SPN-Hash: Improving the Provable Resistance against Differential Collision Attacks*

Jiali Choy[1], Huihui Yap[1], Khoongming Khoo[1], Jian Guo[2], Thomas Peyrin[3,**], Axel Poschmann[3,***], and Chik How Tan[4]

[1] DSO National Laboratories, 20 Science Park Drive, Singapore 118230
{cjiali,yhuihui,kkhoongm}@dso.org.sg
[2] Institute for Infocomm Research, A*STAR, Singapore
ntu.guo@gmail.com
[3] SPMS, Nanyang Technological University, Singapore
{thomas.peyrin,aposchmann}@ntu.edu.sg
[4] Temasek Laboratories, National University of Singapore
tsltch@nus.edu.sg

Abstract. Collision resistance is a fundamental property required for cryptographic hash functions. One way to ensure collision resistance is to use hash functions based on public key cryptography (PKC) which reduces collision resistance to a hard mathematical problem, but such primitives are usually slow. A more practical approach is to use symmetric-key design techniques which lead to faster schemes, but collision resistance can only be heuristically inferred from the best probability of a single differential characteristic path. We propose a new hash function design with variable hash output sizes of 128, 256, and 512 bits, that reduces this gap. Due to its inherent Substitution-Permutation Network (SPN) structure and JH mode of operation, we are able to compute its differential collision probability using the concept of differentials. Namely, for each possible input differences, we take into account all the differential paths leading to a collision and this enables us to prove that our hash function is secure against a differential collision attack using a single input difference. None of the SHA-3 finalists could prove such a resistance. At the same time, our hash function design is secure against pre-image, second pre-image and rebound attacks, and is faster than PKC-based hashes. Part of our design includes a generalization of the optimal diffusion used in the classical wide-trail SPN construction from Daemen and Rijmen, which leads to near-optimal differential bounds when applied to non-square byte arrays. We also found a novel way to use parallel copies of a serial matrix over the finite field $GF(2^4)$, so as to create lightweight and secure byte-based diffusion for our design. Overall, we obtain hash functions that are fast in software, very lightweight in hardware (about 4625 GE for the 256-bit

* The full version of this paper can be found on the eprint archive at http://eprint.iacr.org.

** This author is supported by the Lee Kuan Yew Postdoctoral Fellowship 2011 and the Singapore National Research Foundation Fellowship 2012.

*** This author was supported in part by Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03.

A. Mitrokotsa and S. Vaudenay (Eds.): AFRICACRYPT 2012, LNCS 7374, pp. 270–286, 2012.

hash output) and that provide much stronger security proofs regarding collision resistance than any of the SHA-3 finalists.

Keywords: SPN, wide-trail strategy, Hash Functions, collision resistance.

1 Introduction

For current hash function designs, there are mainly two approaches to obtain provable security. The first approach is to prove collision and/or preimage resistance in relation to *hard* problems. For instance, Contini et al.'s very smooth hash (VSH) [13] is a number-theoretic hash for which finding a collision can be proven to be equivalent to solving the VSSH problem of the same order of magnitude as integer factorization. Concerning preimage, an example is MQ-HASH [8] for which finding a preimage is proven to be as hard as solving a multivariate system of equations. For the SHA-3 candidate FSB [1], finding collisions or preimages imply solving syndrome decoding. The second approach is more practical and less rigorous, and aims at proving a good differential probability bound for a single characteristic path. However, collision resistance is only heuristically inferred from this bound.

The first approach accomplishes more than a proof of resistance to differential cryptanalyis. However, hash function schemes based on this design strategy often suffer significantly in terms of speed and performance. On the other hand, schemes using the second approach enjoy faster speeds but suffer from incomplete proof of collision resistance. In this paper, we seek to reduce the gap between these two approaches by providing a more powerful proof for collision resistance while maintaining similar speed as compared to the symmetric-key design hashes.

Here, we recall that a *differential characteristic* over a composed mapping consists of a sequence of difference patterns such that the output difference from one round corresponds to the input difference in the next round. On the other hand, a *differential* is the set of all differential characteristics with the same first-round input and last-round output differences. Most hash function designs only aim at showing that any single differential characteristic has sufficiently low probability and heuristically infer collision resistance from this. Examples of hash functions which adopt this approach include hashes such as WHIRLPOOL [21] and some SHA-3 finalists like GRØSTL [17] and JH [28]. In addition, this differential characteristic bound is hard to determine for Addition-Rotation-XOR (ARX) designs such as BLAKE [3] and SKEIN [16]. Therefore, the next step in collision resistance proof, as already done by the second-round SHA-3 candidate ECHO [5], is to give a bound on the best differential probability instead of only the best differential characteristic probability. However, note that this security argument only takes into account attackers that limit themselves to a fixed colliding differential (i.e. with a fixed output difference of the internal permutation), while many exist.

Our proposal for a new hash function design is able to achieve a stronger differential collision resistance proof. For example, for our proposed 512-bit hash, we prove that the differential probability of 4 rounds of its internal permutation function, which has a 1024-bit state size, is upper bounded by 2^{-816}. We sum this upper bound over **all output differences that lead to a collision** (2^{512} candidates) in order to find that the differential collision probability of our proposed hash is then upper bounded by $2^{-304} < 2^{-256}$ after the final truncation. In contrast, for the SHA-3 semi-finalist `ECHO` [5], the maximal expected differential probability for four rounds of their 2048-bit `AES` extension, `ECHO.AES`, is 1.055×2^{-452}, but summing over all possible colliding output difference masks (at least 2^{1536} candidates) completely prevents such a collision-resistance argument. For SHA-3 finalist, `GRØSTL`, it is easy to compute the internal collision probability of its compression function f. However, its output transformation, involving a permutation P followed by a truncation, makes such a derivation much less straightforward for the external collision probability of the full `GRØSTL` hash function.

