant practical limitations. First, the lightweight block ciphers are usually designed to save on state bits, so that the block size and key size are usually kept at the edge of the reasonable minimum (it is rather typical in lightweight cryptography to propose a block cipher with a 64-bit block and a 80-bit key). This significantly confines the security level of modes of operation theoretically attainable due to generic attacks. Second, the standard authenticated-encryption modes of operation traditionally aim at high-speed implementations by minimizing the number of block cipher calls and other operations one has to perform per data block processed. For example, OCB [30], which clearly outperforms such wide-spread schemes as AES-GCM and AES-CCM in standard software, requires essentially a single AES call per data block at bulk encryption only. However, such modes usually do not pay too much attention to the amount of memory and the circuit size one needs in a lightweight hardware implementation. For instance, AES-OCB requires at least four 128-bit registers and both AES-encryption and -decryption engines for both encryption/authentication and decryption/verification.

A straightforward solution would be to address the first limitation (small state) by raising the total internal state size of the lightweight primitives, for instance, to 256 bits to avoid generic attacks up to a bound of 2^{128} operations. However, this would in turn take away their major source of advantage and make their area occupation comparable to that of AES-128. That is why we feel that a dedicated authenticated-encryption design can also be based on AES when 128-bit level of security is desired.

In an attempt to mitigate the second limitation (additional memory requirements imposed by modes) one might choose to go for encrypt-then-mac or mac-then-encrypt. However, not only would it jeopardize the highly relevant implementation goal for the scheme to be on-line but also require double message input (being essentially two-pass) and twice more operations per data block than e.g. OCB. In general, there appear to be no single-pass authentication encryption modes of operation for block ciphers preserving the minimal state size required. This emphasizes the demand for a dedicated lightweight authenticated-encryption design.

Moreover, we also want to make this new design fast in software, as opposed to some bit-oriented lightweight ciphers (such as Grain, Trivium, KATAN, PRESENT, etc.) which succeed in attaining a low area in hardware but whose performance in software is not even comparable to that of AES, especially in the presence of the Intel AES instructions. We feel that not much efficiency can be gained by designing a slightly more efficient *generic* authenticated-encryption mode of operation for block ciphers since the bottleneck will remain the one block cipher call per data block. This is not only true for authenticated encryption modes but also for MAC-only and encryption-only modes.

The situation is, however, essentially different if the designer is allowed to look inside the specific underlying block cipher such as AES and to construct a dedicated mode of operation which only uses exactly as many operations of the underlying block cipher as needed. This is the approach taken in the designs of the stream cipher LEX [5] and the

message authentication algorithm Pelican [13]. Lately, similar reasoning was applied to the setting of authenticated encryption resulting in the design of ASC-1 [25].

ALE. This paper proposes a lightweight authenticated encryption algorithm based on AES called *ALE* (*A*uthenticated *L*ightweight *E*ncryption). It is a single-pass nonce-based on-line scheme that preserves the memory alignment of data. The design of ALE combines some ideas of Pelican, LEX and ASC-1 in a lightweight manner. In a nutshell, the algorithm uses Pelican keyed in all rounds (similarly to PC-MAC [28]) for computing the authentication tag and leaks bytes of the state in every round in a LEX-type way for encryption/decryption. It has a 256-bit secret internal state dependent on both key and nonce.

By requiring only 2.5 kGE of area in lightweight ASIC hardware, which is less than 100 GE overhead compared to plain AES-ECB in the smallest implementation available [29], ALE is about half the size of AES-OCB and ASC-1. In terms of speed in the lightweight implementation, ALE is about 2.5 times faster than AES-OCB and about 4.5 times faster than ASC-1 in its smallest implementation.

At a first glance, the overall design philosophy of ALE might seem similar to that of ASC-1. However, as the numbers of relative area and speed above already strikingly suggest, ASC-1 has several crucial shortcomings in the way of practical lightweight implementation. First, ASC-1 needs an internal state which is twice larger than that of ALE. Second, it is slow in its most compact (and most serial) implementations due to the order in which the round keys are used. Finally, ASC-1 does not accept associated data that can be vital in some networking settings while ALE explicitly deals with it.

The remainder of the paper is organized as follows. Section 2 gives a specification of the algorithm. Section 3 introduces some elements of its cryptanalysis. Section 4 provides lightweight implementation numbers of the algorithm in ASIC hardware. We conclude in Section 5.

2 The Authenticated Lightweight Encryption (ALE) algorithm

In this section, we describe ALE – our new authenticated lightweight encryption algorithm. The basic operation of ALE is the AES round transformation and the AES-128 key schedule. In all the following, we assume that the reader is familiar with AES.

2.1 Specification

ALE is an on-line single-pass nonce-based authenticated encryption algorithm with associated data. Its encryption/authentication procedure accepts a 128-bit master key κ, a message μ, associated data α and a 128-bit nonce ν. An equivalent of at most 2^{48} bits are allowed to be authenticated or both authenticated and encrypted with the same master key. The encryption/authentication procedure outputs the ciphertext γ of exactly the same bit length as the message μ and the authentication tag τ of

$\ell_\tau = 128$ bits for both the message μ and associated data α. Its decryption/verification procedure accepts key κ, ciphertext γ, associated data α, nonce ν and tag τ. It returns the decrypted message μ if tag is correct or $\perp$ otherwise.

The encryption/authentication operation is given in Figure 2 and can be described in five steps:

Padding: Message $\mu = m_1||\dots||m_{t-1}||m_t$ consists of $t-1$ full 128-bit blocks $m_1, \dots, m_{t-1}$ and m_t which is either 128 bits or shorter. μ is appended by a single 1 bit followed by the minimum number of 0 bits up to the next multiple of 128 bits if needed. The resulting padded message M is split into t or $t+1$ (if μ already contained $128t$ bits) blocks of 128 bits each, $M = m_1||\dots||m_{t-1}||m'_t$ or $M = m_1||\dots||m_t||1||0^{127}$. Associated data α is appended by a single 1 bit followed by the minimum number of 0 bits up to the next multiple of 128 bits. The padded associated data is split into r blocks of 128 bit each, $A = a_1||\dots||a_r$.

Initialization: The internal state consists of two 128-bit states: the key state (upper line in Figure 2) and the data state (lower line in Figure 2). The key state is initialized with nonce ν encrypted with AES[3] under the master key κ. The data state is initialized with nonce ν AES-encrypted using the initialized key state as key. After that, the final subkey of the last AES encryption is updated one more time using the AES round key schedule with byte round constant x^{10} in $\mathbb{F}_{2^8}$. This value is stored in the key state. Now both states are initialized.

Processing associated data: If there is no associated data, this step is skipped. Otherwise, a_1 is xored to the data state. If there is only one padded associated data block, one proceeds with processing message immediately. Otherwise, if there are at least two padded associated data blocks, A is processed block by block: The data state is encrypted with 4 rounds of AES using the key state as key. The final round subkey is updated one more time using the AES round key schedule with byte round constant x^4 in $\mathbb{F}_{2^8}$. This value is stored in the key state. The next block of A is xored to the data state.

Processing message: M is processed block by block: The data state is encrypted with 4 rounds of AES using the key state as key. 16 bytes are leaked from the data state in the 4 rounds of AES in accordance with the LEX specification.

This leak is xored to the current block of M. The final round subkey is updated one more time using the AES round key schedule with byte round constant x^4 in $\mathbb{F}_{2^8}$. This value is stored in the key state. The current block of M is xored to the data state.

For the last block of M, there are two options. 1) If it is m'_t, then data state is encrypted with 4 rounds of AES using the key state as key. The exact required number of most significant bits is taken from the leak and xored to the last (maybe incomplete) block of μ to produce the last bits of ciphertext and m'_t is xored to the data state. 2) If it is $1||0^{127}$, the data state is encrypted with 4 rounds of AES using the key state as key and $1||0^{127}$ is xored to the data state.

[3] Here and further in the paper, we imply AES-128 whenever we write AES.

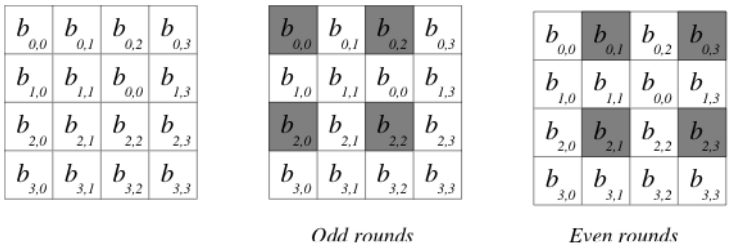


Fig. 1. The positions of the output extracted in the even and odd rounds of LEX [5]

Finalization: The data state is encrypted with the full AES using the master key κ. The output of this encryption is returned as the authentication tag τ for the message and associated data.

The decryption/verification procedure is defined correspondingly. The only two differences are that one works with the ciphertext $\gamma = c_1||\ldots||c_t$ instead of the message μ while xoring with the stream and that the supplied tag value τ is compared to the one computed by the algorithm. We want to stress that only if the tag is correct the decrypted message is returned.

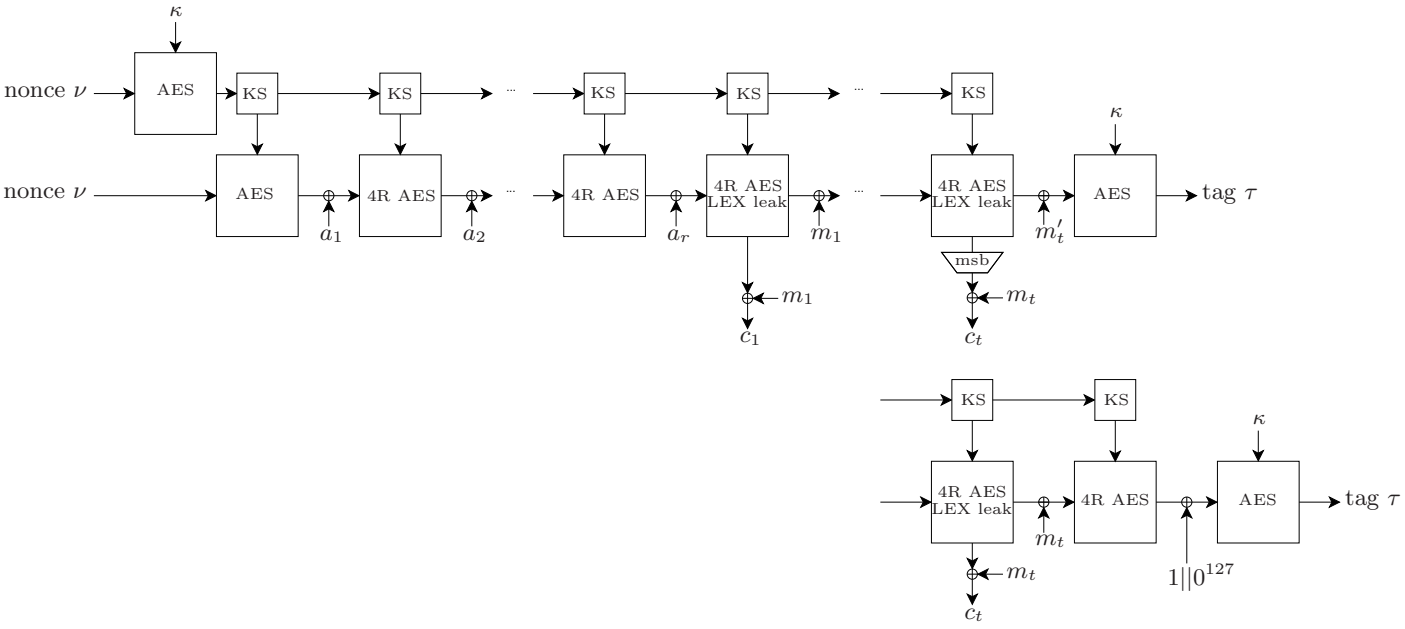


Fig. 2. Encryption and authentication procedure of ALE

2.2 Security assumptions and claims

The security analysis of the algorithm starts from the following three assumptions.

Assumption 1 (Nonce-respecting adversary) *A nonce value is only used once with the same master key for encryption.*

This assumption is quite common among nonce-based designs. Note that on most platforms, this assumption can be easily satisfied by implementing the nonce as a counter.

Assumption 2 (Abort on verification failure) *If the verification step of the algorithm reveals that the ciphertext has been tampered with, then the algorithm returns no information beyond the verification failure. In particular, no plaintext blocks are returned.*

This assumption significantly reduces the impact of chosen-ciphertext attacks, since the adversary obtains very little information from a chosen-ciphertext query. We feel that this assumption is quite natural for authenticated encryption modes. After all, when the verification fails, we know that the integrity of the plaintext has been jeopardised, and there is no reason to output it. The assumption does, however, exclude implementations where decryption is done in a streaming mode, since all plaintext blocks need to be kept inside until the verification has completed successfully.

Assumption 3 (Key lifetime limitation) *At most* 2^{48} *bits of data are authenticated or encrypted and authenticated using the same master key.*

Key lifetime limitation is a usual assumption for designs based on stream ciphers. The restriction to at most 2^{48} bits of data is essentially that of LEX.

Under these three assumptions, the security claims for the algorithm are as follows.

Claim 1 *Any key recovery with complexity equivalent to processing* N *data blocks has a success probability at most* $N2^{-128}$

Claim 2 *Any forgery attack not involving key recovery has a success probability at most* $2^{-\ell_\tau}$.

2.3 Properties

Here we list some of ALE's merits in terms of implementation. Since ALE is based on similar design principles as Pelican MAC and LEX, it also shares many strong properties of these two designs.

– Security analysis benefits from existing analysis on AES as well as Pelican MAC and LEX.

- AES hardware/software implementations might be reused with only a few simple modifications, including the usage of Intel AES instructions.
- Side-channel attack countermeasures developed for the AES will be useful for ALE as well, including threshold implementations in hardware to thwart first-order power- and EM-based differential attacks and bitsliced implementations to mitigate cache-timing leakage.
- For long messages, ALE needs only about 4 AES rounds to both encrypt and authenticate a block of message, which is similar to ASC-1. However, about 10 AES rounds are needed by AES-OCB and 20 AES rounds are required by AES-CCM to process a data block.
- The overhead of ALE per message amounts to 3 AES calls. This is less than the overhead of ASC-1 which is 4 AES calls but more than the overhead of AES-OCB of 2 AES calls. AES-CCM has virtually no overhead but has an excessive cost per block.
  However, in terms of the number of AES rounds, AES-OCB becomes less efficient already for messages longer than 128 bits and ASC-1 is always inferior to ALE. Note that AES-OCB contains a nonce stretching mechanism that effectively saves one AES call overhead if multiple messages are encrypted with the same key and with adjacent counter values as nonces. However, implementing this mechanism in lightweight hardware requires an additional 128-bit state which increases the area requirement by another 700-800 GE.
- Only two 128-bit states are needed by ALE to implement encryption, which makes it lightweight-friendly. At the same time, 4 states needed for AES-OCB and ASC-1. AES-CCM requires 3 states. In fact, 2 states needed by ALE are even less than the encryption-only AES-CTR occupies, where some space needs to be allocated for the counter.
- Only the AES encryption engine is needed by ALE for both encryption/authentication and decryption/verification procedures. AES-OCB requires both encryption and decryption engines for supporting those operations.
- ALE is an on-line scheme meaning it is single-pass and does not have to know message length before the last message block is input. AES-CCM is off-line. Additionally AES-CCM is two-pass. AES-OCB and ASC-1 are also on-line schema.
- ALE accepts associated data while ASC-1 does not. AES-CCM and AES-OCB can both work with associated data. AES-OCB is additionally capable of accepting static associated data (which does not require any recomputation for a new nonce). Note that ALE can easily incorporate the support of static associated data which we omitted in the current paper for the sake of design simplicity.

3 Security analysis

Since ALE combines some ideas of Pelican MAC [13] and LEX [5] it also benefits from existing security analysis. In the following, we briefly recall existing security analysis on these two primitives and discuss their relevance to ALE.

3.1 Internal collisions

Like any MAC derived from the ALRED construction [12] also Pelican Mac enjoys some level of provable security. It is shown that, in the absence of internal collisions, the security of the construction can be reduced to the security of the n-bit underlying block cipher [12, Theorem 1, Theorem 2]. In other words, Pelican cannot be broken with less than $2^{n/2}$ queries unless the adversary also breaks the block cipher itself. However, the security proofs of Pelican Mac rely on the fact that the iteration function is unkeyed. Therefore, they don't carry over to ALE. PC-MAC is another MAC function derived from the original design [28]. PC-MAC uses a keyed iteration function and has a proof of security in the indistinguishibility framework.

For the Pelican Mac two approaches to exploit knowledge of the (unkeyed) iteration function are described by the authors to generate internal collisions and hence forgeries.

Fixed points. Since the iteration function is known in Pelican Mac, one may compute the number of state values that are resulting in fixed points for a given message block m_i. Assume the number of fixed points is x, then the probability that inserting the message block m_i in a message will not impact its tag and hence result in a forgery is $x \cdot 2^{-n}$. However, if the iteration function can be modeled as a random permutation, then the number of fixed points has a Poisson distribution and is expected to be small. Moreover, in ALE the iteration function is keyed with a nonce dependent session key. Since this key changes for every iteration function, one needs a fixed point in both the key state and data state rendering the attack inefficient.

Extinguishing differentials. Extinguishing differentials are very similar to differential cryptanalysis for block ciphers. The main idea is to find pairs of messages (or in our case also ciphertexts) with a certain difference that may result in a zero difference in the state with a high probability after the difference has been injected. However, in the case of Pelican MAC the iteration function consists of 4 rounds of Rijndeal implementing the wide trail design strategy [14], which allows to prove good bounds against differential attacks. In more detail, any differential characteristic spanning over 4 rounds has at least 25 active S-boxes resulting in an upper bound for the differential probability of 2^{-150}. Moreover, the differential probability of any differential can be upper bounded by 2^{-114} [24]. Note that this is not far away from the theoretically optimal bound of $2 \cdot 2^{-128}$. In other words, Pelican Mac and hence also ALE provides good upper bounds for the probability of extinguishing differentials. Moreover, we want to note that in ALE the iteration function is keyed with a nonce dependent session key, which is changed for every encryption/authentication procedure, complicating the application of differential cryptanalysis to ALE, since an attacker also needs to predict the differences in the session key. However, since this session key is generated by encrypting the nonce with the master key using 10 rounds of AES, this seems to be a very difficult task.

3.2 Distinguishing attacks

The encryption component of ALE is inspired by the stream cipher LEX. The keystream bits in LEX are generated by extracting 32 bits from each round of AES in the OFB mode. In [19], Englund et al. describe a distinguishing attack that is applicable to block ciphers in the OFB mode in general. To be more precise, whenever the part of the state that depends on both the key and the IV is smaller than twice the key size (as it is the case for instance in LEX) the attack theoretically succeeds. However, in LEX the attack is thwart by limiting the number of keystream bits that can generated from one master key. In ALE we have a similar restriction but more important the internal state is larger due to the session key (depending on the nonce and the master key) used to key the iteration function.

3.3 Slide attack

A slide attack for an earlier version of LEX has been found by Wu and Preneel in [32] and fixed later by Biryukov in a new version of LEX. To avoid slide attacks two different functions for the initialization and the encryption/authentication should be used. Even a very small difference between the two is sufficient. For instance, the new version of LEX uses the full AES with the XOR of the last subkey for the initialization and AES without the XOR of this subkey for the encryption.

However, ALE uses 10 rounds of AES in the initialization, but only 4 rounds for the encryption/authentication. Moreover, the initial state as well as the key state depend on the same input, a 128-bit nonce. We think that this is sufficient to break the similarities used by slide attacks.

3.4 Exploiting (generic) internal collisions

All published attacks [8,16,33] on Pelican MAC so far are forgery attacks making use of (generic) internal collisions on the internal state. Note that due to the small state size of Pelican Mac of 128 bits, internal collisions can be found with complexity of 2^{64} due to the birthday paradox. However, in ALE a nonce dependent session key is used making it difficult to detect internal collisions on the state unless the session key collides as well, basically doubling the internal state size and giving some reinsurance in the design.

