Efficient Construction of Diamond Structures

Abstract. A cryptographic hash function is a function $H : \{0,1\}^* \rightarrow \{0,1\}^n$, that takes an arbitrary long input and transforms it to an n-bit output, while keeping some basic properties that ensure its security. Because they are very useful in computer security, cryptographic hash functions are amongst the most important primitives in the modern cryptography.

The Merkle-Damgård structure is an iterative construction for transforming a compression function $f : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$ into a hash function, and it is widely used by different hash functions such as MD4, MD5, SHA0 and SHA1. Some generic attacks on this structure were presented in the last 15 years. Some of these attacks use the *diamond structure*, first introduced by Kelsey and Kohno in the herding attack. This structure is a complete binary tree that allows 2^k different inputs to lead to the same hash value, and it used in numerous attacks on the Merkle-Damgård structure. Following the herding attack, other papers analyzed and optimized the *diamond structure*. The best time complexity of constructing a *diamond structure* to date is about $a \cdot 2^{\frac{n+k}{2}+2}$ for $a \approx 2.732$.

In this work we suggest a new method for constructing a diamond structure with better time complexity of $c \cdot 2^{\frac{n+k}{2}+2}$ for $c \approx 1.254$. We present a pseudo-code for this new method, and a recursive formulation of it. We also present analysis supported by experiments of our new method.

1 Introduction

Cryptographic hash functions are one of the important basic primitives in cryptography. Their importance is reflected in their wide use: digital signatures, hashed passwords, message authentication code (MAC), etc.

Design and cryptanalysis of hash functions has become one of the hottest research topics in the last fifteen years, when a series of groundbreaking works showed that some of the hash function designs (including Merkle-Damgård construction) are theoretically insecure [2, 9, 14, 16, 17], and that most of used hash functions (including SHA0, SHA1, MD4, MD5, RIPEMD, etc.) are theoretically (and some of them also practically) insecure [6, 18, 26, 28, 27, 29–31]. These results called for rethinking of the hash functions design methodologies, and invited new designs and their analysis. As part of this rethinking, the National Institute of Standards and Technology (NIST) announced a selection process of a new hash function standard called SHA3, which culminated in the choice of Keccak [5] as SHA3 function in October 2012. The SHA3 process gave rise to numerous new design methodologies and continuously developing cryptanalytic techniques.

One of the main sources for comparison between design strategies are generic attacks. While usually non-practical, they point out structural weaknesses in a strategy that may make us prefer a more conservative (or just a different) design. One of the basic designs of numerous hash functions is the Merkle-Damgård structure [8, 22], which, given a compression function $f : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$, creates a cryptographic hash function $\mathcal{MDH}^f : \{0,1\}^* \rightarrow \{0,1\}^n$, so that the hash function has certain security properties. Numerous generic attacks were presented against this construction, e.g., Joux's multicollision attack on iterative hash functions [14], the expandable messages attack of Kelsey and Schneier [17] and the herding attack of Kelsey and Kohno [16]. Naturally, generic attacks are used at complex algorithms and designs, often become used by other attacks.

This work improves the complexity of attacks based on the *diamond structure*, first introduced by Kelsey and Kohno in the herding attack [16]. Kelsey and Kohno calculated the diamond construction complexity by intuitive reasoning concluded that it is $2^{\frac{n+k}{2}+2}$ compression function calls. Blackburn et al. [7] showed that their calculation is wrong, and the real complexity, by the method presented by Kelsey-Kohno, is actually $\sqrt{k} \cdot 2^{\frac{n+k}{2}+2}$ compression function calls. In [21] Kortelainen and Kortelainen suggested a new method to construct the diamond in $a \cdot 2^{\frac{n+k}{2}+2}$ compression function calls, for $a = 2.732$. In this paper we suggest a new method for constructing the diamond in $c \cdot 2^{\frac{n+k}{2}+2}$ compression function calls, for $1 \leq c \leq 2$. Our experiments and analysis suggest that for our algorithm $c = 1.254$. The advantage of our work over the previous is not only the time complexity improvement, but also the algorithmic improvement: While the Kortelainen algorithm is very complex, our algorithm is simple and intuitive.

This paper is organized as follows: Section 2 gives notations and definitions used in this paper. In Section 3 we quickly recall the herding attack, and most importantly, the construction of diamond structures. We discuss the different methods to construct a diamond structure in Section 4. Our new ideas on how to efficiently build a diamond structure are given in Section 5. Finally, we conclude the paper in Section 6.

2 Notations and Definitions

Definition 1. *A cryptographic hash function is a function $H : \{0,1\}^* \rightarrow \{0,1\}^n$, that takes an arbitrary length input and transforms it to an n-bit output such that $H(x)$ can be computed efficiently, while the function has three basic security properties:*

1. Collisions resistance: It is hard to find (with high probability) an adversary that could find two different messages M, M' such that $H(M) = H(M')$ in less than $\mathcal{O}(2^{n/2})$ calls to $H(\cdot)$.

2. Second pre-image resistance: Given h, M such that $H(M) = h$, an adversary cannot find (with high probability) an additional message $M' \neq M$ such that $H(M') = h$ in less than $\mathcal{O}(2^n)$ calls to $H(\cdot)$.
3. Pre-image resistance: Given a hash value h, an adversary cannot find (with high probability) any message M such that $H(M) = h$ in less than $\mathcal{O}(2^n)$ calls to $H(\cdot)$.

