Computer Security Symposium 2016
11 - 13 October 2016

Parallelizable Message Preprocessing for Merkle-Damgård Hash Functions

Hidenori Kuwakado[1,a)] Shoichi Hirose[2] Masahiro Mambo[3]

Abstract: Since well-known hash functions sequentially process a message, the time for computing a digest strongly depends on the performance of a single processor. That is, even if multi-core processors are available, it is difficult to reduce the time. This paper proposes a message preprocessing that contributes to a reduction of the time for computing a digest. The message preprocessing is a provable collision-resistant hash function in the sense that the collision resistance can be reduced to the hardness of lattice problems. Furthermore, it can be efficiently computed by massively parallel processing. The throughput of the message processing is evaluated via its implementation on graphics processing units (GPUs).

Keywords: hash function, preprocessing, lattice problem, graphics processing unit

1. Introduction

Increasing the number of cores has been the trend in processor development since the performance gains from increasing the operating frequency diminish greatly. Furthermore, most modern processors support single instruction multiple data (SIMD) instructions in order to improve the performance of multimedia use.

Hash functions are a primitive for achieving secure encryption and secure authentication. Well-known hash functions such as the SHA-2 family are serial processing, that is, message blocks are processed in the sequential order. Hence, the throughput of hash functions strongly depends on the performance of a single processor (a single core). It follows that the throughput of hash functions is hardly improved by increasing the number of processors and supporting SIMD instructions.

The objective of this paper is to solve this problem without changing the design of existing hash functions. We are concerned with existing hash functions because their cryptographic security has been extensively studied and their existing software/hardware implementations are available. Accordingly, we propose that a message is irreversibly compressed prior to input to the existing hash functions, that is, a message preprocessing. To achieve the objective, the message preprocessing is designed so that it is suitable for parallel processing to use modern processors fully. We show that a function proposed by Ajtai et al. [1], [3] (an *AGGH function*) satisfies such conditions. The algorithm of the AGGH function is suited to graphics processing units (GPUs). In our experiments, the throughput of SHA-512 with the AGGH function is at most 3.11 times faster than that of the original SHA-512. In the case of Tegra TK1, which is an embedded processor with a GPU, the throughput is improved about 1.33 times. The cryptographic security of the AGGH function has been proved in the sense that there are reductions to an NP-hard lattice problem. However, there are few cryptographic applications of the AGGH function since the AGGH function is a linear function. The message preprocessing of hash functions is a promising application of the AGGH function.

Related Work Although parallelizable hash functions have been shown in articles [4], [10], it seems that they are not widely used. Gueron and Krsnov [5] have proposed an algorithm for accelerating the computations of Davies-Meyer hash function. This algorithm is based on the computation of several message schedules for several message blocks by using SIMD instructions. This algorithm produces message blocks that are identical to those produced by the original message schedule. They also studied simultaneous hashing for multiple messages [6]. Note that both of the papers are for multiple messages, whereas this paper is for a single message. Szydlo and Yin [11] have improved a collision-resistant property of hash functions by message preprocessing. This is for improving the security and is not for improving the throughput.

Lyubashevsky et al. [8] have proposed a SWIFFT function that is based on the hardness of a lattice problem. The SWIFFT function overcomes drawbacks of the AGGH function on the efficiency. The comparison of the AGGH function with the SWIFFT function is described in the later section. Györfi et al. [7] have shown hardware architecture for a SWIFFT function. The SWIFFT function was used

[1] Kansai University
[2] University of Fukui
[3] Kanazawa University
[a)] kuwakado@kanasai-u.ac.jp

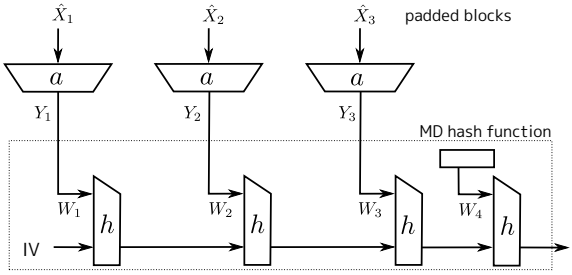

Fig. 1 A new construction ((K_h, K_a) is omitted).

as a primitive of SWIFFTX, which was one of candidates in the SHA-3 competition.