3.5 Exploiting (partial) internal collisions after different numbers of encryptions

Most attacks on LEX published so far use the fact that LEX uses the same round keys repeatedly. For instance the main idea of the key-recovery attack in [8,15] is to find a pair of internal states after different numbers of encryptions that partially collides after 4 rounds. Since the round keys are reused in LEX, the adversary can easily locate two states that collide in the part of the state which contains the round key. Hence, the

complexity of a brute-force search for (partial) internal collisions is determined by the size of the part of the state that contains the ciphertext, i.e. 128 bits.

In ALE however, the round keys are not reused. Hence, the complexity of a brute-force search for (partial) internal collisions is determined by the size of the full internal state, i.e. 256 bits. It follows that finding (partial) internal collisions becomes more difficult rendering the attack infeasible. Note that even if the attack would be applicable, it might only be used to recover the internal state, but not the master key. To recover the master key, the attacker still has to break the full AES itself.

4 Lightweight ASIC hardware implementation

This section presents a lightweight ASIC hardware implementation of ALE. Furthermore, we compare it to the existing authentication encryption schemes such as AES-OCB3, AES-CCM and ASC-1 implemented having the same design goals and using the same lightweight AES engine in their core.

4.1 Hardware architecture

ALE was implemented targeting the lowest ASIC area occupation possible. Our hardware architecture is based on the most compact AES implementation published so far [29]. The original AES implementation has a mixed data-path: it instantiate a single S-box following the proposal of Canright [11] but performs the MixColumn on all the 32 bit in parallel.

As depicted in Figure 3, the base AES design was extended to support our authenticated encryption proposal: control logic for padding, initialization and finalization, key schedule, LEX-type leak, xor for encryption/decryption. Our implementation requires to load two times both the nonce and the key. The nonce is loaded in the first two execution of AES. The key is loaded once during the first execution of the AES and once during the last execution of AES. Also, the input and output values should be maintained in the respective wires and synchronized with the operations of the accelerator on order to correctly perform the additions with the message and the additional data.

4.2 Comparison

Table 1 summarizes the implementation numbers (including area and timing) of ALE as compared to the reference designs. These results were obtained using the STMicroelectronics 65nm CMOS technology and the corresponding standard cell library characterized for LP-HVT (low power high Vt) process. The specific technology was selected because of its suitability for lightweight applications. The synthesis was performed using the tool Synopsys design compiler 2009.06. All the designs were synthesized at a frequency of 20 MHz. The cycle count does not consider the overhead for loading and offloading of the data. As indication, Table 1 reports also the power consumption of

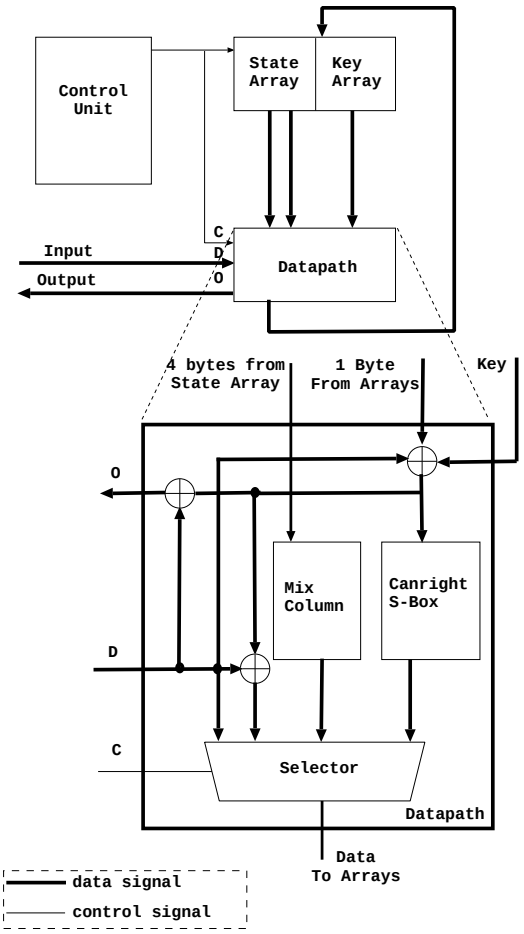


Fig. 3. The lightweight implementation of ALE

each algorithm. The estimation was carried out with Synopsys power compiler 2009.06, using the standard tool parameters for the switching activity.

ALE occupies 2,581 GE, and requires 762 clock cycles to authenticate and encrypt one block of 128 bits. This cycle count includes the overhead of 678 cycles, which is caused by the three invocation of the AES algorithm needed for initialization and finalization. Thus, only 84 clock cycles are needed to process each further 128-bit block of data.

As comparison, we report in Table 1 also the performances of the AES core used as starting point (AES-ECB) of our implementation and the ones of AES-OCB3, ASC-1, and AES-CCM authentication encryption schema. All the algorithms were implemented using the same lightweight AES engine [29] and the same experimental setup as ALE.

In its most compact implementation, ASC-1 is especially slow due to its complex non-serial key schedule which has a high overhead (for instance, one has to know the

11th key of the AES-256 expanded key to compute the first round and the 1st key again to compute the 5th round) which makes backward and forward computations necessary. This implementation is given in Table 1 as ASC-1 A. However, The key schedule overhead can be reduced if an additional 128-bit register is introduced. This implementation is given referred to as ASC-1 B in the table.

It can be observed that the overhead for the support of both encryption/authentication and decryption/verification functionalities (those implementations are marked as 'e/d' in Table 1) is fairly small for ALE and ASC-1. At the same time, since AES-OCB3 additionally requires an AES decryption engine for decryption, the overhead is much more significant there.

ALE occupies an area which is approximately two times lower than those of AES-OCB3 and ASC-1, while providing an overall at least two times higher speed, being particularly suitable for lightweight applications. ALE also nicely compares with the results reported in literature for Hummingbird-2 [18], which in its smallest version has an area of approximately 2,159 (estimated using a different technological library), and Grain-128a with authentication [2] which occupies approximately 2,770 gates (estimated by the designers).

Table 1. Lightweight ASIC implementation numbers for ALE compared to AES-OCB3, AES-CCM and ASC-1. Overhead indicates the number of cycles needed for the initial setup and the finalization of the authenticated encryption. The net per block provides the number of clock cycles required to process each block of data, on top of the overhead per message. The designs marked with 'e/d' incorporate both encryption/authentication and decryption/verification functionalities.

Design	Area (GE)	Net per 128-bit block (clock cycles)	Overhead per message (clock cycles)	Power (uW)
AES-ECB	2,435	226	-	87.84
AES-OCB3	4,563	226	452	165.21
AES-OCB3 e/d	5,783	226	452	201.32
ASC-1 A	4,793	370	904	169.11
ASC-1 A e/d	4,964	370	904	193.71
ASC-1 B	5,517	235	904	199.02
ASC-1 B e/d	5,632	235	904	207.13
AES-CCM	3,472	452	-	128.31
AES-CCM e/d	3,765	452	-	162.15
ALE	**2,581**	**84**	678	95.49
ALE e/d	**2,702**	**84**	678	103.11

5 Conclusion

In this paper, we have proposed ALE – a new Authenticated Lightweight Encryption algorithm based on AES. It is a single-pass nonce-based on-line scheme that combines some ideas of Pelican Mac, LEX and ASC-1 in a highly lightweight manner. ALE is about half the size of ASC-1 and in terms of speed in the lightweight implementation, it is about 4.5 times faster than ASC-1 in its smallest implementation.

By requiring only 2.5 kGE of area in lightweight ASIC hardware ALE is actually significantly smaller than most other authentication encryption modes including the popular modes AES-OCB and AES-CCM. In terms of speed in the lightweight implementation, ALE is about 2.5 times faster than AES-OCB and about 5 times faster than AES-CCM.

Acknowledgments. The authors thank Axel Poschmann for providing the reference implementation of the AES algorithm. Andrey Bogdanov is postdoctoral fellow of the Fund for Scientific Research - Flanders (FWO). This work has been supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State, by the European Commission under contract number ICT-2007-216676 ECRYPT NoE phase II, by KU Leuven-BOF (OT/08/027), and by the Research Council KU Leuven (GOA TENSE).

References

1. ISO/IEC 19772:2009. Information Technology - Security techniques - Authenticated Encryption, 2009.
2. Martin Ågren and Martin Hell and Thomas Johansson and Willi Meier. Grain-128a: a new version of Grain-128 with optional authentication. IJWMC 5(1), pages 48–59. 2011.
3. Jean-Philippe Aumasson, Luca Henzen, Willi Meier and María Naya-Plasencia. Quark: A Lightweight Hash. CHES 2010, LNCS, volume 6225, pp. 1–15, Springer-Verlag, 2010.
4. Steve Babbage and Matthew Dodd. The MICKEY Stream Ciphers. The eSTREAM Finalists, LNCS, volume 4986, pp. 179–190, Springer-Verlag, 2008.
5. Alex Biryukov. Design of a New Stream Cipher-LEX. The eSTREAM Finalists, LNCS, volume 4986, Springer-Verlag, 2008.
6. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. CHES 2007, LNCS, volume 4727, pp. 450–466, Springer-Verlag, 2007.
7. Andrey Bogdanov, Miroslav Knezevic, Gregor Leander, Deniz Toz, Kerem Varici and Ingrid Verbauwhede. Spongent: A Lightweight Hash Function. CHES 2011, LNCS, volume 6917, pp. 312–325, Springer-Verlag, 2011.
8. Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic Search of Attacks on round-reduced AES and Applications. Cryptology ePrint Archive, Report 2012/069, 2012.
9. Christophe De Cannière, Orr Dunkelman and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. CHES 2009, LNCS, volume 5747, pp. 272–288, Springer-Verlag, 2009.
10. Christophe De Cannière and Bart Preneel. Trivium. The eSTREAM Finalists, LNCS, volume 4986, pp. 244–266, Springer-Verlag, 2008.
11. David Canright. A Very Compact S-Box for AES. CHES 2005, LNCS, volume 3659, pp. 441–455 Springer-Verlag, 2005.

12. Joan Daemen and Vincent Rijmen. Refinements of the ALRED construction and MAC security claims. IET Inf. Secur., volume 4, issue 3, pp. 149–157, 2010.
13. Joan Daemen and Vincent Rijmen. The Pelican MAC Function. IACR Cryptology ePrint Archive, Report 2005/088, 2005.
14. Joan Daemen and Vincent Rijmen. The Wide Trail Design Strategy. IMA Int. Conf. 2001, LNCS, volume 2260, pp. 222–238, Springer-Verlag, 2001.
15. Orr Dunkelman and Nathan Keller. A New Attack on the LEX Stream Cipher. ASIACRYPT 2008, LNCS, volume 5350, pp. 539–556, Springer-Verlag, 2008.
16. Orr Dunkelman, Nathan Keller and Adi Shamir. ALRED Blues: New Attacks on AES-Based MAC's. Cryptology ePrint Archive, Report 2011/095, 2011.
17. Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication 800-38D, 66 pages, NIST, 2001.
18. Daniel W. Engels, Markku-Juhani O. Saarinen, Peter Schweitzer, and Eric M. Smith. The Hummingbird-2 Lightweight Authenticated Encryption Algorithm. RFIDSec 2011, LNCS, volume 7055, pages 19–31. Springer-Verlag, 2011.
19. Håkan Englund, Martin Hell, and Thomas Johansson. A Note on Distinguishing Attacks. IEEE Information Theory Workshop on Information Theory for Wireless Networks. pages 87–90, IEEE Press, 2007.
20. Jian Guo, Thomas Peyrin and Axel Poschmann. The PHOTON Family of Lightweight Hash Functions. CRYPTO 2011, LNCS, volume 6841, pp. 222–239, Springer-Verlag, 2011.
21. Martin Hell, Thomas Johansson, Alexander Maximov and Willi Meier. The Grain Family of Stream Ciphers. The eSTREAM Finalists, LNCS, volume 4986, pp. 244–266, Springer-Verlag, 2008.
22. Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jaesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim and Seongtaek Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. CHES 2006, LNCS, volume 4249, pp. 46–59, Springer-Verlag, 2006.
23. Chae Hoon Lim and Tymur Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. WISA 2005, LNCS, volume 3786, pp. 243–258, Springer-Verlag, 2005.
24. Seokhie Hong, Sangjin, Jongin Lim, Jaechul Sung, Dong Hyeon Cheon, and Inho Cho. Provable Security against Differential and Linear Cryptanalysis for the SPN Structure. FSE 2000, LNCS, volume 1978, pp. 273–283, Springer-Verlag, 2000.
25. Goce Jakimoski and Samant Khajuria. ASC-1: An Authenticated Encryption Stream Cipher. SAC 2011, LNCS, volume 7118, Springer-Verlag, 2012.
26. Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. FSE 2011, LNCS, volume 6733, pp. 306–327, Springer-Verlag, 2011.
27. Gregor Leander, Christof Paar, Axel Poschmann and Kai Schramm. New Lightweight DES Variants. FSE 2007, LNCS, volume 4593, pp. 196–210, Springer-Verlag, 2007.
28. Kazuhiko Minematsu, Yukiyasu Tsunoo. Provably Secure MACs from Differentially-Uniform Permutations and AES-Based Implementations. FSE 2006, LNCS, volume 4047, pp. 226–241, Springer-Verlag, 2006.
29. Amir Moradi, Axel Poschmann, San Ling, Christof Paar and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. EUROCRYPT 2011, LNCS, volume 6632, pp. 69–88, Springer-Verlag, 2011.
30. Phillip Rogaway, Mihir Bellare, John Black and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. CCS 2001, pp. 196–205, ACM, 2001.
31. François-Xavier Standaert, Gilles Piret, Neil Gershenfeld and Jean-Jacques Quisquater. SEA: A Scalable Encryption Algorithm for Small Embedded Applications. CARDIS 2006, LNCS, volume 3928, pp. 222–236, Springer-Verlag, 2006.
32. Hongjun Wu and Bart Preneel. Resynchronization Attacks on WG and LEX. FSE 2006, LNCS, volume 4047, pp. 422–432, Springer-Verlag, 2006.
33. Zheng Yuan, Wei Wang, Keting Jia, Guangwu Xu, and Xiaoyun Wang. New Birthday Attacks on Some MACs Based on Block Ciphers. CRYPTO 2009, LNCS, volume 5677, pages 209–230. Springer-Verlag, 2009.

Heavy Quark for secure AEAD

Jean-Philippe Aumasson[1], Simon Knellwolf[2], and Willi Meier[2]

[1] NAGRA, Switzerland
[2] FHNW, Switzerland

Abstract. Lightweight primitives are generally limited to 80- or 128-bit security, because lightweight applications seldom need more than this. However, non-lightweight platforms like multimedia systems-on-chip would also greatly benefit from a smaller hardware footprint, as it reduces development and integration costs, and leaves more circuit area to another component, or to add another functionality. Such systems sometimes need up to 256-bit security, for example to ensure a consistent security level across primitives. This paper thus breaks with the tradition and proposes a 256-bit authenticated encryption scheme with associated data (AEAD), based on the lightweight design Quark. We create a new Quark instance to use in a custom SpongeWrap mode, offering one-pass AEAD supporting arbitrary interleaving of encrypted and associated data, as well as a range of trade-offs between security and usage limit. More than a new primitive, this work provides insights on the scalability of lightweight designs to higher security levels: our new design c-Quark has internal state of 384 bits, and allows the implementation of 256-bit AEAD with in the order of 4000 GE.

1 Introduction

A recent research trend is the design of lightweight hash functions, with proposals Quark [1], photon [2], and spongent [3], all three being sponge functions. Those designs were preceded by reduced versions of Keccak [4] as well as by hash functions based on the block cipher present [5]. The above sponge functions achieve a competitive ratio between security and hardware footprint, due to a second preimage resistance of $n/2$ rather than n bits. In addition to this resource-efficiency, those designs can easily be used to construct other primitives than hash functions like stream ciphers, MACs, key derivation functions, or authenticated encryption schemes with associated data (AEADs).

This paper presents c-Quark, a new Quark instance with state size $b = 384$ bits, and a dedicated AEAD mode based on the SpongeWrap construction [6] of Bertoni et al. Compared to the original general definition of SpongeWrap, our scheme explicitly defines nonce management as well as a padding rule and a usage limit. This instance of SpongeWrap is called c-QuarkWrap. Security of at least 253 bits is ensured against adversaries limited to 2^{64} queries with a given key. Lower bounding to 253 rather than 256 bits simplifies a lot the construction.

We wrote a VHDL description of the new instance c-Quark, and simulated its performance in 90 nm ASIC. We estimate that the compact architecture fits in approximately 4000 gate-equivalents. Both our serial and parallel architectures show combinations of security, area, and speed that are competitive with those of previous designs.

Similar AEAD schemes can be easily constructed for (reduced) Keccak, photon, or spongent. A thorough comparison of the security and efficiency of all those schemes remains to be done.

2 Specification of c-Quark

c-Quark is a new instance of the Quark family, with parameters $r = 64, c = 320, b = 384, n = 384$. Here r is the *rate* (or block size), c is the *capacity* (or security), b is the width (or state size), and n is the digest's size. The description below is succinct and we refer to [1] for more details.

Like previous Quark instances, c-Quark processes a message m in three steps:

Version date: 2012.06.20

(1) **Initialization**: m is padded by appending a '1' bit followed by the minimal (possibly zero) number of '0' bits to reach a length that is a multiple of r.
(2) **Absorbing phase**: the r-bit message blocks are XOR'd with the last r bits of the state (that is $s_{b-r}, \ldots, s_{b-2}, s_{b-1}$), interleaved with applications of the permutation P.
(3) **Squeezing phase**: the last r bits of the state are returned as output, interleaved with applications of P until n bits are returned.

Unlike previous Quark instances, c-Quark's permutation P makes $2b$ rounds rather than $4b$ (see §§4.2). Note that we denote the internal sponge state $s = (s_0, \ldots, s_{b-1})$, where s_0 is the *first* bit of the state. The IV of c-Quark is given in Appendix A.

The P permutation of c-Quark is composed of three feedback shift registers (see Fig. 1):

- An NFSR X of 192 bits, denoted $X^t = (X_0^t, \ldots, X_{191}^t)$, at epoch $t \geq 0$.
- An NFSR Y of 192 bits, denoted $Y^t = (Y_0^t, \ldots, Y_{191}^t)$.
- An LFSR L of 16 bits, denoted $L^t = (L_0^t, \ldots, L_{15}^t)$.

Given a b-bit input and the internal state s of the sponge function, P is initialized as follows:

- $(X_0^0, \ldots, X_{191}^0) := (s_0, \ldots, s_{191})$.
- $(Y_0^0, \ldots, Y_{191}^0) := (s_{192}, \ldots, s_{383})$.
- $(L_0^0, \ldots, L_{15}^0) := (1, \ldots, 1)$.

Then the state update works as follows to compute $(X^{t+1}, Y^{t+1}, L^{t+1})$ from (X^t, Y^t, L^t):

(1) The function h is evaluated upon input bits from X^t, Y^t, and L^t, and the result is denoted h^t: $h^t := h(X^t, Y^t, L^t)$.
(2) X is clocked: $(X_0^{t+1}, \ldots, X_{191}^{t+1}) := (X_1^t, \ldots, X_{191}^t, Y_0^t + f(X^t) + h^t)$.
(3) Y is clocked: $(Y_0^{t+1}, \ldots, Y_{191}^{t+1}) := (Y_1^t, \ldots, Y_{191}^t, g(Y^t) + h^t)$.
(4) L is clocked: $(L_0^{t+1}, \ldots, L_{15}^{t+1}) := (L_1^t, \ldots, L_{15}^t, p(L^t))$.

In c-Quark, the LFSR is of 16 bits (against 10 in previous instances), to facilitate the $\times 32$ parallel implementation. The feedback polynomial is thus adapted, so that $p(L^t)$ returns $L_0^t + L_2^t + L_3^t + L_5^t$ rather than $L_0^t + L_3^t$. The functions f, g, and h are given in Appendix B.

