Hash Functions and RFID Tags: Mind the Gap

Andrey Bogdanov, Gregor Leander, Christof Paar, Axel Poschmann,
Matt J.B. Robshaw, and Yannick Seurin

[1] Horst Görtz Institute for IT Security, Ruhr-University Bochum, Germany
[2] Orange Labs, Issy les Moulineaux, France
leander@rub.de, {abogdanov,cpaar,poschmann}@crypto.rub.de,
{matt.robshaw,yannick.seurin}@orange-ftgroup.com

Abstract. The security challenges posed by RFID-tag deployments are well-known. In response there is a rich literature on new cryptographic protocols and an on-tag hash function is often assumed by protocol designers. Yet cheap tags pose severe implementation challenges and it is far from clear that a suitable hash function even exists. In this paper we consider the options available, including constructions based around compact block ciphers. While we describe the most compact hash functions available today, our work serves to highlight the difficulties in designing lightweight hash functions and (echoing [17]) we urge caution when routinely appealing to a hash function in an RFID-tag protocol.

1 Introduction

With RFID tags on consumer items, the potential for wired-homes, and large-scale sensor networks becoming a reality, we are on the threshold of a pervasive computing environment. But along with these new applications come new, and demanding, security challenges. The cryptographic research community has been quick to identify some of the issues, and device authentication and privacy have received considerable attention. As a result a variety of new protocols have been proposed and in many of them, particularly ones intended to preserve user privacy and to anonymize tag interactions, it is assumed that a cryptographic hash function will be used on the tag.

However which hash function might be used in practice is rarely identified. Looking at dedicated hash functions from the last 20 years, we have become used to their impressive hashing speed (though this is a view that we might have to change in the future). This fast throughput might lead some designers to believe that hash functions are "efficient" in other regards and that they can be routinely used in low-cost environments. This is a mistake, a point that was convincingly made in a paper by Feldhofer and Rechberger [17]. Generally speaking, current hash functions are not at all suitable for constrained environments. They require significant amounts of state and the operations in current dedicated designs are not hardware friendly. This is not surprising since modern hash functions were designed with 32-bit processors in mind, but it means that very few RFID-oriented protocols appealing to a hash function could ever be used on a modestly-capable tag.

In this paper we consider RFID tag-enabled applications and the use of hash functions in RFID protocols. We then turn our attention to the design of hash functions in

E. Oswald and P. Rohatgi (Eds.): CHES 2008, LNCS 5154, pp. 283–299, 2008.

Table 1. An overview of the performance of some current compact algorithms where block ciphers are ordered by block and key size while hash functions are ordered by the size of the output

	Key	Block	Cycles per	Throughput at	Logic	Area	
	size	size	block	100KHz (Kbps)	process	GE	rel.
Block ciphers							
PRESENT-80 [6]	80	64	32	200	0.18μm	1 570	1
PRESENT-80 [7]	80	64	563	11.4	0.18μm	1 075	0.68
DES [42]	56	64	144	44.4	0.18μm	2 309	1.47
mCrypton [32]	96	64	13	492.3	0.13μm	2 681	1.71
PRESENT-128 [6]	128	64	32	200	0.18μm	1 886	1.20
TEA [54]	128	64	64	100	0.18μm	2 355	1.50
HIGHT [24]	128	64	34	188.2	0.25μm	3 048	1.65
DESXL [42]	184	64	144	44.4	0.18μm	2 168	1.38
AES-128 [16]	128	128	1 032	12.4	0.35μm	3 400	2.17
Stream ciphers							
Grain [15]	80	1	1	100	0.13μm	1 294	0.82
Trivium [15]	80	1	1	100	0.13μm	2 599	1.66
Hash functions							
	Hash output		Cycles per	Throughput at	Logic	Area	
	size		block	100KHz (Kbps)	process	GE	rel.
MD4 [17]	128		456	112.28	0.13μm	7 350	4.68
MD5 [17]	128		612	83.66	0.13μm	8 400	5.35
SHA-1 [17]	160		1 274	40.18	0.35μm	8 120	5.17
SHA-256 [17]	256		1 128	45.39	0.35μm	10 868	6.92
MAME [53]	256		96	266.67	0.18μm	8 100	5.16

Section 3 and we explore whether a block cipher makes an appropriate starting point for a compact hash function instead of a dedicated design.[1]

In Section 4 we instantiate lightweight hash functions using literature-based constructions and the compact block cipher PRESENT [6]. This allows us to implement a range of representative constructions that, for their given parameter sets, are the most compact hash functions available today. In Section 5 we then look at some challenging problems in designing hash functions with greater hash output lengths. While the paper reveals positive results, our work also serves to highlight the difficult issue of compact hash functions; we therefore close the paper with problems for future research.

2 Cryptography and RFID Tags

When considering applications based around the deployment of RFID tags we are working with very specific applications with rather unique requirements. The difficulty of implementing cryptography in such environments has spurred considerable research, and this can be roughly divided into two approaches:

[1] This relates to proposals for future work identified in [17].

1. Devise new algorithms and protocols for tag-based applications. Such new protocols might use new cryptographic problems or might be based almost exclusively on very lightweight operations, *e.g.* bitwise exclusive-or and vector inner-products.
2. Optimise existing algorithms and protocols so that they become suitable for RFID tag-based applications. This approach might not give the compact results that some of the more exotic proposals do, but the security foundations may be more stable.

The work in this paper is more in-line with the second approach—seeing what we can do with what we have—though we hope it will be helpful to protocol designers. The current state of the art for new compact block ciphers and stream ciphers is summarised in Table 1 where most compact proposals offering 80-bit security seem to require 1 300–1 600 *gate equivalents* (GE). For hash functions things are more complicated.

