Looking Back at a New Hash Function

Olivier Billet[1], Matthew J.B. Robshaw[1], Yannick Seurin[1], and Yiqun Lisa Yin[2]

[1] Orange Labs, Issy les Moulineaux, France
{forename.surname}@orange-ftgroup.com
[2] Independent Security Consultant
yiqun@alum.mit.edu

Abstract. We present two (related) dedicated hash functions that deliberately borrow heavily from the block ciphers that appeared in the final stages of the AES process. We explore the computational trade-off between the key schedule and encryption in a block cipher-based hash function and we illustrate our approach with a 256-bit hash function that has a hashing rate equivalent to the encryption rate of AES-128. The design extends naturally to a 512-bit hash function.

1 Introduction

After recent cryptanalytic advances [37,38] the need for new hash functions has become acute. In response NIST has made a call for proposals [28] for the development of a new *Advanced Hash Standard (SHA-3)*. However most commentators would probably agree that the field of hash functions has, until recently, been somewhat neglected and that the current knowledge of hash function design is somewhat fragmented. So difficult are the starting conditions for the development of the AHS that it is not always straightforward to exactly articulate the properties we want from a hash function. Even worse, there is little agreement on even the basic features for a successful hash function design.

By way of contrast, if we were to turn the clock back to the start of the AES process, at that time we already had five years of block cipher theory and design *after* the development of linear cryptanalysis [20] and ten years *after* the development of differential cryptanalysis [8]. And while all the AES submissions were very different, their designs had evolved from several years of research experience gained during the mid-1990s.

In this paper we propose two new (related) dedicated hash functions DASH-256 and DASH-512. Whilst they are, in principle, suitable for submission to the NIST hash function development process, this is not our intention. Instead we prefer to see the paper as research-oriented and our work is prompted by the following questions :

1. How close can we stay to AES proposals in the design of a hash function?
2. Can we use an unusual key schedule design to our advantage?

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

2 Background, goals, and design criteria

We informally recapitulate some of the classical goals for a hash function. 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 [21]. For a secure hash function that gives an n-bit output, compromising these properties should require 2^n, 2^n, and $2^{n/2}$ operations respectively. A more thorough set of hash function requirements for the SHA-3 development process is available at [28].

The pioneering work of Merkle and Damgård [14,22] showed how to construct a collision-free hash function from a *compression function* that has a fixed-length input. This input consists of a *chaining variable* and a message extract while the new value of the chaining variable is produced as output. The chaining variable will be denoted by v_i and the message extract will be denoted by m_i. Thus, at iteration i of the Merkle-Damgård construction, we compute $v_{i+1} = \text{COMPRESS}(m_i, v_i)$. The advantages and disadvantages of the Merkle-Damgård approach are, by now, well-established. On the positive side are its simplicity and the proof of security that (loosely speaking) relates the collision-resistance of the hash function to that of the compression function. On the negative side are cryptanalytic results that take advantage of the chaining that is used in a repeated application of the compression function [15,16,18,19]. These results help provide a greater understanding of the Merkle-Damgård approach, particularly when hashing exceptionally long messages.

Design decisions.

Our design philosophy for DASH-256 (and DASH-512) can be summarised as: *keep it simple and use established techniques*. In practise this resulted in the following:

1. We base the hash function around the use of a compression function and the Merkle-Damgård paradigm [14,22]. To avoid some structural deficiencies we use the HAIFA model [7] for formatting the inputs.
2. For the compression function we use a block cipher and *Davies-Meyer* [29].
3. We revisit the AES process and appeal to the vast pool of results [25] to design a block cipher at the heart of the compression function.
4. We push the parameters of a block cipher key schedule so as to better understand the range of options for a practical hash function design.

By way of background, we now consider each issue in turn.

Using Merkle-Damgård and Davies-Meyer.

While there have been proposals for alternatives to Merkle-Damgård, *e.g.* the *sponge construction* [6], we focus on the body of work that considers adjustments to Merkle-Damgård, such as those of Coron *et al* [13], Biham and Dunkelman [7], and Rivest [34]. These proposals share the property that Merkle-Damgård is

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

used almost as is, but that additional inputs are included at each iteration of the compression function. They vary in the form of inputs and the resultant loss of efficiency, but recent work [2] has shown that the more efficient proposal by Rivest [34] does not seem to provide the additional security intended. With this in mind we use the HAsh Iterative FrAmework [7], or HAIFA model.