Definition 2 (Merkle-Damgård structure ($\mathcal{MDH}$)). *The Merkle-Damgård structure [8, 22] is a structure of an iterative hash function, based on a compression function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. The compression function takes an n-bit chaining value and an m-bit message block and transforms them into a new n-bit chaining value, keeping the three basic properties described above. In order to hash a whole message M, the following steps are required (let b be the number of bits in the message, and ℓ be the number of bits used to encode the message length in bits[1]):*

1. Padding step:
   (a) Concatenate '1' to the end of the message.
   (b) Pad a sequence of $0 \leq k < m$ zeros, such that $b+1+k+\ell \equiv 0 \pmod{m}$.
   (c) Append the message with the original message length in bits, encoded in ℓ bits.
2. Divide the message to blocks of m bits, so if the length of padded message is $L \cdot m$ then
$$M = M_0||M_1||\dots||M_{L-1}.$$
3. The iterative chaining value h_i starts with a constant IV, defined as h_{-1} of the hash function, and it updated in every iteration, according to the appropriate message block M_i, to new chaining value:
$$h_i = f(h_{i-1}, M_i).$$
4. The output of this process is:
$$\mathcal{MDH}^f(M) = h_{L-1}$$

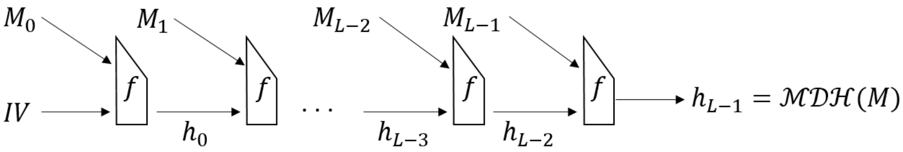

Fig. 1. The Merkle-Damgård structure

Merkle [22] and Damgård [8] proved that if the compression function is collision-resistant then the whole structure (when the padded message includes the original message length) is also collision-resistant.

[1] It is common to set $2^\ell - 1$ as the maximal length of a message.

3 The Herding Attack and the Diamond Structure

A well known generic attack on the Merkle-Damgård structure is the Kelsey-Kohno herding attack [16]. This attack has two phases. In the first phase, the adversary performs some precomputation and commits to a hash value h. In the second phase, given a prefix P, he finds a suffix S s.t.

$$H(P||S) = h.$$

In the precomputation the adversary constructs a complete binary tree called a *diamond structure*. This structure allows 2^k sequences of k message blocks to iteratively lead to the same chaining value. This may seem related to Joux's multicollision attack [14] where 2^k different message blocks result in the same chaining value. However, in the case of Joux's multicollision attack, all these 2^k options start from the same chaining value, whereas in the diamond structure, there are 2^k *different* starting chaining values.

To construct the *diamond structure*, the adversary starts with 2^k different chaining values, and looks for collisions between pairs of these chaining values to map them down to 2^{k-1} chaining values. In Section 4 we discuss different methods to do so. He repeats this process k times, s.t. in every iteration $1 \leq i \leq k$ he maps the 2^{k-i+1} chaining values he received at the end of the previous iteration down to 2^{k-i} chaining values. The output of this process, after k iterations, is a single hash value h. Fig. 2 illustrates this structure for $k = 3$, when the arrows represent message blocks, and the values $h_{i,j}$ represent chaining values. This structure generates a multicollision of 2^k messages, when k is the diamond width. After the construction of the diamond structure the adversary commits the output h.[2] According to Kelsey and Kohno, the work done to construct the diamond structure is about $2^{\frac{n+k}{2}+2}$ compression function calls, and we will discuss it later in Section 4.

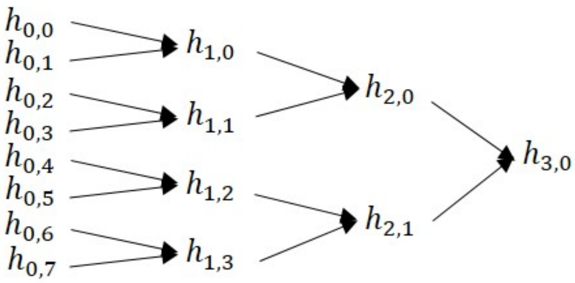

Fig. 2. Diamond Structure for $k = 3$

[2] He should consider the length of the prefix P, or at least the maximum length of it, and if the real length is less than he considered, he can add some blocks to the M_{link} block, which will defined in the second phase.

In the second phase, the adversary is challenged with a prefix P, and he has to find a suffix S yielding the desired result $H(P||S) = h$. To do so, he looks for a message block M_{link} that links the chaining value yielded from the prefix P to one of the leaves $\{h_{0,i}\}$ of the diamond structure. Given a chaining value h_{0,i_0}, traversing the tree from this leave to the root, appending the message blocks from each edge, leads the string Q, which creates, together with M_{link}, the desired suffix $S = M_{link}||Q$. Since the diamond has 2^k leaves, the work done to find the M_{link} block is about 2^{n-k} compression function calls, and thus the total work (according to Kelsey and Kohno) is about

$$2^{\frac{n+k}{2}+2} + 2^{n-k}.$$

We note that the diamond structure was later used in [1, 2, 4] to offer second pre-image attacks against Merkle-Damgård hash functions, including dithered hash functions [25].

4 Previous Methods for Constructing Diamond Structure

After Kelsey and Kohno published their attack based on the diamond structure and suggested their method and its analysis [16], several papers were published on methods for constructing and analyzing the diamond structure. We now discuss them.

4.1 Kelsey-Kohno's Method

In [16] Kelsey and Kohno suggest a method for constructing a diamond structure. We now describe their method for the first level of the diamond, and the application for the other levels is immediate.

Given 2^k starting chaining values, $\{h_{0,0}, h_{0,1}, \cdots, h_{0,2^k-1}\}$, the adversary should find 2^{k-1} collisions between pairs of them to map them down to 2^{k-1} new chaining values. i.e., he should find a partition to pairs of $\{h_{0,0}, h_{0,1}, \cdots, h_{0,2^k-1}\}$, and for each pair $(h_{0,i}, h_{0,j})_{i \neq j}$ he should find message blocks M, M' (not necessarily different) such that $f(h_{0,i}, M) = f(h_{0,j}, M')$.