2.1 Hash Functions and Protocols for RFID Tags

Informally, a cryptographic hash function H takes an input of variable size and returns a hash value of fixed length while satisfying the properties of preimage resistance, second preimage resistance, and collision resistance [33]. For a hash function with n-bit output, compromising these should require 2^n, 2^n, and $2^{n/2}$ operations respectively. These properties make hash functions very appealing in a range of protocols. For tag-based applications, the protocols in question are often focused on authentication or on providing some form of anonymity and/or privacy [1,2,13,18,20,31,39]. However some estimates suggest that no more than 2 000 GE are available for security in low-cost RFID tags [26] and a glance at Table 1 shows that the hash functions available are unsuitable in practice. When we consider what we need from a hash function in an RFID tag-based application the following issues can be identified:

1. In tag-based applications we are unlikely to hash large amounts of data. Most tag protocols require that the hash function process a challenge, an identifier, and/or perhaps a counter. The typical input is usually much less than 256 bits.
2. In many tag-based applications we do not need the property of *collision resistance*. Most often the security of the protocol depends on the *one-way* property. In certain situations, therefore, it is safe to use hash functions with smaller hash outputs.
3. Applications will (typically) only require moderate security levels. Consequently 80-bit security, or even less, may be adequate. This is also the position taken in the eSTREAM project [14]. An algorithm should be chosen according to the relevant security level and in deployment, where success depends on every dollar and cent spent, there is no point using extra space to get a 256-bit security level if 64-bit security is all that is required.
4. While the physical space for an implementation is often the primary consideration, the peak and average power consumption are also important. The time for a computation will matter if we consider how a tag interacts with higher-level communication and anti-collision protocols.
5. Some protocols use a hash function to build a *message authentication code* (MAC), often by appealing to the HMAC construction [38]. When used as a MAC a number of interesting issues arise such as choosing an appropriate key length and understanding whether keys will be changed, something that will almost certainly be

impossible in most tag-enabled applications. There might also be the possibility of side-channel analysis on the MAC. However such attacks will rarely be worthwhile for cheap tag-enabled applications and we do not consider this issue further.

Taking account of these considerations allows us to make some pragmatic choices. There will be applications that just require one-wayness and the application may only require 80-bit or 64-bit security. Note that this is the view adopted by Shamir in the proposal SQUASH for use in RFID tags [49]. For other applications we might like to see 80-bit security against collision attacks.

Since current hash functions of dedicated design are either too big or broken, we first consider hash functions that are built around block ciphers. In particular we use the compact block cipher PRESENT [6] as a building block and we consider the implementation of a range of hash functions offering 64-bit and 128-bit outputs using established techniques. We also consider hash functions that offer larger outputs and we highlight some design directions along with their potential hardware footprint.

3 Hash Function Constructions

Hash functions in use today are built around the use of a *compression function* and appeal to the theoretical foundations laid down by Merkle and Damgård [11,34]. The compression function h has a fixed-length input, consisting of a *chaining variable* and a message extract, and gives a fixed-length output. A variety of results [12,25,27] have helped provide a greater understanding of this construction and while there are some limitations there are some countermeasures [4]. Since our goal is to obtain *representative* performance estimates, we will not go into the details of hash function designs. Instead we will assume that our hash function uses a compression function in an appropriate way and that the compression function takes as input some words of chaining variable, represented by H_i, and some words of (formatted) message extract, represented by M_i. We then restrict our focus to the cost of implementing the compression function.

In the hash function literature it is common to distinguish between two popular ways of building a compression function. The first is to use a compression function of a dedicated design and the second is to use an established, and trusted, block cipher.

3.1 Dedicated Constructions

The separation of dedicated constructions from block cipher-based constructions tends to disguise the fact that even dedicated hash functions like SHA-1 [36] and MD5 [46] are themselves built around a block cipher. Remove the feed-forward from compression functions in the MD-family and we are left with a reversible component that can be used as a block cipher (such as SHACAL [19] in the case of SHA-1). However the underlying block cipher we are left with is rather strange and has a much larger-than-normal block and key size combination. The problem with dedicated hash functions is that recent attacks [51,52] have shown that there is much to learn in designing block ciphers with such strange parameter sizes. There is therefore some value in considering approaches that use a more “classical” block cipher as the basis for a compression function.

3.2 Block Cipher Constructions

The use of a block cipher as a building block in hash function design [10] is as old as DES [35]. The topic has been recently revisited and Black *et al.* [5] have built on the work of Preneel [44] to present a range of secure $2n$- to n-bit compression functions built around an n-bit block cipher that takes an n-bit key. Among these are the well-known *Davies-Meyer*, *Matyas-Meyer-Oseas*, and *Miyaguchi-Preneel* constructions.

A hash function with an output of n bits can only offer a security level of 2^n operations for pre-image and second pre-image attacks and $2^{n/2}$ operations against finding collisions. While a security level of 128 bits is typical for mainstream applications, 80-bit security is often a reasonable target for RFID tag-based applications. Either way, there is a problem since the hash functions we need cannot always be immediately constructed out of the block ciphers we have to hand. This is not a new problem. But it is not an easy one to solve either, and there has been mixed success in constructing $2n$-bit hash functions from an n-bit block cipher [8,10,28,29,30,43,45]. While limitations have been identified in many constructions, work by Hirose [21,22] has identified a family of double-block-length hash functions that possess a proof of security. These use block ciphers with a key length that is twice the block length. Such a property is shared by AES-256 [37] and PRESENT-128 [6] and so in Section 4.2 we consider the performance of an Hirose-style construction instantiated using PRESENT-128.

When it comes to providing a replacement for SHA-1, the parameter sizes involved provide a difficult challenge. If we are to use a 64-bit block cipher like PRESENT-128, then in arriving at a hash function with an output of at least 160 bits we need a construction that delivers an output three times the block size (thereby achieving a 192-bit hash function). There are no "classical" constructions for this and so Sections 5.1 and 5.2 illustrate two possible design directions. These give representative constructions and we use them to gauge the hardware requirements of different design approaches. We hope that this will be of interest to future hash function designers.

4 Compact Hashing for 64- and 128-Bit Hash Outputs

In this section we will consider a variety of approaches to compact hashing when we use the block cipher PRESENT [6] as a building block. PRESENT is a 64-bit SPN block cipher which can be used with either an 80-bit or a 128-bit key. These will be referred to as PRESENT-80 and PRESENT-128 and full details and design rationale for these ciphers can be found in [6]. We denote encrypting a message M under the key K with PRESENT-80 or PRESENT-128 to obtain the ciphertext C as $C = E(M,K)$ and $A||B$ denotes the concatenation of A and B.