For the compression function itself we use a block cipher in Davies-Meyer mode. This is a mode for which there is a proof-of-security, *i.e.* the security of Davies-Meyer can be reduced to that of the underlying block cipher. If we denote encryption of a plaintext p under a key k by $\text{ENC}_k(p)$ then the output of the Davies-Meyer mode is given by $\text{ENC}_k(p) \oplus p$. When used as a compression function, for which the chaining variable input is denoted v_i and the message input is denoted m, the next value of the chaining variable output from the compression function is given by $v_{i+1} = \text{ENC}_m(v_i) \oplus v_i$. We note that there are some unusual properties of Davies-Meyer. For instance it is easy to find fixed points for this construction. By choosing $v_i = \text{ENC}_m^{-1}(0)$ we have that $\text{COMPRESS}(m, v_i) = v_i$ since $\text{ENC}_m(v_i) \oplus v_i = v_i$. However the HAIFA model helps to mitigate the effect of these, as well as countering other recent cryptanalytic work on long-message attacks [15,16,18]. By contrast the Davies-Meyer mode has one major advantage over other single block cipher constructions [11]. Note that for the Davies-Meyer transformation $\text{ENC}_m(v_i) \oplus v_i$ the block size is given by $|v_i|$ and the key size by $|m|$. For the AES these are restricted to 128 bits and 128/192/256 bits respectively. However our design allows the block size to vary between 256 and 512 bits while the "key" length is eight times larger; this permits larger message inputs on each iteration and a more competitive throughput. And for block cipher designs there is no better place to look than the AES process [25].

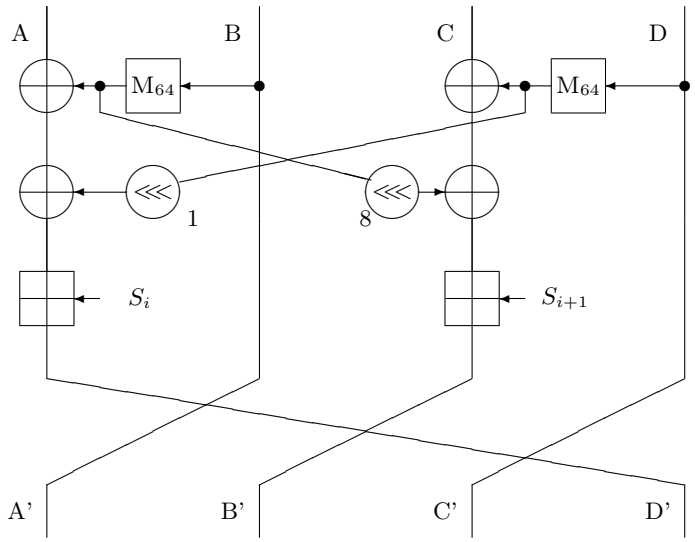
Revisiting the AES process.

Returning to the AES process with the benefit of hindsight is an interesting experience. We are not the first to do so: the designers of PRESENT [12] used the AES finalist Serpent [9] as a starting point for the development of their ultra-compact block cipher. We therefore hope to be able to make similarly advantageous observations by considering two other finalists: Rijndael (*i.e.* the AES [24]) and RC6 [32]. Rijndael is now very well known. Like Rijndael, RC6 was a simple proposal that offered good software performance on modern processors [3]. However the 32-bit squaring operation didn't scale quite as well to 8-bit processors or hardware implementations. However, of particular interest to us here is the key schedule for RC6. While it is computationally heavy, it allows very long keys. This is ideal for a hashing application as was observed by the RC6 designers during the AES process [33].

So the block cipher that lies at the heart of DASH-256 and DASH-512 will use a topology that is similar to RC6 and CLEFIA [36] along with a key schedule that is almost identical to that used in RC5 and RC6. However we will make changes to some of the operations used to improve scalability and to reduce the potential exposure to side-channel analysis in MAC-applications [27].

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

Fig. 1. One round of the encryption routine for $\mathcal{A}_{256}$ with the 256-bit input $(A||B||C||D)$ being transformed into the 256-bit output $(A'||B'||C'||D')$.



Block ciphers and key schedules.

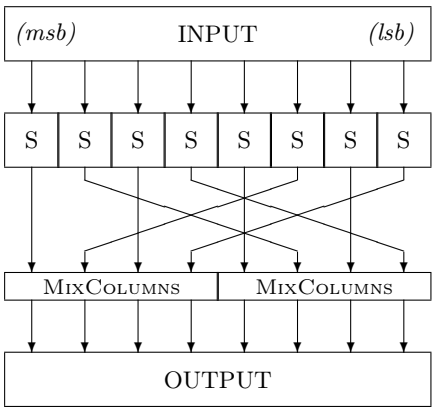
The performance of a block cipher is dependent on the cost of both the encryption routine and key setup. For bulk encryption the cost of a single key setup is amortized over the entire encryption session. However, when used as the basis for a hash function, the cost of the key schedule becomes a significant factor. Most modern ciphers, including the AES, tend to have a lightweight key schedule. In this paper, we want to explore what happens when we put more work into the key schedule. Given the importance of key schedule performance for hashing, at first sight this appears to be the wrong direction. Indeed for constructions where the message and chaining value must be the same size, such as Mateas-Meyer-Oseas [21] and Miyaguchi-Preneel [21] (which is used in Whirlpool [5]) this would be the case. However, the Davies-Meyer construction allows us to take a very large message block as "key" and provided there is sufficient mixing of the key there is no reason why a good performance profile cannot be attained.