In addition, we have to consider that for some hash function constructions, it is necessary to prove low related-key differential probability instead of just low fixed-key differential probability. For example, consider the Davies-Meyer compression function instantiated with `AES`. The main `AES` cipher has very low differential characteristic probability which is bounded by 2^{-150} for every four rounds. However, in the Davies-Meyer mode, each input message block to the hash corresponds to the cipher key of the `AES`-based compression function. This makes the compression function vulnerable to the multicollision attack by Biryukov et al. [9], because AES does not have good resistance against related-key differential attack.

1.1 Our Contributions

In this paper, we propose a new hash function design, `SPN-Hash`, with variable output sizes of 128, 256, and 512 bits. It is specially constructed to circumvent the weaknesses in the proofs of differential collision resistance as well as to resist common attacks against hash functions.

Concerning the internal permutations, we use the Substitution-Permutation Network (SPN) structure as the building block for `SPN-Hash` to ensure that the maximum probability taken over all differentials (not only differential characteristics) will be low enough. In [23], Park et al. presented an upper bound for the maximum differential probability for two rounds of an SPN structure, where the linear transformation can have any value as its branch number. This bound is found to be low for SPN structures. For instance, the maximum differential probability for four rounds of `AES` is bounded by 1.144×2^{-111}. Based on Park's result, we deduce an upper bound for the differential collision probability of `SPN-Hash`. We use this bound to show that our hash functions are secure against a differential collision attack. Furthermore for our internal permutations, we need to consider non-square byte-arrays of size $m \times n$ where $m \neq n$. The designers of AES [15] gave a construction for $m \times n$ arrays where $m < n$ using optimal diffusion

maps, but the differential bound is the same as that of an $m \times m$ array, which is sub-optimal for mn-byte block size. By their method, a 256-bit permutation would be constructed by a 4×8 byte-array that has the same differential bound 1.144×2^{-111} as a 4×4 byte-array. This is not close enough to $2^{-blocksize} = 2^{-256}$ for our security proof. **We generalize the optimal diffusion map of [15] to construct $m \times n$ byte-arrays where $m > n$, which can achieve near optimal differential bound close to $2^{-blocksize}$.**

We also analyzed the security of our internal permutations against the latest rebound-like attacks [25]. More precisely, we present distinguishing attacks on three versions of the internal permutation P for 8 out of 10 rounds. For the 256-bit permutation P, the 8-round attack requires time 2^{56} and memory 2^{16}. For the 512-bit permutation P, the 8-round attack requires time 2^{48} and memory 2^8, while for the 1024-bit permutation P, the 8-round attack requires time 2^{88} and memory 2^{16}.

Concerning the operating mode, we use the `JH` mode of operation [28], a variant of the Sponge construction [6]. In this design, assuming a block size of $2x$ bits, each x-bit input message block is XORed with the first half of the state. A permutation function P is applied, and the same message block is XORed with the second half of P's output. For this construction, the message blocks are mapped directly into the main permutation block structure instead of via a key schedule. **This eliminates the need to consider related-key differentials when analyzing protection against collision attacks**. Furthermore, the `JH` mode of operation is able to provide second preimage resistance of up to 2^x bits for an x-bit hash as compared to only $2^{x/2}$ for the Sponge construction with the same capacity.

To summarize, our `SPN-Hash` functions use `AES`-based internal permutations with fixed-key and a generalized optimal diffusion to ensure low and provable maximum differential probability. Then our `JH`-based operating mode allows us to apply directly our security reasoning and obtain a bound on the maximum probability of an attacker looking for collisions using a fixed input difference. To the best of our knowledge, **this is the only known function so far that provides such a security argument**.

The performances of `SPN-Hash` are good since the internal permutation is very similar to the one used in the `SHA-3` finalist `GRØSTL`. We propose a **novel construction to use parallel copies of the** `PHOTON` 8×8 **serialized MDS matrix over $GF(2^4)$ from [18], to create a secure and very lightweight byte-based diffusion for our design in hardware.**[1] Moreover, the area of `SPN-Hash` is also lowered by the relatively small internal memory required by the `JH` mode of operation. Hardware implementations require 4625 GE for 256-bit hash output, while current best results for the `SHA-3` competition finalists require 10000 GE or more. Overall, **our proposal achieves both excellent software speed and compact lightweight implementations**.

[1] Note that the approach of [18] to do an exhaustive search for serialized MDS matrix over $GF(2^8)$ by MAGMA is only feasible for $n \times n$ matrix up to size $n = 6$. Therefore we need our current approach to construct serailized 8×8 matrix over $GF(2^8)$.

Our paper is organized as follows: We state some necessary preliminaries concerning differential cryptanalysis in Section 2. Then we describe our proposed `SPN-Hash` design and give instantiations of 128-, 256-, and 512-bit `SPN-Hash` in Section 3 before proceeding to a summary of our security analysis results against differential collision, preimage, second preimage, and rebound attacks in Section 4. Lastly in Section 5, we show some performance comparisons.