4.1 Two Compact 64-Bit Proposals: DM-PRESENT-80 and DM-PRESENT-128

There are a variety of choices for building a 64-bit hash function from a 64-bit block cipher. We will illustrate these with the *Davies-Meyer* mode where a single 64-bit chaining variable H_i is updated using a message extract M_i according to the computation $H_i' = E(H_i, M) \oplus H_i$.

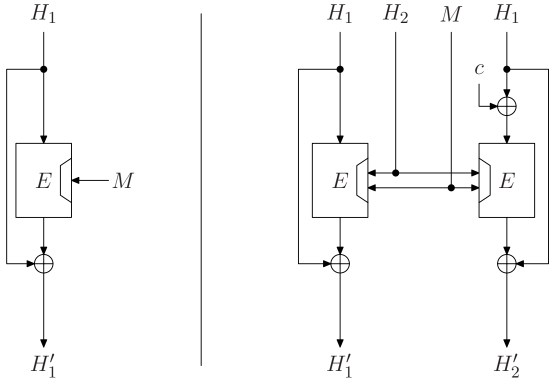


Fig. 1. Compression functions for the 64-bit and 128-bit hash functions: DM-PRESENT-80 and DM-PRESENT-128 (on the left) as well as H-PRESENT-128 (on the right)

In our case E denotes encryption with either PRESENT-80 or PRESENT-128, see Figure 1. Such hash functions will only be of use in applications that require the one-way property and 64-bit security.[2] At each iteration of the compression function 64 bits of chaining variable and 80 bits (*resp.* 128 bits) of message-related input are compressed. Therefore the two proposals DM-PRESENT-80 and DM-PRESENT-128 provide a simple trade-off between space and throughput. We also provide figures for a serial and parallel implementation of PRESENT, see Table 2.

While we have focused on using Davies-Meyer, it is important to note that these figures are a good indication of the cost for *any* single block-length hash function construction. If one prefers to implement *Matyas-Meyer-Oseas* or *Miyaguchi-Preneel* based on PRESENT (instead of *Davies-Meyer*) then the cost of DM-PRESENT-80 will be a reasonable guide. Moving away from PRESENT to a different block cipher will almost certainly cause an increase to the space required for an implementation.

4.2 A Compact 128-Bit Proposal: H-PRESENT-128

When designing a 128-bit hash function from the 64-bit output block cipher PRESENT, we have to appeal to so-called double-block-length hash function constructions. Natural candidates are MDC-2 [10] and Hirose's constructions [21,22]. These schemes possess security proofs in the ideal cipher model, where the underlying block cipher is modeled as a family of random permutations, one permutation being chosen independently for each key. However MDC-2 is not an ideal construction [50] and so we base our 128-bit hash function H-PRESENT-128 on the construction studied in [22].

The scheme H-PRESENT-128 is illustrated in Figure 1. The compression function takes as input two 64-bit chaining variables and a 64-bit message extract, denoted by the triple (H_1, H_2, M), and outputs the pair of updated chaining variables (H'_1, H'_2) according to the computation

[2] These properties are identical to those offered by the proposal SQUASH [49].

$$H_1' = E(H_1, H_2\|M) \oplus H_1 \quad \text{and} \quad H_2' = E(H_1 \oplus c, H_2\|M) \oplus H_1$$

where E denotes PRESENT-128 and c is a non-zero constant that needs to be fixed [9]. Thus the chaining variable $H_1\|H_2$ is 128 bits long and 64 bits of message-related input are hashed per iteration.

Hirose showed that, in the ideal cipher model, an adversary has to make at least 2^n queries to the cipher in order to find a collision with non-negligible advantage, where n is the block size of the cipher. It is possible to make the same kind of analysis for preimage resistance (see proof of Theorem 4 in [23]) and to show that any adversary has to make at least 2^{2n} queries to the cipher to find a preimage. As for Section 4.1 our implementation results are presented for both a parallel and serial implementation of PRESENT-128, see Table 2. These results should be viewed as indicative of the cost of a double-block-length construction using PRESENT. Since only one key schedule needs to be computed per iteration of the compression function, the Hirose construction is probably one of the most efficient constructions of this type, e.g. in the case of PRESENT around 1 000 GE can be saved in this way.

5 Compact Hashing for $\geq$ 160-Bit Hash Outputs

It is possible that some tag-enabled applications might need collision-resistance at a security level of 2^{80} operations. For this we need a hash output of 160 bits or greater. However this is where the problems really begin and we consider two directions.

For the first, we continue the approach of the paper so far and we consider building a hash function with a hash output greater than 160 bits from PRESENT *as is*. So in Section 5.1 we try to use PRESENT in this way and, using established results in the literature, we make a proposal. However, at the same time, we use the very same results to demonstrate that this approach is unlikely to be successful, a sentiment that is supported by our implementation results. Instead, for the second direction that is described in Section 5.2, we move towards a dedicated hash function though we keep elements of PRESENT in our constructions. Our dedicated proposals are deliberately simple and obvious, and in this way we aim to provide some first results on the impact different design choices might have in moving towards a new, compact, hash function.

5.1 Longer Hash Outputs Using PRESENT

Our goal is to build a hash function out of PRESENT that offers an output of at least 160 bits. Since PRESENT has a 64-bit block size, this means that we are forced to consider a triple-block-length construction and we will end up with a 192-bit hash function.

Unfortunately very few designs for l-block length hash function with $l \geq 3$ have been studied so far. However Peyrin *et al.* [41] have identified some necessary conditions for securely combining *compression functions* to obtain a new compression function with a longer output. We can use these results and so, in the case we consider here, our constituent compression functions will be based around PRESENT-128, *i.e.* we will use DM-PRESENT-128 as the building block.