Once initialized, the state of Quark is updated $2b$ times. The output is defined as the final value of the NFSRs X and Y, using the same bit ordering as for the initialization. That is, the new internal state of the sponge construction is set to

$$s = (s_0, \ldots, s_{b-1}) = (X_0^{2b}, X_1^{2b}, \ldots, Y_{b/2-2}^{2b}, Y_{b/2-1}^{2b}) \ .$$

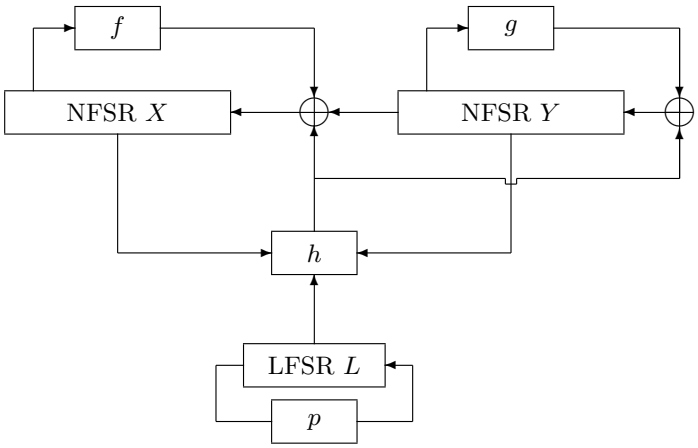


Fig. 1. Diagram of the permutation of Quark.

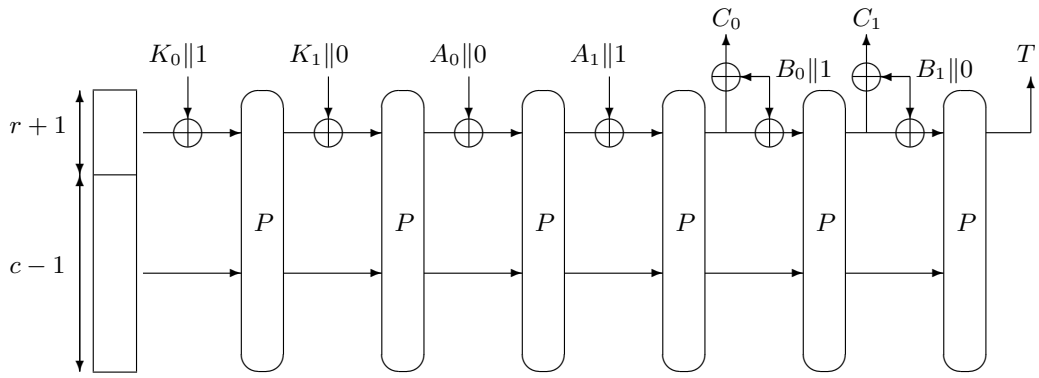


Fig. 2. Generic SpongeWrap mode a $2r$-bit (padded) key K, a $2r$-bit (padded) header A, and a $2r$-bit (padded) body B, returning an r-bit tag T.

3 Authenticated encryption with c-Quark

We construct an authenticated encryption scheme with associated data (AEAD) by using c-Quark in SpongeWrap mode [6, §5], an AEAD construction based on the duplex construction. Below we briefly describe the original SpongeWrap and then how to refine it to specify a concrete AEAD based on c-Quark; for convenience of reference we name this construction c-QuarkWrap.

3.1 The original SpongeWrap mode

The SpongeWrap mode is essentially the sponge construction wherein

(1) The key K is absorbed,
(2) Data to be absorbed is given as pairs (A, B), where the *header* A is returned unencrypted, and the *body* B is encrypted by xoring each r-bit block with the r-bit block squeezed from the current internal state.
(3) A tag T is returned as the r-bit blocks squeezed after the last B block processed (empty blocks are processed if more than r bits are needed).

Moreover, each block absorbed is flagged by a *frame bit* (see [6, §§5.1]), set to '1' for

- All key blocks except the last,
- The last header block,
- All body blocks except the last,

and it is set to '0' for all other blocks. This frame bit is absorbed by extending the original rate r of one bit.

The SpongeWrap construction supports an arbitrary number of (A, B) "wraps", and thus can interleave associated and encrypted data, and can return intermediate authentication tags. We refer to Algorithm 3 in [6] for a complete, formal, description of SpongeWrap.

Fig. 2 shows an example with a $2r$-bit (padded) key K, a $3r$-bit (padded) header A, and a $2r$-bit (padded) body B, returning an r-bit tag T.

3.2 The c-QuarkWrap AEAD

We propose to use the SpongeWrap mode with c-Quark to construct an AEAD supporting interleaved encrypted and (unencrypted) associated data. As SpongeWrap is very generic, specific parameters have to be specified. The version of SpongeWrap with our particular parameters is called c-QuarkWrap.

Padding. The SpongeWrap mode supports any sponge-compliant padding rule [6, Def.1]. c-QuarkWrap uses the same padding as for c-Quark in sponge mode, namely, to append a '1' bit followed by zero to $r-1$ zeroes. This is the simplest sponge-compliant padding rule. Note that each block processed has to be padded, as per the definition of the duplex construction (see Algorithm 2 in [6])

Key. SpongeWrap supports any key length. c-QuarkWrap keys are of 256 bits, and are thus absorbed as four 64-bit blocks, each followed by the frame bit and a padding bit.

Usage exponent. [7, §§5.2] defines the *usage exponent* as the value a such that the implementation imposes an upper limit of 2^a uses of given key. c-QuarkWrap sets this limit to 2^{64}. As c-QuarkWrap has an actual capacity of $384-61-1-1=318$ (due to the duplex padding and to the frame bit) this guarantees a security of at least $2^{318-1}/2^{64}=2^{253}$ (see [7] for a proof).

Nonces. SpongeWrap does not specify how nonces should be handled (if at all). c-QuarkWrap takes a 64-bit nonce, and each value should be used at most once within a same c-QuarkWrap object. As noted in [6, §§2.2], a nonce repetition implies that an adversary can learn the xor of two plaintext bodies, but does not affect the authentication functionality. As there are more distinct nonces than uses allowed by the usage exponent, outpassing the latter does not imply a nonce repetition.

After the SpongeWrap object is initialized with the key, the 64-bit nonce is copied to A_0, that is, the first bits of the first header processed. A nonce should be specified for each wrap call. Of course if the nonce is identical for consecutive wrap calls, only one copy of it can be transmitted along.

Tag length. SpongeWrap supports the generation of tags of arbitrary length. c-QuarkWrap returns 64-bit tags, which is sufficient for most applications. Longer tags can be produced by following the SpongeWrap specification.

Initial state. SpongeWrap relies on the duplex mode, which sets the initial state to the all-zero string. Instead, c-QuarkWrap uses a specific initial state (given in Appendix A), because an all-zero state may facilitate differential attacks on reduced-round versions of P.

3.3 Properties of c-QuarkWrap

As a particular version of SpongeWrap, c-QuarkWrap inherits its functional properties, which include

- Single-pass processing of associated and encrypted data.
- Support for arbitrarily interleaved associated and encrypted data.
- Generation of intermediate tags after each "wrap".
- Non-expanding encryption.
- Possibility to reuse the permutation to implement a hash function (or other cryptographic primitives).

Encrypting $8m$ bytes without any associated data requires $4+1+m$ calls to P, or $1+m$ if the secret internal state has been precomputed. Adding $8m'$ bytes of associated data adds m' calls to P.

Algorithm 1 c-QuarkWrap

function INIT(K)
  $s = IV$
  $s = P(s \oplus K_0 \| 11)$
  $s = P(s \oplus K_1 \| 01)$
  $s = P(s \oplus K_2 \| 01)$
  $s = P(s \oplus K_3 \| 01)$
end function

function AE(N, B) ▷ authenticated encryption of $B = B_0 \| \cdots \| B_w$ with nonce N
  $s = P(s \oplus N \| 11))$
  $C_0 = B_0 \oplus (s_{320 \ldots 383})$
  for $i = 0 \rightarrow w - 1$ **do**
    $s = P(s \oplus B_i \| 11)$
    $C_{i+1} = B_{i+1} \oplus (s_{320 \ldots 383})$
  end for
  $s = P(s \oplus B_w \| 01)$
  $T = s_{320 \ldots 383}$
  return $(N, C_0 \| \cdots \| C_w, T)$
end function

function AEAD(N, A, B) ▷ authenticated encryption of B with associated data $A_0 \| \cdots \| A_v$ and nonce N
  $s = P(s \oplus N \| 01)$
  for $i = 0 \rightarrow v - 1$ **do**
    $s = P(s \oplus A_i \| 01)$
  end for
  $s = P(s \oplus A_i \| 11)$
  $C_0 = B_0 \oplus (s_{320 \ldots 383})$
  for $i = 0 \rightarrow w - 1$ **do**
    $s = P(s \oplus B_i \| 11)$
    $C_{i+1} = B_{i+1} \oplus (s_{320 \ldots 383})$
  end for
  $s = P(s \oplus B_w \| 01)$
  $T = s_{320 \ldots 383}$
  return $(N, A, C_0 \| \cdots \| C_w, T)$
end function

4 Security

4.1 Security with ideal permutation

As previously observed, c-Quark is actually used in c-QuarkWrap with a capacity of 318 bits, due to the SpongeWrap frame bit and the duplex padding bit adding up to the 64 bits of effective data xored to the state. As per the analysis reported in [6, Th.1], this guarantees a security (with respect to confidentiality and authenticity) of approximately 2^{159} in the general case. However, by defining an upper bound of 2^{64} on the number of usages of a given key, a bound of 2^{253} on the complexity of a generic attack succeeding with high probability is obtained.

4.2 Security of the permutation

Reduced-round P permutations of previous Quark instances were attacked [1] with cube testers, truncated differential attacks, and conditional differentials attacks, with the latter method outperforming the first two. While the three original instances of Quark have $4b$ rounds, fewer than b rounds could be attacked. This security margin, along with the results of the analysis below, is the reason why we reduced the number of rounds to $2b$ for c-Quark.

We first applied the same kind of truncated differential analysis to c-Quark than to the original Quark's: 348 rounds could be distinguished from an ideal permutation with complexity 2^{20}, by exploiting a biased difference in s_0 given an input difference in s_3.

We then applied the more advanced technique of conditional differential cryptanalysis, with two specific approaches:

(1) Control the propagation of the single-input difference in s_3 as far as possible.
(2) Find an input difference (of arbitrary weight) that leads to a single-bit difference in the state bit s_3 after q rounds. After q rounds, the propagation is not controlled anymore.

The details and results of both approaches are described in the following.

Conditional Differential Cryptanalysis (1). The following conditions prevent the propagation of the difference whenever possible in the first 50 rounds (5 conditions in total):

In round 36:

$$s_{49} = 0,\ s_1+s_9+s_{14}s_{48}s_{66}s_{78}s_{95}+s_{14}s_{48}+s_{14}s_{78}s_{128}s_{160}+s_{14}+s_{18}s_{52}s_{70}s_{82}s_{99}+s_{18}s_{52}+s_{18}s_{82}s_{132}s_{164}+s_{18}+s_{26}s_{47}s_{196}+s_{26}s_{47}s_{252}+s_{29}+s_{30}s_{51}s_{200}+s_{30}s_{51}s_{256}+s_{33}+s_{35}+s_{39}+s_{41}+s_{45}+s_{47}s_{56}s_{196}+s_{47}s_{56}+s_{47}s_{196}s_{252}+s_{48}s_{66}s_{158}s_{160}+s_{51}s_{60}s_{200}+s_{51}s_{60}+s_{51}s_{200}s_{256}+s_{52}s_{70}s_{162}s_{164}+s_{56}s_{196}+s_{56}s_{252}+s_{60}s_{200}+s_{60}s_{256}+s_{66}s_{78}s_{95}s_{110}s_{128}s_{146}+s_{66}s_{78}s_{95}+s_{66}+s_{70}s_{82}s_{99}s_{114}s_{132}s_{150}+s_{70}s_{82}s_{99}+s_{70}+s_{78}+s_{82}+s_{86}+s_{90}+s_{95}s_{110}s_{146}s_{158}+s_{95}s_{110}+s_{95}+s_{99}s_{114}s_{150}s_{162}+s_{99}s_{114}+s_{99}+s_{110}s_{128}s_{146}s_{158}s_{160}+s_{110}+s_{113}+s_{114}s_{132}s_{150}s_{162}s_{164}+s_{114}+s_{117}+s_{128}s_{146}s_{158}+s_{128}+s_{132}s_{150}s_{162}+s_{132}+s_{141}+s_{142}+s_{145}+s_{147}+s_{150}+s_{151}+s_{153}+s_{157}+s_{158}s_{160}+s_{158}+s_{162}s_{164}+s_{162}+s_{193}+s_{195}+s_{197}+s_{199}+s_{226}+s_{230}+s_{252}+s_{253}+s_{255}+s_{256}+s_{257}+s_{259}+s_{280}+s_{284}+s_{292}+s_{296}+s_{331}+s_{335}+s_{341}+s_{345}=0$$

In round 38:

$$s_{183}+1=0,\ s_{132}s_{147}+s_{165}+1=0,\ s_5+s_9+s_{18}s_{52}s_{70}s_{82}s_{99}+s_{18}s_{52}+s_{18}s_{82}s_{132}s_{164}+s_{18}+s_{30}s_{51}s_{200}+s_{30}s_{51}s_{256}+s_{33}+s_{39}+s_{45}+s_{51}s_{60}s_{200}+s_{51}s_{60}+s_{51}s_{200}s_{256}+s_{52}s_{70}s_{162}s_{164}+s_{60}s_{200}+s_{60}s_{256}+s_{70}s_{82}s_{99}s_{114}s_{132}s_{150}+s_{70}s_{82}s_{99}+s_{70}+s_{82}+s_{90}+s_{99}s_{114}s_{150}s_{162}+s_{99}s_{114}+s_{99}+s_{114}s_{132}s_{150}s_{162}s_{164}+s_{114}+s_{117}+s_{132}s_{150}s_{162}+s_{132}+s_{145}+s_{146}+s_{150}+s_{151}+s_{157}+s_{162}s_{164}+s_{162}+s_{197}+s_{199}+s_{230}+s_{256}+s_{257}+s_{259}+s_{284}+s_{296}+s_{335}+s_{345}+1=0$$

On a sample that satisfies these conditions we found a stronger bias on the difference in s_0 after 348 rounds (the bias can be reliably detected on samples of size 2^{12} at significance level $\alpha = 0.001$), but no additional was detected at later rounds.

Conditional Differential Cryptanalysis (2). Let $q = 30$, that is, we aim for a single bit difference in s_3 after 30 rounds. By backward computation with linearized update functions one can find an initial difference that has differences at the following bits (25 in total): s_1, s_3, s_4, s_5, s_{17}, s_{30}, s_{192}, s_{193}, s_{195}, s_{196}, s_{198}, s_{199}, s_{200}, s_{201}, s_{202}, s_{203}, s_{204}, s_{205}, s_{206}, s_{208}, s_{210}, s_{212}, s_{214}, s_{216}, and s_{218}. The following conditions make sure that this difference boils down to a single bit difference after 30 rounds (22 conditions in total):

In rounds 0 to 9:
$s_{25}s_{46}+s_{46}s_{55}+s_{46}s_{251}+s_{55} = 0$ $s_{229}s_{253}s_{287}s_{305}+s_{229}+s_{287}s_{326}s_{352} = 0$ $s_{26}s_{47}+s_{47}s_{56}+s_{47}s_{252}+s_{56} = 0$ $s_{231}s_{255}s_{289}s_{307}+s_{231}+s_{289}s_{328}s_{354} = 0$ $s_{28}s_{49}+s_{49}s_{58}+s_{49}s_{254}+s_{58} = 0$ $s_{51}s_{69}s_{81}s_{98}+s_{51}+s_{81}s_{131}s_{163} = 0$ $s_{29}s_{50}+s_{50}s_{59}+s_{50}s_{255}+s_{59} = 0$ $s_{233}s_{257}s_{291}s_{309}+s_{233}+s_{291}s_{330}s_{356} = 0$ $s_{30}s_{51}+s_{51}s_{60}+s_{51}s_{200}+s_{51}+s_{60} = 0$ $s_{31}s_{52}+s_{52}s_{61}+s_{52}s_{257}+s_{61} = 0$ $s_{32}s_{53}+s_{53}s_{62}+s_{53}s_{258}+s_{62} = 0$ $s_{33}s_{54}+s_{54}s_{63}+s_{54}s_{259}+s_{63} = 0$ $s_{34}s_{55} + s_{55}s_{64} + s_{55}s_{260} + s_{64} = 0$

In rounds 10 to 19:
$s_{35}s_{56} + s_{56}s_{65} + s_{56}s_{261} + s_{65} = 0$ $s_{36}s_{57} + s_{57}s_{66} + s_{57}s_{262} + s_{66} = 0$ $s_{38}s_{59} + s_{59}s_{68} + s_{59}s_{264} + s_{68} = 0$ $s_{40}s_{61} + s_{61}s_{70} + s_{61}s_{266} + s_{70} = 0$ $s_{64}s_{82}s_{94}s_{111} + s_{64} + s_{94}s_{144}s_{176} = 0$ $s_{42}s_{63} + s_{63}s_{72} + s_{63}s_{268} + s_{72} = 0$ $s_{44}s_{65} + s_{65}s_{74} + s_{65}s_{270} + s_{74} = 0$

In rounds 20 to 29:
$s_{46}s_{67} + s_{67}s_{76} + s_{67}s_{272} + s_{76} = 0$
$s_{48}s_{69} + s_{69}s_{78} + s_{69}s_{274} + s_{78} = 0$

A sample that satisfies all these conditions can be generated by setting the following bits of the initial state to zero (and assigning random values to the other bits): s_{46}, s_{47}, s_{49}, s_{50}, s_{51}, s_{52}, s_{53}, s_{54}, s_{55}, s_{56}, s_{57}, s_{58}, s_{59}, s_{60}, s_{61}, s_{62}, s_{63}, s_{64}, s_{65}, s_{66}, s_{67}, s_{68}, s_{69}, s_{70}, s_{72}, s_{74}, s_{76}, s_{78}, s_{81}, s_{94}, s_{229}, s_{231}, s_{233}, s_{287}, s_{289}, and s_{291}. Using a sample of size 2^{20}, a bias can always be detected in the difference of s_0 after $348 + 30 = 378$ rounds (at significance level $\alpha = 0.001$).

It is possible to choose a slightly larger q (and hence to further increase the number of rounds). Table 1 shows the weight of the input difference and the number of required conditions for some larger values of q. Not only grows the number of conditions quickly, but they also get very complicated, which means that large parts of the state must be fixed.

Table 1. Hamming weight of the input difference and number of conditions for different q (the bias can be detected after $348 + q$ rounds).

Controlled rounds (q)	30	32	34	36	38	40	42	44	46	48
Difference weight	25	28	32	35	37	39	36	38	40	42
Number of conditions	22	24	28	33	41	49	53	61	67	78

Conclusion. It seems very unlikely that the c-Quark permutation with $2b = 768$ rounds admits differential properties that can be exploited for an attack on the hash function or its use in the proposed AEAD mode. Using conditional differential cryptanalysis we could "attack" $348 + 48 = 396$ rounds, which leaves a reasonable security margin.

Obviously better distinguishers may exist on P, and "shortcut" attacks on c-Quark as a hash function may exist too. As a first step, cryptanalysts may consider reduced-round versions of c-Quark as well modified versions with weaker feedback functions. Although we do not claim resistance against related-key attacks, they could be considered in the security evaluation of c-Quark.

5 Hardware efficiency

5.1 Methodology

We wrote VHDL descriptions of a serial architecture of c-Quark (1-bit datapath; most compact, slowest) and of a parallel architecture (32-bit datapath; less compact, fastest). We synthesized them for a 90 nm TSMC technology using Synopsys DC Ultra (2011 version) with the tcbn90lphp standard cell library.

Compared to the hardware evaluation in [1], we report only a basic analysis with *pre-place-and-route* area evaluation for the c-Quark sponge function. Much more reliable results would be obtained from a working post-layout design. Extra logic and memory is necessary to implement the complete c-QuarkWrap mode.

5.2 Results

Below we report the main performance metrics for each of the architectures implemented, showing efficiency estimates competitive with other lightweight designs[3]:

Serial architecture. In this architecture the P permutation has a latency of $2b = 768$ cycles, and processes 64 bits. At 100 kHz, this is a throughput of $100/768 \times 64 \approx 8.33$ kbps. The synthesis of our RTL design gave a circuit of approximately 3125 gate-equivalents (GE); assuming a density of 80% after place-and-route, this gives an area of approximately 4000 GE, that is, an efficiency of approximately $8333/4000 \approx 2$ bps/GE.