Organization Section 2 describes the specification of the Merkle-Damgård hash function with a message preprocessing function. After requirements to the message preprocessing function are mentioned, it is shown that the AGGH function fills the requirements. Section 3 shows the throughput of SHA-512 with the AGGH function which is implemented on GPUs. The throughput can be improved by using the AGGH function. Section 4 proves that the Merkle-Damgård hash function with a message preprocessing function is collision-resistant if the compression function of the underlying Merkle-Damgård hash function and the message preprocessing function are collision-resistant. Section 5 concludes this paper.

2. Parallelizable Preprocessing

2.1 New Construction

Let h be a family of functions $h : \mathcal{K}_h \times \{0,1\}^{b+v} \to \{0,1\}^v$ and let a be a family of functions $a : \mathcal{K}_a \times \{0,1\}^{\lambda} \to \{0,1\}^b$. The value of b/λ is called a *compression rate* of a, which is denoted by α ($0 < \alpha < 1$). Let $\mathcal{K}$, $\mathcal{D}$, and $\mathcal{R}$ be

$$\mathcal{K} = \mathcal{K}_h \times \mathcal{K}_a, \quad \mathcal{D} = \bigcup_{i=0}^{\frac{\lambda}{b}2^{\ell}-1} \{0,1\}^i, \quad \mathcal{R} = \{0,1\}^v,$$

respectively. For a key $(K_h, K_a) \in \mathcal{K}$ and a message $X \in \mathcal{D}$, a family of functions $G((K_h, K_a), X) : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ is defined as follows (Fig. 1):

(1) Choose (K_h, K_a) from $\mathcal{K}$ accoding to the uniform distribution on $\mathcal{K}$.

(2) Append a single '1' to X, and then append '0' so that the length of the padded message becomes a multiple of λ. Parse it as n λ-bit blocks, denoted by $\hat{X}_1 \parallel \hat{X}_2 \parallel \ldots \parallel \hat{X}_n$.

(3) For $i = 1, 2, \ldots, n$, compute $Y_i = a(K_a, \hat{X}_i)$ where the bit length of Y_i is b.

(4) For $i = 1, 2, \ldots, n+1$, assign W_i as

$$W_i = \begin{cases} Y_i & i = 1, 2, \ldots, n \\ 1 \parallel 0^{b-1-\ell} \parallel (nb)_2 & i = n+1 \end{cases}$$

where $(nb)_2$ means an ℓ-bit binary representation of nb, which is the bit length of $W_1 \parallel W_2 \parallel \ldots \parallel W_n$.

(5) After a v-bit fixed initial vector (IV) is assigned to V_0, compute $V_i = h(K_h, W_i \parallel V_{i-1})$ for $i = 1, 2, \ldots, n+1$.

(6) Output V_{n+1} ($\in \mathcal{R}$) as a digest of X.

If $Y_1 \parallel Y_2 \parallel \ldots \parallel Y_n$ is regarded as a message, then the procedure from (4) to (6) is identical to the Merkle-Damgård hash function. The family of functions G is considered as the Merkle-Damgård hash function with a message preprocessing function a.

2.2 Conditions on the Message Preprocessing Function

The objective of a message preprocessing is to increase the throughput of computing a digest. Let t_a be the throughput of the message preprocessing function a (e.g., [MiB/s]) and let t_h be that of the compression function h. The following inequality has to hold to achieve the objective.

$$\frac{\lambda}{t_a} + \frac{b}{t_h} < \frac{\lambda}{t_h}, \tag{1}$$

where the right-hand side is the time for computing h for an input with length λ and the left-hand side is that for computing the composite function $h \circ a$ for an input with length λ. Rearranging Eq. (1) gives

$$\frac{\lambda}{\lambda - b} t_h < t_a. \tag{2}$$

By using the compression rate α, Eq. (2) is transformed to

$$\frac{1}{1-\alpha} t_h < t_a.$$

The message preprocessing function a has to be at least $1/(1-\alpha)$ times faster than the compression function h.

On the security, Theorem 1 in Sect. 4 will show that the family of functions G is collision-resistant if both of the compression function h and the message preprocessing function a are collision-resistant.