More background to the construction framework is given in [41]. However, within this framework, efficiency demands that we keep to a minimum the number of compression functions that we need to use, where each compression function is instantiated by DM-PRESENT-128. For reasons of simplicity and greater design flexibility we restrict ourselves to processing only a single 64-bit message extract, and so our inputs to C-PRESENT-192, where we use C as shorthand for "constructed", consist of a quadruplet (H_1, H_2, H_3, M) while the output is a triplet (H_1', H_2', H_3'). The compression function C-PRESENT-192 is illustrated in Appendix I and the output is computed as

$$\begin{aligned}
H_1' &= f^{(1)}(H_3, H_1, H_2) \oplus f^{(3)}(H_3 \oplus M, H_1, H_2) \oplus f^{(5)}(H_2, H_3, M)\\
H_2' &= f^{(1)}(H_3, H_1, H_2) \oplus f^{(4)}(H_1, H_3, M) \oplus f^{(6)}(H_1 \oplus H_2, H_3, M)\\
H_3' &= f^{(2)}(M, H_1, H_2) \oplus f^{(4)}(H_1, H_3, M) \oplus f^{(5)}(H_2, H_3, M) \ ,
\end{aligned}$$

with $f^{(i)}(A, B, C) = E(A \oplus c_{i-1}, B \| C) \oplus A$ for different constants c_i and E denotes encryption with PRESENT-128.

This construction might seem too complicated, but this is exactly the point we wish to make. The particular set of parameter values that are forced upon us when trying to build a large-output hash function from a small block cipher means that there will be *no* simple construction. More precisely, work in [41] shows that for *any* construction that uses a compression function with parameters equivalent to PRESENT-128 along with linear mixing layers to combine chaining variables and intermediate values, at least six compression functions are needed to resist all currently-known generic attacks. We *must* therefore use at least six independent calls to DM-PRESENT-128. The construction C-PRESENT-192 attains this minimum and more details of the construction are given in Appendix I. Our implementation results and estimates are given in Table 2, but the performance profile for C-PRESENT-192 suggests that building directly on PRESENT is unlikely to be a good way forward. Instead we consider some dedicated elements.

5.2 Dedicated Design Elements Inspired by PRESENT

Hash function design is notoriously difficult and so an interesting first step is to identify some general approaches and to understand their security and performance trade-offs. In this section we describe the results of some prototyping which tests a range of approaches and provides good background to our ongoing work. Our basic premise is to stay close to the design elements of PRESENT and to modify the design so as to give a block cipher with a much larger block size. We then adapt the key schedule in two alternative ways with the first being a natural proposal and the second having strong similarities to Whirlpool [3]. We give implementation results for both approaches.

Our schemes will continue to be based on the Davies-Meyer (DM) scheme $H_{i+1} = E(H_i, M_i) \oplus H_i$ though the form of our encryption function E will now change. In general, the encryption function E can be described as:

$$\begin{aligned}
E &: \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n \ ,\\
E &: \text{PLAINTEXT} \times \text{KEY} \mapsto \text{CIPHERTEXT}
\end{aligned}$$

The detailed description of PRESENT can be found in [6]. At a top-level we can write the r-round encryption of the plaintext STATE as:

```
for $i = 1$ to $r$ do
    $\text{STATE} \leftarrow \text{STATE} \oplus \text{eLayer}(\text{KEY}, i)$
    $\text{STATE} \leftarrow \text{sBoxLayer}(\text{STATE})$
    $\text{STATE} \leftarrow \text{pLayer}(\text{STATE})$
    $\text{KEY} \leftarrow \text{genLayer}(\text{KEY}, i)$
end for
$\text{STATE} \leftarrow \text{STATE} \oplus \text{eLayer}(\text{KEY}, r+1)$,
```

where eLayer describes how a subkey is combined with a cipher STATE, sBoxLayer and pLayer describe how the STATE evolves, and genLayer is used to describe the generation of the next subkey.

When used in the DM mode we recast the plaintext and ciphertext as hash function STATE and use the (formatted) message extract as the key. For ease of design we will choose the parameters k and n so that $k|n$ and $4|n$, and both our proposals will have the following (unmodified) structure:

```
for $i = 1$ to $r$ do
    $\text{STATE} \leftarrow \text{STATE} \oplus \text{eLayer}(\text{MESSAGE}, i)$
    $\text{STATE} \leftarrow \text{sBoxLayer}(\text{STATE})$
    $\text{STATE} \leftarrow \text{pLayer}(\text{STATE})$
    $\text{MESSAGE} \leftarrow \text{genLayer}(\text{MESSAGE}, i)$
end for
$\text{STATE} \leftarrow \text{STATE} \oplus \text{eLayer}(\text{MESSAGE}, r+1)$
```

The following building blocks are unchanged between the two proposals and are merely generalizations of the PRESENT structure to larger 160-bit block sizes.

1. sBoxLayer: This denotes use of the PRESENT 4-bit to 4-bit S-box S and it is applied $n/4$ times in parallel.
2. pLayer: This is an extension of the PRESENT bit-permutation and moves bit i of STATE to bit position $P(i)$, where

$$P(i) = \begin{cases} i \cdot n/4 \mod n-1, & \text{if } i \in \{0,\ldots,n-2\} \\ n-1, & \text{if } i = n-1. \end{cases}$$

It is in the specification of genLayer, which transforms the message of length k from round-to-round, and eLayer : $\mathbb{F}_2^k \rightarrow \mathbb{F}_2^n$, that describes how the message extract is combined with cipher state, that the two proposals differ.

PROP-1. For ease of comparison with PRESENT we keep exactly the same 80-bit key input and the same 80-bit key schedule. Thus we modify a 160-bit chaining variable using an 80-bit message input and, to make an implementation estimate, we use 64 rounds. This is equivalent to the parameters $n = 160$, $k = 80$, and $r = 64$. The sBoxLayer and pLayer are as above and eLayer and genLayer are described as follows:

1. $\text{eLayer}(\text{MESSAGE}, i) = \text{MESSAGE} || \text{genLayer}(\text{MESSAGE}, i)$
2. $\text{genLayer}(\text{MESSAGE}, i)$ is defined as the 80-bit key schedule of PRESENT. Thus, MESSAGE is rotated by 61 bit positions to the left, the left-most four bits are passed through the PRESENT S-box, and the round counter i is exclusive-ored with some bits of MESSAGE.

In words, we use the key schedule of PRESENT-80 exactly as is and at each round we use what would be two successive 80-bit round keys. At each round the key schedule is updated only once, so the same subkey is used once on the right-hand side and, in the following round, on the left-hand side.

PROP-2. For the second proposal, we consider a structure that has some similarity to Whirlpool. Our parameter set is $n = 160$ and $k = 160$ which allows us to use a longer message extract at each iteration of the compression function. For prototyping and implementation estimates we set $r = 80$. The building blocks eLayer and genLayer are specified as:

1. $\text{eLayer}(\text{MESSAGE}, i) = \text{MESSAGE}$
2. $\text{genLayer}(\text{MESSAGE}, i) = \text{pLayer}(\text{sBoxLayer}(\text{MESSAGE} \oplus i))$, being just a copy of the data path with round constant addition.