3 The Specification of DASH-256 and DASH-512

Both DASH-256 and DASH-512 are built around a dedicated block cipher which we will name $\mathcal{A}_{256}$ and $\mathcal{A}_{512}$ respectively. For ease of exposition we will concentrate on DASH-256 and $\mathcal{A}_{256}$ and describe the cipher in terms of an encryption routine and a key schedule. The necessary changes for DASH-512 are given in Section 3.4.

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

Fig. 2. The mixing operation M_{64} that is used in $\mathcal{A}_{256}$. Note that the S-boxes and MDS transformations are those specified in the AES.



3.1 The encryption routine for $\mathcal{A}_{256}$

One encryption round (out of the 30 required) is illustrated in Figure 1. Each strand represents a 64-bit word and the key schedule, see Section 3.2, generates 64 subkeys of which two are used as pre-whitening for strands B and D, 60 are used during the encryption process (two in each round), and two are used as post-whitening on strands A and C before output. The data-dependent rotations and multiplication in RC6 have been replaced in $\mathcal{A}_{256}$ with a confusion/diffusion operation closely inspired by the AES. The mixing operation M_{64} is the natural restriction of the AES diffusion layer to two columns, see Figure 2, and uses the S-boxes and AES MDS transformation directly. This allows us to combine the scalability of RC5 and RC6 with the AES diffusion operations.[3] However, AES diffusion is somewhat structured so the one-bit and eight-bit rotations help to break some alignments and avoid some trivial linear approximations.

3.2 The key schedule for $\mathcal{A}_{256}$

By using a key schedule that is close to that used in RC5 and RC6 we aim to leverage its long-standing in the literature and the opportunities for analysis during the AES process. We also take advantage of the fact that it allows long key inputs. The original key schedule can be found in either of [30,32] though we follow the example set in the encryption routine and replace the single data-dependent rotation in the key schedule with the AES-inspired diffusion operation M_{64}. This is illustrated in Figure 3. The input to be hashed at an iteration of the compression function (after HAIFA formatting) will be 256 bytes long and loaded into an array of 32 words of 64 bits $L[0], \ldots, L[31]$. From this we generate

[3] Naturally other MDS transformations [17] and S-boxes may offer other advantages.

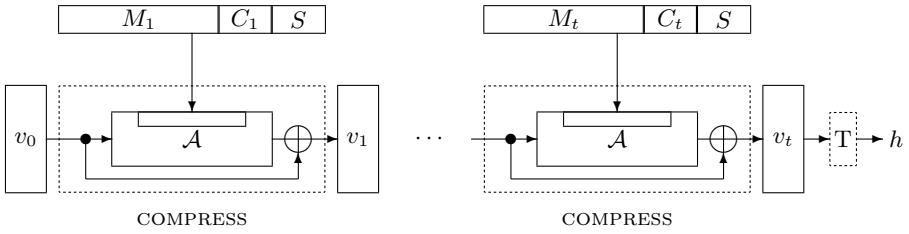
Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

Fig. 3. The key schedule for $\mathcal{A}_{256}$. The input is represented as an array $L[\cdot]$ of 32 64-bit words and the output is a set $S[\cdot]$ of 64 64-bit subkeys. The constants $P_{64} =$ `0xB7E151628AED2A6B` and $Q_{64} =$ `0x9E3779B97F4A7C15` are those used in RC5 and RC6.

$S[0] = P_{64}$
for $i = 1$ **to** 63 **do** $S[i] = S[i-1] + Q_{64}$

$A = B = i = j = 0$
for $s = 1$ **to** (3×64) **do**
{
$A = S[i] = (S[i] + A + B) \lll 3$
$B = L[j] = (L[j] + A + B) \oplus \mathrm{M}_{64}(A + B)$
$i = (i + 1) \bmod 64$
$j = (j + 1) \bmod 32$
}

Fig. 4. The chained iteration of the compression function. In the HAIFA model, the initial value v_0 is computed from an IV (Section 3.3) which we choose to be `0x FEDCBA9876543210` || `0123456789ABCDEF` || `FDB97531ECA86420` || `02468ACE13579BDF`.



64 words of 64 bits which are stored in an array $S[0, ..., 63]$ and used as subkeys during encryption.