Parallel architecture. In this architecture the P permutation has a latency of $2b/32 = 24$ cycles, and processes 64 bits. At 100 kHz, this is a throughput of $100/24 \times 64 \approx 266.67$ kbps. The synthesis of our RTL design gave a circuit of approximately 7100 gate-equivalents (GE); assuming a density of 80% after place-and-route, this gives an area of approximately 8875 GE, that is, an efficiency of approximately $266\,666/8875 \approx 30$ bps/GE.

References

1. Aumasson, J.P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: a lightweight hash (full version). http://131002.net/quark/quark_full.pdf appeared in CHES 2010.
2. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: CRYPTO. (2011)
3. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: SPONGENT: A lightweight hash function. In: CHES. (2011)
4. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak SHA-3 submission. Submission to NIST (Round 3), http://keccak.noekeon.org/Keccak-submission-3.pdf (2011)
5. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash functions and RFID tags: Mind the gap. In: CHES. (2008)
6. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the sponge: single-pass authenticated encryption and other applications. http://sponge.noekeon.org/SpongeDuplex.pdf appeared in SAC 2011.
7. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the security of the keyed sponge construction. http://sponge.noekeon.org/SpongeKeyed.pdf appeared in SKEW 2011.

A Initial state

The initial state of c-Quark is the SHA-384 digest of the string "c-quark", and like for previous Quark instances the first state bit s_0 (copied to X_0 in P) is defined as the most significant bit of the first byte of this value (`2b`):

```
3b4503ec7662c3cb30e00837ec8d38bbe5ff5acd6901a2495750f9198e2e3b5852dcaa1662b7dad65fcb5a8a1f0d5fcc
```

[3] For example, at a same 100 kHz frequency, the serial and parallel architectures of spongent-256/512/256 have respective efficiencies $350/5110 = 0.068$ bps/GE and $66490/9944 = 6.68$ bps/GE.

B Boolean functions

We define the Boolean functions f, g, and h used for computing the P permutation of c-Quark:

$f(X) = X_0+X_{13}+X_{34}+X_{65}+X_{77}+X_{94}+X_{109}+X_{127}+X_{145}+X_{157}+X_{140}+X_{159}X_{157}+X_{109}X_{94}+X_{47}X_{13}+X_{157}X_{145}X_{127}+X_{94}X_{77}X_{65}+X_{159}X_{127}X_{77}X_{13}+X_{157}X_{145}X_{109}X_{94}+X_{159}X_{157}X_{65}X_{47}+X_{159}X_{157}X_{145}X_{127}X_{109} + X_{94}X_{77}X_{65}X_{47}X_{13} + X_{145}X_{127}X_{109}X_{94}X_{77}X_{65}$

$g(Y) = Y_{i+21} + Y_{i+57} + Y_{i+60} + Y_{i+94} + Y_{i+112} + Y_{i+125} + Y_{i+133} + Y_{i+152} + Y_{i+157} + Y_{i+146} + Y_{i+159}Y_{i+157} + Y_{i+125}Y_{i+112} + Y_{i+36}Y_{i+21} + Y_{i+157}Y_{i+152}Y_{i+133} + Y_{i+112}Y_{i+94}Y_{i+60}+ Y_{i+159}Y_{i+133}Y_{i+94}Y_{i+21} + Y_{i+157}Y_{i+152}Y_{i+125}Y_{i+112} + Y_{i+159}Y_{i+157}Y_{i+60}Y_{i+36}+ Y_{i+159}Y_{i+157}Y_{i+152}Y_{i+133}Y_{i+125}+Y_{i+112}Y_{i+94}Y_{i+60}Y_{i+36}Y_{i+21}+Y_{i+152}Y_{i+133}Y_{i+125}Y_{i+112}Y_{i+94}Y_{i+60}$

$h(X, Y, L) = X_{i+25} + Y_{i+59} + Y_{i+3}X_{i+55} + X_{i+46}X_{i+55} + X_{i+55}Y_{i+59} + Y_{i+3}X_{i+25}X_{i+46}+ Y_{i+3}X_{i+46}X_{i+55} + Y_{i+3}X_{i+46}Y_{i+59} + X_{i+25}X_{i+46}Y_{i+59}L_i + X_{i+25}L_i + L_i + X_{i+4} + X_{i+28} + X_{i+40} + X_{i+85} + X_{i+112} + X_{i+141} + X_{i+146} + X_{i+152} + Y_{i+2} + Y_{i+33} + Y_{i+60} + Y_{i+62} + Y_{i+87} + Y_{i+99} + Y_{i+138} + Y_{i+148}$

C Test values

We give intermediate values of the c-Quark internal state when hashing the empty message, that is, after padding, the message block `80000000`:

Initial state after XOR with the message block `80000000`:

```
3b4503ec7662c3cb30e00837ec8d38bbe5ff5acd6901a2495750f9198e2e3b5852dcaa1662b7dad6dfcb5a8a1f0d5fcc
```

State after applying the only permutation of the absorbing phase:

```
b9a4d5653dff49af0e9c01c202e33ce30df6dc988a3f7df674ed10280b74152b0b7542795236945e1cb9770ee7c25fa9
```

States after each permutation of the squeezing phase:

```
9d4607ec0e3a744447d6f79343970a4986a6d7b5dcfa0b52f5ea3cbbc54ed1056eadbfe16ccfeafbdce2c9464578337c
97078af8b39dec11810d275fe1ee072aa766a82cffadf8e875df86c85802ebc68fa919f69aeb28e469c7e26cb4f1bdf4
eb09b18152c593c24e24b4313a92134ebe6e88099dfeeefc793f1165c9f1585910133da0b3fe393b4869f1a93639f1f3
57cec14c521600e91936829170737bfb66f9adf818abb10f6e44b1121a5916043a11a5706b4c987b60b888975ff9ffee
ea1477b135ff77cd78585224a2d224e9f6e48b812021bf68b02125f329d2310e731d0bee58c56b1b880d2c499108a27a
```

Digest returned:

```
1cb9770ee7c25fa9dce2c9464578337c69c7e26cb4f1bdf44869f1a93639f1f360b888975ff9ffee880d2c499108a27a
```

An Improved Hardware Implementation of the Grain-128a Stream Cipher

Shohreh Sharif Mansouri and Elena Dubrova

Department of Electronic Systems, School of ICT,
KTH - Royal Institute of Technology, Stockholm
Email:{shsm,dubrova}@kth.se

Abstract. In this paper we study efficient hardware implementations of the Grain-128a family of stream ciphers. To achieve higher throughput compared to the standard design, we apply four different techniques in combination: isolation of the authentication section, Fibonacci-to-Galois transformation of the feedback shift registers, multi-frequency implementation and internal pipelining. The combined effects of all these techniques, when a two-level pipeline is used, enable an average 52% increase in throughput among all the ciphers. All techniques are standard cell techniques and are therefore easy to apply. They introduce an average 9% area penalty and an average 5% power overhead.

1 Introduction

Recently, hardware authentication devices with very tight power budgets, such as RFID tags, have become ubiquitous. Deploying secure cryptographic algorithms in these systems is extremely important and failure to do so would block off many potential applications [1].

Feedback Shift Registers (FSR)-based stream ciphers with low hardware footprint are one of the most promising candidates for deployment in low-power authentication devices [2]. This motivates research on FSR-based stream ciphers. The eSTREAM project [3], launched in 2004, identified in 2008 a portfolio of three promising FSR-based stream ciphers for deployment in highly-constrained environments: Grain [4], Mickey [5] and Trivium [6].

Since FSR-based stream ciphers target highly-constrained environments, designing them efficiently is important. Hardware efficiency was one of the main parameters used for grading the ciphers during the eSTREAM project. However, only a straightforward implementation was performed and since recently only straightforward implementations of the ciphers were made [7]. In 2010, special techniques to improve the hardware figures-of-merit of FSR-based stream ciphers were introduced in [8] and [9].

In 2011, Agren and co-workers introduced a new version of Grain-128 which natively supports authentication, with a maximal tag length of 32 bits, called Grain-128a [10], which is described in Section 2. Five parallelized versions of Grain-128a have also been introduced. To our best knowledge, no study on the

hardware implementation of these ciphers has been conducted until now. In this work we aim on finding the best implementation of Grain-128a in terms of throughput. We first implement the original Grain-128a ciphers (Section 4) using a straightforward design flow. Then, we apply four different techniques to improve their throughput: in Section 5 we isolate the authentication section of the ciphers using flip-flops, in Section 6 we transform the Fibonacci FSRs of the ciphers to Galois FSRs, in Section 7 we introduce dual-frequency implementations of the ciphers and in Section 8 we pipeline their pre-outputs functions.

The final results, reported in Section 9, show that among the six versions of the cipher, using two levels of pipeline, we obtain an average 52% throughput improvement at the expense of an average 9% area overhead and a 5% power overhead. The techniques we apply are standard cell techniques and are therefore compatible with the standard ASIC design flow and easy to apply.

2 The Grain-128a Cipher

The Grain-80 stream cipher (also known as Grain-v1) was developed by Hell, Johansson and Meier [4] and proposed at the eSTREAM project [3]. Grain-80 supports an 80 bits key and an 80 bits public Initial Value (IV). Recognizing the growing need for 128 bits keys, the Grain-128 stream cipher, built on the same concepts as Grain-80 but supporting a 128 bits key and a 96 bits public IV, was developed [11] in 2006.

In 2011, the Grain-128a stream cipher was proposed [10]. It is an extension of the Grain-128 stream cipher, the main difference with Grain-128 being the fact that it natively supports authentication, with a variable tag size up to 32 bits. Also, the non-linear functions of Grain-128a are slightly different compared to those of Grain-128, that have been strengthened against known attacks.

A complete schematic of the grain-128a cipher is shown in Figure 1. The cipher is divided into two parts: the *keystream generator*, which generates a pre-output stream, and the *authentication section*.

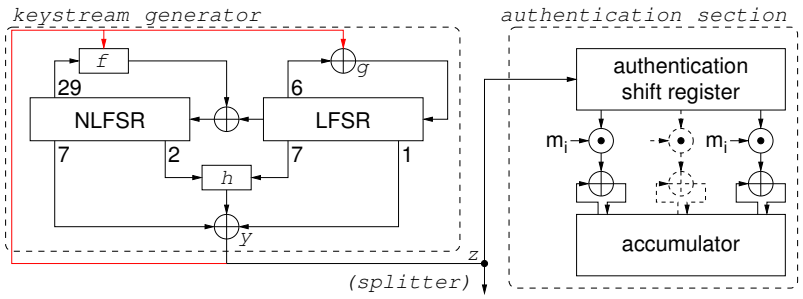


Fig. 1. The Grain-128a cipher.

2.1 Keystream Generator

The keystream generator contains a 128-bit Linear FSR (LFSR) and a 128-bits Non-Linear FSR (NLFSR). The contents of the 128-bits LFSR are denoted as $s_0, s_1, ..., s_{127}$; the contents of the 128-bits NLFSR are denoted as $b_0, b_1, ..., b_{127}$. All memory elements of the LFSR and the NLFSR are updated simultaneously. s_i is updated to s_{i+1} for $0 \leq i \leq 126$; s_{127} is updated to $f(s)$, where $f(s)$ is:

$$f(s) = s_0 + s_7 + s_{38} + s_{70} + s_{81} + s_{96}$$

b_i is updated to b_{i+1} for $0 \leq i \leq 126$; b_{127} is updated to $g(b, s_0)$, where $g(b, s_0)$ is:

$$\begin{aligned} g(b, s_0) = \; & s_0 + b_0 + b_{26} + b_{56} + b_{91} + b_{96} + b_3 b_{67} + b_{11} b_{13} + b_{17} b_{18} + b_{27} b_{59} \\ & + b_{40} b_{48} + b_{61} b_{65} + b_{68} b_{84} + b_{88} b_{92} b_{93} b_{95} + b_{22} b_{24} b_{25} + b_{70} b_{78} b_{82} \end{aligned}$$

The function $h(b, s)$ is:

$$h(b, s) = b_{12} s_{1+8} + s_{13} s_{20} + b_{95} s_{42} + s_{60} s_{79} + b_{12} b_{95} s_{94}$$

The pre-output function $y(b, s)$ is:

$$y(b, s) = h(b, s) + s_{93} + b_2 + b_{15} + b_{36} + b_{45} + b_{64} + b_{73} + b_{89}$$

The sequence of pre-output bits output by the y function are denoted as $y_0, ..., y_i$. The output function $z(b, s) = y_{64+2i}$ outputs all pre-output bits of even index except the first 64. The first 64 pre-output bits and all bits of odd index are used for authentication.

2.2 Authentication Section

Two authentication registers, the *accumulator* and the *authentication shift register*, both of size 32, are used. The content of the accumulator is denoted as $a_0, ..., a_{31}$. The content of the authentication shift register is denoted as $r_0, ..., r_{31}$. During an initial 64-cycles authentication initialization phase, the first 32 pre-output elements $y_0, ..., y_{31}$ of y_i are stored in the authentication shift register ($r_i = y_i$) while the following 32 elements $y_{32}, ..., y_{63}$ of y_i are stored in the accumulator ($a_i = y_{32+i}$).

In every cycle i, r_{31} is updated to the new pre-output bit $y_{64+2i+1}$ while all the other 31 elements r_j are updated to r_{j+1}. All bits a_j in the accumulator are updated to $a_j + m_i r_j$, where m_i is the bit of the message $m = m_0, ..., m_{L-1}$ that is being encrypted in cycle i. The final content of the accumulator once encryption is concluded is denoted as the tag t and can be used for authentication ($t_i = a_i$).

If the tag size is $w < 32$, only the part of the tag t with the w highest indexes is used as a tag, and the other parts are discarded.

2.3 Cipher Phases

Summarizing, after having been loaded with the key and the initial value, the cipher goes through the following phases: (1) *keystream initialization phase*, 256 clock cycles in which the cipher does not produce any output bit and the output of the y function is fed back to the LFSR and the NLFSR (red lines in Figure 1); (2) *authentication initialization phase*, 64 clock cycles in which all the pre-output bits are stored in the accumulator and the authentication shift register; (3) *operational phase* in which half the pre-output bits are output as keystream and half are fed to the authentication section of the cipher. The *keystream generation phase* includes both phases (2) and (3). The phases of the cipher are summarized in Figure 2.

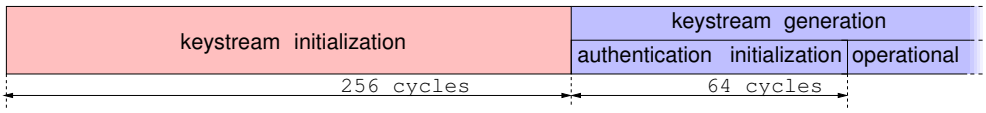


Fig. 2. Cipher phases.

2.4 Parallelized Versions

It is possible to change the NLFSR and the LFSR so that they shift their content by k positions per clock cycle, and to implement k times the functions f, g, h and y. The resulting ciphers generate keystream bits k times faster compared to the base version of the cipher. The degree of parallelism k can only be 2, 4, 8, 16 or 32. We refer to a version of Grain-128a parallelized k times as Grain-128aXk.

Grain-128aXk produces $\frac{k}{2}$ bits of keystream per cycle and delivers $\frac{k}{2}$ pre-output bits per cycle to the authentication section of the cipher. The authentication section of the cipher must be parallelized to support the incoming stream of pre-output bits (in the non-parallelized Grain-128a, $k = 1$ and the authentication section receives only one pre-output bit every two cycles).

3 Implementation and Analysis Methodology

All timing, area and power figures reported in this paper are obtained by designing the ciphers at Register Transfer Level (RTL) in Verilog, and then synthesizing the code for best performances using Cadence RTL Compiler for the TSMC 90 nm ASIC technology. All ciphers use parallel initialization, which is less sensitive to physical attacks, i.e. all key and IV bits are loaded at the same time in the LFSR and the NLFSR. To determine the phases of the cipher, the cipher uses an LFSR counter [12], the smallest and fastest type of counter.

We specify all non-standard commands we issue during synthesis. Timing and area figures are obtained from the synthesis tool; power figures are obtained

Table 1. Timing in the original versions of Grain-128a.

	X1	X2	X4	X8	X16	X32
Min. clock period (ps)	472	493	499	521	588	665
Critical path	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}	D_{hya}	D_{hya}

using the following procedure: the post-synthesis gate-level netlist is exported by the synthesis tool; a gate-level simulation is performed using the Cadence Incisive logic simulator with a set of random test vectors and a clock frequency of $10MHz$; the switching activity of all nets in the system is saved to a VCD file and read back by Cadence RTL Compiler, which then estimates the power consumption of the system.

4 Straightforward Implementation

We first implement Grain-128aXk using a standard design flow and optimizing the system for the highest throughput.

The cipher throughput is determined by its critical path which is the longest combinational propagation delay in the system. To improve the throughput of the different versions of the Grain-128a cipher, we study the location of the critical paths in the synthesized ciphers. We define the following delays:

- D_n: maximal propagation delay from any NLFSR flip-flop to any other NLFSR flip-flop. D_l is the LFSR counterpart.
- D_{hy}: maximal propagation delay from any NLFSR or LFSR flip-flop through the h and y functions to the output of the cipher.
- D_{hya}: maximal propagation delay from any NLFSR or LFSR flip-flop through the h and y functions to any accumulator flip-flop.
- D_a: maximal propagation delay from any flip-flop in the authentication section of the cipher to any accumulator flip-flop.

Two additional delays, active only during the keystream initialization phase, are defined:

- D_{hyn}: maximal propagation delay from a flip-flop of the NLFSR or LFSR through the h and y functions to the first flip-flop of the NLFSR. D_{hyl} is the LFSR counterpart.

Table 1 reports the minimal clock period and the critical paths for all the versions of the cipher. The first observation is that Grain-128aX16 and Grain-128aX32 can benefit from breaking the h-y-accumulator path. This is discussed in the next section.

5 Isolating the Authentication Section

In a parallelized cipher Grain-128aXk with $k \geq 4$, the value of a_j must be updated to

$$a_j + \sum_{u=0}^{u<\frac{k}{2}} m_{i+u} \cdot r_{j+u}$$

in every cycle i. The accumulator logic must perform more multiply and accumulate operations for higher k; therefore D_a increases with the degree of parallelism. The implementation is straightforward for $j \leq 31 - \frac{k}{2}$. However, for $j > 31 - \frac{k}{2}$, the accumulator logic would need to access values of r_j with $j > 31$. These values can be seen as "future values" of the r bits. Since r shifts its elements by $\frac{k}{2}$ positions every clock cycle and loads $\frac{k}{2}$ new elements from $\frac{k}{2}$ outputs of the k parallel h/y functions, $\frac{k}{2}$ future values of the r elements can always be found on the output lines of the h/y functions, and can be accessed by the accumulator logic to implement the authentication functionality.

For high degrees of parallelism, this straightforward solution, used in Section 4, involves a long combinational path D_{hya} through the h/y functions and the accumulator logic that limits the performances of Grain-128aX16 and Grain-128aX32.

To increase the performances of these ciphers and eliminate the D_{hya} path, flip-flops are inserted in the authentication section of the cipher on the outputs of the h/y functions, as shown in Figure 3. This solution adds one cycle latency in the production of the tag, but has no effect on cipher security.

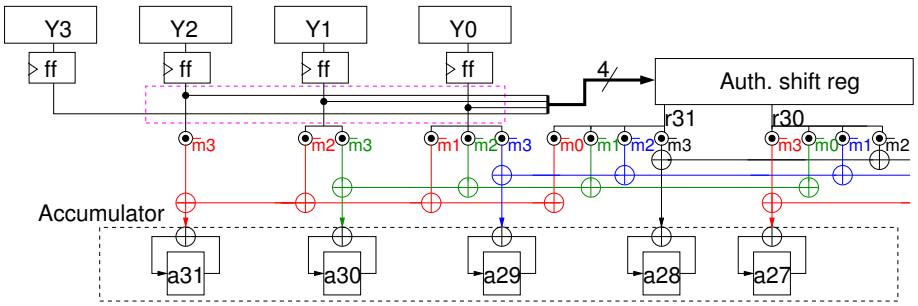


Fig. 3. Isolation of the authentication section for Grain-128aX8.