In words, we imagine that our message extract is a 160-bit key and we process the key in a key-schedule that is identical to the encryption process.

We estimated the hardware figures for different architectures when implementing PROP-1 and PROP-2. Our implementation estimates range from a 4-bit width data path (highly serialized) up to a 160-bit width data path which offers one round of processing in one cycle. Since PROP-2 uses a very similar key schedule (*i.e.* message path) and encryption routine, we can give a further two different implementation options: one with a shared sBoxLayer between the data path and the message path and one with an individual sBoxLayer. The results are summarized below with the efficiency *eff.* being measured in bps/GE.

Understanding the best trade-offs for the different approaches is not easy. As one can see, all three implementations scale nicely, though it seems that PROP-2 is more efficient in terms of throughput per area when compared to PROP-1. On the other hand PROP-1 offers a lower minimal achievable gate count, though at the cost of a higher cycle count. Much would also depend on a thorough security analysis of any final proposal and while some initial analysis in Appendix II suggests the possibility of optimizations to an approach like PROP-2, this is something to explore in future work during the design of an explicit proposal.

data path	PROP-1			PROP-2 (shared)			PROP-2 (ind.)		
width	area (GE)	cycles	*eff.*	area (GE)	cycles	*eff.*	area (GE)	cycles	*eff.*
4	2 520	5 282	1.2	3 010	6 481	0.82	3 020	3 281	1.62
16	2 800	1 322	4.33	3 310	1 621	2.92	3 380	821	5.77
32	3 170	662	7.64	3 730	811	5.11	3 860	411	10.09
80	4 270	266	14.09	4 960	325	9.29	5 300	165	18.3
160	4 830	134	24.73	5 730	163	15.29	6 420	83	30.03

6 Implementation of the Standard Constructions

To consider the efficiency of the standard constructions, we implemented two different architectures (round-based and serial) in VHDL and simulated using *Mentor*

Table 2. The performance of different hash functions based on the direct application of PRESENT. For comparison with our hash functions with 128-bit output we include estimates for the AES-based 128-bit hash function in Davies-Meyer mode. For comparison with MAME we include estimates for the 256-bit hash function built from the AES in Hirose's construction.

	Hash output size	Data path size	Cycles per block	Throughput at 100KHz (Kbps)	Efficiency (bps/GE)	Logic process	Area GE
MD4 [17]	128	32	456	112.28	15.3	0.13μm	7 350
MD5 [17]	128	32	612	83.66	10	0.13μm	8 400
SHA-1 [17]	160	32	1 274	40.19	4.9	0.35μm	8 120
SHA-256 [17]	256	32	1 128	45.39	4.2	0.35μm	10 868
MAME [53]	256	256	96	266.67	32.9	0.18μm	8 100
In this paper							
DM-PRESENT-80	64	64	33	242.42	109.5	0.18μm	2 213
DM-PRESENT-80	64	4	547	14.63	9.1	0.18μm	1 600
DM-PRESENT-128	64	128	33	387.88	153.3	0.18μm	2 530
DM-PRESENT-128	64	4	559	22.9	12.1	0.18μm	1 886
H-PRESENT-128	128	128	32	200	47	0.18μm	4 256
H-PRESENT-128	128	8	559	11.45	4.9	0.18μm	2 330
C-PRESENT-192	192	192	108	59.26	7.4	0.18μm	8 048
C-PRESENT-192	192	12	3 338	1.9	0.41	*estimate*	4 600
AES-based *DM scheme*	128	8	$> 1\,032$	< 12.4	< 2.8	*estimate*	$> 4\,400$
AES-based *Hirose scheme*	256	8	$> 1\,032$	< 12.4	< 1.3	*estimate*	$> 9\,800$

Graphics Modelsim SE PLUS 6.3a. *Synopsys DesignCompiler* version *Z-2007.03-SP5* was used to synthesize the design to the *Virtual Silicon* (VST) standard cell library *UMCL18G212T3*, which is based on the *UMC L180 0.18μm 1P6M* logic process and has a typical voltage of 1.8 Volt. For synthesis we advised the compiler to use a clock frequency of 100 KHz, a typical operating frequency for RFID applications.

We used *Synopsys Power Compiler* version *Z-2007.03-SP5* to estimate the power consumption of our implementations. At a clock frequency of 100 KHz DM-PRESENT-80 consumes 6.28 μW in the round-based implementation and 1.83 μW in the serialized implementation. The figures for the other designs are as follows: DM-PRESENT-128 7.49 μW and 2.94 μW, H-PRESENT-128 8.09 μW and 6.44 μW, and C-PRESENT-192 9.31 μW (round-based). Note that it is not easily possible to compare power consumption of designs implemented in different technologies, hence we did not include these figures in Table 2. However, the figures for SHA-256 (15.87 μW) and SHA-1 (10.68 μW) provided by Feldhofer et Rechberger [17] are in the same range as ours.

The area requirements of DM-PRESENT-80 and DM-PRESENT-128 comprise of the area requirements of the appropriate PRESENT cores and a 64-bit register to store the chaining variable (around 510 GE). Additionally, in the round-based variants a 64-bit XOR gate (170 GE) is required and in the serial variants a 4-bit XOR gate (10 GE).[3]

[3] Note that contrary to the constructions presented in this article the round-based PRESENT cores do not require a finite state machine nor do they contain clock-gated flip-flops. Therefore all in this paper presented constructions require additional logic which increases the area.

For the area estimates of the AES-based *Davies-Meyer* and *Hirose* schemes we used the smallest known (3 400 GE) AES implementation [16]. We estimated the area requirements for storing one bit to be 8 GE as stated in [16]. For the AES-based *Davies-Meyer scheme* we assumed that at least one additional register would be required to store the 128-bit value H_1 (1 024 additional GE), summing up to at least 4 400 GE in total.