2.3 The AGGH Function

We here discuss a function proposed by Ajtai, Goldreich, Goldwasser, and Halevi [1], [3] (called an *AGGH function*) as the message preprocessing function a.

Let q be a positive integer, $\mathbb{Z}_q$ be the set of all integers modulo q, and $\mathbf{A}$ be a $\mu \times \lambda$ random matrix with entries in $\mathbb{Z}_q$. For an input $X \in \{0,1\}^{\lambda}$, the AGGH function is defined as

$$Y = a(\mathbf{A}, X) = \mathbf{A} \cdot X \bmod q, \tag{3}$$

where $Y \in \mathbb{Z}_q^{\mu}$. Since Y is the input to h, μ and q are chosen so that the bit length of Y is equal to b (i.e., $\mu \lg(q) = b$)[*1]. Since $\mathbf{A}$ is chosen uniformly at random from a set of all the $\mu \times \lambda$ matrices, $\mathbf{A}$ itself plays a role of K_a.

The AGGH function has the following cryptographic properties.

- Ajtai [1] has proved that when $\mathbf{A}$ is chosen uniformly at random, the average-case complexity of inverting the AGGH function is at least as hard as the worst-case

[*1] lg denotes the logarithm of base 2 or the number of bits for expressing a given number.

Table 1 The number of operations.

Operation	SHA-512
bitwise NOT	80
bitwise AND	400
bitwise inclusive OR	736
bitwise exclusive OR	816
shift	1600
modular addition	752
Total	4384

complexity of approximating an NP-hard lattice problem.

- Goldreich, Goldwasser, and Halevi [3] have proved that when $\mathbf{A}$ is chosen uniformly at random, it is computationally hard to find $X^{(1)}, X^{(2)}$ such that $a(\mathbf{A}, X^{(1)}) = a(\mathbf{A}, X^{(2)})$, that is, the AGGH function is collision-resistant.

Despite of these remarkable properties, the AGGH function itself is not suitable for cryptographic applications because the AGGH function is a linear function.

Next, we discuss the throughput of the AGGH function when the SHA-512 compression function is used as the compression function h. Equation (3) requires at most $\mu(\lambda - 1)$ additions over $\mathbb{Z}_q$. For example, suppose that $b = 1024$, $q = 256$, and $\alpha = 1/8$. Equation (3) requires at most 1048448 $(= 2^7(2^{13} - 1))$ additions because

$$\lg(q) = \lg(256) = 2^3,\ \mu = \frac{b}{\lg(q)} = \frac{1024}{2^3} = 2^7,$$
$$\lambda = \frac{b}{\alpha} = 1024 \cdot 8 = 2^{13},$$

and the SHA-512 compression function requires 4384 operations as shown in Table 1[*2]. Their comparison suggests that the throughput of the AGGH function is much lower than that of SHA-512 when they are sequentially computed.

However, the computation of Eq. (3) can be completed in the time of $\lg(\lambda - 1)$ additions if an ideal parallel processing is possible. We can expect that the throughput of the AGGH function increases considerably by using massively parallel processing units such as graphics processing units (GPUs).

Another drawback of Eq. (3) is the memory size for $\mathbf{A}$, which is $\mu\lambda\lg(q)$ bits. For example, suppose that $b = 1024$, $q = 256$, and $\alpha = 1/8$. The size of $\mathbf{A}$ is about 2^{23} [bits], and the size of constants in SHA-512 is 5632 $(\approx 2^{12.45})$[bits]. Thus, the AGGH function requires larger memory than the SHA-512 compression function.

In fact, above-mentioned drawback of the AGGH function (i.e., the low throughput and the large memory) have been known. To overcome drawbacks, Lyubashevsky et al. [8] have proposed a SWIFFT function that is based on the following idea.

- A skew-circulant random matrix is used as $\mathbf{A}$.
- A prime is used as q.

The size of the skew-circulant random matrix is $1/\mu$ times smaller than that of the random matrix. Owing to the skew-circulant matrix and the prime q, the function that is equivalent to Eq. (3) is efficiently performed with the fast Fourier transform. However, this paper does not use the SWIFFT function, and uses the AGGH function for the following reasons.

- The reduction to lattice problems is unclear when the skew-circulant random matrix is used.
- To use single instruction multiple data (SIMD) instructions for the addition over $\mathbb{Z}_q$, q has to be 2^8 or 2^{16}.