If he fixes pairs of them, like $\{(h_{0,0}, h_{0,1}), (h_{0,2}, h_{0,3}), \ldots, (h_{0,2^k-2}, h_{0,2^k-1})\}$, then he should generate about $2^{\frac{n}{2}}$ candidate message blocks for each pair to find a collision, and thus the work done to find 2^{k-1} collisions is about $2^{k-1} \cdot 2^{\frac{n}{2}} = 2^{\frac{n}{2}+k-1}$ compression function calls.

Instead, he generates about $2^{\frac{n-k+1}{2}}$ candidate message blocks from each starting chaining value $h_{0,i}$, and then looks for collisions between all the possibility pairs dynamically. When he finds a collision, i.e., two starting chaining values $h_{0,i}, h_{0,j}$ and two message blocks M, M' s.t. $f(h_{0,i}, M) = f(h_{0,j}, M') = h$, he chooses these message blocks for these chaining values, and takes the new chaining value h for the next level of the diamond.

Kelsey and Kohno expected to find 2^{k-1} such collisions (we present their computation in Section 4.2), i.e., $2^k \cdot 2^{\frac{n-k+1}{2}} = 2^{\frac{n+k+1}{2}}$ message blocks should be sufficient to map all the 2^k starting chaining values into 2^{k-1} new chaining values will be in the next level of the diamond.

4.2 Kelsey-Kohno's Original Analysis

Kelsey and Kohno calculate the complexity of the diamond construction, and conclude the following:

> "The work done to build the diamond structure is based on how many messages must be tried from each of 2^k starting values, before each has collided with at least one other value. Intuitively, we can make the following argument, which matches experimental data for small parameters: When we try $2^{\frac{n}{2}+\frac{k}{2}+\frac{1}{2}}$ messages spread out from 2^k starting hash values (lines), we get $2^{\frac{n}{2}+\frac{k}{2}+\frac{1}{2}-k}$ messages per line, and thus between any pair of these starting hash values, we expect about $(2^{\frac{n}{2}+\frac{k}{2}+\frac{1}{2}-k})^2 \times 2^{-n} = 2^{n+k+1-2k-n} = 2^{-k+1}$ collisions. We thus expect about $2^{-k+k+1} = 2$ other hash values to collide with any given starting hash value." [16]

According to [16], if the adversary generates $2^{\frac{n-k+1}{2}}$ message blocks for each starting chaining value then the probability of a collision between any two starting chaining values is 2^{-k+1}. Thus, given a starting chaining value, since it could collide with any other starting chaining value, he expects about $2^{-k+1+k} = 2$ collisions. Since for any starting chaining value he expects at least one collision, he expects to map all the 2^k starting chaining values down to 2^{k-1} new chaining values.

Now, to construct the whole diamond he should repeat this work k times, for every $1 \leq i \leq k$, and thus the total complexity is about

$$\sum_{i=1}^{k} 2^{\frac{n+i+1}{2}} \approx 2^{\frac{n+k}{2}+2}.$$

4.3 On the Inaccuracy of Kelsey-Kohno's Analysis

Blackburn et al. [7] show that although this calculation is correct, their conclusion is wrong. They suggest to model the Kelsey-Kohno's method by the Erdös-Rényi random graph $G(n,p)$. The $G(n,p)$ is a random graph with n vertices, and there is an edge between any two vertices with probability p independently of other edges. In Kelsey-Kohno's case we get $G(2^k, 2^{-k+1})$, where $V = \{h_{0,0}, h_{0,1}, \ldots, h_{0,2^k-1}\}$ and $(h_{0,i}, h_{0,j}) \in E$ if and only if there exist two message blocks M, M' s.t. $f(h_{0,i}, M) = f(h_{0,j}, M')$, and it happens with probability of 2^{-k+1}, as described in Kelsey-Kohno's calculation. In this perspective, mapping the 2^k starting chaining values into 2^{k-1} new chaining values is equal to the existence of a perfect matching in G. In [10–12] Erdös and Rényi show that for a random graph $G(n,p)$, if $p > \frac{(1+\epsilon)\ln n}{n}$ then G will almost surely be connected and will contain a perfect matching, and if $p < \frac{(1-\epsilon)\ln n}{n}$ then G will almost surely contain isolated vertices and thus it will not contain any perfect matching. It means that $p = \frac{\ln n}{n}$ is a sharp threshold for the existence of a perfect matching in $G(n,p)$. In Kelsey-Kohno's case we get that (for $k > 2$)

$$p = 2^{-k+1} < k \cdot \ln 2 \cdot 2^{-k} = \frac{\ln(2^k)}{2^k}.$$

Thus, the graph will almost surely contain isolated vertices and thus it will not contain any perfect matching.

Hence, despite of the correct conclusion of Kelsey and Kohno that about $2^{-k+k+1} = 2$ edges for each vertex are expected, there will be many isolated vertices.[3]

To fix the problem, Blackburn et al. [7] conclude that we should generate about $\sqrt{k} \cdot 2^{\frac{n-k}{2}}$ message blocks from each vertex. Thus, to construct the entire diamond structure about

$$\sum_{i=1}^{k} \sqrt{i} \cdot 2^{\frac{n+i}{2}} \approx \sqrt{k} \cdot 2^{\frac{n+k}{2}+2}$$

message blocks are needed.[4]

4.4 Kortelainen-Kortelainen's Method

In [21] Kortelainen and Kortelainen suggest a new method to construct the diamond structure. Their general idea is to divide the construction into steps such that in every step exactly two vertices are matched to a single chaining value. If a vertex is matched they stop generating more message blocks for it. We describe here their algorithm for the first level of the diamond, the application for the other levels is immediate.