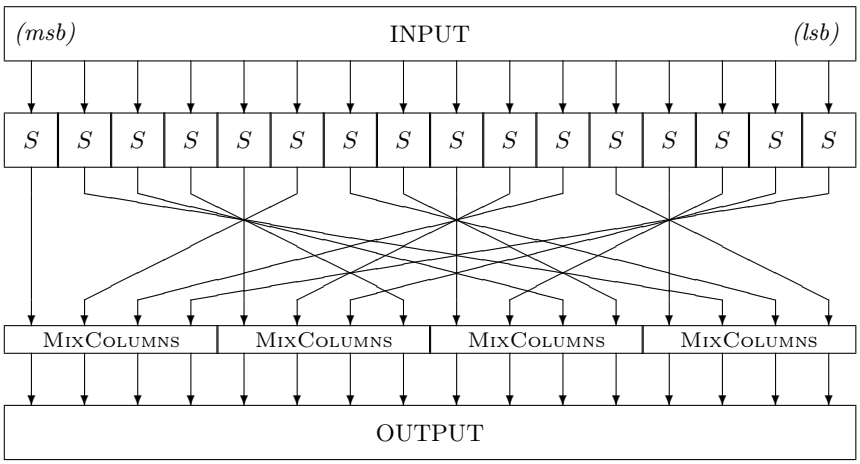
3.3 The full specification of DASH-256

We restrict ourselves to the case of DASH-256 and since we follow the HAIFA construction there are three inputs to the hash function; a message $\mathcal{M}$ of length n bits with $n < 2^{64}$, a salt value S of 64 bits, and the length d of the hash output or message digest. Internally, we use a 64-bit counter that takes a value denoted C_i at iteration i of the compression function. The counter stores the value—in little-endian notation—of the number of bits of $\mathcal{M}$ that have been hashed so far.

To deal with incomplete blocks we pad $\mathcal{M}$ to give a related message $\mathcal{M}'$. The input to the compression function x_i is of the form $[M_i||C_i||S]$ with $|C_i| = |S| = 64$ and so we generate a padded message $\mathcal{M}'$ that is of length $t \times 1920$ bits where

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

Fig. 5. The mixing operation M_{128} that is used in $\mathcal{A}_{512}$. Note that this is exactly the diffusion layer specified in the AES.



t is the smallest integer for which $t \times 1920 > n + 73$, the strange number being explained by what follows: padding is always applied and appends a single '1' bit and as many '0' bits as needed so as to leave room for nine bits that are set[4] to the binary representation of the hash output length d and a further 64 bits that are reserved for the binary representation of n. The resultant $\mathcal{M}'$ is then divided into t blocks $M_1, \ldots, M_t$, with each M_i being of length 240 bytes.

At each iteration of the compression function there are two inputs; the current value of the chaining variable v_i which is a 256-bit input and the 2048-bit $x_i = [M_i||C_i||S]$ that is being processed and we have $v_i = \text{COMPRESS}(x_i, v_{i-1})$ for $1 \leq i \leq t$. The initial value v_0 is computed as $v_0 = \text{COMPRESS}(d||\text{IV}||0)$ for a master IV, as required in HAIFA, and the output is given by the value v_t. A hash value of any shorter length, such as 224 bits, can be derived by truncation from the left, *i.e.* we use the rightmost bits. This is indicated by T in Figure 4 and would, of course, require that the representation of d in the padding and computation of v_0 be changed accordingly.

3.4 The specification of DASH-512

The essential difference between DASH-256 and DASH-512 is that the first operates on 64-bit words while the second operates on 128-bit words. This is a direct benefit of the elegant scalability designed into RC5 [30]. All-but-one of the operations in DASH-256 scale obviously between the versions, the one exception being the M_{64} function. However for DASH-512 we use M_{128} which is a 128-bit permutation that is *identical* to one round of the AES without the key addition.

[4] This is required in the HAIFA model.

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

This is illustrated in Figure 5. All other parts of the algorithm, illustrated in Figures 1, 3, and 4, scale in the obvious way and any 64-bit word operation is replaced by a 128-bit word operation. In Figure 3 the equivalent constants P_{128} and Q_{128} can be defined as described in [30] for $\mathcal{A}_{512}$. Future analysis will reveal the appropriate number of rounds for $\mathcal{A}_{512}$ while padding will follow the HAIFA model and this can also be used to compute a 384-bit hash value by truncation.

4 Security Analysis

The security of DASH-256 and DASH-512 can be split into a consideration of the underlying block cipher and then of the compression function and chaining mode. The latter concerns are handled by the results of Damgård [14], Merkle [22], Black *et al* [11], and Biham and Dunkelman [7] so for reasons of space we concentrate on the cipher within the compression function and particularly on $\mathcal{A}_{256}$.

4.1 The encryption routine in the component $\mathcal{A}_{256}$

Many cryptanalytic tools for block ciphers can be used against hash function so we consider these classical techniques first.

Differential cryptanalysis. We can easily identify a lower bound on the number of active S-boxes for a differential in $\mathcal{A}_{256}$ (and $\mathcal{A}_{512}$) when the expanded message words $S[\cdot]$ are the same for both pairs in a differential. The situation where the expanded message words might induce a difference is considered in the case of *local collisions* below.

Without loss of generality, we can suppose that we have a non-zero exclusive-or difference in strand `A`. This will pass across a single round of $\mathcal{A}_{256}$ and $\mathcal{A}_{512}$ trivially. However it must induce a, for $1 \leq a \leq 8$, active S-boxes in the following round which, in turn, induce more active S-boxes in the rounds that follow. To establish a lower bound on the number of active S-boxes we can appeal to the properties of the MDS operation in M_{64} and M_{128} and observe that over two adjacent active rounds there must be at least five active S-boxes. Thus over any three rounds of $\mathcal{A}_{256}$ and $\mathcal{A}_{512}$—for which there is no difference in the expanded message words—there will be at least five active S-boxes. This gives a differential probability of less than 2^{-30}. Since there are 30 rounds to $\mathcal{A}_{256}$ this leads to a simple upper bound of 2^{-300} over the full encryption routine.