2 Preliminaries

Substitution Permutation Network. One round of an SPN structure consists of three layers: key addition, substitution, and linear transformation. In the key addition layer, a round subkey is XORed with the input state. The substitution layer is made up of small non-linear substitutions called S-boxes implemented in parallel. The linear transformation layer is used to provide a good spreading effect of the cryptographic characteristics in the substitution layer. As such, the SPN structure has good confusion and diffusion properties [26]. One round of the SPN structure is shown in Figure 1 in Appendix A.

Maximum Differential Probability of an S-Box. In this paper, we follow the standard definitions related to differential cryptanalysis, such as those in [15]. We take all S-boxes to be bijections from $GF(2^s)$ to itself. Consider an SPN structure with an st-bit round function. Let each S-box S_i be an s-bit to s-bit bijective function $S_i : GF(2^s) \rightarrow GF(2^s)$, $(1 \leq i \leq t)$. So the S-box layer consists of t s-bit S-boxes in parallel.

Definition 1. *For any given* $\Delta x, \Delta y \in GF(2^s)$*, the* differential probability of each S_i *is defined as*

$$DP^{S_i}(\Delta x, \Delta y) = \frac{\#\{x \in GF(2^s) \mid S_i(x) \oplus S_i(x \oplus \Delta x) = \Delta y\}}{2^s},$$

where we consider Δx *to be the input difference and* Δy *the output difference.*

Definition 2. *The* maximal differential probability of S_i *is defined as*

$$DP((S_i)_{max}) = \underset{\Delta x \neq 0, \Delta y}{max}\, DP^{S_i}(\Delta x, \Delta y).$$

Definition 3. *The* maximal value of $DP((S_i)_{max})$ *for* $1 \leq i \leq t$ *is defined as*

$$p = \underset{1 \leq i \leq t}{max}\, (DP(S_i)_{max}).$$

An S-Box S_i is strong against differential cryptanalysis if $DP((S_i)_{max})$ is low enough, while a substitution layer is strong if p is low enough.

A *differentially active* S-box is an S-box having a non-zero input difference. A differentially active S-box always has a non-zero output difference and vice versa. In order to evaluate security against differential cryptanalysis, other than the differential probabilities of the S-box or S-box layer, one also has to consider

the number of active S-boxes whose value is determined by the linear transformation layer.

Substitution-Diffusion-Substitution Function. In order to ease the analysis of the SPN structure, we define an SDS (Substitution-Diffusion-Substitution) function as shown in Figure 2. Let the linear transformation layer of the SDS function be defined by L, its input difference by $\Delta x = x \oplus x^*$, its output difference by $\Delta y = y \oplus y^* = L(x) \oplus L(x^*)$. If L is linear, we have $\Delta y = L(\Delta x)$. The number of differentially active S-boxes on the input/output of the SDS function is given by the branch number of the linear transformation layer.

Definition 4. *The* branch number *of a linear transformation layer L is defined as*

$$\beta_d = \min_{v \neq 0} \{wt(v) + wt(L(v))\},$$

where the $wt(x)$ is the number of non-zero s-bit characters in x.

If we want to find the number of active S-boxes in two consective rounds of the SPN structure, we only need to consider the SDS function. β_d gives a lower bound on the number of active S-boxes in two consecutive rounds of a differential characteristic approximation.

Definition 5. *A linear transformation layer on t elements is* maximal distance separable (MDS) *if $\beta_d = t + 1$.*

Maximum differential Probability of an SPN. The differential probability, which is the sum of all differential characteristic probabilities with the same input and output difference, gives a more accurate estimate of resistance against differential cryptanalysis (than that of a single characteristic path). In [23], Park et al. proved an upper bound for the maximum differential probability for 2 rounds of the SPN structure.

Theorem 1. *[23, Theorem 1] Let L be the linear transformation of an SPN structure and β_d be the branch number of L from the viewpoint of differential cryptanalysis. Then the maximum differential probability for 2 rounds of the SPN structure is bounded by*

$$\max \left\{ \max_{1 \le i \le t} \max_{1 \le u \le 2^s - 1} \sum_{j=1}^{2^s-1} \{DP^{S_i}(u, j)\}^{\beta_d}, \max_{1 \le i \le t} \max_{1 \le u \le 2^s - 1} \sum_{j=1}^{2^s-1} \{DP^{S_i}(j, u)\}^{\beta_d} \right\}.$$

As a consequence, we get the following theorem.

Theorem 2. *[23, Corollary 1] Let L be the linear transformation of an SPN structure and β_d be the branch number of L from the viewpoint of differential cryptanalysis. Then the maximum differential probability for 2 rounds of the SPN structure is bounded by $p^{\beta_d - 1}$, where p is the maximal value of $DP((S_i)_{max})$ for $1 \le i \le t$.*

3 The SPN-Hash Functions

In this section we describe our proposed hash function design, SPN-Hash , with variable hash output sizes of 128, 256, and 512 bits. We adopt the JH mode of operation [28], a variant of the Sponge construction [6], operating on a state of $b = r + c$ bits. b is called the width, r the rate, and c the capacity. Our design is a simple iterated construction based on a fixed-length unkeyed permutation P, where $r = c$. The internal state of P can be represented by an $n \times m$ matrix of 8-bit cells, where n is the number of bytes in a bundle, and m is the number of bundles. Thus, P operates on a width of $b = 8nm$ bits, the rate and capacity are $4nm$-bit each, and the output is a $4nm$-bit hash value.