[*2] The circular shift operation is achieved by two shift operations and one bitwise inclusive OR operation.

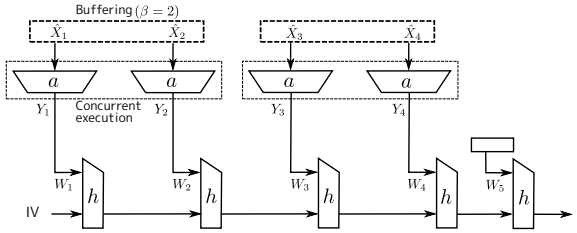

Fig. 2 The preprocessing with message buffering.

Table 2 Throughput of the AGGH function. [MiB/s]

GPU	α		β
	1/8	1/16	
Tegra TK1	66.0	66.8	128
Tegra TX1	102.8	112.0	256
GTX 960M	278.9	284.3	512
GTX 690 (single)	685.5	698.6	1024
GTX TITAN X	1015.6	1010.7	2048

3. Implementation

3.1 Message Buffering

GPUs have more cores than necessary to compute the single AGGH function. For example, NVIDIA GeForce GTX TITAN X, which is a high-end GPU, contains 3072 cores. Buffering a sufficiently long message allows a GPU to compute a number of the AGGH functions independently. Consequently, the throughput is probably improved.

Suppose that a GPU computes β AGGH functions concurrently after buffering β blocks (Fig. 2). Let $t_a(\beta)$ be the throughput (e.g., [MiB/s]) of the AGGH function in such a case and let t_h be the throughput of the compression function. Then, since the input size is $\beta\lambda$ and the time is given by

$$\frac{\beta\lambda}{t_a(\beta)} + \frac{\beta b}{t_h} = \frac{\beta(\lambda t_h + b t_a(\beta))}{t_a(\beta) t_h},$$

the throughput is given by

$$\frac{\beta\lambda \cdot t_a(\beta) t_h}{\beta(\lambda t_h + b t_a(\beta))} = \frac{t_a(\beta) t_h}{t_h + \alpha t_a(\beta)}$$

where $\alpha\ (= b/\lambda)$ denotes the compression rate. Hence, the throughput is $t_a(\beta)/(t_h + \alpha t_a(\beta))$ times faster than that of the underlying Merkle-Damgård hash function owing to the message preprocessing with buffering.

3.2 Results

The throughput was measured for a sufficiently long message under the following parameters: $b = 1024$, $q = 256$. The throughput was measured in such a way that the time

Table 3 Throughput of SHA-512. [MiB/s]

CPU	Throughput
Tegra TK1 (ARM Cortex A15)	35.9
Tegra TX1 (ARM A57)	97.3
Intel Core i7-2600S	199.3
Intel Core i7-6700HQ	206.4
Intel Core i5-3570	255.5
Intel Xeon E3-1275 V2 [2]	411.5

for transferring data between a CPU and a GPU through the PCI Express bus was included.

Table 2 shows the throughput of the AGGH function with buffering that was implemented on NVIDIA GPUs. In Table 2, α and β denotes the compression rate and the number of buffering blocks. Their codes were written in CUDA 7.0 or CUDA 5.5 and used the SIMD instruction that operates on quads of a byte[*3]. The throughput was the highest when the compression rate and the number of buffering blocks were values shown in the Table 2. When β is fixed, the above discussion showed that smaller α gave better throughput. However, the throughput $t_a(\beta)$ actually depends on not only β but also α.

Table 3 shows the throughput of the SHA-512 compression function that was implemented on ARM CPU or Intel CPU. Their codes are written in standard C. The throughput of Intel Xeon E3-1275 V2 in Table 3 is an excerpt from eBASH [2].

Tegra TK1 is a system on a chip (SoC) for GPU-accelerated parallel processing in embedded systems. Owing to the message preprocessing, the throughput on Tegra TK1 is about 1.66 ($= 66.8/(35.9 + 66.8/16)$) times faster than that of SHA-512. Using GTX TITAN X, which is usually equipped to personal computers of Intel CPU, the throughput is about 1.88 to 3.11 times faster than that of SHA-512.