This basic analysis is crude in two significant ways. First, on the positive side for the algorithm, it *significantly* under-estimates the number of active S-boxes. Second, on the negative side, this crude analysis doesn't immediately capture situations where the array $S[\cdot]$ might be used to introduce a difference. However analysis of the key schedule, see Section 4.2, and of *local collisions* later in this section suggest that more complex differential phenomena are highly unlikely and while more sophisticated analysis is underway, we expect this to confirm the difficulty of applying differential techniques.

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

Linear cryptanalysis. The fixed rotations during encryption with $\mathcal{A}_{256}$ (and $\mathcal{A}_{512}$), see Figure 1, are intended to hinder the evolution of linear approximations. Note that without the fixed rotations it would be straightforward to identify linear approximations that held with probability 1 across infinitely many rounds. In particular, if we were to use Γ_i to denote the single-bit parity mask with a single one in position i, then we would have the following linear approximation for a single round of $\mathcal{A}_{256}$ with no rotations:

$$(\Gamma_0, 0, \Gamma_0, 0) \xrightarrow{\text{ONE ROUND (NO ROTATIONS)}} (0, \Gamma_0, 0, \Gamma_0).$$

This would hold with probability 1, *i.e.* with the maximum bias of $\frac{1}{2}$.

However the simple fixed rotations prevent such simple linear approximations from developing. We note that there is an interesting effect if we were to remove, or to change into exclusive-or, the operation used to introduce the expanded message words $S[\cdot]$. Let us call such a round a *linearised-round* and for this linearised variant of $\mathcal{A}_{256}$ we will consider the parity mask consisting of *all* bits, *i.e.* a mask of $\Gamma_p =$ `0xFFFFFFFFFFFFFFFF`. Then we would have that:

$$(\Gamma_p, 0, \Gamma_p, 0) \xrightarrow{\text{ONE LINEARISED ROUND}} (0, \Gamma_p, 0, \Gamma_p)$$

with probability 1. Thus the *linearised* version of $\mathcal{A}_{256}$ (and $\mathcal{A}_{512}$) would be vulnerable to this kind of analysis, a common enough situation when ciphers are modified to facilitate analysis. However, with integer addition and an effective key schedule such parity relations are quickly destroyed.

Three-round local collisions. Here we consider a typical *disturbance correction* strategy and how it might be used against $\mathcal{A}_{256}$. We consider the following perturbative-corrective pattern for a three-round local collision and the linearised version of $\mathcal{A}_{256}$, *i.e.* where the expanded message words $S[i]$ are introduced using exclusive-or. Consider the follow three rounds of expanded message

$$(\Delta S[i], 0, \Delta S[i+2], \Delta S[i+3], 0, \Delta S[i]),$$

where $\Delta S[i]$ is a low-weight perturbative vector, and $\Delta S[i+2]$ and $\Delta S[i+3]$ are deduced from the best differential of the AES S-box, *i.e.*

$$\Delta S[i+2] = (\Delta S'[i]) \lll 1 \quad \text{and} \quad \Delta S[i+3] = \Delta S'[i]$$

where $\Delta S'[i]$ is such that $\Pr_C[\mathrm{M}_{64}(C \oplus \Delta S[i]) \oplus \mathrm{M}_{64}(C) = \Delta S'[i]]$ is maximal. The maximal differential probability of the AES S-box is 2^{-6}, hence whatever $\Delta S[i]$ and $\Delta S'[i]$, the probability of such a local collision for the linearised variant of $\mathcal{A}_{256}$ is upper bounded by 2^{-12}.

If we now return to the real $\mathcal{A}_{256}$ where the words $S[i]$ are mixed through modular addition, we can make the following analysis. For each difference bit in `A` and `B`, `A` + `C` and `B` + `C` differ only in the same bits as `A` and `B` with probability upper bounded by 2^{-r}, where r is the number of different bits, with the exception

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

of the *most significant bit (MSB)*, in `A` and `B`. There are four additions to take into account, one for each non-zero input expanded message word. Due to the MDS property in M_{64} and M_{128}, one must have $\mathrm{Hwt}(\Delta S[i])+\mathrm{Hwt}(\Delta S[i+2]) \geq 5$ as well as $\mathrm{Hwt}(\Delta S[i]) + \mathrm{Hwt}(\Delta S[i+3]) \geq 5$ where we use Hwt to denote the Hamming weight.

This means that there are at least six active bits across integer addition that are not in the most significant position. Hence the probability of such a local collision is upper bounded by $2^{-12} \times 2^{-6} = 2^{-18}$. To do better than the birthday attack, an attack on $\mathcal{A}_{256}$ would need an expanded message difference that combines seven or less such local collisions. However this would imply that the remaining 64-bit words in $S[\cdot]$ are identical for the two messages and there is a vanishingly small chance that an attacker can manipulate message inputs so as to give two arrays $S[\cdot]$ with the required values.