After the application of this solution to all versions of Grain-128a, the minimal clock periods and critical paths of the ciphers improve, as reported in Table 2. Timing improves for all versions of the cipher.

The critical paths of all versions of the cipher are now given by D_{hyn}. Since h, y and g are optimized together by the synthesis tool, D_{hyn} depends on both the h/y functions (19 literals in total) and on the $g(b, s_0)$ feedback function of the NFLSR (30 literals).

To improve D_{hyn}, the best option seems to be to reduce the maximal propagation delay of the paths going from the NLFSR bits to the first bit of the

Table 2. Timing in the versions of Grain-128a after the isolation of the authentication section.

	X1	X2	X4	X8	X16	X32
Min. clock period (ps)	440	490	483	517	545	580
Critical path	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}

NLFSR. This is done by applying to the cipher FSRs the Fibonacci to Galois transformation suggested in [13], that is the object of the next section.

6 Fibonacci to Galois Transformation

A Feedback Shift Register (FSR) consists of n binary storage elements, called *bits* [14]. Each bit i has an associated state variable x_i which represents the current value of bit i and a *feedback function* $f_i(x_0, ..., x_{n-1})$ which determines how the value of i is updated. All updates take place simultaneously.

The update functions (f_i) are expressed in the Algebraic Normal Form, with "$\oplus$" indicating XOR operations and "$\cdot$" indicating AND operations. For Linear Feedback Shift Registers (LFSRs), all functions f_i take the form:

$$f(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{n-1} c_i \cdot x_i$$

For Non Linear Feedback Shift Registers (NLFSRs):

$$f(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{2^n-1} c_i \cdot x_0^{i_0} \cdot x_1^{i_1} \cdot \ldots \cdot x_{n-1}^{i_{n-1}}$$

where $c_i \in \{0, 1\}$ and $(i_0 i_1 \ldots i_{n-1})$ is the binary expansion of i with i_0 being the least significant bit. For NLFSRs, we call all terms of the ANF *product terms*.

Thr FSRs can be implemented in two configurations, *Fibonacci* or *Galois*. An FSR is in Fibonacci configuration if all feedback functions f_i except f_{n-1} take the form $f_i = x_{i+1}$. If some functions f_i with $i \neq n-1$ are not in this form, then the FSR is in *Galois* configuration.

For LFSRs, this definition is more general than the traditional definition of Galois LFSRs, which corresponds to that of fully-shifted Galois LFSRs [13]. However, to keep the presentation simple, in this paper we use this definition for both NLFSRs and LFSRs.

As discussed in [13] a Fibonacci n-bit FSR can be transformed into an equivalent Galois FSR having the same output stream (i.e. the values of x_0 in the two FSRs are always identical). The tranformation can be done by moving a set of product terms P from f_i to f_j while the index of each variable x_k of each product term in P is changed to x_{k-i+j}. To guarantee the equivalence of a Fibonacci

NLFSR to a Galois NLFSR, product terms cannot be shifted to positions lower that the *minimum terminal bit* τ_{min} which is calculated as:

$$\tau_{min} = \max_{p_i \in P} (max\_index\,(p_i) - min\_index\,(p_i))$$

where $min\_index(p_i)$ and $max\_index(p_i)$ denote respectively the minimum and maximum index of the variables in product term p_i. In other words, τ_{min} is the maximal value of $max\_index(p_i) - min\_index(p_i)$ among all product terms in g_{n-1}. Proof of equivalence between the Fibonacci and the Galois FSRs can be found in [13].

When this transformation is applied to stream cipher's FSRs, as discussed in [8], the feedback functions in which a product term p_i can be moved are also limited by:

- *minimal index in the product term*: no product term p_i can be moved to a feedback function of grade lower than $n - 1 - min\_index(p_i)$.
- *combinational functions inputs*: to preserve the original encryption algorithm, no product term can be moved to a feedback function of grade lower than the highest state bit used as input of any combinational function.
- *degree of parallelization*: in an FSR parallelized k times, all feedback functions f_i except $n - j \cdot k - 1,\ \forall j = \{0, 1, \ldots, \lfloor (n-1)/k \rfloor - 1\}$ should have feedback functions of type $f_i = x_{i+1}$.

6.1 Throughput Optimization

Tranformation from Fibonacci to Galois for a single FSR can generally result into many different designs. To choose the best design in term of throughput, a heuristic algorithm was developed in [9]. This algorithm tries to find the fastest Galois FSR equivalent to a given Fibonacci FSR, i.e. the Galois FSR with the shortest critical path [9]. The algorithm does not consider area overhead.

The algorithm contains a simple critical path model of the FSRs, i.e. every FSR is associated to a cost, which is an estimation of its critical path, and the algorithm tries to find the minimal-cost Galois FSR. Normally, given a Fibonacci FSR, the algorithm can choose among more than one minimal-cost Galois FSR. Although all of them have in principle the same throughput, they have slightly different area overheads.

In this paper, we use the same algorithm suggested in [9] to define the best Fibonacci-to-Galois transformation; however, we have introduced a final area optimization stage to it.

6.2 Area Optimization

The main idea of the area optimization stage is that area savings occur when two or more products in different feedback functions can be implemented using shared gates.

Once the algorithm in [9] has identified a minimal-cost FSR, its feedback functions are scanned to search for product terms in the form $x_i x_j$... and $x_{i+k} x_{j+k}$.... These product terms will be encountered if the original Fibonacci FSR also contained product terms in the form $x_{i^*} x_{j^*}$... and $x_{i^*+k^*} x_{j^*+k^*}$.... This is a common occurrence for cryptographic FSRs.

The algorithm removes the two product terms from the feedback functions and tries to place them exactly at distance k from each other, with the first above the second. All available positions are scanned to find a suitable placement. The product-term movement takes place only if it does not increase the cost of the FSR, i.e. its estimated critical path. If a suitable placement for the two product terms is found, the products $x_i x_j$ and $x_{i+k} x_{j+k}$ are transformed to the same product $x_{i^{**}} x_{j^{**}}$, and both products can be implemented using a single shared AND gate. The main idea of this optimization is shown in Figure 4: the Galois configuration allows to implement the two product terms of the Fibonacci FSR using a single AND gate.

The algorithm continues until all suitable product terms have been placed and no further area optimization is possible.

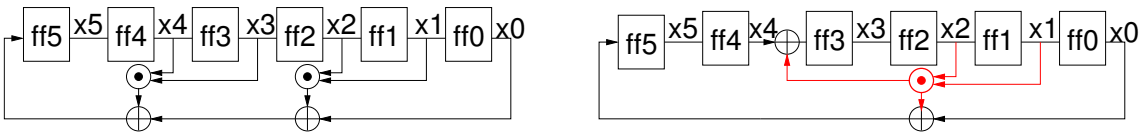


Fig. 4. Area savings obtained through shared AND gates.

6.3 NLFSR Fibonacci to Galois Transformation

By transforming the FSRs of Grain-128aXk from a Fibonacci to a Galois configuration, as shown in Table 3, Galois FSRs (both LFSR and NLFSR) run faster compared to the original Fibonacci FSRs. The highest improvement in timing is 67% for Grain-128aX1's NLFSR; the average timing improvement is 34%. In average, the areas of the FSRs improve by 2%. However, for some of the FSRs the area increases; the maximal area increase is 7% for the LFSR of Grain-128aX1.

The Galois LFSRs and NLFSRs of Grain-128aXk which are obtained from the algorithms in [9] and the area optimization algorithm from Subsection 6.2 can be found in Appendix A.

6.4 Effect on the Cipher

Table 4 reports the timing figures and critical paths for the Grain-128Xk ciphers after the Fibonacci to Galois transformation. Despite the high timing improvements in the cipher's FSRs (see Table 3), the minimal clock periods of the complete ciphers improve only by 9% on average (compare Tables 4 and 2).

Table 3. Timing (maximum frequency) and area in the optimized Galois (Opt.) and Fibonacci FSRs of Grain-128a implemented as stand-alone systems.

FSR		X1		X2		X4		X8		X16		X32	
		L	N	L	N	L	N	L	N	L	N	L	N
Frequency (GHz)	Opt.	5.6	4.5	5.5	4.0	5.3	3.7	4.4	3.1	4.0	2.7	4.0	2.7
	Fib.	3.5	2.7	3.6	2.6	3.5	2.6	3.5	2.5	3.4	2.4	4.0	2.7
	impr.	60%	67%	53%	54%	51%	42%	26%	24%	18%	13%	0%	0%
Area (μm^2)	Opt.	2649	2736	2554	2876	2680	3493	3034	4624	3581	6576	4732	11676
	Fib.	2481	3044	2600	3154	2615	3453	3301	4830	3785	6782	4732	11676
	impr.	-7%	10%	2%	9%	-3 %	-1%	8%	4%	5%	3%	0%	0%

Table 4. Timing in the versions of Grain-128a after the Fibonacci to Galois transformation.

	X1	X2	X4	X8	X16	X32
Min. clock period (ps)	440	445	472	482	573	580
Critical path	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}

The critical paths for all versions of grain-128a are given by D_{hyn}, i.e. performances are limited by the initialization path from the FSRs bits through the h, y and g functions to the first NLFSR bit.

This path is active only during the keystream initialization phase. However, the performances of the cipher are determined by the throughput during the operational phase. To increase further the throughput of the cipher, as suggested in [8], we use different clocks in different cipher phases. This solution is described in the next section.

7 Dual Frequency Implementation

In [8], to support efficiently both initialization and key generation phases, the authors suggested a dual-frequency implementation of Grain-80 in which the cipher works with a slower clock during the keystream initialization phase and a faster clock during the keystream generation phase (the phase that determines throughput).

To dynamically change the frequency of the cipher we use a clock division block, shown in Figure 5. The cipher receives only one external clock signal. This clock is used by the cipher during the keystream generation phase ($run = 1$); the frequency of the clock is divided by two during the keystream initialization phase ($run = 0$). A clock divider by two is sufficient to achieve the desired functionality, as we will show further on in this section. As shown in Figure 5, the clock division block is very simple and its area overhead is very small compared to that of the cipher.

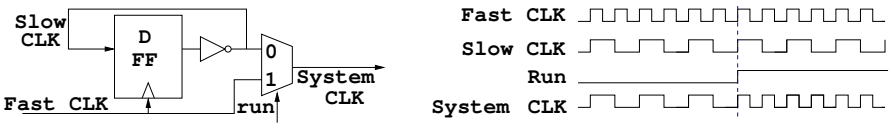


Fig. 5. Clock divider by two.

Table 5. Timing in the versions of Grain-128a during the keystream initialization (K.I.) and generation (K.G.) phases after the introduction of the dual frequency solution.

Phase		X1	X2	X4	X8	X16	X32
K.I.	Min. clock period (ps)	561	598	607	639	751	840
	Critical path	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}
K.G.	Min. clock period (ps)	389	372	389	403	446	498
	Critical path	D_{hy}	D_{hy}	D_{hy}	D_{hy}	D_n	D_n

To further increase the advantages of our method, we synthesize the ciphers optimizing them for operation during the keystream generation phase by defining the paths from the outputs of the y functions to the inputs of the LFSR and NLFSR as false paths during synthesis, i.e. we instruct the synthesis tool not to optimize any combinational path going from the outputs of the y functions to the inputs of the LFSR and NLFSR. This makes D_{hyn} larger, but reduces D_n, D_a and D_{hy} because it pushes the tool to optimize them as much as possible.

The timing figures and critical paths of the different Grain-128a ciphers in both the keystream initialization and keystream generation phases are reported in Table 5. Based on the results in Table 5, clock division by two is sufficient to ensure correct operation for all degrees of parallelization, because the frequencies at which the ciphers can run during the initialization phase are more than half the frequencies at which the ciphers can run during the keystream generation phase.

For high-parallelism versions of the cipher, the critical path during the operational phase is given by one of the feedback functions of the NLFSR, which are hard to optimize further. However, for $k \leq 8$, the performances of the cipher are limited by the propagation delay through the h and y functions to the output. In the next section, we pipeline these functions to increase the throughput of the cipher.

8 Internal Pipelining

To keep the presentation simple, in this section we consider the cascade of the h and y functions as a single non-linear function hy. This hy function is pipelined using a 2-level or 3-level pipeline, using the solution shown in Figure 6 (for two levels). During the keystream generation phase, the output of the hy function goes to the output and the authentication section of the cipher. There is no feedback to the keystream generator; therefore, pipelining the hy function does

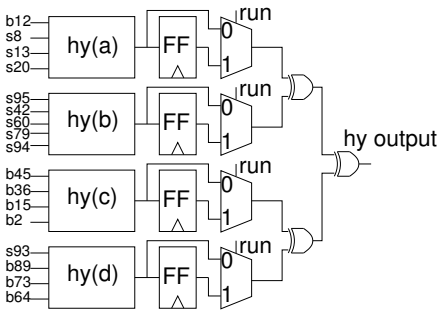


Fig. 6. Pipelined *hy* function (two levels).

Table 6. Timing in the versions of Grain-128a during the keystream initialization (K.I.) and generation (K.G.) phases after pipelining the *hy* function with 2 or 3 pipeline levels (P.L.).

P.L.	Phase		X1	X2	X4	X8	X16	X32
2	K.I.	Min. clk period (ps)	579	604	629	657	830	904
		Critical path	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}
	K.G.	Min. clk period (ps)	303	286	328	350	417	488
		Critical path	D_{hy}	D_{hy}	D_{hy}	D_n	D_a	D_a
3	K.I.	Min. clk period (ps)	569	623	668	685	827	881
		Critical path	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}	D_{hyn}
	K.G.	Min. clk period (ps)	283	280	305	350	410	475
		Critical path	D_{hy}	D_{hy}	D_{hy}	D_n	D_a	D_a

not alter the functionality of the cipher but only introduces a latency delay in the generation of the output stream and the authentication tag, which does not have any effect on the security of the cipher.

During the keystream initialization phase, the output of the *hy* function is fed back to the first bits of the NLFSR and the LFSR; if the *hy* function is pipelined during this phase, the functionality of the cipher is altered. Therefore, multiplexers are implemented in the pipeline to bypass the flip-flops and deactivate it during the keystream initialization phase. The initialization path through the multiplexers is defined as a false path during synthesis to push the tool to optimize for the operational phase.

Table 6 shows timing figures and critical paths for versions of the cipher with 2-level and 3-level pipelines on the *hy* function. Pipelining the *hy* function improves the timing of all versions of Grain-128a. The best improvement is obtained for Grain-128aX2.

Pipelining the *hy* function has also some drawbacks: flip-flops have to be inserted to implement the pipeline; the propagation delay through the *hy* function during the keystream initialization phase increases due to the delays of the multiplexers. The drawbacks increase with the number of pipeline levels.

9 Final Comparison

Table 7 compares the different versions of the ciphers that have been implemented in this work. Five implementations of the ciphers are compared: (ORG) are the original ciphers (see Section 4); (F2G) are the original ciphers after the isolation of the authentication section and the Fibonacci to Galois transformation of the FSRs (see Sections 5 and 6); (2F) are the F2G ciphers modified with the implementation of the dual frequency solution (see Section 7); (P2) and (P3) are the 2F ciphers after internal pipelining respectively with two and three levels (see Section 8).

The results are reported in terms of maximal frequency, throughput ($f_{max} \cdot \frac{k}{2}$), area and power. For every value, we report also the improvement over the original cipher. With a 2-level (3-level) pipeline, the average improvement in throughput

Prop.	k	ORG	F2G		2F		P2		P3	
		v.	v.	imp. (%)	v.	imp. (%)	v.	imp. (%)	v.	imp. (%)
Freq. (GHz)	X1	2.1	2.3	10	2.6	24	3.3	57	3.5	67
	X2	2.0	2.3	15	2.7	35	3.5	75	3.6	80
	X4	2.0	2.1	5	2.6	30	3	50	3.3	65
	X8	1.9	2.1	10	2.5	32	2.9	53	2.9	53
	X16	1.7	1.7	0	2.2	29	2.4	41	2.4	41
	X32	1.5	1.7	13	2.0	33	2	33	2.1	40
Through. (Gb/s)	X1	1.1	1.15	10	1.3	24	1.65	57	1.75	67
	X2	2.0	2.3	15	2.7	35	3.5	75	3.6	80
	X4	4.0	4.2	5	5.2	30	6	50	6.6	65
	X8	7.6	8.4	10	10	32	11.6	53	11.6	53
	X16	13.6	13.6	0	17.6	29	19.2	41	19.2	41
	X32	24	27.2	13	32.0	33	32	33	33.6	40
Area (μm^2)	X1	5876	5747	2	5689	3	5791	1	5856	0
	X2	6972	6872	1	6910	1	7280	-4	7314	-5
	X4	8299	8153	2	8499	-2	8949	-8	9145	-10
	X8	10778	10725	0	10925	-1	11087	-3	11538	-7
	X16	15709	16681	-6	16516	-5	18053	-15	17653	-12
	X32	23430	25554	-9	26572	-13	30127	-28	28917	-23
Power (μW)	X1	96.9	95.5	1	94.5	2	93.5	3	94.1	3
	X2	106.1	103.7	2	105.3	1	122.2	-15	113.1	-7
	X4	120.6	130.6	-8	122.9	-2	130.4	-8	136.1	-13
	X8	176.4	169.4	4	179.5	-2	174.6	1	184.4	-4
	X16	247.8	269.0	-8	246.4	1	291.6	-18	275.4	-11
	X32	417.9	403.5	3	402.9	4	474.7	-14	415.1	1

Table 7. Implementation results

among all the versions of the cipher is 52% (58%). The highest improvement is

75% (80%) for Grain-128aX2 and the minimal improvement is 33% (40%) for Grain-128aX32. These improvements come at the expense of an average 9% (9%) area penalty. There is no area penalty for Grain-128aX1 and the penalty increases with k. The average power consumption of the ciphers increases in average by 8% (5%). For Grain-128aX1, the techniques we apply have a very good effect because they considerably increase throughput and also reduce marginally area and power.

10 Conclusion

In conclusion, we have shown that it is possible to considerably improve the hardware timing figures of merit of the different versions of the Grain-128a cipher by applying a combination of different techniques. The throughput of the ciphers improved on average 52% or 58% depending on the number of pipeline stages. The improvements came at the expense of only an average 9% area penalty. The power figures of the ciphers increased on average by 8% or 5%. Only easy to apply standard cell techniques compatible with the standard ASIC design flow were considered.

Acknowledgment: This work was supported in part by a project No 621-2010-4388 from Swedish Research Council.

References

1. A. Juels, "Rfid security and privacy: a research survey," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 2, pp. 381–394, Feb. 2006.
2. T. Good and M. Benaissa, "ASIC hardware performance," *New Stream Cipher Designs: The eSTREAM Finalists, LNCS 4986*, pp. 267–293, 2008.
3. M. Robshaw and O. Billet, Eds., *New Stream Cipher Designs: The eSTREAM Finalists*. Berlin, Heidelberg: Springer-Verlag, 2008.
4. M. Hell, T. Johansson, A. Maximov, and W. Meier, "The Grain family of stream ciphers," *New Stream Cipher Designs: The eSTREAM Finalists, LNCS 4986*, pp. 179–190, 2008.
5. S. Babbage and M. Dodd, "The mickey stream ciphers," pp. 191–209, 2008.
6. C. Cannière and B. Preneel, "Trivium," *New Stream Cipher Designs: The eSTREAM Finalists, LNCS 4986*, pp. 244–266, 2008.
7. T. Good and M. Benaissa, "Hardware results for selected stream cipher candidates," in *of Stream Ciphers 2007 (SASC 2007), Workshop Record*, 2007, pp. 191–204.
8. S. Mansouri and E. Dubrova, "An improved hardware implementation of the grain stream cipher," in *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, sept. 2010, pp. 433 –440.
9. J.-M. Chabloz, S. S. Mansouri, and E. Dubrova, "An algorithm for constructing a fastest galois nlfsr generating a given sequence," in *Proceedings of the 6th international conference on Sequences and their applications*, ser. SETA'10. Springer-Verlag, 2010, pp. 41–54.