4. Security Proof

4.1 Definition

A hash function is a family of functions $H : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ where the domain $\mathcal{D}$ is a wider space than the range $\mathcal{R}$. Let K be a value sampled from the uniform distribution on $\mathcal{K}$. Then, $H_K : \mathcal{D} \to \mathcal{R}$ is defined as

$$H_K(M) = H(K, M) \tag{4}$$

for all $M \in \mathcal{D}$.

Definition 1 Let $H : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be a hash function and let A be an algorithm (adversary). The advantage of A with respect to collision resistance is defined by

$$\mathbf{Adv}_H^{\mathrm{CR}}(A) = \Pr\left[\mathbf{Exp}_H^{\mathrm{CR}}(A) = 1\right] \tag{5}$$

where $\mathbf{Exp}_H^{\mathrm{CR}}(A)$ is defined as follows:

(1) Choose K from $\mathcal{K}$ according to the uniform distribution on $\mathcal{K}$ and give K to A.
(2) Let $(X^{(1)}, X^{(2)})$ be the output of $A(K)$.
(3) If $X^{(1)}, X^{(2)} \in \mathcal{D} \wedge X^{(1)} \neq X^{(2)} \wedge H_K(X^{(1)}) = H_K(X^{(2)})$, then return 1. Otherwise, return 0.

Suppose that the size of $\mathcal{K}$ is exponential in ι. If there exists a negligible function $\mathrm{neg}(\iota)$ such that $\mathbf{Adv}_H^{\mathrm{CR}}(A) \leq \mathrm{neg}(\iota)$ for all probabilistic polynomial-time algorithms A in ι, H is called a collision-resistant hash function.

[*3] All the GPUs in Table 2 support such SIMD instructions. When no SIMD instructions are used, for example, the throughput of Tegra TK1 at $\alpha = 1/16$ is 41.7 [MiB/s].

4.2 Proof of Collision Resistance

We evaluate the advantage of an adversary A_G against G with respect to collision resistance. The theorem to be proved is as follows:

Theorem 1 Let h be a family of functions $\mathcal{K}_h \times \{0,1\}^{b+v} \to \{0,1\}^v$ and a be a family of functions $\mathcal{K}_a \times \{0,1\}^{\lambda} \to \{0,1\}^b$. A family of functions $G : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ is constructed from h and a in a manner of Sect. 2.1. If there exists an adversary A_G that can find collisions in G, then there exists an adversary A_h that can find collisions in h or an adversary A_a that can find collisions in a such that

$$\mathbf{Adv}_G^{\mathrm{CR}}(A_G) \leq \mathbf{Adv}_h^{\mathrm{CR}}(A_h) + \mathbf{Adv}_a^{\mathrm{CR}}(A_a). \tag{6}$$

The running time of A_h or that of A_a is dominated by the sum of the following three items.

- the running time of A_G
- the time to compute $G(K, X^{(1)})$ and $G(K, X^{(2)})$ where $(X^{(1)}, X^{(2)})$ is the collision in G found by A_G
- the time to sort message blocks of $X^{(1)}, X^{(2)}$

We introduce Lemma 1 for proving Theorem 1. The proof of Lemma 1 is described in Appendix A.1.

Lemma 1 Let h and G denote families of functions in Theorem 1. If there exists an adversary A_G that can find collisions in G, then there exists an adversary A_h that can find collisions in h such that

$$\Pr\left[\mathbf{Exp}_h^{\mathrm{CR}}(A_h) = 1\right] \geq \Pr\left[\mathtt{col}_h | \mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right] \cdot \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right], \tag{7}$$

where $\mathtt{col}_h$ denotes the event such that a collision found by A_G results in a collision in h. The running time of A_h is dominated by that of A_G and the time to compute $G(K, X^{(1)})$ and $G(K, X^{(2)})$ where $(X^{(1)}, X^{(2)})$ is the collision in G found by A_G. Similarly, there exists an adversary A_a that can find collisions in a such that

$$\Pr\left[\mathbf{Exp}_a^{\mathrm{CR}}(A_a) = 1\right] \geq \Pr\left[\mathtt{col}_a | \mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right] \cdot \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right], \tag{8}$$

where $\mathtt{col}_a$ denotes the event such that a collision found by A_G results in a collision in a. The running time of A_a dominated by that of A_G, the time to compute $G(K, X^{(1)})$ and $G(K, X^{(2)})$, where $(X^{(1)}, X^{(2)})$ is the collision in G found by A_G and the time to sort message blocks of $X^{(1)}, X^{(2)}$.