They divide the process into k phases (in reduced order), such that in every phase $2 \leq j \leq k$ they match exactly 2^{j-1} vertices, and in the last phase $j = 1$ they match the two remaining vertices, so at the end of these phases all the vertices are matched. Every phase $2 \leq j \leq k$ is divided into 2^{j-2} steps, such that in every step they match exactly two vertices to a new chaining value. The last step is done by finding a collision between the last two remaining vertices. Before the process, an initialization phase is performed as follow: Given the initial hash values (denoted by $A_{k,0}$), create a set $M_{k,0}$ of $2^{\frac{n-k}{2}-1}$ message blocks, such that the cardinality of $H_{k,0} := f(A_{k,0}, M_{k,0}) = \{f(a,m) | a \in A_{k,0}, m \in M_{k,0}\}$ is $2^{\frac{n+k}{2}-1}$, i.e., there are no collisions by these message blocks.[5] In Addition, they

[3] The degree of each vertex follows $Poi(2)$ distribution. Thus, for each vertex, the probability that it is an isolated vertex is e^{-2}, and thus we expect to about $2^k \cdot e^{-2}$ isolated vertices.

[4] Blackburn et al. [7] discuss another model to represent the diamond construction: *Sampling With Replacement Random Intersection Graph* $G_{SWR}(\nu, m, L)$ random graph, defined as follow: Let V be a set of vertices where $|V| = \nu$ (in our case $\nu = 2^k$), and F be a colors set where $|F| = m$ (in our case $m = 2^n$). For each vertex $v \in V$ generate a subset $F_v \subset F$ by sampling uniformly with replacement L colors from F (in Kelsey-Kohno's case $L = 2^{\frac{n-k+1}{2}}$). Finally, $(v,u) \in E \iff F_v \cap F_u \neq \phi$. They achieve from this model the same results as from the $G(n,p)$ model.

[5] Although usually we are looking for collisions, this requirement about the cardinality of $H_{k,0}$ is needed for their analysis. Later, by our method, we will show how to use such collisions. If there are collisions, they replace the appropriate message blocks one by one until the required cardinality is obtained.

initialize a pairing set, denoted by B_k, as an empty set. This set contains pairs of the form (h_i, D_i) where h_i is a chaining value, and D_i is a message block such that $f(h_i, D_i)$ will be in the next layer of the diamond structure.

For the algorithm they define the following integers:

> Let $r \geq 2, n$ be positive integers. Define the integers $s_{r,0}, s_{r,1}, \dots s_{r,2^{r-2}}$ as follow:
>
> $$s_{r,0} = \left\lceil 2^{\frac{n-r}{2}-1} \right\rceil, s_{r,j+1} = s_{r,j} + \left\lceil \frac{2^{\frac{n-r}{2}+1}}{2^r - 2j} \right\rceil, j = 0, 1, \dots 2^{r-2} - 1$$

They prove that:

$$\forall j \in \{0, 1, \dots 2^{r-2}\} : s_{r,j} \geq \frac{2^{\frac{n-r}{2}-1}}{2^r - 2j}$$

Let $i \in \{k, k-1, \dots, 3, 2\}, j \in \{0, 1, \dots, 2^{i-2} - 1\}$, The input for the step j in the phase i, denoted by $S(i, j)$, is the set $A_{i,j}$ of the $2^i - 2j$ unmatched vertices, the set $M_{i,j}$ of the $s_{i,j} \geq \frac{2^{\frac{n+i}{2}-1}}{2^i - 2j}$ message blocks generated until now, and the set $H_{i,j} = f(A_{i,j}, M_{i,j}) = \{f(a,m) | a \in A_{i,j}, m \in M_{i,j}\}$, such that

$$|H_{i,j}| = |A_{i,j}| \cdot |M_{i,j}| = (2^i - 2j) \cdot s_{i,j} \geq (2^i - 2j) \cdot \frac{2^{\frac{n+i}{2}-1}}{2^i - 2j} = 2^{\frac{n+i}{2}-1}$$

Now, they create a set $M'_{i,j}$ of $s_{i,j+1} - s_{i,j}$ message blocks such that $|f(A_{i,j}, M'_{i,j})| \geq 2^{\frac{n-i}{2}+1}$. They look for a collision, i.e., $h_{ij}, h'_{ij} \in A_{i,j}, m_{ij} \in M_{i,j}, m'_{ij} \in M'_{i,j}$ such that $f(h_{ij}, m_{ij}) = f(h'_{ij}, m'_{ij})$. Note, that since $|H_{i,j} \times f(A_{i,j}, M'_{i,j})| \geq 2^n$ the expected number of collisions is at least one, and they assume that it is exactly one.[6] Let $A_{i,j+1} := A_{i,j} \setminus \{h_{ij}, h'_{ij}\}, M_{i,j+1} := M_{i,j} \cup M'_{i,j}$, and $H_{i,j+1} := f(A_{i,j+1}, M_{i,j+1})$. In addition set up $B_k = B_k \cup \{(h_{ij}, m_{ij}), (h'_{ij}, m'_{ij})\}$. These sets are the output of this step, and the input for the next step. The pseudo-code for this algorithm is given in Algorithm 1.

They concluded that the total message complexity of the whole diamond construction is

$$a \cdot \left(\sum_{i=2}^{k} 2 \cdot 2^{\frac{n+i}{2}} + 4 \cdot 2^{\frac{n}{2}} \right) \leq a \cdot 2^{\frac{n+k}{2}+2}$$

for $a = \frac{1}{4} \cdot \left[1 + \frac{1}{\sqrt{2}} + 2\frac{e}{e-1} \left(1 + \frac{1}{\sqrt{2}}\right)^2 \right] \approx 2.732$ (the detailed analysis is in [20]).[7]

4.5 Memoryless Diamond Structure Construction

A different optimization target is reducing the memory used in the construction of the diamond structure. Barham et al. [4] show that it is possible to eliminate

[6] If not, they replace some message blocks one by one until it is obtained.