The H-PRESENT-128 implementation consists of a modified PRESENT-128 core, a PRESENT data path (1 010 GE), a 64-bit register for the chaining variable (510 GE), and two 64-bit multiplexer (340 GE). Additionally, the round-based variant requires two 64-bit XOR gates (340 GE) and the serial variant two 4-bit XOR gates (20 GE). The AES-based *Hirose scheme* requires an AES implementation with 256-bit key length. However, no such low-cost implementation has been reported so far. Therefore we estimate the area requirements starting from the Feldhofer *et al.* [16] implementation with a 128-bit key. At least 128 additional key bits (1 024 GE) have to be stored to achieve an AES implementation with 256 bits key length, summing up to at least 4 400 GE. The *Hirose scheme* requires two instantiations of the block cipher and the storage of one intermediate value H_1, which has the same size as the block size. All together we estimate the AES-based *Hirose scheme* to require at least 9 800 GE. The serial variant of C-PRESENT-192 was not implemented, because the figures for the round-based variant and the estimations indicate large area requirements with more than 4 500 GE. In fact this large area requirement for both variants of C-PRESENT-192 was the main reason to look for other constructions such as PROP-1 and PROP-2.

Table 2 summarizes our results and compares them to other hashing functions and AES-based schemes. When the hash output length is 128 bits or lower, a construction based around PRESENT seems to have potential. Certainly they are far more competitive than current hash functions, the primary reason being that there exist efficient block cipher-based constructions for this size of hash output. Even a larger block cipher such as AES makes for a more compact hash function than current dedicated designs at this security level, though the throughput suffers.

7 Conclusions

While compact hash functions are often proposed in protocols for RFID tags, there are currently no sufficiently compact candidates to hand. Here we have explored the possibility of building a hash function out of a block cipher such as PRESENT. We have described hash functions that offer 64- and 128-bit outputs based on current design strategies. For their parameter sets these are the most compact hash function candidates available today. In particular, H-PRESENT-128 requires around 4 000 GE, which is similar to the best known AES implementation and about 50% smaller than the best reported MD5 implementation. At the same time, H-PRESENT-128 requires between 20–30 *times* fewer clock cycles than compact AES and MD5 implementations, giving it a major time-area advantage.

Obviously 128-bit hash functions are relevant for applications where a security-performance trade-off is warranted. To obtain larger hash outputs there are severe complications and we suspect that dedicated designs could be more appropriate. Clearly there are many areas of open research, not least the design of very compact hash functions. In parallel, it might also be worth revisiting tag-based protocols that use hash functions to see if the same goals can be achieved in a different way.

References

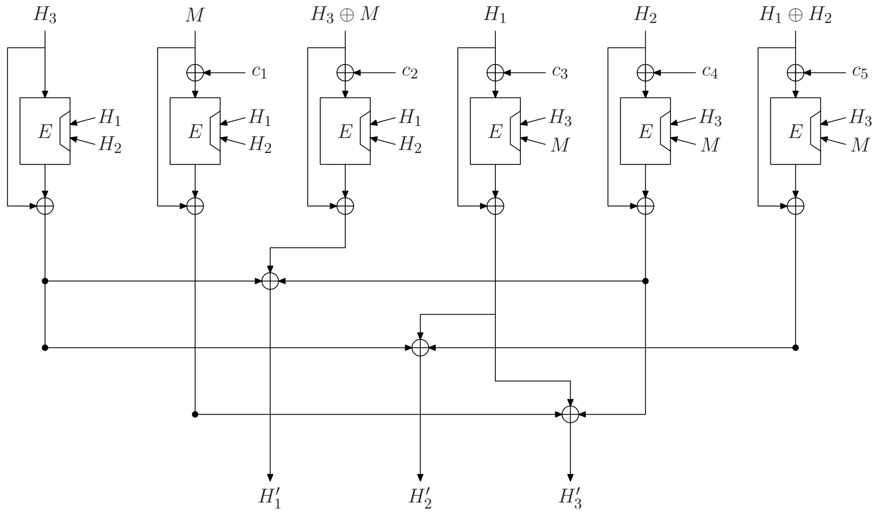
1. An, Y., Oh, S.: RFID System for User's Privacy Protection. In: IEEE Asia-Pacific Conference on Communications, pp. 16–519. IEEE Computer Society, Los Alamitos (2005)
2. Avoine, G., Oechslin, P.: A Scalable and Provably Secure Hash-based RFID Protocol. In: 3rd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops), pp. 110–114. IEEE Computer Society, Los Alamitos (2005)
3. Baretto, P., Rijmen, V.: The Whirlpool Hashing Function, `http://paginas.terra.com.br/informatica/-paulobarreto/WhirlpoolPage.html`
4. Biham, E., Dunkelman, O.: A Framework for Iterative Hash Functions - HAIFA. Presented at Second NIST Cryptographic Hash Workshop, August 24-25 (2006), `csrc.nist.gov/groups/ST/hash/`
5. Black, J., Rogaway, P., Shrimpton, T.: Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 320–335. Springer, Heidelberg (2002)
6. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: Present: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
7. Rolfes, C., Poschmann, A., Leander, G., Paar, C.: Ultra-Lightweight Implementations for Smart Devices - Security for 1000 Gate Equivalents. In: CARDIS 2008. Springer, Heidelberg (to appear, 2008)
8. Brown, L., Pieprzyk, J., Seberry, J.: LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications. In: Seberry, J., Pieprzyk, J.P. (eds.) AUSCRYPT 1990. LNCS, vol. 453, pp. 229–236. Springer, Heidelberg (1990)
9. Chang, D.: A Practical Limit of Security Proof in the Ideal Cipher Model: Possibility of Using the Constant As a Trapdoo. In: Several Double Block Length Hash Functions. IACR Cryptology ePrint Archive, Report 2006/481, `http://eprint.iacr.org/2006/481`
10. Coppersmith, D., Pilpel, S., Meyer, C.H., Matyas, S.M., Hyden, M.M., Oseas, J., Brachtl, B., Schilling, M.: Data authentication using modification detection codes based on a public one way encryption function. U.S. Patent No. 4,908,861 (March 13, 1990)
11. Damgård, I.: A Design Principle for Hash Functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, Heidelberg (1990)
12. Dean, R.D.: Formal Aspects of Mobile Code Security. PhD thesis, Princeton University (1999)
13. Dimitriou, T.: A Lightweight RFID Protocol to Protect Against Traceability and Cloning Attacks. In: Proceedings of IEEE International Conference on Security and Privacy of Emerging Areas in Communication Networks (SecureComm 2005), pp. 59–66. IEEE Computer Society, Los Alamitos (2005)
14. ECRYPT Network of Excellence. The Stream Cipher Project: eSTREAM, `http://www.ecrypt.eu.org/stream`
15. Good, T., Chelton, W., Benaissa, M.: Hardware Results for Selected Stream Cipher Candidates. In: SASC 2007 (February 2007), `http://www.ecrypt.eu.org/stream/`
16. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong Authentication for RFID Systems Using the AES Algorithm. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 357–370. Springer, Heidelberg (2004)