We here prove Theorem 1.

Proof When $\mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1$, an event $\mathtt{col}_h \vee \mathtt{col}_a$ is a certain event. That is,

$$\Pr\left[\mathtt{col}_h \vee \mathtt{col}_a | \mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right] = 1. \tag{9}$$

Since $\mathtt{col}_h$ and $\mathtt{col}_a$ are not always disjoint, the following

inequality holds.

$$\begin{aligned}&\Pr\left[\mathtt{col}_h \vee \mathtt{col}_a | \mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right] \\ &\leq \Pr\left[\mathtt{col}_h | \mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right] + \Pr\left[\mathtt{col}_a | \mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right]\end{aligned} \tag{10}$$

Adding Eq. (7) to Eq. (8) gives

$$\begin{aligned}&\Pr\left[\mathbf{Exp}_h^{\mathrm{CR}}(A_h) = 1\right] + \Pr\left[\mathbf{Exp}_a^{\mathrm{CR}}(A_a) = 1\right] \\ &\geq \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right] \\ &\cdot \left(\Pr\left[\mathtt{col}_h | \mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right] + \Pr\left[\mathtt{col}_a | \mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right]\right) \\ &\geq \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right] \\ &\quad \cdot \Pr\left[\mathtt{col}_h \vee \mathtt{col}_a | \mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right] \quad (\because \text{Eq. (10)}) \\ &\geq \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G) = 1\right]. \quad (\because \text{Eq. (9)})\end{aligned}$$

From Definition 1, the inequality above is transformed as

$$\mathbf{Adv}_h^{\mathrm{CR}}(A_h) + \mathbf{Adv}_h^{\mathrm{CR}}(A_a) \geq \mathbf{Adv}_G^{\mathrm{CR}}(A_G).$$

The running time of A_h and that of A_a are the same as that described in Lemma 1. □

5. Conclusions

This paper has proposed the Merkle-Damgård hash function with the message preprocessing that meets the trend in processor development, and has shown that the AGGH function is suitable for the message preprocessing. Our measuring results showed that the AGGH function was efficiently computed with GPUs and the throughput was improved owing to the message preprocessing of the AGGH function. This paper has also proved that the Merkle-Damgård hash function with the message preprocessing is collision-resistant if the compression function of the Merkle-Damgård hash function and the message preprocessing function are collision-resistant.

A hash-based message authentication code (HMAC) is an important application of the Merkle-Damgård hash function [9]. It is future work to analyze the security of HMAC that is based on the Merkle-Damgård hash function with the message preprocessing.

References

[1] Ajtai, M.: Generating Hard Instances of Lattice Problems, *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, STOC '96*, pp. 99–108 (1996).

[2] eBASH, `http://bench.cr.yp.to/results-hash.html`, 2016.

[3] Goldreich, O., Goldwasser, S. and Halevi, S.: *Collision-Free Hashing from Lattice Problems*, pp. 30–39, Springer Berlin Heidelberg (2011).

[4] Gueron, S.: A j-lanes tree hashing mode and j-lanes SHA-256, Cryptology ePrint Archive, Report 2012/476 (2012). `http://eprint.iacr.org/`.

[5] Gueron, S. and Krasnov, V.: Parallelizing message schedules to accelerate the computations of hash functions, Cryptology ePrint Archive, Report 2012/067 (2012). `http://eprint.iacr.org/`.

[6] Gueron, S. and Krasnov, V.: Simultaneous hashing of multiple messages, Cryptology ePrint Archive, Report 2012/371 (2012). `http://eprint.iacr.org/`.

[7] Györfi, T., Creţ, O., Hanrot, G. and Brisebarre, N.: High-Throughput Hardware Architecture for the SWIFFT/SWIFFTX Hash Functions, Cryptology ePrint Archive, Report 2012/343 (2012). `http://eprint.iacr.org/`.