Firstly, the input message x of length N bits is padded and divided into blocks of $r = 4nm$ bits each. The padding function produces the padded message, x', of length a multiple of $4nm$. It follows "Padding Method 2" in [22, Algorithm 9.30]: first append the bit '1' to x, followed by a sequence of $z = (-N - 2nm - 1 \mod 4nm)$ '0' bits. Finally, append the $2nm$-bit representation of $l = (N + z + 2nm + 1)/4nm$. The integer l represents the number of message blocks in the padded message x'. The maximum message length for $4nm$-bit SPN-Hash is thus set as $4nm \cdot (2^{2nm} - 1) - 2nm - 1$.

Then, all the bits of the state are initialized to the value of an Initialization Vector (IV). The IV of $4nm$-bit SPN-Hash is taken to be the $8nm$-bit binary representation of $4nm$. That is, in big-endian notation, the IVs are 0x00...0080 for 128-bit SPN-Hash , 0x00...0100 for 256-bit SPN-Hash, and 0x00...0200 for 512-bit SPN-Hash .

For each padded message block, the JH mode of operation iteratively XORs the incoming $4nm$-bit input message block M_i into the left half of the state, applies the permutation $P : GF(2)^{8nm} \rightarrow GF(2)^{8nm}$ to the internal state and XORs M_i into its right half. After all the message blocks have been processed, the right half of the last internal state value is the final message digest and therefore our construction produces a $4nm$-bit hash. It is summarized as follows:

$$
\begin{aligned}
&Padded\ Input = M_0, M_1, \ldots, M_{N-1} \\
&(H_{0,L}, H_{0,R}) = IV \\
&\text{For } i = 0 \text{ to } N-1\text{:} \\
&\quad (H_{i+1,L}, H_{i+1,R}) = P((M_i \oplus H_{i,L}, H_{i,R})) \oplus (0, M_i) \\
&Hash = H_{N,R}
\end{aligned}
$$

where $M_i \in GF(2)^{4nm}, (H_{i,L}, H_{i,R}) \in GF(2)^{8nm}$ and N is the total number of padded message blocks. A diagram of our JH mode of operation is shown in Figure 3 in Appendix A.

Using appropriate parameters m and n such that m is even and m divides n, we will be able to construct a wide range of hash functions of different output sizes:

128-bit SPN-Hash : $m = 4, n = 8$
256-bit SPN-Hash : $m = 8, n = 8$
512-bit SPN-Hash : $m = 8, n = 16$

3.1 The Internal Permutation P

The $8nm$-bit permutation P iterates a round function for 10 rounds. Its internal state can be represented by an $n \times m$ matrix of 8-bit cells, where n is the number of bytes in a bundle, and m is the number of bundles. Here, each column can be viewed as a bundle consisting of n bytes. In each round, there is a substitution layer, followed by an MDS layer, a generalized optimal diffusion layer, and lastly, an XOR with a round constant. Thus, the linear transformation layer of the SPN structure introduced in Section 2 is actually a composition of the MDS layer and the generalized optimal diffusion layer while the "round keys" of the SPN structure are taken to be the round dependant constants. A diagram of the permutation function P is shown in Figure 4 in Appendix A.

The Substitution Layer σ. takes in a $8nm$-bit input and splits it into nm bytes. It then applies the `AES` 8-bit S-box [15] to each of these bytes in parallel. This is chosen due to its low maximum differential and linear approximation probabilities of 2^{-6}, which strengthens resistance against differential and linear attacks. In hardware, it is possible to achieve a very compact implementation of the `AES` S-box using "tower-field" arithmetic, as proposed in [12]. In software, one could use the Intel AES-NI instruction set [14] for efficient implementation.

The MDS Layer θ. combines consecutive n bytes into bundles and applies on each of these m bundles an MDS transformation described in Section 3.3.

The Generalized Optimal Diffusion Layer π. is a permutation of bytes that achieves good spreading effect. It is an instantiation of the generalized optimal diffusion which we define in Section 3.2. We write this layer π as $(\pi_1, \pi_2, \ldots, \pi_n)$, where $0 \leq \pi_i \leq m-1$. This notation indicates that row i is rotated by π_i positions to the left:

128-bit `SPN-Hash`: $\pi = (0, 0, 1, 1, 2, 2, 3, 3)$
256-bit `SPN-Hash`: $\pi = (0, 1, 2, 3, 4, 5, 6, 7)$
512-bit `SPN-Hash`: $\pi = (0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7)$

These byte permutations are indeed generalized optimal diffusions since exactly n/m bytes from each column is sent to each of the m columns.

The Round Constant $RCon_i$. that is XOR-ed with the state is different for every round i. This is to defend against slide attacks [10,11] and to prevent fixed points present over reduced rounds from being propagated to the entire permutation P. Each $RCon_i$ can be viewed as an $n \times m$ matrix A, where $A_{x,y}$ $(0 \leq x < n,\ 0 \leq y < m)$ denotes the entry in row x and column y. Then for $RCon_i$ used in round i,

$$A_{x,y} = \begin{cases} y \oplus i & \text{if } x = 0 \\ 0 & \text{otherwise,} \end{cases}$$

where i is the round number viewed as an 8-bit value. These values of round constants are chosen as they are light in hardware.

3.2 Generalized Optimal Diffusion

Definition 6. Generalized Optimal Diffusion*: Let m divide n. We consider a concatenation of n bytes as a* bundle *and we consider a concatenation of m bundles as a* block*. A linear transform, π, mapping a block of m bundles to m bundles is called a* (m, n)-generalized optimal diffusion *if for each input bundle of a block, n/m bytes of that input bundle is mapped to each of the m output bundles.*

Our (m, n)-generalized optimal diffusion is a generalization of the optimal diffusion layer used in the wide-trail strategy of Rijmen and Daemen [15]. The latter corresponds to the case $m = n$ and the ShiftRows function in `AES` is a particular instantiation of it. For our hash function design, m must be even and m must divide n.