17. Feldhofer, M., Rechberger, C.: A Case Against Currently Used Hash Functions in RFID Protocols. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4277, pp. 372–381. Springer, Heidelberg (2006)
18. Gao, X., Xian, Z., Wang, H., Shen, J., Huang, J., Song, S.: An Approach to Security and Privacy of RFID System for Supply Chain. In: IEEE International Conference on E-Commerce Technology for Dynamic E-Business, pp. 164–168. IEEE Computer Society, Los Alamitos (2004)
19. Handschuh, H., Knudsen, L.R., Robshaw, M.J.B.: Analysis of SHA-1 in Encryption Mode. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 70–83. Springer, Heidelberg (2001)
20. Henrici, D., Götze, J., Müller, P.: A Hash-based Pseudonymization Infrastructure for RFID Systems. In: Georgiadis, P., Lopez, J., Gritzalis, S., Marias, G. (eds.) SecPerU 2006, pp. 22–27. IEEE Computer Society Press, Los Alamitos (2006)
21. Hirose, S.: Provably Secure Double-Block-Length Hash Functions in a Black-Box Model. In: Park, C.-s., Chee, S. (eds.) ICISC 2004. LNCS, vol. 3506, pp. 330–342. Springer, Heidelberg (2005)
22. Hirose, S.: Some Plausible Constructions of Double-Block-Length Hash Functions. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 210–225. Springer, Heidelberg (2006)
23. Hirose, S.: How to Construct Double-Block-Length Hash Functions. In: Second Cryptographic Hash Workshop, Santa Barbara (August 2006)
24. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.-S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A New Block Cipher Suitable for Low-Resource Device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
25. Joux, A.: Multi-Collisions in Iterated Hash Functions. Application to Cascaded Constructions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (2004)
26. Juels, A., Weis, S.A.: Authenticating Pervasive Devices With Human Protocols. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 198–293. Springer, Heidelberg (2005)
27. Kelsey, J., Schneier, B.: Second Preimages on n-bit Hash Functions for Much Less than 2^n Work. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 474–490. Springer, Heidelberg (2005)
28. Knudsen, L.R., Lai, X.: New Attacks on all Double Block Length Hash Functions of Hash Rate 1, Including the Parallel-DM. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 410–418. Springer, Heidelberg (1995)
29. Lai, X., Massey, J.L.: Hash Functions Based on Block Ciphers. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 55–70. Springer, Heidelberg (1993)
30. Lai, X., Waldvogel, C., Hohl, W., Meier, T.: Security of Iterated Hash Functions Based on Block Ciphers. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 379–390. Springer, Heidelberg (1994)
31. Lee, S., Hwang, Y., Lee, D., Lim, J.: Efficient Authentication for Low-Cost RFID Systems. In: Gervasi, O., Gavrilova, M.L., Kumar, V., Laganá, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K. (eds.) ICCSA 2005. LNCS, vol. 3480, pp. 619–627. Springer, Heidelberg (2005)
32. Lim, C., Korkishko, T.: mCrypton - A Lightweight Block Cipher for Security of Low-cost RFID Tags and Sensors. In: Song, J., Kwon, T., Yung, M. (eds.) WISA 2005. LNCS, vol. 3786, pp. 243–258. Springer, Heidelberg (2005)
33. Menezes, A.J., Vanstone, S.A., Van Oorschot, P.C.: Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton (1996)
34. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer, Heidelberg (1990)
35. National Institute of Standards and Technology. FIPS 46-3: Data Encryption Standard (DES) (October 1999), http://csrc.nist.gov

36. National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard (August 2002), http://csrc.nist.gov
37. National Institute of Standards and Technology. FIPS 197: Advanced Encryption Standard (AES) (November 2001), http://csrc.nist.gov
38. National Institute of Standards and Technology. FIPS 198: The Keyed-Hash Message Authentication Code (March 2002), http://csrc.nist.gov
39. Ohkubo, M., Suzuki, K., Kinoshita, S.: Cryptographic Approach to Privacy-Friendly Tags. In: RFID Privacy Workshop, MIT, Cambridge (2003)
40. Okeya, K.: Side Channel Attacks Against HMACs Based on Block-Cipher Based Hash Functions. In: Batten, L.M., Safavi-Naini, R. (eds.) ACISP 2006. LNCS, vol. 4058, pp. 432–443. Springer, Heidelberg (2006)
41. Peyrin, T., Gilbert, H., Muller, F., Robshaw, M.J.B.: Combining Compression Functions and Block Cipher-Based Hash Functions. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 315–331. Springer, Heidelberg (2006)
42. Poschmann, A., Leander, G., Schramm, K., Paar, C.: New Lightweight DES Variants Suited for RFID Applications. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 196–210. Springer, Heidelberg (2007)
43. Preneel, B., Bosselaers, A., Govaerts, R., Vandewalle, J.: Collision-Free Hash Functions Based on Block Cipher Algorithms. In: Proceedings 1989 International Carnahan Conference on Security Technology, pp. 203–210. IEEE, Los Alamitos (1989)
44. Preneel, B.: Ph.D. thesis. Katholieke Universiteit Leuven (1993)
45. Quisquater, J.-J., Girault, M.: 2n-bit Hash-Functions Using n-bit Symmetric Block Cipher Algorithms. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 102–109. Springer, Heidelberg (1990)
46. Rivest, R.L.: RFC 1321: The MD5 Message-Digest Algorithm (April 1992), http://www.ietf.org/rfc/rfc1321.txt
47. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004)
48. Seurin, Y., Peyrin, T.: Security Analysis of Constructions Combining FIL Random Oracles. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 119–136. Springer, Heidelberg (2007)
49. Shamir, A.: SQUASH - a New MAC With Provable Security Properties for Highly Constrained Devices Such As RFID Tags. In: Nyberg, K. (ed.) FSE 2008, Springer, Heidelberg (to appear, 2008)
50. Steinberger, J.P.: The Collision Intractability of MDC-2 in the Ideal-Cipher Model. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 34–51. Springer, Heidelberg (2007)
51. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
52. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
53. Yoshida, H., Watanabe, D., Okeya, K., Kitahara, J., Wu, J., Kucuk, O., Preneel, B.: MAME: A Compression Function With Reduced Hardware Requirements. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 148–165. Springer, Heidelberg (2007)
54. Yu, Y., Yang, Y., Fan, Y., Min, H.: Security Scheme for RFID Tag. Auto-ID Labs white paper WP-HARDWARE-022, http://www.autoidlabs.org/