4.2 The key schedule in the component $\mathcal{A}_{256}$

By choice the key schedule for $\mathcal{A}_{256}$ is closely related to that used in RC5 and RC6. In moving to the key schedule in $\mathcal{A}_{256}$ and $\mathcal{A}_{512}$ we have added some non-linearity via a series of AES S-boxes. While experiments have shown an improved avalanche of change as a result, this does not exclude some dedicated analysis.

The attack of Saarinen on RC6. During the first round of the AES process, Saarinen made some interesting observations about the RC6 key schedule when very long keys were used [35]. Let us assume that we choose a key length so that the arrays $L[\cdot]$ and $S[\cdot]$ are of equal length.[5] The important feature of the key expansion, see Figure 3, is that state information is carried between the two arrays by two words A and B. If we take two keys that are nearly equal except for the last few words then, on the first pass through, only the last few words of the $L[\cdot]$ and $S[\cdot]$ arrays will change. If the cryptanalyst is lucky, or if we can find a high probability differential of the right form, the difference in the values of A and B at the start of the second pass will be zero. When this happens, no change is carried into the second pass and only the last few words of the $L[\cdot]$ and $S[\cdot]$ arrays will have a non-zero difference.

Moving on, if we are lucky (since we cannot rely on a differential of sufficiently high probability) the difference in the values of A and B at the start of the third pass will be zero. If this happens then, on the third and final pass through the arrays, only the later words in $S[\cdot]$ will change. Saarinen [35] was therefore able to demonstrate ciphertexts generated by related keys that had an average Hamming distance between them of 4.2 bits. This was later extended [23] to demonstrate the existence of equivalent keys for this particular instance.

Two features are important for this attack. First, being able to identify a short cancelling differential for the first pass. Second, the number of times we pass through the $L[\cdot]$ array. In the case of RC6 with 128-bit blocks and 1308-bit

[5] This is the simplest case, but variants exist for different array sizes.

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

keys (the case looked at by Saarinen) we start the $L[\cdot]$ array three times. For the first pass the difference in A and B is zero (by definition). For the second pass it is zero by construction of the differential, and for the third pass we can use the birthday paradox to find a pair of messages that generate a zero-difference in A and B from a pool of 2^{32} possibilities.

In the case of $\mathcal{A}_{256}$ and $\mathcal{A}_{512}$, however, the $S[\cdot]$ array will always be twice the size of the $L[\cdot]$ array. Thus we will pass through the $L[\cdot]$ array six times. The conditions we need on A and B at the start of each pass is a condition on 128 or 256 bits respectively. The first time it is trivially satisfied and we might pessimistically assume that it can be satisfied with probability one the second time.[6] Then there remain three times for which the condition on A and B must hold by chance before we process the $S[\cdot]$ array for the final time. Thus we have a condition on 3×128 bits (3×256 resp.) which we expect to see fulfilled from a pool of 2^{192} messages (2^{384} resp.) using the birthday paradox. However this is worse than brute-force.

In fact the conditions to avoid the attack of Saarinen can be generalised and we need to pass through the $L[\cdot]$ array at least four times and the $S[\cdot]$ array at least three times. This is what we accomplish in DASH-256 and DASH-512. Interestingly, in [33] the RC6 designers propose a 1024-bit key length when using RC6, which is based on 32-bit words, for the most efficient hashing configuration. This also satisfies our general requirement.

On the potential for collisions in the expansion. Consider two different inputs M_i and M_i' to an iteration of the compression function. These will be used to initialise the $L[\cdot]$ array and after the expansion phase will give the final values to the $S[\cdot]$ array. Clearly, if we derive the same $S[\cdot]$ array from different inputs then we trivially have a collision over one iteration of the compression function. However provided there is sufficient mixing of the $L[\cdot]$ and $S[\cdot]$ arrays, there are no known weak or equivalent key phenomena for RC5 or RC6 and a brute-force attack seems to pose a far greater concern. Of course this isn't the full picture. Even arrays that are identical only part of the time, or in the earlier words, can still be useful to the cryptanalyst. However, assuming sufficiently thorough mixing of the values in the $S[\cdot]$ array, collisions in the chaining variable would seem to be easier to find than pairs of messages where five or more words in $S[\cdot]$ are identical. Given 30 rounds, it is highly unlikely an exploitable weaknesses will occur by chance.

The oneway-ness of the key schedule. The key schedule expands the message-related input into a set of 64 subkeys. This is done in a complicated way and it has been noted by various commentators that this delivers a certain amount of *one-wayness* [30,35]. So even if attacks on the encryption process leak information about the subkeys $S[\cdot]$ it would be very hard to relate this information to the input M_i at the i^{th} iteration of the compression function. Yet

[6] This would require a sophisticated differential through several AES S-boxes.