The following two results compute the maximum differential probability of `SPN-Hash`. Their proofs can be found in Appendix **??**.

Theorem 3. *Let $\theta : [GF(2^8)^n]^m \to [GF(2^8)^n]^m$ be an MDS layer formed by concatenating m $n \times n$ MDS transforms over $GF(2^8)$. Let $\pi : [GF(2^8)^n]^m \to [GF(2^8)^n]^m$ be a (m, n)-generalized optimal diffusion mapping m bundles to m bundles. Then $\pi \circ \theta \circ \pi$ is a $m \times m$ MDS transform over $GF(2^{8n})$.*

Theorem 4. *The probability of any non-zero input-output differential for the internal permutation P described in Section 3.1 is upper bounded by*

$$\left(126 \times (2^{-7})^{n+1} + (2^{-6})^{n+1}\right)^m.$$

3.3 MDS Layer

The MDS layer provides an independent linear mixing of each column. In the following, we describe the mixing function of each column and show that it is indeed an MDS transform.

128- and 256-bit `SPN-Hash`. In [18], the authors proposed a method for generating the 8×8 MDS transform over $GF(2^4)$ in a serial way that is very compact. However, it is difficult to find an 8×8 serialized MDS matrix over $GF(2^8)$ using the exhaustive search method of [18]. Thus, we show here a way to construct such a matrix using two parallel copies of the `PHOTON` 8×8 serialized MDS matrix[2] over $GF(2^4)$ [18, Appendix C]. This method of construction, similar to

[2] We use `PHOTON` 's serialized matrix as we verified that it has the lowest binary weight over $GF(2^4)$.

the one used for the MDS layer of ECHO [5], produces an MDS transform that is very lightweight as compared to, for example, the 8×8 matrices[3] over $GF(2^8)$ used in WHIRLPOOL [4] or GRØSTL [17].

In what follows, we describe this MDS transform for 128- and 256-bit SPN-Hash. Label the 8 bytes in each column as $a_1, a_2, \ldots, a_8$. We may write each byte as a concatenation of two 4-bit values, $a_i = a_i^L \parallel a_i^R$. Let $a^L = (a_1^L, a_2^L, \ldots, a_8^L)$ and $a^R = (a_1^R, a_2^R, \ldots, a_8^R)$. Let Q be the 8×8 MDS matrix over $GF(2^4)$ used in the PHOTON [18, Appendix C] hash function, i.e.

$$Q = (A_{256})^8 = \begin{pmatrix} 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 2&4&2&11&2&8&5&6 \end{pmatrix}^8 = \begin{pmatrix} 2&4&2&11&2&8&5&6 \\ 12&9&8&13&7&7&5&2 \\ 4&4&13&13&9&4&13&9 \\ 1&6&5&1&12&13&15&14 \\ 15&12&9&13&14&5&14&13 \\ 9&14&5&15&4&12&9&6 \\ 12&2&2&10&3&1&1&14 \\ 15&1&13&10&5&10&2&3 \end{pmatrix}$$

The matrix Q is chosen as it can be implemented with a very low area footprint in hardware. This is due to the shifting property of A which simply updates the last cell of the column vector with a linear combination of all the vector components, and then rotates the vector by one position towards the top. The MDS layer is thus composed of 8 applications of A to the input column vector. This allows reuse of existing memory without need for temporary storage or additional control logic. Furthermore, the hash function using Q can be implemented efficiently in software using precomputed tables that combine the S-box and matrix coefficients.

We compute $b^L = Q \cdot a^L = (b_1^L, b_2^L, \ldots, b_8^L)$ and $b^R = Q \cdot a^R = (b_1^R, b_2^R, \ldots, b_8^R)$. For field multiplication over $GF(2^4)$, the irreducible polynomial $x^4 + x + 1$ is chosen with compactness as the main criterion. Then the output of the local diffusion layer is taken to be $(b_1, b_2, \ldots, b_8)$, where each b_i is a concatenation of the two 4-bit values, $b_i = b_i^L \parallel b_i^R$.

It can be shown that this transform is indeed MDS over $GF(2^8)$. Suppose the input a is non-zero. Then at least one of a^L or a^R is non-zero. Without loss of generality, suppose a^L is non-zero. Since Q is MDS, this means that the number of non-zero 4-bit values in (a^L, b^L) is at least 9. Hence, the number of non-zero bytes in (a, b) is at least 9.

512-Bit SPN-Hash. The choice of matrix for the 16×16 MDS is left open to the reader. One possibility is to use the matrix proposed by Nakahara *et al.* in [20]. Note that Nakahara *et al.*'s matrix may not be lightweight due to its large size necessitating a large number of primitive operations. However, this is not an issue since it is unlikely that a 512-bit hash function will be used for lightweight purposes.

[3] A comparison of their hardware estimations can be found in Section 5.2.

4 Security Analysis of SPN-Hash

In this section, we give a summary of our security analysis results against various attacks [4].

4.1 Differential Collision Attack

We analyzed the security of SPN-Hash against differential collision attacks. While some hash functions such as ECHO [5] do provide upper bounds on the maximum differential probability for a certain number of rounds, in actual fact, one has to *sum the maximum differential probability bound over all the possible colliding output differences* for a sharper estimation of the collision resistance of a hash function. To the best of our knowledge, no known hash function has yet provided such a collision resistance proof.