[8] Lyubashevsky, V., Micciancio, D., Peikert, C. and Rosen, A.: SWIFFT: A Modest Proposal for FFT Hashing, *Fast Software Encryption, FSE 2008, Lecture Notes in Computer Science*, Vol. 5086, pp. 54–72 (2008).

[9] National Institute of Standards and Technology: The Keyed-Hash Message Authentication Code (HMAC), *Federal Information Processing Standards Publication, FIPS PUB 198-1* (2008). `http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf`.

[10] Sarkar, P. and Schellenberg, P. J.: A Parallel Algorithm for Extending Cryptographic Hash Functions, *Progress in Cryptology – INDOCRYPT 2001, Lecture Notes in Computer Science*, Vol. 2247, pp. 40–49 (2001).

[11] Szydlo, M. and Yin, Y. L.: Collision-Resistant usage of MD5 and SHA-1 via Message Preprocessing, Cryptology ePrint Archive, Report 2005/248 (2005). `http://eprint.iacr.org/`.

Appendix

A.1 Proof of Lemma 1

Lemma 1 Let h and G denote families of functions in Theorem 1. If there exists an adversary A_G that can find collisions in G, then there exists an adversary A_h that can find collisions in h such that

$$\Pr\left[\mathbf{Exp}_h^{\mathrm{CR}}(A_h)=1\right] \geq \Pr\left[\mathtt{col}_h|\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right] \cdot \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right],$$

where $\mathtt{col}_h$ denotes the event such that a collision found by A_G results in a collision in h. The running time of A_h is dominated by that of A_G and the time to compute $G(K, X^{(1)})$ and $G(K, X^{(2)})$ where $(X^{(1)}, X^{(2)})$ is the collision in G found by A_G. Similarly, there exists an adversary A_a that can find collisions in a such that

$$\Pr\left[\mathbf{Exp}_a^{\mathrm{CR}}(A_a)=1\right] \geq \Pr\left[\mathtt{col}_a|\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right] \cdot \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right],$$

where $\mathtt{col}_a$ denotes the event such that a collision found by A_G results in a collision in a. The running time of A_a dominated by that of A_G, the time to compute $G(K, X^{(1)})$ and $G(K, X^{(2)})$, where $(X^{(1)}, X^{(2)})$ is the collision in G found by A_G and the time to sort message blocks of $X^{(1)}, X^{(2)}$.

Proof Consider adversary A_h taking K_h as an input described in Fig. A·1. In Fig. A·1, the statement from '▷' to the end of a line is a comment. Analyzing the procedure of Fig. A·1 gives

$$\mathbf{Exp}_h^{\mathrm{CR}}(A_h) = \begin{cases} 0 & \text{(line 6, line 17)} \\ 1 & \text{(line 9, line 14).} \end{cases}$$

When line 17 is performed, although A_G has succeeded in finding a collision in G, it is not a collision in h. The success of A_G does not mean that of A_h. Hence, we have

$$\Pr\left[\mathbf{Exp}_h^{\mathrm{CR}}(A_h)=1\right] = \Pr\left[\mathtt{col}_h|\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right] \cdot \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right].$$

The running time of A_h is dominated by that of A_G and the time to compute variables in line 4 (i.e., $G(K, X^{(1)})$ and $G(K, X^{(2)})$).

Next, consider adversary A_a taking K_a as an input described in Fig. A·2. The algorithm of A_a is based on the observation that if $(\hat{X}_i^{(1)}, \hat{X}_j^{(2)})$ is a collision in a, then two different messages $X^{(1)}, X^{(2)}$ that gives a collision in G an be obtained easily. Conversely, a collision in G does not always give a collision in a. In addition, even if A_G does not give a collision in G, the output of A_G might give a collision in a. Analyzing the procedure of Fig. A·2 gives

$$\mathbf{Exp}_a^{\mathrm{CR}}(A_a) = \begin{cases} 0 & \text{(line 9)} \\ 1 & \text{(line 7).} \end{cases}$$