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

it is information about the input M_i that is needed to compromise either the compression function or the resultant hash function.

The role of the key schedule. It is well-known that many hash function designs are built around a dedicated block cipher. In such cases there is some message mixing, *i.e.* a key schedule, and some state processing, *i.e.* an encryption routine. In MD5 [31] the message-mixing involves message block repetition while in SHA-1 [26] "key expansion" is a little more involved. However it remains simple and without a strong "encryption" process it is somewhat vulnerable.

By contrast, in $\mathcal{A}_{256}$ and $\mathcal{A}_{512}$ we might view the key schedule as a complex non-linear message expansion. This idea of "expansion" as a form of pre-processing appears in [1] and has been used in other hash functions [10]. Given such an expansion phase, we can then view the "encryption" as a complicated way of distilling information into the 256-bit (or 512-bit) output. But is it better to have more work done in the "expansion" or in the "distillation"? When looked at in this way, traditional hash functions of the MD-family have a computationally lightweight (almost trivial) expansion phase and compensate for this with a heavier mixing phase. For DASH-256 and DASH-512 this is reversed and we have a computationally heavy expansion paired with a lighter (though strong) distillation phase. We believe that this approach is worth exploring and could be better suited to the hashing environment where an attacker has *complete* control over the inputs to the compression function. For compression functions based on block ciphers, a simple key schedule will place a significant burden on the encryption routine.

5 Performance

Assessing the performance of a cryptographic algorithm is tricky and often incomplete. When we look at the operations in DASH-256 it is likely that most software implementations will use table look-ups for the S-box operation and that this will be the dominant operation. For DASH-256 there are $30 \times 2 \times 8 = 480$ table look-ups during encryption and $192 \times 8 = 1536$ during key expansion giving a total of 2016. Since 240 bytes are processed per compression function iteration we have 8.4 look-ups per byte. In comparison encryption with the AES-128 requires $10 \times 16 = 160$ look-ups for encryption but only 16 bytes of plaintext are encrypted giving an encryption cost of 10 look-ups per byte. Thus we might expect the bulk processing performance of DASH-256 to be comparable to the bulk encryption rate of AES-128. This seems to be a very natural target since they both offer the same 128-bit security. We can compare results from Wei Dai (see [39] for details) with our first-cut optimised version of DASH-256 where the key expansion (*exp.*) and processing time (*pro.*) are separated out.[7] The results compare well to some other recent hash function proposals [4].

[7] Note that *pro.* is not the encryption rate. If we used DASH-256 for encryption then we would need 480 table look-ups to encrypt 32 bytes giving a rate of 15 look-ups/byte.

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

platform	*clock (GHz)*	*AES-128 (cycles/byte)*	*SHA-256 (cycles/byte)*	*DASH-256 (cycles/byte)*		
[39] Opteron	2.4	15.9	21.5	-		
				exp.	*pro.*	*total*
Opteron	2.2	-	-	14.4	3.1	17.5

Unfortunately the performance of DASH-256 suffers on 32-bit machines. On the P4, for example, a first-cut implementation runs at half the speed of AES-128 and we see that even basic operations over 64-bit words can exact a heavy price.

6 Conclusions

We have presented two, closely-related, dedicated hash functions. In contrast to some other recent hash function proposals we have stayed close to known constructions and deliberately looked back at the AES process to use techniques that were analysed and discussed there. At the same time we have explored the role of a computationally-heavy key schedule which allows us to hash a large amount of message at each iteration. We believe that an appropriate balance between security and speed can be achieved in this way and we encourage others to explore the advantages and disadvantages of this approach.

Independently of the success of DASH-256, we can see several directions in which to take this work. Certainly we believe that some variants of DASH-256 may offer room for improvement. For instance, while the key schedule in DASH-256 has many interesting attributes, we feel that the design is too complex. And while its long standing is a good sign, it would be more satisfying to say something concrete about the security offered when such very long keys (messages) are being used. The state size of DASH-256 is large, though so is that of some other hash function proposals, and we note that there would be an overhead when hashing short inputs. The most significant downside, however, is that DASH-256 is oriented to 64-bit operations. Instead we feel (with hindsight) that a new design geared towards 32-bit operations would be a better starting point.

We believe that these are all interesting avenues to explore, as is the more general question of the role of the key schedule when a block cipher is used as the basis for a hash function. With this in mind, we hope that the simplicity of our proposal will promote new and independent analysis of DASH-256/512 in particular and hash functions in general; something that we strongly encourage.

References

1. W. Aiello, S. Haber, and R. Venkatesan. New Constructions for Secure Hash Functions. In S. Vaudenay, editor, *Proceedings of FSE '98*, volume 1372 of *Lecture Notes in Computer Science*, pages 150–167. Springer-Verlag, 1998.
2. E. Andreeva, C. Bouillaguet, P.-A. Fouque, J. Hoch, J. Kelsey, A. Shamir, and S. Zimmer. Second Preimage Attacks on Dithered Hash Functions. In N. Smart, editor, *Proceedings of Eurocrypt 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 270–288. Springer-Verlag, 2008.