Appendix I

Using current best practice we outline a representative design for a 192-bit hash function, named C-PRESENT-192 in the text. Some of the background to the design and its implementation are given in Section 5.1 with some additional explanation below.



Motivation. Our goal was to design a 192-bit hash function using PRESENT as the fundamental building block and to use linear mixing layers both before and after the compression functions. Within this framework, it has been shown that the minimum number of compression functions that can be used is six. In addition, the output mapping should be a $(6,3,3)$ binary linear error-correcting code, while the input mapping must satisfy the following constraints:

1. Every external output block must depend on all external input blocks, no matter which invertible transformation of the external inputs and outputs is applied.
2. Every pair of external input blocks must appear as an identified pair for every invertible combination of external output blocks, where a pair (A,B) is said to be identified when A and B both appear within the internal inputs to some $f^{(i)}$, and this no matter which invertible transformation of the external inputs is applied.

The input mapping for our representative was selected from among those that satisfy these conditions and that also minimize the number of key schedules used to hash one block of message. By reducing the number of key schedules we increase the performance of the scheme and, potentially, reduce the space required by an implementation. It can be proved that for the parameter sets of interest to us here, the minimal number of key schedules is two.

For the results of Peyrin *et al.* to hold, the compression functions $f^{(i)}$ have to be *ideal* compression functions with respect to collision and preimage resistance (that is, finding a collision or a preimage must require on average $\Theta(2^{n/2})$ and $\Theta(2^n)$ evaluations of the function respectively) and must behave independently. Each inner compression function $f^{(i)}$ is built around PRESENT-128 in a way similar to the Davies-Meyer mode. That way, the results of Black *et al.* [5] ensure that, in the ideal cipher model, finding a collision (resp. a preimage) for the compression functions $f^{(i)}$ requires $\Theta(2^{n/2})$ (resp. $\Theta(2^n)$) queries to the cipher. Hence, in the ideal cipher model, each inner compression function $f^{(i)}$ is ideal in the sense defined above.

Making the six compression functions $f^{(i)}$ independent is not so easy. The most secure way to do this would be to "tweak" the block cipher with *e.g.* the XE or XEX construction of Rogaway [47]. However, these constructions are only efficient when one has to compute ciphertexts for the same key and many different tweaks, which is not our case. Using any known provably secure construction of a tweakable block cipher for the C-PRESENT-192 scheme would imply one supplementary cipher call for each key, thus increasing the number of block cipher calls per message block to eight. Instead we might consider using the same kind of technique that is used in the Hirose construction and we use five constants $c_1, \ldots, c_5$ to make the six instances of the compression function independent. In the absence of a structural weakness in PRESENT this is sufficient for our purposes. Further, we are trying to estimate the space required for a construction of this type and so this approach will help yield conservative estimates. The constants were chosen to be linearly independent and of low Hamming weight. They are given by $\mathtt{c_0} = \mathtt{0}$ and $\mathtt{c_i} = (\mathtt{0x0000000000000001}) \ll (\mathtt{i}-\mathtt{1})$ for $i \geq 1$. While some limitations of this construction follow from [48], assuming we can consider the inner compression functions independent, Peyrin *et al.* show that there is no currently-known attack with *computational* complexity less than brute-force on the larger compression function.

Appendix II

Our proposed design elements are not intended to be specifications. Nevertheless, some preliminary analysis follows from the simple structures proposed. In particular, for a fixed message block and two different chaining values we can apply Theorem 1 of [6] directly. This states that at least 10 active S-boxes are involved in any 5-round differential characteristic. However, for the more important case of two different message blocks, the analysis has to be slightly modified. The following two results on the differential behavior of the proposals can be viewed as a first step towards a deeper analysis:

Theorem 1. *Let $P^{(3)}_{(\Delta_1,\Delta_2)\mapsto\Delta}$ be the probability of a differential characteristic over 3 rounds of PROP-1 with $\Delta_2 \neq 0$, i.e. the probability that*

$$\text{PROP-}1_3(H\oplus\Delta_1, M\oplus\Delta_2) = \text{PROP-}1_3(H,M)\oplus\Delta,$$

where PROP-1_3 *denotes three rounds of PROP-1. Then each 3-round differential characteristic of this form has at least 4 active S-boxes and therefore $P^{(3)}_{(\Delta_1,\Delta_2)\mapsto\Delta} \leq 2^{-8}$.*

Theorem 2. *Let $P_{(\Delta_1,\Delta_2)\mapsto\Delta}$ be the probability of a differential characteristic such that*

$$\text{PROP-}2(H\oplus\Delta_1, M\oplus\Delta_2) = \text{PROP-}2(H,M)\oplus\Delta$$

for $\Delta_2 \neq 0$. Then $P_{(\Delta_1,\Delta_2)\mapsto\Delta} \leq 2^{-400}$ for PROP-2.

Theorem 1 indicates that the probability of each 64-round differential characteristic can be upper-bounded by $(2^{-8})^{\frac{64}{3}} \approx 2^{-170}$. This observation as well as Theorem 2 show that the differential properties may be strong enough to thwart pre-image, second pre-image and collision attacks for the both proposals. Furthermore, Theorem 2 indicates that one could probably decrease the number of rounds in PROP-2 without unduly compromising the security. The most appropriate trade-off remains an area of research.