10. M. Agren, M. Hell, T. Johansson, and W. Meier, "Grain-128a: a new version of grain-128 with optional authentication," *Int. J. Wire. Mob. Comput.*, vol. 5, pp. 48–59, Dec. 2011.
11. M. Hell, T. Johansson, A. Maximov, and W. Meier, "A stream cipher proposal: Grain-128," in *Information Theory, 2006 IEEE International Symposium on*, July 2006, pp. 1614–1618.
12. T. Balph, "Lfsr counters implement binary polynomial generators," *MOTOROLA SEMICONDUCTOR, EDN*, vol. 43, pp. 155–116, 1998.
13. E. Dubrova, "A transformation from the fibonacci to the galois nlfsrs," *Information Theory, IEEE Transactions on*, vol. 55, no. 11, pp. 5263 –5271, nov. 2009.
14. S. Golomb, *Shift Register Sequences*. Aegean Park Press, 1982.

Appendix A: Fibonacci to Galois Transformation of FSRs in Grain-128a

NLFSR Fibonacci to Galois Transformation

We use the algorithm described in [9] and the area optimization algorithm from Subsection 6.2 to transform the NLFSRs of Grain-128aXk from a Fibonacci to a Galois configuration. For Grain-128a, the product term with the maximal difference in variable indexes is b_3b_{67}, i.e. $\tau_{min} = 64$ (see Section 6). Product terms cannot be allocated to functions f_i of grade $i < 95$ because bit b_{95} is used in function h (see Section 6).

The area optimization algorithm (see Subsection 6.2) places the Fibonacci product terms $b_{88}b_{92}b_{93}b_{95}$, $b_{22}b_{24}b_{25}$ and $b_{70}b_{78}b_{82}$ respectively 27, 11 and 30 feedback functions downer than the product terms $b_{61}b_{65}$, $b_{11}b_{13}$ and $b_{40}b_{48}$.

The following Galois NLFSR is obtained:

$$
\begin{aligned}
f_{127} &= s_0 \oplus b_0 \\
f_{126} &= b_{127} \oplus b_{39}b_{47} \\
f_{125} &= b_{126} \oplus b_{59}b_{63} \\
f_{124} &= b_{125} \oplus b_0b_{64} \\
f_{123} &= b_{124} \oplus b_{52} \\
f_{116} &= b_{117} \oplus b_0b_2 \\
f_{105} &= b_{106} \oplus b_0b_2b_3 \\
f_{110} &= b_{111} \oplus b_0b_1 \\
f_{102} &= b_{103} \oplus b_{71} \\
f_{101} &= b_{102} \oplus b_0 \\
f_{100} &= b_{101} \oplus b_0b_{32} \\
f_{99} &= b_{100} \oplus b_{63} \\
f_{98} &= b_{99} \oplus b_{59}b_{63}b_{64}b_{66} \\
f_{97} &= b_{98} \oplus b_{38}b_{54} \\
f_{96} &= b_{97} \oplus b_{39}b_{47}b_{51}
\end{aligned}
$$

Here and in the remainder of the paper, unspecified feedback functions are in the form $f_i = x_{i+1}$.

For Grain-128aX2 the Fibonacci product terms of the original NLFSR can be moved only to feedback functions f_{127}, f_{125}, f_{123}, f_{121}, f_{119}, f_{117}, f_{115}, f_{113}, f_{111}, f_{109}, f_{107}, f_{105}, f_{103}, f_{101} and f_{99}. The following Galois NLFSR is obtained after application of the timing and area optimization algorithms:

$$
\begin{aligned}
f_{127} &= b_0 \oplus s_0 \\
f_{125} &= b_{126} \oplus b_1 b_{65} \\
f_{123} &= b_{124} \oplus b_{57} b_{61} \\
f_{121} &= b_{122} \oplus b_5 b_7 \\
f_{119} &= b_{120} \oplus b_9 b_{10} \\
f_{115} &= b_{116} \oplus b_{15} b_{47} \\
f_{113} &= b_{114} \oplus b_{12} \\
f_{111} &= b_{112} \oplus b_6 b_8 b_9 \\
f_{109} &= b_{110} \oplus b_{73} \\
f_{107} &= b_{108} \oplus b_{62} b_{58} b_{50} \\
f_{105} &= b_{106} \oplus b_{18} b_{26} \\
f_{103} &= b_{104} \oplus b_{72} \\
f_{101} &= b_{102} \oplus b_{30} \\
f_{99} &= b_{100} \oplus b_{40} b_{56} \\
f_{97} &= b_{98} \oplus b_{58} b_{62} b_{63} b_{65}
\end{aligned}
$$

The area optimization algorithm places the Fibonacci product terms $b_{88}b_{92}b_{93}b_{95}$ and $b_{70}b_{78}b_{82}$ respectively 26 and 16 feedback functions downer than the Fibonacci product term $b_{61}b_{65}$. Also, the $b_{22}b_{24}b_{25}$ product term is placed 10 feedback functions downer than the $b_{11}b_{13}$ product term. This allows sharing gates among some of the parallelized feedback functions.

For Grain-128aX4 the Fibonacci product terms of the original NLFSR can be moved only to feedback functions f_{127}, f_{123}, f_{119}, f_{115}, f_{111},f_{107}, f_{103} and f_{99}. The NLFSR is transformed to:

$$
\begin{aligned}
f_{127} &= s_0 \oplus b_0 \oplus b_3 b_{67} \\
f_{123} &= b_{124} \oplus b_{22} \oplus b_{52} \oplus b_{23} b_{55} \\
f_{119} &= b_{120} \oplus b_9 b_{10} \oplus b_3 b_5 \\
f_{115} &= b_{116} \oplus b_{70} b_{66} b_{58} \\
f_{111} &= b_{112} \oplus b_6 b_8 b_9 \\
f_{107} &= b_{108} \oplus b_{68} b_{72} b_{73} b_{75} \\
f_{103} &= b_{104} \oplus b_{72} \oplus b_{37} b_{41} \\
f_{99} &= b_{100} \oplus b_{40} b_{56} \oplus b_{63} \oplus b_{12} b_{20}
\end{aligned}
$$

The area optimization algorithm places the Fibonacci $b_{88}b_{92}b_{93}b_{95}$ product term 8 feedback functions downer than the product term $b_{70}b_{78}b_{82}$; the product term $b_{22}b_{24}b_{25}$ is placed 8 feedback functions downer than the product term $b_{11}b_{13}$.

For Grain-128aX8 the Fibonacci product terms of the original NLFSR can be moved only to feedback functions f_{127}, f_{119}, f_{111} and f_{103}. The NLFSR is

transformed to:

$$f_{127} = s_0 \oplus b_0 \oplus b_3 b_{67} \oplus b_{88} b_{92} b_{93} b_{95}$$
$$f_{119} = b_{120} \oplus b_9 b_{10} \oplus b_3 b_5 \oplus b_{32} b_{40} \oplus b_{60} b_{76}$$
$$f_{111} = b_{112} \oplus b_{10} \oplus b_{40} \oplus b_{11} b_{43} \oplus b_{75} \oplus b_6 b_8 b_9$$
$$f_{103} = b_{104} \oplus b_{72} \oplus b_{37} b_{41} \oplus b_{46} b_{54} b_{58}$$

For Grain-128aX16 the Fibonacci product terms of the original NLFSR can be moved only to feedback functions f_{127} and f_{111}. The NLFSR is transformed to:

$$f_{127} = s_0 \oplus b_0 \oplus b_{56} \oplus b_3 b_{67} \oplus b_{11} b_{13} \oplus b_{40} b_{48} \oplus b_{22} b_{24} b_{25} \oplus b_{70} b_{78} b_{82}$$
$$f_{111} = b_{112} \oplus b_{10} \oplus b_{75} \oplus b_{80} \oplus b_1 b_2 \oplus b_{11} b_{43} \oplus b_{45} b_{49} \oplus b_{72} b_{76} b_{77} b_{79} \oplus b_{68} b_{52}$$

For Grain-128aX32 the Fibonacci product terms of the original NLFSR can be moved only to feedback functions f_{127}, i.e. the NLFSR cannot be transformed into a Galois NLFSR.

LFSR Fibonacci to Galois Transformation

The NLFSR optimization algorithm is used also to optimize the LFSR of Grain-128a.

For Grain-128a, Grain-128X2 and Grain128X4 the LFSR is transformed to:

$$f_{127} = s_0 \oplus s_7$$
$$f_{123} = s_{124} \oplus s_{34}$$
$$f_{119} = s_{120} \oplus s_{62}$$
$$f_{115} = s_{116} \oplus s_{85}$$
$$f_{111} = s_{112} \oplus s_{80}$$

For Grain-128X8, the LFSR is transformed to:

$$f_{127} = s_0 \oplus s_7 \oplus s_{38}$$
$$f_{119} = s_{120} \oplus s_{62}$$
$$f_{111} = s_{112} \oplus s_{65}$$
$$f_{103} = s_{104} \oplus s_{72}$$

For Grain-128X16, the LFSR is transformed to:

$$f_{127} = s_0 \oplus s_7 \oplus s_{38} \oplus s_{70}$$
$$f_{111} = s_{112} \oplus s_{65} \oplus s_{80}$$

For Grain128a-X32, it is impossible to generate a Galois equivalent of the cipher's LFSR.

Suggestions for Hardware Evaluation of Cryptographic Algorithms

Frank K. Gürkaynak

Microelectronics Designs Center, ETH Zurich, Switzerland
kgf@ee.ethz.ch

Abstract. Public competitions to determine new cryptographic standards such as the search for AES, SASC, SHA-3 have been very successful. Many researchers contribute to such competitions to evaluate different aspects of the candidates among others also the performance of hardware and software implementations. While significant effort has been put into hardware evaluations, somehow the result of these studies could not reliable be tabulated, at least not with the same efficiency as it was done for software evaluations. This paper investigates different shortcomings of current hardware evaluations and proposes changes to future calls, so that the results of hardware evaluators can be combined and compared more reliably.

1 Introduction

Public competitions to determine new cryptographic standards have enjoyed significant success. It is especially remarkable to see the progress that has been made in evaluating the software performance for these competitions. In little more than 10 years time, the software benchmarking has evolved from an ad-hoc discipline to a systematic and automated approach as demonstrated by the success of eBASH. The main advantage of the approach shown by eBASH is that it allows different implementations to be compared and accumulated objectively.

At the moment, hardware evaluation for these competitions are not at the same level as the software evaluations. Although the number of studies on hardware studies have increased both in number and quality, the results are still not directly comparable.

Hardware design is a difficult and tedious process that still relies heavily on human experience. This effort is only justified if the design involved is trying to achieve the highest level of performance, efficiency, or is trying reduce the cost of implementation as much as possible. While the design methodology to achieve these goals is almost identical, the optimization techniques and design decisions would differ significantly depending on the goal of the implementation. This is the main reason why a *general* comparison of hardware is not trivial. There are many buttons and knobs that can be adjusted during the design process. While in each of the cases, a functional cryptographic implementation would be realized, two resulting circuits optimized for different parameters and constraints may not be really comparable.

The goal of this paper is to outline some ideas that we believe should be implemented in future calls for hardware evaluations, in order to ensure that reports from various contributors can be directly compared. This may at one point lead to an automated evaluation system similar to eBASH.

In our opinion the main reason why results on hardware implementations can not be directly compared is due to two facts:

- The calls do not set clear goals for hardware implementations. Designers are therefore forced to make the decisions themselves. Not only are they prone to make (slightly) different decisions, but these decisions could directly affect the outcome or ranking of the candidates.
- The boundary conditions for the implementations are not given. These include how the environment communicates with the cryptographic core, how various options (key changes, initialization vectors, padding, etc.) are set and supported. Once again the designers are forced to make arbitrary decisions which results in hardware that may do things differently and will therefore not really be compatible.

This paper outlines four concrete suggestions that should be included in future calls for hardware evaluations to counter these problems. These suggestions are described in some detail in the following section. There are some other issues related to hardware evaluations that should also be considered which are briefly addressed in Section 3. Finally a summary of the suggestions is given in Section 4.

2 Requirements in the Call

When compared to software implementations, the hardware implementation of an algorithm is more involved and certainly more costly in terms of design and development time. As such hardware design is only used if the application has extreme requirements in circuit size, power/energy consumption or performance that can not be met with a software implementation.

When mapping a cryptographic algorithm onto a hardware architecture, designers usually enjoy a large degree of freedom. They can choose a technology to implement the design, adapt their architecture for different performance requirements and interfaces. In an industrial design, the system specifications set limits on most of these choices, which in turn guides the design process. Such limits are simply not there during hardware evaluations performed for recent calls. Different groups end up making different design choices (scenario, technology, interface). The results they produce are therefore not directly comparable[1].

In order to allow *fair* comparisons, or even a system where an independent entity collects all hardware implementations and tabulates the results like eBASH, the call should set some limits/goals for the hardware implementations. This section discusses the main points.

2.1 Defining Scenarios

One of the main difficulties in comparing the hardware performance of different cryptographic algorithms in recent competitions was that each group used different application scenarios, which emphasized some parameters more than the others. The problem was not that any of the assumptions were wrong, it was more that every researcher had a different assumption resulting in a slightly different architecture which in turn did not help when comparing results between different groups. If the application scenario was given in the call, the researchers would all be producing results for the same performance parameters, which in turn will simplify comparisons.

It is both expected and normal that different algorithms exhibit different strengths when implemented in hardware. When a group decides on any given parameter (i.e. consider the fastest algorithm), the result will already be skewed in favor of some candidates. The problem is that the evaluator will be charged with making this decision. For a fair comparison, the person evaluating should not be allowed to choose what performance parameter to concentrate as this will directly affect the result.

Obviously, it is desirable to have an application scenario that closely matches the anticipated application of the cryptographic algorithm in question. If such an application is known in advance, defining such an application scenario will be easy. The problem with most cryptographic algorithms is that they can be employed in a variety of different applications with contrasting requirements. I.e. high performance, high throughput networking, and wireless sensor nodes.

Note that, it is not very important what exactly the application scenario is. The benefit of having an application scenario is to make sure all groups put the same emphasis in their designs, and are not asked to figure out which performance parameter is more important. However, arguments will most probably be made to support different scenarios, citing the conflicting requirements for general purpose cryptographic algorithms. It would be possible to define more than one scenario to placate such requests, although doing so would have several disadvantages:

- More time will be spent in defining these scenarios in the call.
- If real-world requirements are not known, defining more scenarios will not really *improve* the quality of the scenarios. Instead of one unrepresentative scenario, there will be potentially several *equally bad* scenarios.
- The results will be split between scenarios, reducing the number of directly comparable results.

In the following, some scenarios are listed to serve as examples.

- **Scenario 1: Sensor node**
  Messages of 32 to 512 bits in length, expected throughput 1000 kbits/s, key change every 3 months. Absolute energy consumption highest priority, small area second, throughput third.

[1] How can you comment on the difference between a minimal area crypto core with an 8-bit interface designed using a 130 nm technology and a high-speed implementation of the same algorithm in a 28 nm technology with a 256-bit parallel I/O.

– **Scenario 2: Embedded core**
  Messages of 32 bits to 4 Mbits in length, expected throughput 100 Mbits/s, key change every 10 ms. Smallest area priority, Energy/bit highest priority.
– **Scenario 3: Data center**
  Messages of 2^{10} bits to 2^{64} bits, expected throughput 10 Gbit/s, key change every 2^{14} bits, Throughput per area most important, power second, area third.

2.2 Defining Reference Technology / Device

Hardware design principally maps the hardware description to a given ASIC technology or FPGA device. The actual performance varies depending on the technology (or FPGA device) used. An implementation in a more recent technology (a newer FPGA device) is more likely to result in higher performance, smaller area and less power consumption.

Even though some relations can be established to compare results from different technologies, all such approaches will have significant error margins (in the range of $\pm 20\%$). Such errors could be eliminated if futures calls werre to include a specific technology/standard cell library and FPGA device for which the results should be evaluated. Once again what the choice is, will not make a real difference, it will just serve to simplify comparisons between groups.

For FPGA devices, the choosing a device is not very hard. There are a few known FPGA vendors (Xilinx, Altera, Lattice) and the design automation tools for all these vendors are capable of producing results for all possible FPGA devices that are being offered. It would be better to choose a device that is commonly avaliable on an evaluation board (i.e. Xilinx XC6VLX240T-1FFG1156 for the ML605 evaluation board).

As for the ASIC technology, a technology that is available for research institutes and universities through MPW (Multi Project Wafer) services (Europratice, CMP, Mosis) should be favored. The technology should not only be limited to the general name (i.e. 180 nm) but should also specify which process option (i.e. low-leakage, 6 metal,...), as well as the concrete standard cell library (i.e. fsa0a_c_generic_core_2009Q2v2.0) used.

While modern technologies (45 nm, 28 nm) seem to attract more attention in publication these days, it should be noted that there is no inherent advantage in using a specific technology for comparative purposes, since absolute performance numbers would not be relevant[2]. A circuit that is $2\times$ faster in 350 nm technology is most likely going to be $2\times$ faster also in the 28 nm technology. Any difference in the results is more likely to be due to the differences in the standard cell library rather than the technology.

Considering the possibility of actual ASIC implementations, it would be wise to choose a technology that is affordable and widely available (has many runs per year).

2.3 Defining the Interface

In principle when evaluating a new algorithm, the core functionality of an algorithm should be the most important part of the implementation. However, if not defined properly, the boundary conditions of the core can take both considerable design effort and influence the overall performance.

How the cryptographic core communicates with the environment, what options it supports, and how it supports them could have a significant effect on the performance. As an example, if the I/O is limited, additional storage might be needed at the inputs and outputs increasing the circuit area. What options the hardware actually supports can have even more impact on the results. Consider the recent SHA-3 competition. Which message digest size should have been considered. Only one or all at the same time. If only one, which one was more relevant? If such decisions are left to the designers, they will make arbitrary decisions and the resulting hardware would differ, making comparisons difficult[3].

It is important that the call define the interface for the hardware implementations. The most effective method would be to include a test-bench written in HDL that is able to interpret the KAT (Known Answer Test) files generated for the software models directly. Such a test-bench would have not only define the exact I/O structure of the cryptographic core, but would also define how the core communicates with the outside world (for example how a new secret key is loaded), and what options it supports, in what manner. The same test-bench would be used for both ASIC and FPGA designs.

[2] The application scenario could be adjusted to the expected performance of the selected technology

[3] How do you compare an AES implementation that supports 128, 192, and 256-bit keys for encryption only, and a second one that supports both encryption and decryption but with only 128-bit keys

2.4 Requesting HDL Implementations

All previous calls for public competitions for cryptographic algorithms required a software implementation as part of the submission package. We suggest to add an HDL implementation (Verilog or VHDL) for the submission package as well. There are several important benefits for this:

- If HDL implementations are made part of the submission package, the submitter would directly see the effects of various design choices in hardware. This would most certainly result in submissions that are more *hardware friendly*.
- There would be a *golden model* for hardware implementations. Other groups could concentrate on improving on this base implementation.
- An initial hardware evaluation using the submission packages could be started directly, much earlier than in previous submissions.

Of all the suggestions in this paper, we expect this one to be the most argued against. After all not all groups contributing in cryptographic algorithms are experienced in hardware design. Such a requirement could be consdiered a barrier for some submitters. On the other hand, there are many research groups that focus on hardware implementations[4] and which would be willing to co-operate with groups who have little or no experience.

3 Other Points

3.1 Side Channel Attack Resistance

Various forms of side channel attacks will always remain a serious threat to security. As such it would be very interesting to know if one of the candidate algorithms in a public competition is more resistant to possible side-channel attacks.

The main problem with side-channel attacks is that the side-channels are a by-product of the specific implementation of a given algorithm. It may be the case that certain algorithmic structures might render some known side-channel attacks more difficult. However until now, there is no established *way* of determining the susceptibility of a given algorithm to all possible side-channel attacks that can be mounted against any given implementation of this algorithm.

Any evaluation against side-channel resistance, will be based on the particular implementation of the algorithm (and the specific attack). While the current proposal does not add any points for the side channel evaluation, it will offer the following advantages.

- All submission would include an HDL implementation, making sure that there is at least one valid/verified implementation that can be directly used for side-channel related studies.
- The interface to control all the hardware evaluations will be common making it easier to compare different implementations from different groups
- Earlier availability of verified HDL code will make it easier to develop ASICs. This will lead to actual measurements on devices much earlier than in other competitions.