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

3. K. Aoki and H. Lipmaa. Fast Implementations of AES Candidates. Available from `http://csrc.nist.gov`.
4. J.P. Aumasson, W. Meier, and R. Phan. The Hash Function Family LAKE. In K. Nyberg, editor, *Proceedings of FSE 2008*, to appear.
5. P. Baretto and V. Rijmen. The Whirlpool Hashing Function, Available from `paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html`.
6. G. Bertoni, J. Daemen, M. Peeters, and G. van Assche. Sponge Functions. Presentation at ECRYPT Hash Workshop, May 24-25, 2007. Available via `www.ecrypt.eu.org`.
7. E. Biham and O. Dunkelman. A Framework for Iterative Hash Functions - HAIFA. Presented at Second NIST Cryptographic Hash Workshop, August 24-25, 2006. Available via `csrc.nist.gov/groups/ST/hash/`.
8. E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer Verlag, 1993.
9. E. Biham, L.R. Knudsen, and R.J. Anderson. Serpent: A New Block Cipher Proposal. In S. Vaudenay, editor, *Proceedings of FSE '98*, LNCS, volume 4586, pages 222–238, Springer Verlag.
10. O. Billet, M.J.B. Robshaw, and T. Peyrin. On Building Hash Functions from Mutivariate Quadratic Equations. In J. Pieprzyk, H. Ghodosi, and E. Dawson, editors, *Proceedings of ACISP 2007*, LNCS, volume 4586, pages 82–95, Springer Verlag.
11. J. Black, P. Rogaway, and T. Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer-Verlag, 2002.
12. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *Proceedings of CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450-466. Springer-Verlag, 2007.
13. J-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430-448. Springer-Verlag, 2005.
14. I. Damgård. A Design Principle for Hash Functions. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, 1989.
15. R.D. Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, 1999.
16. A. Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In M.K. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer-Verlag, 2004.
17. P. Junod and S. Vaudenay. Perfect Diffusion Primitives for Block Ciphers— Building Efficient MDS Matrices. In H. Handschuh and M. Anwar-Hasan, editors, *Proceedings of SAC 2004*, LNCS, volume 3357, pages 84–98, Springer-Verlag, 2004.
18. J. Kelsey and B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer-Verlag, 2005.
19. J. Kelsey and T. Kohno. Herding Hash Functions and the Nostradamus Attack. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer-Verlag, 2006.

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.

20. M. Matsui. Linear Cryptanalysis Method for DES Cipher. In T. Helleseth, editor, *Proceedings of Eurocrypt '93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397, Springer-Verlag, 1994.
21. A.J. Menezes, S.A. Vanstone, and P.C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.
22. R.C. Merkle. One Way Hash Functions and DES. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer-Verlag, 1989.
23. H. Mizuno, H. Kuwakado, and H. Tanaka. Equivalent keys in RC6-32/20/176. IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences. Vol. E84-A, No. 10, pages 2474–2481.
24. National Institute of Standards and Technology. FIPS 197: Advanced Encryption Standard, November 2001. Available via `csrc.nist.gov`.
25. National Institute of Standards and Technology. AES Archive. Available via `csrc.nist.gov`.
26. National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard, August 2002. Available via `csrc.nist.gov`.
27. National Institute of Standards and Technology. FIPS 198: The Keyed-Hash Message Authentication Code (HMAC), March 2002. Available via `csrc.nist.gov`.
28. National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Available via `csrc.nist.gov`.
29. B. Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.
30. R.L. Rivest. The RC5 Encryption Algorithm. In B. Preneel, editor, *Proceedings of FSE 1994*, LNCS, volume 1008, pages 363–366, Springer-Verlag, 1994.
31. R. L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm, April 1992 Available via `www.ietf.org/rfc/rfc1321.txt`.
32. R.L. Rivest, M.J.B. Robshaw, R. Sydney, and Y.L. Yin. The Block Cipher RC6. Available via `csrc.nist.gov`.
33. R.L. Rivest, M.J.B. Robshaw, and Y.L. Yin. The Case for RC6 as the AES. Available via `csrc.nist.gov`.
34. R.L. Rivest. Abelian Square-Free Dithering for Iterated Hash Functions. Presented at First NIST Cryptographic Hash Workshop, October 31 - November 1, 2005. Available via `csrc.nist.gov/groups/ST/hash/`.
35. M.-J. O. Saarinen. A Note Regarding the Hash Function Use of MARS and RC6. Available via `csrc.nist.gov`.
36. T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata. The 128-bit Block Cipher CLEFIA. In A. Biryukov, editor, *Proceedings of FSE 2007*, LNCS, volume 4593, pages 181–195, Springer Verlag.
37. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer-Verlag, 2005.
38. X. Wang, Y.L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer-Verlag, 2005.
39. W. Dai. Crypto++ 5.5 Benchmarks, at `www.cryptopp.com/benchmarks.html`.

Appears in Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 239–253.