[7] We note that the analysis of [20] uses a slightly different definition of a, but for more natural comparison with previous methods, we took a as the coefficient of $2^{\frac{n+k}{2}+2}$.

Algorithm 1 Kortelainen-Kortelainen's method

```
Input: $H_k \subseteq 0, 1^n, |H_k| = 2^k$
$A_{k,0} \leftarrow H_k$
Create a set $M_{k,0}$ of $2^{\frac{n-k}{2}-1}$ message blocks s.t. $|f(A_{k,0}, M_{k,0})| = 2^{\frac{n-k}{2}-1}$. (Initialization part)
$H_{k,0} \leftarrow f(A_{k,0}, M_{k,0})$
$B_k \leftarrow \phi$
for $i = k$ downto 2 do
    for $j = 0$ to $2^{i-2} - 1$ do
        Create a set $M'_{i,j}$ of $s_{i,j+1} - s_{i,j}$ message blocks s.t. $M_{i,j} \cap M'_{i,j} = \phi$ and $|f(A_{i,j}, M'_{i,j})| \geq 2^{\frac{n-i}{2}+1}$.
        look for a collision: $h_{i,j}, h'_{i,h} \in A_{i,j}, m_{i,j} \in M_{i,j}, m'_{i,j} \in M'_{i,j}$ s.t. $f(h_{i,j}, m_{i,j}) = f(h'_{i,j}, m'_{i,j})$.
        $A_{i,j+1} \leftarrow A_{i,j} \setminus \{h_{i,j}, h'_{i,j}\}$
        $M_{i,j+1} \leftarrow M_{i,j} \cup M'_{i,j}$
        $H_{i,j+1} \leftarrow f(A_{i,j+1}, M_{i,j+1})$
        $B_k \leftarrow B_k \cup \{(h_{i,j}, m_{i,j}), (h'_{i,j}, m'_{i,j})\}$
    end for
end for
```

the memory used in the construction of the diamond structure. In this approach, the chaining values that build the diamond are not kept in the memory. Thus, we can use this construction when at the online phase not all the diamond structure's chaining values are needed.

Similarly to Merkle hash tree [22], they decide in advance which leave collides with which leave. Now, for every pair they use a memoryless collision search [19, 23, 24] to find a collision. When a collision between two chaining values is found, i.e.,

$$f(h_{i,2j}, M) = f(h_{i,2j+1}, M') = h_{i+1,j},$$

it is possible to build the diamond's part that is needed to obtain the chaining value $h_{i+1,j+1}$, and then to look for a collision between the two new chaining values $h_{i+1,j}, h_{i+1,j+1}$. The memory complexity of this construction is about $k+1$ blocks. The time complexity of this construction is about $2^{\frac{n}{2}+k+1}$, since the construction includes $2^{k+1}-1$ collisions, and each collision requires about $2^{\frac{n}{2}}$ compression function calls. Although the time complexity of this method is larger than that of the previous methods, its memory complexity is very small.

5 Our New Method

In Section 4.3 we discussed the time complexity when all the messages are generated simultaneously and uniformly between all vertices. If we generate about $2^{\frac{n-k+1}{2}}$ message blocks per vertex, the expected number of collisions for a vertex is 2. In this case the $G(2^k, 2^{-k+1})$ random graph contains some isolated vertices (about $2^k \cdot e^{-2}$), and thus it does not contain any perfect matching. The conclusion of Blackburn et al. [7] is that about $\sqrt{k} \cdot 2^{\frac{n-k}{2}}$ message blocks per vertex are needed for the existence of a perfect matching in the graph. The disadvantage of this method is that we need to generate many message blocks for each vertex to get a perfect matching, of which only a small portion of the found collisions is used.

We now suggest a new method to construct the diamond. Our method is based on two ideas, described in the next sections:

1. Messages-Layers Trade-off: We generate less than $\sqrt{k} \cdot 2^{\frac{n-k}{2}}$ message blocks. Since we expect to have isolated vertices, we expect to have more than 2^{k-1} vertices in the next layer. Thus, we should add some layers to the construction to reach a single chaining value at the root of the diamond structure. We show that by generating less message blocks in each layer in exchange for more layers in the construction, we can reduce the total time complexity.
2. Match While Generate (MWG): We generate the message blocks one by one and look for collision after every generation. If two vertices collide with each other, we match them and do not generate more message blocks for them. We show that by this method we can further reduce the time complexity.

After their description, we show how to use them together to obtain the best time complexity for constructing the diamond structure.

5.1 Messages-Layers Trade-off

The first idea, which we call "Messages-Layers Trade-off" is that we can weaken the requirement of a perfect matching in each layer exchange for more layer in the construction of the diamond. Instead of generating $\sqrt{k} \cdot 2^{\frac{n-k}{2}}$ message blocks per vertex, we generate fewer messages. The result is that in the $G(2^k, p)$ graph we get that $p < \frac{\ln n}{n}$, and thus it does not contain a perfect matching. We then match all vertices we can (e.g., by a greedy algorithm like Karp-Sipser [15] or its variants [3]) and for the remaining vertices we choose an arbitrary message block to get an arbitrary chaining value in the next layer. Since G does not contain a perfect matching, the number of vertices in the second layer is greater than 2^{k-1}. Similarly, the number of vertices in any layer $1 \leq i \leq k$ is greater than 2^{k-i}. Thus, we need to add some layers to get a single chaining value at the end of this process. We note that our experiments show that the number of layers does not increase by much. Moreover, the additional layers have almost no affect on attacks on Merkle-Damgård hash functions, and have a small impact on dithered hash functions.