3.2 Automated Comparisons

One of the main criticisms directed at hardware evaluation efforts has been the lack of a fully automated system that would tabulate the results for a variety of different options similar to eBASH. Although there have been some efforts in the FPGA domain (ATHENA), reliable ASIC comparisons are still not centralized.

The main obstacle in realizing such an automated system is that there are many degrees of freedom in hardware design. Trying out every conceivable design option is not feasible.

If the suggestions in this paper are adopted, the degrees of freedom will be severely restricted. Especially limiting the designs to one (or up to three) different scenarios, and providing a standard test-bench will force the designers to generate HDL codes that could be considered comparable. This could then lead to an automated evaluation system,

[4] The ETH Zurich being one of them

that would accept submissions from different groups, realize all the designs using the same parameters and tabulate the results.

The counter argument for this proposal is that, the suggestions in this paper would limit the options of the designers, and while it would generate *more* comparable results, it would mask potential advantages of individual architectures in the design space that is not being covered with the scenarios/interfaces specified in the call. Personally we think having a conclusive result with a narrow focus, is more preferable to having a set of disconnected results each focused on a different aspect. At the same time, making concrete specifications in the call should not discourage researchers to investigate and designs for other implementation scenarios. The call should just set a higher priority results for a specific scenario and interface.

3.3 Limitations of Comparisons between different Technology and Tools

Unlike the tools used for software development (computers, compilers), the electronic design automation tools, access to technology information and design libraries are very expensive. Almost all research groups are granted access to these tools and libraries at significantly reduced cost through educational agreements that have strict limitations on usage. In general, these companies are not pleased when their tools and libraries are compared to their competitors, and many agreements have clauses that explicitly forbids such comparisons.

If several different tools or design libraries are used, any comparison between implementations will also serve as a comparison between different tools and design libraries. It is important to keep this in mind when developing platforms that disseminate comparative information.

4 Conclusions

We suggest that in future calls for a public competition for a cryptographic algorithm, a more reliable hardware evaluation can be made if the following suggestions are adapted:

- The call clearly describes one scenario (or at most three scenarios) for which the hardware will be implemented. The scenario would define which design parameters will be more relevant, and will not leave this decision to the evaluators.
- The call describes a reference technology for both ASIC and FPGA implementations. Submitters would be required to present results for the reference technology/device but would be encouraged to use other options.
- The call provides a working test-bench that determines the input/output of the hardware macro. The test-bench will be designed in a way to allow the same KAT tests to be applied both to HW and SW at the same time.
- The call asks for a plain HDL implementation (no special FPGA/macro block) as part of the submission package.

Permutation-based encryption, authentication and authenticated encryption

Guido Bertoni[1], Joan Daemen[1], Michaël Peeters[2], and Gilles Van Assche[1]

[1] STMicroelectronics
[2] NXP Semiconductors

Abstract. While mainstream symmetric cryptography has been dominated by block ciphers, we have proposed an alternative based on fixed-width permutations with modes built on top of the sponge and duplex construction, and our concrete proposal Keccak. Our permutation-based approach is scalable and suitable for high-end CPUs as well as resource-constrained platforms. The latter is illustrated by the small Keccak instances and the sponge functions Quark, Photon and Spongent, all addressing lightweight applications. We have proven that the sponge and duplex construction resist against generic attacks with complexity up to $2^{c/2}$, where c is the capacity. This provides a lower bound on the width of the underlying permutation. However, for keyed modes and bounded data complexity, a security strength level above $c/2$ can be proven. For MAC computation, encryption and even authenticated encryption with a passive adversary, a security strength level of almost c against generic attacks can be attained. This increase in security allows reducing the capacity leading to a better efficiency. We argue that for keyed modes of the sponge and duplex constructions the requirements on the underlying permutation can be relaxed, allowing to significantly reduce its number of rounds. Finally, we present two generalizations of the sponge and duplex constructions that allow more freedom in tuning the parameters leading to even higher efficiency. We illustrate our generic constructions with proposals for concrete instantiations calling reduced-round versions of the Keccak-f[1600] and Keccak-f[200] permutations.

1 Introduction

In the last decades, mainstream symmetric cryptography has been dominated by block ciphers: block cipher modes of use have been employed to perform encryption, MAC computation and authenticated encryption. Moreover, most hash functions internally call a compression function with a block cipher structure at its kernel. From a design perspective these hash functions merely consist of block ciphers in some dedicated mode of use. One could argue that the "swiss army knife" title usually attributed to the hash function belongs to the block cipher.

In the last five years, we have proposed new modes of use for, a.o., hashing, MAC computation and (plain or authenticated) encryption that make use of a fixed-width permutation instead of a block cipher [5,7,6]. These modes make use of the sponge and duplex constructions, illustrated in Figures 1 and 2.

In both constructions the width b of the underlying permutation is split in two: an outer part with size r and an inner part with size c. The rate r determines the efficiency of the construction and the capacity c the attainable security strength, so for a given permutation with width b, the equation $b = c + r$ expresses a trade off between security and efficiency.

The first concrete instantiation of such a permutation-based sponge function has been our design Keccak [10]. With its seven associated permutations, it goes from a toy primitive to a wide sponge function. In the meanwhile several other sponge functions have been proposed: Quark [2], Photon [20] and Spongent [12]. Remarkably, the latter three were all put forward as lightweight hash functions. All four designs are based on permutation families covering multiple widths rather than a single width. All together, these permutations cover a large number of widths ranging from 25 to 1600 bits.

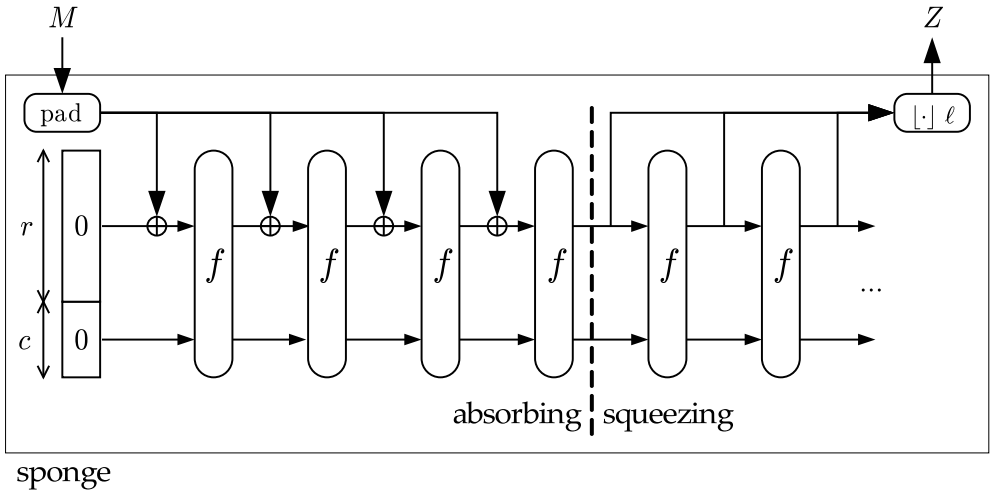


Fig. 1. The sponge construction

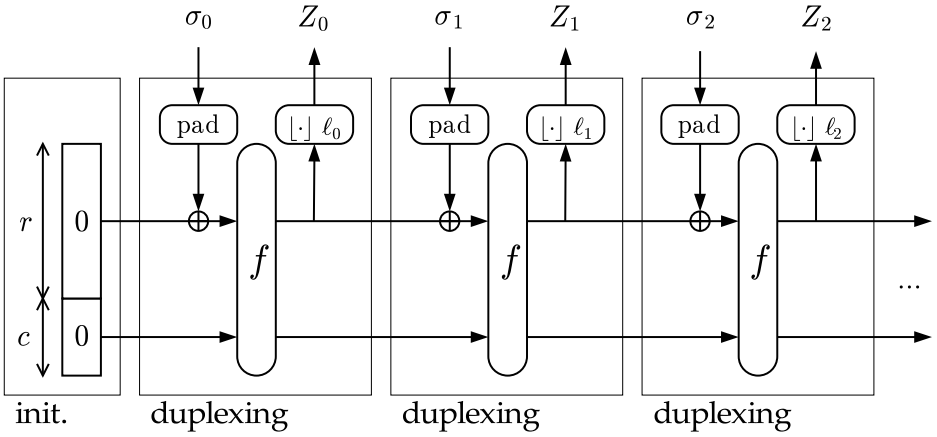


Fig. 2. The duplex construction

Additionally, numerous other cryptographic designs have iterated permutations at their core, such as the hash function Grøstl [19] whose specification counts two 512-bit permutations P_{512} and Q_{512} and two 1024-bit permutations P_{1024} and Q_{1024}, JH [27] that makes use of a 1024-bit permutation called E8, or the stream ciphers Salsa and ChaCha [3] that are based on 512-bit permutations. Moreover, most block ciphers can be converted into an iterated permutation by fixing the key to some constant. All these permutations can be used in sponge functions and duplex objects.

In this note, we focus on the intuition behind our parameter choices rather than formal proofs. The outline is as follows. In Section 2, we look at the generic security of keyed sponge and duplex modes and exploit the known results to increase the efficiency for authentication and (plain or authenticated) encryption. In section 3 we argue that cryptanalytic results provide evidence that primitives are much harder to attack in keyed modes than in un-keyed modes and use that observation to propose keyed modes based on reduced-round variants of Keccak-f. Finally, we propose in Section 4 variants dedicated to authentication (based on Alred) and (authenticated or plain) encryption.

Throughout the text, we use the concept of *security strength* as used by NIST in its cryptography standards. It is defined in [22] as *a number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system*. Security strength levels are expressed in bits and a level of n bits implies that the amount of work to break the system is of the order 2^n operations. As exhaustive key search of an n-bit key takes an amount of work 2^n, we will adopt for a target security strength of n, keys of length n. In its standards NIST targets five specific security strength levels: 80, 112, 128, 192 and 256 bits. The concrete instances we propose in this note address 80 and 128 bits for the lightweight instances and 128 and 256 bits for the other ones.

2 Increasing the rate in the sponge and duplex constructions

Using the indifferentiability framework we have proven that the sponge and duplex constructions are secure against generic attacks with complexity below $2^{c/2}$ [4]. With this bound, a desired security strength level of 80 bits implies a capacity of 160 bits and hence imposes a minimum width for the permutation, which may hinder lightweight applications. Moreover, permutations with a width just slightly above 160 bits will inevitably lead to small rates.

When a sponge function or duplex object is used in conjunction with a key, one can prove more refined bounds taking into account the data complexity. In [8] we have proven that if the data complexity is limited to 2^a r-bit blocks, the keyed mode withstands generic attacks with time complexity up to 2^{c-a} calls of the underlying permutation. If $a < c/2$, this results in an increase of the security strength from $c/2$ to $c - a$.

The bound $c - a$ assumes a very powerful adaptive adversary and the intuition behind it is the following. Given sufficient output of a keyed sponge (or duplex) object, an attacker can make guesses for the inner c bits of the state and verify for each guess whether it is consistent with the observed output. In keyed modes of the sponge or duplex construction, the knowledge of the inner part of the state is as valuable as knowing the key. The probability of success of a single guess is 2^{-c} and hence for typical values of r, i.e. $r \geq 8$, the expected workload of this attack is very close to 2^{c-1} executions of the underlying permutation f. If $r > c$, this corresponds with generically solving a constrained-input constrained output (CICO) problem [6] for f with c unknown bits at its input and c unknown bits at its input. In general an adaptive attacker can reduce this expected workload by a factor close to 2^a at the cost of 2^a adaptively chosen sponge (or duplex) input blocks by converting this CICO problem in a multi-target CICO problem. She must just apply inputs to the keyed sponge or duplex instance in such a way that the outer r-bit parts of the state at the input of f has some chosen value for multiple executions of f. In the duplex mode this is in general not difficult. The $r - 2$ outer bits can be fixed to zero by feeding as σ in a duplexing call the first $r - 2$ bits of the output Z of the previous duplexing call.

If the adversary can force M executions of f with the *same* outer r bits but *different* inner c bits, the probability of success for a guess becomes $M2^{-c}$ instead of 2^{-c}, reducing the expected workload roughly by a factor M. We call M the multiplicity of the attack. The bound 2^{c-a} is a consequence of the fact that in the worst case the multiplicity M may come very close to the data complexity 2^a.

In specific use cases an adversary may not have the possibility to enforce the r outer bits to some fixed value and hence achieving a high multiplicity may be out of reach. She can count on luck to have collisions in the r outer bits and from observed sponge or duplex output blocks extract the outer value that occurs most often. The number of times this outer value is observed is then the multiplicity. If $a < 2r$, M is expected to be only 1 or

2. Once a comes close to r this value starts to grow. Some use cases in which the attacker cannot enforce the values of the outer bits are the following:

- MAC computation with the sponge construction where the input is the key in a first sequence of blocks, followed by the message in following blocks. In this case the attacker can choose the message input blocks, but does not know the r outer bits of the state prior to their absorbing. Note that by MACing many messages with the same first message block one can enforce the outer part of the state after absorbing the first message block to be the same, but this also holds for the inner part so it does not contribute to the multiplicity.
- Keystream generation with the sponge construction where the input is a key followed by a nonce. In this case specifying the same nonce will also not contribute to the multiplicity.
- Authenticated encryption with the duplex construction (SpongeWrap), with an adversary that does not have active access to the message encryptor. Clearly, in that case even an adversary that knows the plaintext can only observe duplex inputs and outputs and not choose their values. Note that access to the message decryptor will typically only provide plaintexts upon receiving messages with a valid tag.

We have discussed this issue under the name of the passive state recovery problem and proven an upper bound for the generic success probability in [5] .

Clearly, these security bounds decrease the required capacity for a given target security strength and hence open up to smaller permutations or higher rates.

If we look at real-world constraints, we think assuming $M \leq 2^{64}$ for any attacker is reasonable. Going beyond this limit would imply an attacker that is able to present over 2^{64} chosen input blocks to keyed duplex instances under attack and manage the outputs. We will assume this limit in our choice of parameters in the remainder of this note. Generalizing to other values is however straightforward.

3 Reducing the number of rounds in the underlying permutation

In the design of Keccak we have chosen for a large safety margin by taking a number of rounds in Keccak-f that is almost the double of what we estimate to be sufficient for the absence of shortcut attacks more efficient than generic attacks.

In the published cryptanalysis of concrete primitives we observe that most primitives offer a much higher resistance against attacks in keyed modes than in unkeyed modes. The typical examples of attacks on an unkeyed hash function are the generation of collisions and second pre-images. Attacking a keyed mode ranges from key retrieval attacks to distinguishers. Note that also determining a first pre-image can be seen as an attack on a keyed mode, where the pre-image is the key. Examples that illustrate this difference in resistance between keyed and unkeyed modes include:

- MD5 [24]: despite painstaking efforts there is little progress in MD5 pre-image generation [25] while MD5 collisions [26] have been generated that are practically exploitable
- Panama [15]: the Panama stream cipher is as yet unbroken while for the Panama hash function collisions can be generated instantaneously [23,13]
- Keccak: In the Keccak crunchy crypto contest [9] collision challenges have been broken up to 4 rounds while pre-image challenges have been broken only up to 2 rounds.

Clearly, the situations for an adversary attacking an un-keyed instance of a sponge function (e.g., used for collision-resistant hashing) and that of attacking a keyed instance

are very different. In a keyed instance, after the key has been absorbed, the inner c bits of the state are unknown to the attacker. In sponge-based MAC generation, during the absorbing phase even the complete state is unknown to the attacker.

In the remainder of this paper we explore how the safety margin in KECCAK can be relaxed for keyed applications in the light of these facts. We compare the performance of these instances with KECCAK[], the KECCAK instance with default parameters $r = 1024$ and $c = 576$ and KECCAK[r=40, c=160]. Of course the same exercise can be conducted for Quark, Photon, Spongent or in general any iterated permutation used in a keyed sponge or duplex mode.

In this section we limit ourselves to applying the standard sponge and duplex constructions to round-reduced versions of KECCAK-f. Note that such round-reduced versions are covered by the KECCAK specifications in [10] and the KECCAK reference code [11]. To avoid confusion with KECCAK instances in which KECCAK-f has the nominal number of rounds, we denote these primitives by the name KECCUP.

Definition 1. KECCUP-$f[b, n]$ *is a family of permutations parameterized by its width* b *and its number of rounds* n*, where* KECCUP-$f[b, n]$ *is identical to* KECCAK-$f[b]$ *reduced to* n *rounds [10]. Similarly,* KECCUP$[r, c, n]$ *is a sponge function or duplex object using* KECCUP-$f[r + c, n]$.

As we do not modify the sponge or duplex constructions but just apply them to round-reduced instances of KECCAK-f, the proven security bounds with respect to generic attacks still apply. However, it may well be that this reduction of the number of rounds leads to specific attacks breaking the security claims.

We do the exercise for two KECCUP-f permutation widths: 1600 and 200.

3.1 Using KECCUP-$f[1600, n]$

We target two security strength levels: 128 and 256. We will assume that the length k of the secret keys corresponds to the security strength.

Consider a duplex instance with a target security strength of $k = 128$ bits. Assuming the multiplicity is bound by $M = 2^a \leq 2^{64}$ results in a capacity of $c = k + a = 192$ bits, leaving 1408 bits of rate. From our experiments related to trail search [10,14], we think that the minimum weights for differential or linear trails over four rounds of KECCAK-$f[1600]$ is higher than 64. On the other hand, for higher-order differential cryptanalysis, cube attacks and interpolation attacks, the algebraic degree of KECCAK-f increases only by a factor two each round. If we settle for a degree of 1024, 10 rounds is the choice. Using KECCUP$[r = 1408, c = 192, n = 10]$ would give a speed-up with respect to KECCAK[] of $24/10 \times 1408/1024 \approx 3.3$.

Targeting a security strength of $k = 256$ bits, a similar reasoning leads to a capacity of $c = k + a = 320$ bits and 11 rounds, so KECCUP$[r = 1280, c = 320, n = 11]$. Here the speed-up with respect to KECCAK[] becomes $24/11 \times 1280/1024 \approx 2.7$.

3.2 Using KECCUP-$f[200, n]$

Thanks to their small state size, sponge and duplex instances based on KECCAK-$f[200]$ are well suited for use in resource-constrained environments. Its 25-byte state is actually smaller than AES-128, with its 16-byte data path and 16-byte round keys. On the downside, this small state limits the achievable security strength. We will address security strength levels 80 and 128 and compare their performance with KECCAK$[r = 40, c = 160]$, the instance with capacity 160 bits based on the KECCAK-$f[200]$ permutation.

A target security strength of $k = 80$ bits combined with $M \leq 2^{64}$ gives a capacity of $c = k + a = 144$ bits and a rate of 56 bits. This relatively small rate value severely limits degrees of freedom in differential attacks. As in the case of KECCUP-$f[1600, n]$, our choice of the number of rounds is mostly inspired by possible weaknesses due to the limited algebraic degree. We think 9 rounds is on the safe side, so we propose KECCUP$[r = 56, c = 144, n = 9]$. Compared to KECCAK$[r = 40, c = 160]$, the speedup is $18/9 \times 56/40 \approx 2.8$.

A target security strength of $k = 128$ bits leads to a capacity of 192 bits and a rate of only 8 bits. Thanks to the very small value of this rate, attackers engaged in cube and similar attacks are expected to be forced applying multiple blocks. This relaxes the constraints on the number of rounds somewhat. On the other hand, clearly there must be some number of rounds between the injection of differences and the appearance of their effect at the output. Here we estimate that 6 rounds would be sufficient, so we propose KECCUP$[r = 8, c = 192, n = 6]$. Still, the security strength of 128 bits comes at a high cost. Compared to KECCAK$[r = 40, c = 160]$, the loss of speed is $18/6 \times 8/40 = 0.6$.

4 Variants

In this section we present two variants of the sponge and duplex constructions that do not comply to the standard definitions. In defining these variants we take the liberty of varying the rate and the number of rounds of the underlying permutation across the different phases. We are aware that variants and generalizations to the standard sponge definition have already been proposed. For example, in [20] it was observed that taking different rate/capacity pairs for the absorbing and squeezing phase can also have an effect on the security strength. Other variants are proposed as part of the Parazoa generalization of sponge functions [1], although not all are based on a permutation.