Let $\Delta Input$ denote the input differential and $\Delta Output$ be the output differential of the 4 last rounds of the SPN-Hash internal permutation. A collision for the hash function can occur either by an internal collision (a collision on the full $8nm$-bit internal state) or by an external collision during the last iteration (a collision on the right side of the $8nm$-bit internal state, the left side being truncated before outputing the hash value).

In the case of an external collision, this corresponds to P having an output differential of the form $(\Delta x, \Delta M) \in GF(2)^{4nm} \times GF(2)^{4nm}$, where XOR with the message difference ΔM in the right half will give a zero difference. Then we can show that $Pr(External\ Collision) < 2^{4nm} \times \left[(2^{-6n})(2^{-n} + 2^{-6})\right]^m < 2^{-2nm}$, where the complexity of a generic birthday attack is $2^{4nm/2} = 2^{2nm}$.

In the case of an internal collision on the $8nm$-bit permutation P, since there is no truncation, the differential probability is given by $Pr(\Delta Input \xrightarrow{4R} (0, \Delta M_i))$ for all possible message differences ΔM_i. By Theorem 4 and in the same way as the computation above, we can show that this is lower than 2^{-2nm}, the complexity of a generic birthday attack for the hash function.

Applying these bounds, we can conclude that the differential collision probabilities of 128-bit, 256-bit, and 512-bit SPN-Hash are upper bounded by $2^{-86.73} < 2^{-64}$, $2^{-173} < 2^{-128}$, and $2^{-303.99} < 2^{-256}$ respectively. This means that a differential collision attack will not perform better than a generic birthday attack.

In summary, we are able to show that SPN-Hash can provide good maximum differential probability upper bounds for 4 rounds of its internal permutation and that its operating mode allows us to go further to prove that the sum of all colliding differential probabilities is still much lower than what an attacker would get with a generic birthday collision attack.

4.2 (Second)-Preimage Attack

In the JH mode of operation, there is an XOR of the message in the right half at the end of the permutation to make the meet-in-the-middle (MITM)

[4] A full description of the security analysis can be found on the eprint archive at http://eprint.iacr.org.

attacks, originally applicable to the Sponge construction, invalid. This MITM attack on the Sponge construction can easily be modified into a second preimage attack, which is also defeated by the feedforward XOR in the JH mode of operation.

The preimage attack against the JH-512 hash function by Bhattacharyya et al. [7] has time and memory complexity 2^{507}, which remains a theoretical result because the complexity is of the same magnitude as brute force search. Moreover, a generic time-memory trade-off (TMTO) attack will perform much better with 2^{512} pre-computation complexity, 2^{507} memory, and 2^{10} time complexity.

4.3 Rebound Attack - Distinguishing Attack on Permutation P

We analyzed the security of our internal permutations against the latest rebound-like attack [25] which uses the non-full-active Super S-box cryptanalysis technique. More precisely, we present distinguishing attacks on three versions of the internal permutation P for 8 out of 10 rounds. For the 256-bit permutation P, the 8-round attack requires time 2^{56} and memory 2^{16}. For the 512-bit permutation P, the 8-round attack requires time 2^{48} and memory 2^{8}, while for the 1024-bit permutation P, the 8-round attack requires time 2^{88} and memory 2^{16}. In comparison, the complexity of attacking an ideal permutation is 2^{64}, 2^{96} and 2^{256} for 256-bit, 512-bit, and 1024-bit permutations P respectively.

To the best of our knowledge, the differential paths presented are among the longest paths with the least complexities. Since P comprises ten round functions, the distinguishing attacks do not threaten the security of the hash function.

4.4 Exploiting the MDS Layer Structure for 128- and 256-bit SPN-Hash

Since the MDS layer is built by applying a diffusion matrix over $GF(2^4)$ two times independently, an attacker could try to exploit this special structure in a truncated differential attack by forcing the differences at some stage to remain on the left or on the right side of the bytes processed. This observation was used in the first attacks [24] on the ECHO hash function.

However, we believe such a strategy would very likely fail because this right/left property would be destroyed by the application of the AES S-box. Alternatively, forcing this property to be maintained for each active byte through the S-box layer would imply a big cost for the attacker, bigger than the gain from the truncated differential transitions. This has been confirmed by experiments, as there is no strong bias through the AES S-box in order to reach an all-right or an all-left difference (forward or backward). In the case of ECHO this is not true since the 128-bit ECHO S-box is implemented by two AES rounds, for which forcing good truncated differential paths at no cost is easy.

5 Implementation

5.1 Software Performance

Due to its similarity with GRØSTL concerning the construction of the internal permutation, it is interesting to analyze SPN-Hash's software speed in the light of this SHA-3 candidate. The internal permutation of 256-bit SPN-Hash is comparable with GRØSTL-256 since their (individual) internal permutation are of the same size, and their amount of message bits per call to the internal permutation is the same (256-bit SPN-Hash compression function processes 256 bits message in 10 round operations, compared to 512-bit message in 20 very similar round operations by GRØSTL-256). The round function of these two hash functions should take about similar amount of time due to: 1) The equal number of substitution operations using the same AES sbox; 2) The speed of MDS multiplication is independent of the MDS coefficients in most table-based implementations; 3) The ShiftByte in GRØSTL is done together with step 2 in table-based implementations; 4) The round constants are a bit simpler than those in GRØSTL; and 5) There are three times $\oplus$ for 512 bits in GRØSTL and twice 256-bit $\oplus$ in 256-bit SPN hash. We did a simple and unoptimized implementation based on table lookups, which turns to be 34 cycles per byte on a Intel(R) Xeon(R) CPU E5640 clocked at 2.67GHz. We believe that there remains an important room for improvements by implementing SPN-Hash with optimized assembly instructions. Similar comparison argument applies when one considers implementations with the AES new instruction set, while GRØSTL-256 runs at 12 cycles per byte with internal parallelization of the two permutations P and Q, we expect SPN-Hash-256 to run at 12 to 24 cycles per byte mostly due to the fact that similiar parallelization is not possible. Note that the 128-bit SPN-Hash shall run as fast as the 256-bit version, since its compression function takes half the message bits, and uses roughly half the amount of operations. Test vectors are provided in Appendix Table 2.