We tested this idea on the Kelsey-Kohno's case, i.e., when we generate about $2^{\frac{n-k+1}{2}}$ message blocks per vertex. In this case we get the $G(2^k, 2^{-k+1})$ model, and we know that the degree of each vertex follows $Poi(2)$ distribution. According to the handshaking lemma we know that $\sum_{i=1}^{|V|} deg(v_i) = 2|E|$, where E is the edges' set. We also know that if $X_1, \dots X_t \sim Poi(\lambda)$ then $\sum\limits_{i=1}^{t} X_i \sim Poi(t \cdot \lambda)$. Thus, in our graph we get $|E| \sim Poi(|V|)$.[8] We generated such a graph $G = (V, E)$ where $|V| = 2^k$ and the edges' set E determined by sampling the $|E|$ according to $Poi(|V|)$, and for each edge sampling its two vertices uniformly between all vertices. Now we match the vertices to each other according to their degrees from low to high as follow: Let $M = \phi$ be an empty set. We run over the vertices and if $deg(v) = 1$ we add v and its single connected vertex u to the matching, i.e., $M = M \cup \{v, u\}$. In addition, we remove the vertices (and all their edges) from the graph, i.e., $G = G \setminus \{u, v\}$. We repeat it until the graph has no edges. Now we move on to the next layer with $2^k - \frac{|M|}{2}$ vertices. Clearly, this method has no advantage when $|V|$ is quite small. Thus, for the sake of simplicity, we repeat this method until $|V| \leq 16$ and then we use the Blackburn et al. [7] computation.[9]

We tested this algorithm on some different parameters for k and n. We performed 100 experiments for each pair (k, n). The output of each experiment is the number of message blocks required to construct a diamond structure with 2^k leaves, using a compression function of n bits, and the diamond's length. We present here the average of these outputs from the 100 experiments and the sample standard deviation. Table 1 lists the number of message blocks ($\log_2$ of them) and the diamond's length.

5.2 Match While Generate (MWG)

Intuitive Explanation of the Idea As we discussed earlier, when we generate all the message blocks and then look for collisions, about $\sqrt{k} \cdot 2^{\frac{n+k}{2}}$ message blocks are required such that the $G(2^k, p)$ graph contains a perfect matching. In this case the expected number of collisions for each vertex is k, but in the matching we use only one of them. Our second idea, named "Match While Generate (MWG)", is to look for collisions throughout the process, i.e., after we generate a message block M_j for v_i we check if the candidate $f(v_i, M_j)$ is already generated by another vertex u. If yes, we match them and do not generate any additional message blocks for them. Our second idea is inspired by Hoch's thesis [13]. The process is as follow:

We keep in a Boolean array, denoted by $isMatched$, the state of each vertex, i.e., $isMatched[i] = True \iff v_i$ is already matched. The set of the chaining values for the next level is denoted by $nextV$. We start with an empty set $Candidates \subseteq \{0,1\}^n \times V$ where $\{0,1\}^n$ is the set of all available chaining values, and V is the set of all vertices (initial values). In each iteration j we generate a new message block M_j, and run over the vertices to calculate the chaining value

[8] We tested this idea on more cases: when $|E| \sim Poi(a \cdot |V|), a = 0.5 + \frac{t}{20}, \forall t \in \{0, 1, 2, \dots, 19\}$. The difference between the results in the Kelsey-Kohno's case and the best results is quite small (less than one standard deviation).

[9] It is easy to see that if we switch to the Blackburn et al. process earlier, the expected length of the diamond will decrease.

$n\backslash k$			14	16	18	20	22
28	Blocks	Average	$2^{23.681}$	$2^{24.683}$	$2^{25.683}$	$2^{26.682}$	$2^{27.683}$
		S.D.	$2^{16.374}$	$2^{16.395}$	$2^{16.413}$	$2^{16.418}$	$2^{16.566}$
		Min	$2^{23.662}$	$2^{24.668}$	$2^{25.677}$	$2^{26.68}$	$2^{27.68}$
		Max	$2^{23.717}$	$2^{24.695}$	$2^{25.689}$	$2^{26.685}$	$2^{27.685}$
	Length	Average	20.13	22.85	25.8	28.34	31.35
		S.D.	1.522	1.41	1.583	1.32	1.424
		Min	18	21	24	26	30
		Max	26	31	32	34	36
32	Blocks	Average	$2^{25.683}$	$2^{26.683}$	$2^{27.682}$	$2^{28.683}$	$2^{29.683}$
		S.D.	$2^{18.117}$	$2^{18.267}$	$2^{18.553}$	$2^{18.237}$	$2^{18.541}$
		Min	$2^{25.666}$	$2^{26.665}$	$2^{27.677}$	$2^{28.679}$	$2^{29.681}$
		Max	$2^{25.709}$	$2^{26.692}$	$2^{27.689}$	$2^{28.686}$	$2^{29.684}$
	Length	Average	20.39	22.8	25.81	28.67	31.39
		S.D.	1.984	1.443	2.043	1.735	1.435
		Min	18	21	24	26	29
		Max	29	27	35	37	36
36	Blocks	Average	$2^{27.682}$	$2^{28.682}$	$2^{29.683}$	$2^{30.683}$	$2^{31.683}$
		S.D.	$2^{20.126}$	$2^{20.262}$	$2^{20.455}$	$2^{20.411}$	$2^{20.51}$
		Min	$2^{27.661}$	$2^{28.667}$	$2^{29.677}$	$2^{30.68}$	$2^{31.681}$
		Max	$2^{27.697}$	$2^{28.694}$	$2^{29.689}$	$2^{30.685}$	$2^{31.685}$
	Length	Average	20.24	22.76	25.68	28.62	31.43
		S.D.	1.525	1.443	1.723	1.523	1.416
		Min	18	21	24	27	29
		Max	27	30	33	34	36

Table 1. The number of required message blocks (represented by their $\log_2$), and the diamond's length, using the Messages-Layers Trade-off method.

candidate $h_{candidate} = f(v_i, M_j)$ for each vertex v_i. If $h_{candidate}$ is already generated by another vertex, i.e., $\exists 0 \leq r \neq i \leq 2^k - 1 : (h_{candidate}, v_r) \in Candidates$, then match v_i and v_r, add $h_{candidate}$ to $nextV$, and do not generate any additional message blocks for them. Otherwise, add $(h_{candidate}, v_i)$ to $Candidates$.