4.1 The donkeySponge construction

This mode is inspired by the Alred construction for MAC functions [16] and its instance Pelican-MAC [17,18]. The Alred construction allows building efficient MAC functions from block ciphers. Pelican-MAC is an instantiation based on AES, that is for long messages about 2.5 times faster than an AES-based CBC-MAC. Pelican-MAC takes as input a MAC key and a message and operates in three phases. In a first phase a secret state is initialized by applying AES to a 16-byte all-zero string with the MAC key as key. Then 16-byte message blocks are XORed into the secret state, interleaved by a permutation consisting of 4 unkeyed AES rounds. After applying all message blocks, the tag is obtained by applying AES to the secret state, again with the MAC key as key and (possibly) truncating the result. For the security of the Alred construction it is crucial that an adversary shall not be able to reconstruct the secret state or to generate inner collisions (pairs of partial messages leading to the same state).

Clearly the second phase resembles the absorbing phase of a sponge, but with rate equal to the width. In fact generalizing the Alred construction to support a b-bit permutation rather than a block cipher does not require much imagination. We did the exercise and call the result donkeySponge. We can summarize it as follows:

- The function takes as input a MAC key and a message and it returns a tag.
- The b-bit state is initialized with the MAC key and subject to n_{init} rounds of the permutation resulting in the secret state. The number of rounds n_{init} must be chosen such that all bits of the secret state depend on the MAC key.

- The b-bit blocks of the message are XORed into the secret state, interleaved with n_{absorb}-round permutations. The number n_{absorb} must be chosen to make the success probability of generating inner collisions negligible.
- The tag is obtained by applying a n_{squeeze}-round permutation to the secret state and truncating the result to ℓ bits. The number of rounds n_{squeeze} shall be high enough to prevent an adversary in reconstructing the inner state from outputs observed for chosen inputs. The number $b - \ell$ must be large enough to prevent state reconstruction by exhaustive search, namely, $b - \ell \geq k$.

Note that during the absorbing phase the rate becomes equal to the permutation width. This reduces the capacity to zero and precludes applying the proven generic security strength bound of the sponge construction. Alred has a provable reduction of retrieval of the MAC to the breaking of the block cipher. In donkeySponge the key can be readily computed from the secret state, so this is a security feature that donkeySponge does not have. On the other hand, Alred shares with donkeySponge that retrieval of the secret state is catastrophic for the security. In Alred, reconstruction of this state requires generating inner collisions or breaking the underlying block cipher. In donkeySponge, the former depends on the differential probability (DP) of differentials over n_{absorb} rounds. For the latter, breaking the underlying block cipher is replaced by successfully applying a chosen-input-difference attack on a truncated permutation with unknown input, whose success depends on the DP of differentials over n_{squeeze} rounds.

From an efficiency point of view, Alred requires working memory for the data path and the key schedule while donkeySponge only has a data path. For a given amount of working memory available, in donkeySponge message differences diffuse over the full memory, while in Alred the propagation of these differences is confined to the data path part. Hence in principle donkeySponge makes better use of available resources.

The donkeySponge construction is illustrated in Figure 3 and a formal specification is given in Algorithm 1. It has the following parameters:

- f: permutation family parameterized by the number of rounds, where the n-round member is denoted by $f[n]$;
- n_{init}: number of rounds applied after the key has been put in the state;
- n_{absorb}: number of rounds in between absorbing of message blocks;
- n_{squeeze}: number of rounds before squeezing of the tag (and in between squeeze operations if requested tag length exceeds rate);
- r: rate during squeezing.

We have done the exercise to see what parameter values would be reasonable for a Keccup-based instances of this and the result is the following. For instances calling Keccup-$f[1600, n]$ with security strength 128 and 256 and calling Keccup-$f[200, n]$ with security strength 128 and 80, we would propose the following values:

- $n_{\text{init}} = 3$: after three rounds each state bit depends on some key bit, even for keys with very weak entropy.
- $n_{\text{absorb}} = 6$: from our differential propagation experiments [10,14] we believe that there are no 6-round trails with weight below 128 for Keccak-$f[200]$ and below 256 for Keccak-$f[1600]$.
- $n_{\text{squeeze}} = 12$: we take it as the double of $n_{\text{absorb}} = 6$ to accomodate for strength against variants such as higher-order differential attacks.

For Keccak-$f[200]$ with target security strength 128 the length of the tag is limited to 72 bits. If a longer tag is desired, additional squeezing steps must be performed. For

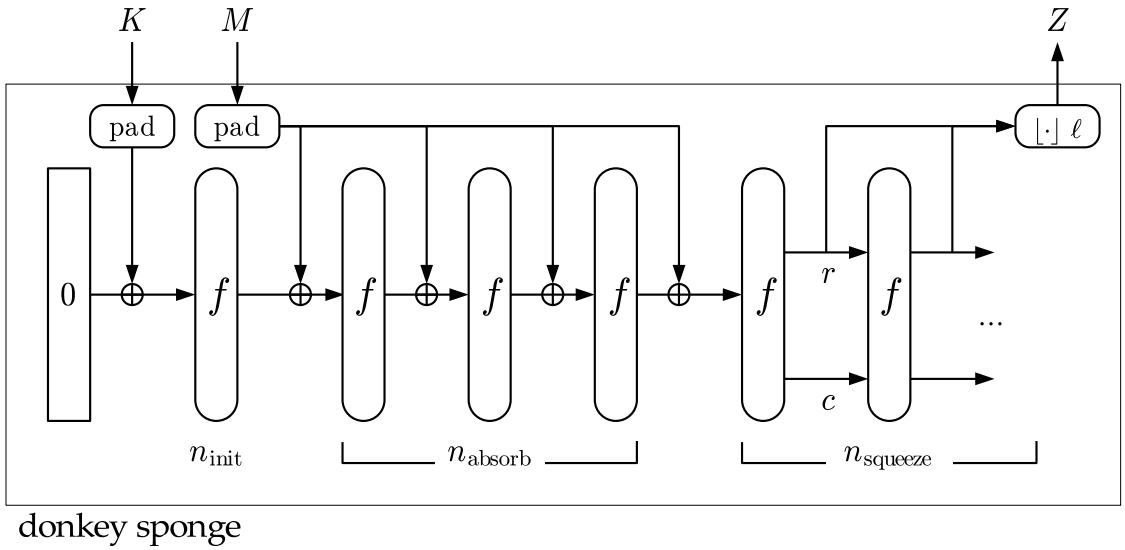


Fig. 3. The donkey sponge construction

long messages the KECCUP-$f[1600, n]$-based variants are a factor $24/6 \times 1600/1024 = 6.25$ faster than KECCAK[] and the KECCUP-$f[200, n]$-based variants are a factor $18/6 \times 200/40 = 15$ faster than KECCAK$[r = 40, c = 160]$. The fixed cost in the computation of a MAC is executing 15 rounds of KECCUP-f.

4.2 The monkeyDuplex construction

The authenticated encryption with associated data (AEAD) mode SPONGEWRAP based on the duplex construction [7] only guarantees confidentiality if for the same key and different messages the associated data is unique. In other words, the associated data should behave as a nonce. Violating this results in the encryption of different plaintexts with the same keystream. However, it does not jeopardize the key.

In this section we propose a variant of the duplex construction called monkeyDuplex whose security requires the uniqueness of a nonce. This makes this mode more fragile and we are aware that nonce uniqueness may not be imposed in all applications. However, if it is possible, it results in a considerable security gain.

The principles underlying monkeyDuplex are the following:

- It is an object that upon creation is loaded with a MAC key and a nonce. Once created, one can make duplexing calls to it, providing it with an input string σ and requesting an output string Z. The output string depends on all previous input strings, the key and the nonce.
- Upon creation, the b-bit state is initialized with the concatenation of the key and the nonce and subject to n_{init} rounds. Informally speaking, the number of rounds n_{init} must be chosen such that an active attacker has no advantage in combining outputs of duplex objects loaded with different nonces. In other words, if the differential properties of $f[n_{\text{init}}]$ are strong enough, state values of monkeyDuplex objects with different nonce values can be considered independent. If so, state recovery by an attacker with access to multiple objects does not give an advantage over state recovery from a single object.

Algorithm 1 donkeySponge$[f, n_{\text{init}}, n_{\text{absorb}}, n_{\text{squeeze}}, r]$

Require: $r < b$

Interface: $Z = \text{donkeySponge}(K, M, \ell)$ with K, M and Z bitstrings, ℓ an integer, $|K| < b$ and $|Z| = \ell$
$s = K||\text{pad10}^*[b](|K|)$
$s = f[n_{\text{init}}](s)$

$P = M||\text{pad10}^*[b](|M|)$
for $i = 0$ to $|P|_b - 2$ **do**
  $s = s \oplus P_i$
  $s = f[n_{\text{absorb}}](s)$
end for
$s = s \oplus P_{|P|_b - 1}$

$Z =$ empty string
while $|Z| < \ell$ **do**
  $s = f[n_{\text{squeeze}}](s)$
  $Z = Z||\lfloor s \rfloor_r$
end while
return $\lfloor T \rfloor_\ell$

– The duplexing calls are qualitatively the same as in the duplex construction. However, the number of rounds of the permutation, n_{duplex}, can be reduced significantly as compared to the plain duplex construction. We rely on the initialization phase and its nonce to make differential attacks infeasible: an attacker has no control whatsoever over state differences between pairs of monkeyDuplex objects. This limits his attack path to state reconstruction.

The monkeyDuplex construction is illustrated in Figure 4 and a formal specification is given in Algorithm 2. It has the following parameters:

– f: permutation family parameterized by the number of rounds, where the n-round member is denoted by $f[n]$;
– ℓ_{key}: length of the key;
– ℓ_{nonce}: length of the nonce;
– n_{init}: number of rounds applied after the key and nonce have been put in the state;
– n_{duplex}: number of rounds in a duplex call;
– r: rate during duplexing.

The difficulty of state recovery grows with increasing values of n_{duplex} and decreasing values of r, while the efficiency is determined by the ratio r/n_{duplex}. For achieving a given efficieny one can vary these two parameters where increasing the rate r necessitates increasing n_{duplex} and vice versa. This results in a spectrum of possible choices with at the two ends two particular approaches:

– Blockwise: $r \geq b/2$. the problem of state recovery consists in solving a CICO problem for the permutation $f[n_{\text{duplex}}]$. The choice of n_{duplex} is based on estimating the difficulty of solving this problem.
– Streaming: $n_{\text{duplex}} = 1$. The problem of state recovery consists in determining the state from the knowledge of a small part of the state for a number n_{unicity} of subsequent rounds. The difficulty of solving this problem depends on n_{unicity} and the nature of the round function. We have $n_{\text{unicity}} = \lfloor b/r \rfloor$. The value of r can hence be derived from the minimum value of n_{unicity} for which the state recovery problem is estimated to have expected workload above 2^k.

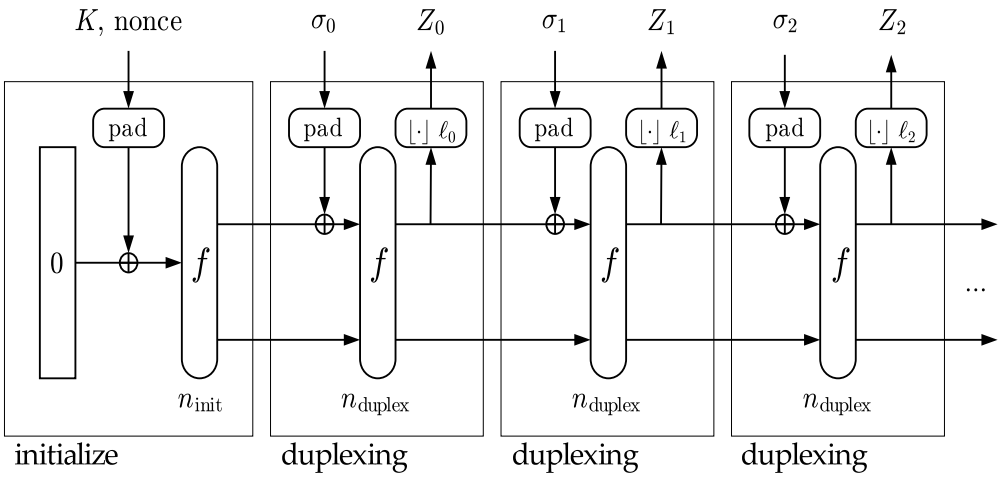


Fig. 4. The monkey duplex construction

Algorithm 2 monkeyDuplex$[f, \ell_{\text{key}}, \ell_{\text{nonce}}, n_{\text{init}}, n_{\text{duplex}}, r]$

Require: $r < b$
Require: $\ell_{\text{key}} + \ell_{\text{nonce}} \leq b - 1$

Interface: D.initialize(K, nonce) with $|K| = \ell_{\text{key}}$ and $|\text{nonce}| = \ell_{\text{nonce}}$
$s = K||\text{nonce}||\text{pad10}^*[b](\ell_{\text{key}} + \ell_{\text{nonce}})$
$s = f[n_{\text{init}}](s)$

Interface: $Z = D$.duplexing(σ, ℓ) with Z and σ bitstrings, integer ℓ satisfying $0 \leq \ell \leq r$ and $|\sigma| \leq r - 2$ and $|Z| = \ell$
$P = \sigma||\text{pad10}^*1[r](|\sigma|)||0^{b-r}$
$s = s \oplus P$
$s = f[n_{\text{duplex}}](s)$
return $\lfloor s \rceil_\ell$

We have done the exercise to see what parameter values would be reasonable for a Keccup-based instances. We denote these instances by the name Ketje. The result is the following. For all instances we propose $n_{\text{init}} = 12$: this is motivated by the same arguments as the choice of n_{squeeze} in donkeySponge. In the choice of r and n_{duplex} we have blockwise and streaming instances.

For the streaming instances ($n_{\text{duplex}} = 1$):

- $b = 1600$, security strength 256: $r = 128$ yielding $n_{\text{unicity}} = 12$. This is a factor $24 \times 128/1024 = 3$ faster than Keccak[].
- $b = 200$, security strength 80: $r = 16$ yielding $n_{\text{unicity}} = 12$. This is a factor $18 \times 16/40 = 7.2$ faster than Keccak$[r = 40, c = 160]$.

We only propose a blockwise instance for $b = 1600$. A value of $b = 200$ does not allow taking $r > b/2$ and at the same time supporting a security strength of 80 and a multiplicity of 2^{64}. We have:

- $b = 1600$, security strength 256: $r = 1280$ and $n_{\text{duplex}} = 8$. This is a factor $24/8 \times 1280/1024 = 3.75$ faster than Keccak[].

The monkeyDuplex construction can be used in different modes. The two most important ones are:

– keystream generation, where all duplexing calls are blank, i.e. with σ = empty string;
– authenticated encryption, similar to SpongeWrap [7].

5 Conclusions

We have discussed ways to speed up keyed modes based of on permutations. These are based on the following observations:

– Constructions can reach a higher security strength level with respect to generic attack in keyed modes than in unkeyed modes.
– Concrete primitives reach a higher security strength against attacks in keyed used cases than in un-keyed use cases.

We have proposed two new constructions that provide a speed advantage over the standard sponge and duplex constructions at the expense of a reduction of generality. The donkeySponge construction is basically a keyed sponge dedicated to MAC computation. The monkeyDuplex construction is a keyed duplex construction that depends on nonces for its cryptographic security and can be used for efficient keystream generation and authenticated encryption. We have illustrated the potential of our proposals with instances based on reduced-round versions of Keccak-f[1600] and Keccak-f[200], but they can be applied to any iterated permutation.

References

1. E. Andreeva, B. Mennink, and B. Preneel, *The parazoa family: generalizing the sponge hash functions*, Int. J. Inf. Sec. **11** (2012), no. 3, 149–165.
2. J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, *Quark: A lightweight hash*, in Mangard and Standaert [21], pp. 1–15.
3. D. Bernstein, *The salsa20 family of stream ciphers*, The eSTREAM Finalists (M. Robshaw and O. Billet, eds.), Lecture Notes in Computer Science, vol. 4986, Springer, 2008, pp. 84–97.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *On the indifferentiability of the sponge construction*, Advances in Cryptology – Eurocrypt 2008 (N. P. Smart, ed.), Lecture Notes in Computer Science, vol. 4965, Springer, 2008, `http://sponge.noekeon.org/`, pp. 181–197.
5. ———, *Sponge-based pseudo-random number generators*, in Mangard and Standaert [21], pp. 33–47.
6. ———, *Cryptographic sponge functions*, January 2011, `http://sponge.noekeon.org/`.
7. ———, *Duplexing the sponge: single-pass authenticated encryption and other applications*, Selected Areas in Cryptography (SAC), 2011.
8. ———, *On the security of the keyed sponge construction*, Symmetric Key Encryption Workshop (SKEW), February 2011.
9. ———, Keccak *crunchy crypto collision and pre-image contest*, 2011, `http://keccak.noekeon.org/crunchy_contest.html`.
10. ———, *The* Keccak *reference*, January 2011, `http://keccak.noekeon.org/`.
11. ———, *Reference and optimized implementations of* Keccak, 2012, `http://keccak.noekeon.org/`.
12. A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varıcı, and I. Verbauwhede, *SPONGENT: A lightweight hash function*, CHES (B. Preneel and T. Takagi, eds.), Lecture Notes in Computer Science, vol. 6917, Springer, 2011, pp. 312–325.
13. J. Daemen and G. Van Assche, *Producing collisions for PANAMA, instantaneously*, Fast Software Encryption 2007 (A. Biryukov, ed.), LNCS, Springer-Verlag, 2007, pp. 1–18.
14. ———, *Differential propagation analysis of* Keccak, Fast Software Encryption 2012, 2012.
15. J. Daemen and C. S. K. Clapp, *Fast hashing and stream encryption with PANAMA*, Fast Software Encryption 1998 (S. Vaudenay, ed.), LNCS, no. 1372, Springer-Verlag, 1998, pp. 60–74.
16. J. Daemen and V. Rijmen, *A new MAC construction ALRED and a specific instance ALPHA-MAC*, Fast Software Encryption (H. Gilbert and H. Handschuh, eds.), Lecture Notes in Computer Science, vol. 3557, Springer, 2005, pp. 1–17.
17. ———, *The Pelican MAC function*, IACR Cryptology ePrint Archive **2005** (2005), 8.
18. ———, *Refinements of the ALRED construction and MAC security claims*, IET information security **4** (2010), 149–157.

19. P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen, *Grøstl – a SHA-3 candidate*, Submission to NIST (round 3), 2011.
20. J. Guo, T. Peyrin, and A. Poschmann, *The PHOTON family of lightweight hash functions*, Crypto (P. Rogaway and R. Safavi-Naini, eds.), Lecture Notes in Computer Science, vol. 6841, Springer, 2011, pp. 222–239.
21. S. Mangard and F.-X. Standaert (eds.), *Cryptographic hardware and embedded systems, CHES 2010, 12th international workshop, Santa Barbara, CA, USA, August 17-20, 2010*, Lecture Notes in Computer Science, vol. 6225, Springer, 2010.
22. NIST, *NIST special publication 800-57, recommendation for key management (revised)*, March 2007.
23. V. Rijmen, B. Van Rompay, B. Preneel, and J. Vandewalle, *Producing collisions for PANAMA*, Fast Software Encryption 2001 (M. Matsui, ed.), LNCS, no. 2355, Springer-Verlag, 2002, pp. 37–51.
24. R. Rivest, *The MD5 message-digest algorithm*, Internet Request for Comments, RFC 1321, April 1992.
25. Y. Sasaki and K. Aoki, *Finding preimages in full md5 faster than exhaustive search*, EUROCRYPT (A. Joux, ed.), Lecture Notes in Computer Science, vol. 5479, Springer, 2009, pp. 134–152.
26. M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. Osvik, and B. de Weger, *Short chosen-prefix collisions for md5 and the creation of a rogue ca certificate*, CRYPTO (S. Halevi, ed.), Lecture Notes in Computer Science, vol. 5677, Springer, 2009, pp. 55–69.
27. H. Wu, *The hash function JH*, Submission to NIST (round 3), 2011.