5.2 Hardware Performance

We have implemented 128-bit and 256-bit SPN hash in VHDL and used *Synopsys DesignCompiler A-2007.12-SP1* to synthesize it to the *Virtual Silicon* (VST) standard cell library *UMCL18G212T3*, which is based on the *UMC L180 0.18μm 1P6M* logic process with a typical voltage of 1.8 V. We used *Synopsys Power Compiler* version *A-2007.12-SP1* to estimate the power consumption of our ASIC implementations. For synthesis and for power estimation we advised the compiler to keep the hierarchy and use a clock frequency of 100 KHz.

Table 1 in Appendix A compares our implementations of SPN hash with the remaining five SHA-3 candidates with regards to area, latency and a FOM proposed by [2]. In order to have a fair comparison, we only include figures for fully-autonomous low-area ASIC implementations and omit figures for implementations that are optimized for high throughput. Among the SHA-3 candidates BLAKE, GRØSTL, and SKEIN, 256-bit SPN-Hash is by far the most compact proposal. Though it has only the second highest FOM, our estimates for a 64-bit

datapath implementation indicate that it can achieve the highest FOM, while still being 35% smaller than the most compact SHA-3 candidate.

References

1. Augot, D., Finiasz, M., Gaborit, P., Manuel, S., Sendrier, N.: SHA-3 Proposal: FSB. Submission to NIST (2008)
2. Aumasson, J.-P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A Lightweight Hash. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 1–15. Springer, Heidelberg (2010), http://131002.net/quark/
3. Aumasson, J.-P., Henzen, L., Meier, W., Phan, R.C.-W.: SHA-3 Proposal BLAKE. Candidate to the NIST Hash Competition (2008), http://131002.net/blake/
4. Barreto, P., Rijmen, V.: The Whirlpool Hashing Function, http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html
5. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 Proposal: ECHO. Submission to NIST (2009) (updated)
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge Functions. In: ECRYPT Hash Workshop (2007)
7. Bhattacharyya, R., Mandal, A., Nandi, M.: Security Analysis of the Mode of JH Hash Function. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 168–191. Springer, Heidelberg (2010)
8. Billet, O., Robshaw, M.J.B., Peyrin, T.: On Building Hash Functions from Multivariate Quadratic Equations. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) ACISP 2007. LNCS, vol. 4586, pp. 82–95. Springer, Heidelberg (2007)
9. Biryukov, A., Khovratovich, D., Nikolić, I.: Distinguisher and Related-Key Attack on the Full AES-256. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 231–249. Springer, Heidelberg (2009)
10. Biryukov, A., Wagner, D.: Slide Attacks. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer, Heidelberg (1999)
11. Biryukov, A., Wagner, D.: Advanced Slide Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 589–606. Springer, Heidelberg (2000)
12. Canright, D.: A Very Compact S-Box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Heidelberg (2005); The HDL specification is available at the author's official webpage http://faculty.nps.edu/drcanrig/pub/index.html
13. Contini, S., Lenstra, A.K., Steinfeld, R.: VSH, an Efficient and Provable Collision-Resistant Hash Function. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 165–182. Springer, Heidelberg (2006)
14. Intel Corporation. Advanced Encryption Standard (AES) Instruction Set (October 30, 2008), http://softwarecommunity.intel.com/articles/eng/3788.htm
15. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
16. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family. Submission to NIST, Round 2 (2009)
17. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schlaffer, M., Thomsen, S.S.: Grøstl addendum. Submission to NIST (2009) (updated)

18. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)
19. Henzen, L., Aumasson, J.-P., Meier, W., Phan, R.C.W.: VLSI Characterization of the Cryptographic Hash Function BLAKE. IEEE Transactions on Very Large Scale Integration (VLSI) Systems (99), 1–9
20. Nakahara Jr., J., Abrahão, É.: A New Involutary MDS Matrix for AES. International Journal of Network Security 9(2), 109–116 (2009)
21. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 126–143. Springer, Heidelberg (2009)
22. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996)
23. Park, S., Sung, S.H., Lee, S., Lim, J.: Improving the Upper Bound on the Maximum Differential and the Maximum Linear Hull Probability for SPN Structures and AES. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 247–260. Springer, Heidelberg (2003)
24. Peyrin, T.: Improved Differential Attacks for ECHO and Grøstl. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 370–392. Springer, Heidelberg (2010)
25. Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., Ohta, K.: Non-full-active Super-Sbox Analysis: Applications to ECHO and Grøstl. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 38–55. Springer, Heidelberg (2010)
26. Shannon, C.: Communication Theory of Secrecy System. Bell System Technical Journal 28, 656–715 (1949)
27. Tillich, S., Feldhofer, M., Issovits, W., Kern, T., Kureck, H., Mühlberghuber, M., Neubauer, G., Reiter, A., Köfler, A., Mayrhofer, M.: Compact Hardware Implementations of the SHA-3 Candidates Arirang, Blake, Grøstl, and Skein. IACR ePrint archive, Report 2009/349 (2009)
28. Wu, H.J.: The Hash Function JH. Submission to NIST (September 2009) (updated), `http://ehash.iaik.tugraz.at/uploads/1/1d/Jh20090915.pdf`