Algorithm 2 presents the Match While Generate algorithm.

Analysis of the Algorithm Here we suggest a recursive presentation for the diamond structure construction. We present here the construction of the first layer of the diamond structure (the application for the other layers is immediate). For the reading convenience we fix an arbitrary order of the vertices $v_0, \ldots v_{2^k-1}$. Let us define some random variables, depending on the process "time" t, i.e., the generation of the t'th candidate:

[10] We can use a hash table to keep the *Candidates* elements, to maintain a constant search time.

Algorithm 2 Match While Generate (MWG)

```
1: $Candidates \leftarrow \phi$
2: $nextV \leftarrow \phi$
3: for $i = 0$ to $2^k - 1$ do
4:     $isMatched[i] \leftarrow False$
5: end for
6: $j \leftarrow 0$
7: $n_{matched} \leftarrow 0$
8: while $n_{matched} < 2^k$ do
9:     Generate a message block $M_j$
10:    for $i = 0$ to $2^k - 1$ do
11:        if $isMatched[i]$ then
12:            Go to 10
13:        end if
14:        Calculate $h_{candidate} = f(v_i, M_j)$
15:        if $\exists 0 \leq r \leq 2^k - 1, r \neq i : (h_{candidate}, v_r) \in Candidates$[10] $\wedge \neg isMatched[r]$ then
16:            $nextV \leftarrow nextV \cup \{h_{candidate}\}$
17:            $isMatched[i] \leftarrow True$
18:            $isMatched[r] \leftarrow True$
19:            $n_{matched} \leftarrow n_{matched} + 2$
20:        else
21:            $Candidates \leftarrow Candidates \cup \{(h_{candidate}, v_i)\}$
22:        end if
23:    end for
24:    $j \leftarrow j + 1$
25: end while
```

- $m(t)$:=The number of matched vertices at time t.
- $c(t)$:=The number of candidates available for matching at time t.
- $bu(t)$:=The number of candidates generated from the current vertex at time t, i.e., sum of values in $Candidates$ of the form (h_i, v).
- $be(t)$:= The number of unmatched vertices before the current vertex at time t.
- $af(t)$:= The number of unmatched vertices after the current vertex at time t.
- We also use a counter variable to count the number of self-collisions, denoted by sc.

The initialization of the variables is: $sc = m(0) = c(0) = bu(0) = be(0) = 0$, and $af(0) = 2^k - 1$. At time $t + 1$ we generate a new candidate, and we compute the value of all the variables, given the values of all variables at time t. When we generate a new candidate from a vertex v_i, there are four possible cases:

1. The new candidate leads to a self-collision. In this case we continue to generate new candidates from the current vertex until it is not a self-collision.

For each new message block, we increment the *sc* counter by one. This case happens with probability

$$\frac{bu(t)}{2^n}.$$

2. The new candidate leads to a matching with v_j, for $j < i$. We have a matching, and since $j < i$ we match a vertex whose position is before the current position. In addition, we remove these two vertices, where v_i with $bu(t)$ candidates, and v_j with $bu(t) + 1$ candidates. Finally, we move to the next vertex. Thus, the updating of the variables is as follow: $m(t+1) = m(t)+2$, $be(t+1) = be(t) - 1$, $af(t+1) = af(t) - 1$ and $c(t+1) = c(t) - 2 \cdot bu(t) - 1$. This case happens with probability

$$\frac{be(t) \cdot (bu(t) + 1)}{2^n}.$$

3. The new candidate leads to a matching with v_j, for $j > i$. We have a matching, and since $j > i$, we match a vertex whose position is after the current position. In addition, we remove these two vertices, both with $bu(t)$ candidates. Finally, we move to the next vertex. Thus, the updating of the variables is as follow: $m(t+1) = m(t)+2$, $af(t+1) = af(t)-2$, $be(t+1) = be(t)$ and $c(t + 1) = c(t) - 2 \cdot bu(t)$. This case happens with probability

$$\frac{af(t) \cdot bu(t)}{2^n}.$$

4. The new candidate does not lead to any matching. There is no matching, and we add the new candidate. In addition, we move to the next vertex, so that the current vertex is added to the vertices that before the next, and the next removed from the vertices that are after it. Thus, the updating of the variables is as follow: $m(t + 1) = m(t)$, $be(t + 1) = be(t) + 1$, $af(t + 1) = af(t) - 1$ and $c(t + 1) = c(t) + 1$. This case happens with probability

$$1 - \frac{bu(t)}{2^n} - \frac{be(t) \cdot (bu(t) + 1)}{2^n} - \frac{af(t) \cdot bu(t)}{2^n}.$$

Finally, if $af(t + 1) < 0$ (after the above updates), it means that we finished an iteration over the unmatched vertices, and at time $t+1$ we start a new iteration. Thus, $be(t + 1) = 0, af(t + 1) = 2^k - m(t + 1) - 1$ and $bu(t + 1) = bu(t) + 1$. If $af(t+1) \geq 0$ it means that we remain at the same iteration and thus $bu(t+1) = bu(t)$.

Using this recursion, we tested the MWG algorithm with some parameters for k and n. We performed 100 experiments for each pair (k, n). The output of each experiment is the number of message blocks required to construct a diamond structure with 2^k leaves, using a compression function of n bits. We present in Table 2 the average of these numbers from the 100 experiments and the sample standard deviation.