1: **procedure** ADVERSARY $A_h(K_h)$
2: Choose K_a from$\mathcal{K}_a$ according to the uniform distribution on $\mathcal{K}_a$ and give (K_h, K_a) to A_G.
3: Let $(X^{(1)}, X^{(2)})$ be the output of $A_G((K_h, K_a))$.
4: For $\tau = 1, 2$, compute $W_i^{(\tau)}, V_i^{(\tau)}$ where $i = 1, 2, \ldots, n^{(\tau)}+1$ in a manner of Sect. 2.1.
5: **if** $V_{n^{(1)}+1}^{(1)} \neq V_{n^{(2)}+1}^{(2)}$ **then**
6: **return** $(\perp, \perp)$. ▷ $\perp \notin \{0,1\}^v$, A_h fails.
7: **end if**
8: **if** $n^{(1)} \neq n^{(2)} \vee V_{n^{(1)}}^{(1)} \neq V_{n^{(2)}}^{(2)}$ **then** ▷ $V_{n^{(1)}+1}^{(1)} = V_{n^{(2)}+1}^{(2)}$
9: **return** $(W_{n^{(1)}+1}^{(1)} \parallel V_{n^{(1)}}^{(1)}, W_{n^{(2)}+1}^{(2)} \parallel V_{n^{(2)}}^{(2)})$. ▷ A_h succeeds.
10: **end if**
11: Let $n = n^{(1)}$. ▷ $n^{(1)} = n^{(2)} \wedge V_{n^{(1)}}^{(1)} = V_{n^{(2)}}^{(2)}$
12: **for** $i = n, n-1, \ldots, 1$ **do** ▷ $V_i^{(1)} = V_i^{(2)}$
13: **if** $W_i^{(1)} \parallel V_{i-1}^{(1)} \neq W_i^{(2)} \parallel V_{i-1}^{(2)}$ **then**
14: **return** $(W_i^{(1)} \parallel V_{i-1}^{(1)}, W_i^{(2)} \parallel V_{i-1}^{(2)})$. ▷ A_h succeeds.
15: **end if**
16: **end for**
17: **return** $(\perp, \perp)$ ▷ A_h fails.
18: **end procedure**

Fig. A·1 Adversary A_h.

1: **procedure** ADVERSARY $A_a(K_a)$
2: Choose K_h from $\mathcal{K}_h$ according to the uniform distribution on $\mathcal{K}_h$ and give (K_h, K_a) to A_G.
3: Let $(X^{(1)}, X^{(2)})$ be the output of $A_G((K_h, K_a))$.
4: For $\tau = 1, 2$, compute $\hat{X}_i^{(\tau)}, Y_i^{(\tau)}$ where $i = 1, 2, \ldots, n^{(\tau)}$ in a manner of Sect. 2.1.
5: Find (i, j) such that $\hat{X}_i^{(1)} \neq \hat{X}_j^{(2)} \wedge Y_i^{(1)} = Y_j^{(2)}$ where $i \in \{1, 2, \ldots, n^{(1)}\}$ and $j \in \{1, 2, \ldots, n^{(2)}\}$.
6: **if** such a (i, j) is found **then**
7: **return** $(\hat{X}_i^{(1)}, \hat{X}_j^{(2)})$ as a collision in a. ▷ A_a succeeds.
8: **else**
9: **return** $(\perp, \perp)$. ▷ A_a fails.
10: **end if**
11: **end procedure**

Fig. A·2 Adversary A_a.

Let $\mathtt{suc}$ denote the event that the statement in line 7 is performed. Then,

$$\begin{aligned} &\Pr\left[\mathbf{Exp}_a^{\mathrm{CR}}(A_a)=1\right] \\ &= \Pr\left[\mathtt{suc}|\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=0\right] \cdot \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=0\right] \\ &\quad + \Pr\left[\mathtt{suc}|\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right] \cdot \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right] \\ &\geq \Pr\left[\mathtt{suc}|\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right] \cdot \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right] \\ &\geq \Pr\left[\mathtt{col}_a|\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right] \cdot \Pr\left[\mathbf{Exp}_G^{\mathrm{CR}}(A_G)=1\right], \end{aligned}$$

which is the inequality of Eq. (8). The running time of A_a is dominated by that of A_G, the time to compute variables in line 4 (i.e., $G(K, X^{(1)})$ and $G(K, X^{(2)})$), and the time to find (i, j) in line 5. The search in line 5 can be done in $O((n^{(1)}+n^{(2)})\log(n^{(1)}+n^{(2)}))$ by using an efficient sorting algorithm (e.g., a quick sort). □