A Tables and Figures

Table 1. Comparison of Low-Area Hardware implementations of SPN hash and a selection of SHA-3 finalists

Digest size	Alg.	Ref.	Msg. size	Technology	Area [GE]	Latency [clk]	T'put@100KHz [kbps]	FOM [nbps/GE^2]
128	SPN-Hash-128		256	UMC 0.18	2 777	710	36.1	2 338
	SPN-Hash-128		256	*estimate*	4 600	230	55.7	2 627
256	SPN-Hash-256		512	UMC 0.18	4 625	1 430	35.8	837
	SPN-Hash-256		512	*estimate*	8 500	230	111.3	1 541
	BLAKE-32	[19]	512	UMC 0.18	13 575	816	62.8	340
	GRØSTL-224/256	[27]	512	AMS 0.35	14 622	196	261.2	1 222
	SKEIN-256-256	[27]	256	AMS 0.35	12 890	1 034	24.8	149

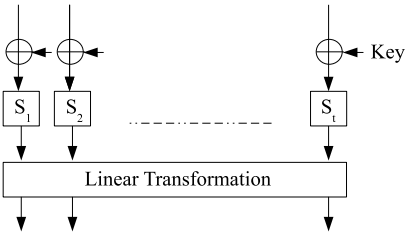

Fig. 1. One round of a SPN structure

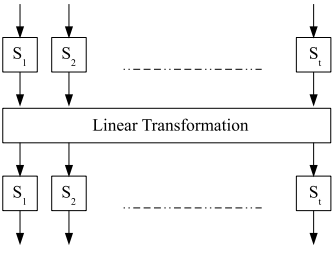

Fig. 2. The SDS function

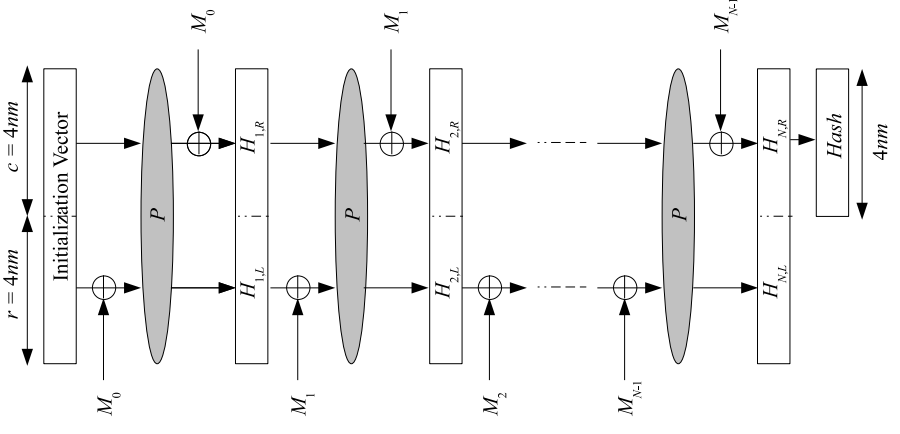

Fig. 3. The JH mode of operation

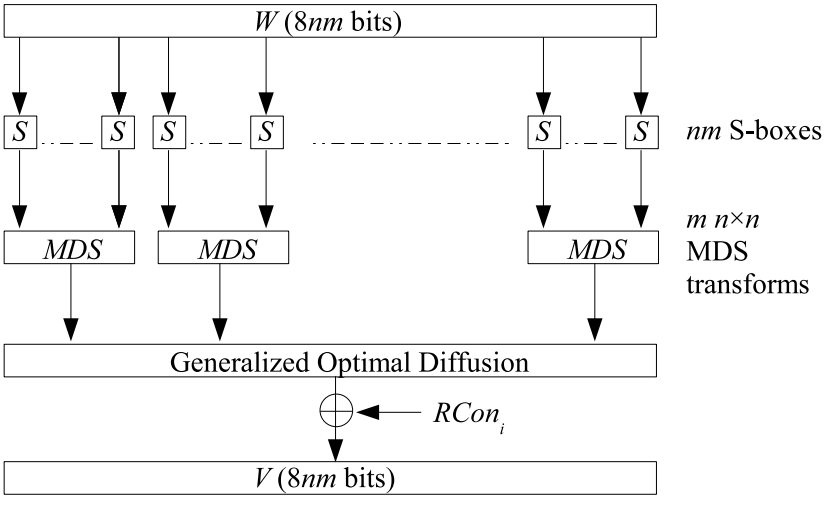

Fig. 4. The round function in permutation P

B Test Vector

We hash the message "SPN-Hash: Improving the Provable Resistance Against Differential Collision Attacks" with three variants of the `SPN-Hash` family, and the following are digests generated by our reference implementation.

Table 2. Test vectors for three variants of `SPN-Hash` family

SPN-Hash-128	2b021df78220afd2a41fa3592dc7d284
SPN-Hash-256	eabd18110d48e81d0663a7034b265462bf93f8019ca292e58ec1d830f90d67c5
SPN-Hash-512	f3e4a3dcc44acb2cf4d6f5f67bd8ce50ef030f55e0189a322136b5fc46af3cf5 e071f1ee9bf1851bbd854540da1ccc496d679b43090f8e24f486d6866092ac02