$n\backslash k$		14	16	18	20	22
28	Average	$2^{23.399}$	$2^{24.411}$	$2^{25.418}$	$2^{26.423}$	$2^{27.43}$
	S.D.	$2^{16.175}$	$2^{16.289}$	$2^{16.65}$	$2^{16.67}$	$2^{16.666}$
	Min	$2^{23.374}$	$2^{24.396}$	$2^{25.411}$	$2^{26.42}$	$2^{27.428}$
	Max	$2^{23.425}$	$2^{24.423}$	$2^{25.428}$	$2^{26.428}$	$2^{27.431}$
32	Average	$2^{25.4}$	$2^{26.411}$	$2^{27.417}$	$2^{28.421}$	$2^{29.423}$
	S.D.	$2^{18.245}$	$2^{18.683}$	$2^{18.839}$	$2^{18.795}$	$2^{18.881}$
	Min	$2^{25.375}$	$2^{26.389}$	$2^{27.403}$	$2^{28.417}$	$2^{29.421}$
	Max	$2^{25.429}$	$2^{26.427}$	$2^{27.426}$	$2^{28.424}$	$2^{29.425}$
36	Average	$2^{27.399}$	$2^{28.41}$	$2^{29.416}$	$2^{30.419}$	$2^{31.422}$
	S.D.	$2^{20.237}$	$2^{20.465}$	$2^{20.831}$	$2^{20.788}$	$2^{20.975}$
	Min	$2^{27.371}$	$2^{28.397}$	$2^{29.406}$	$2^{30.416}$	$2^{31.419}$
	Max	$2^{27.425}$	$2^{28.429}$	$2^{29.426}$	$2^{30.423}$	$2^{31.424}$

Table 2. The number of required message blocks (represented by their $\log_2$), using the MWG algorithm.

Actual Experiments In addition to the simulations which are based on the mathematical model, we tested this algorithm on a diamond structure with 2^{18} leaves, using a 28-bit compression function (we used the 28 first bits of SHA1), i.e., $k = 18, n = 28$. According to Kelsey and Kohno [16] we should generate about $2^{\frac{28+18}{2}+2} = 2^{25}$ message blocks. Blackburn et al. [7] prove that by Kelsey-Kohno's method about $\sqrt{18} \cdot 2^{\frac{28+18}{2}+2} > 2^{27}$ message blocks are needed. According to Kortelainen and Kortelainen we should generate about $a \cdot 2^{\frac{28+18}{2}+2} > 2^{26}$ message blocks. We constructed a diamond structure 100 times with different initial values. The mean value of the number of required message blocks was $\mu = 45672583.18 \approx 2^{25.445}$, and the sample standard deviation was $\sigma = 104866.202 \approx 2^{16.678}$. Fig. 3 illustrates the distribution of the number of message blocks required to construct it (the numbers are represented by their $\log_2$). According to the t-Test, the data follows $Norm(\mu, \sigma^2)$ distribution, with statistical significance of $2.5091 \cdot 10^{-14}$.

We note that the mean value of the experiment is greater than those of the recursion by a few standard deviation units. This difference is a subject for a future research.

5.3 Combining these Two Ideas Together

It is possible to improve the MWG algorithm, using the method described in Section 5.1. Before we present the improvement, we want to look at the number of message blocks required for each collision, throughout the construction of a diamond structure's first layer. Intuitively, at the beginning of the construction there are only a few candidates for matching, and thus we need to generate many message blocks for a collision. Similarly, near the end of the layer, since only a few unmatched vertices remain, i.e., only a few candidates for matching

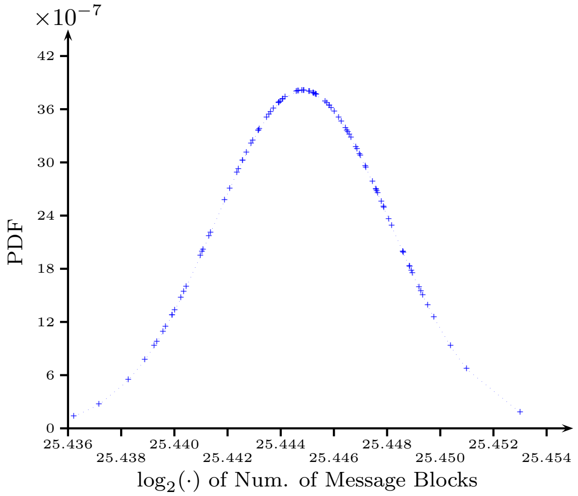

Fig. 3. Sampled distribution of the number of message blocks (represented by its $\log_2$) required to construct a diamond structure with 2^{18} leaves, and compression function of 28 bits, using the MWG algorithm.

exist, we need to generate many message blocks for a collision. In the middle part of the layer, on the one hand we already generated a significant amount of message blocks per vertex, and on the other hand still have a significant amount of unmatched vertices, so we have enough candidates, thus, we expect a collision after fewer messages. We tested this intuition, and Fig. 4 shows an example for the average number of message blocks generated between 256 collisions using the MWG algorithm, where $k = 18, n = 28$.

As we discussed above, the investment of the beginning is needed to ensure we have enough candidates. At the same time, the last matchings require a lot of message blocks in return of a small benefit. Thus, we can stop the construction near the end, as was discussed in Section 5.1, and move to the next layer. As a result we will have to add some layers. To do so, we improve the MWG algorithm by adding a boundary on the number of message blocks. Clearly, this method has no advantage over the previous when $|V|$ is quite small, and it may even be the case that it is less efficient. Thus, for the sake of simplicity, we repeat this method until $|V| \leq 16$ and then we use the original MWG algorithm.

We tested this improved algorithm by bounding the number of message blocks by $2^{\frac{28+18+1}{2}}$ (Kelsey-Kohno's number). We adapted the recursion presented in Section 5.2 to this improved algorithm, and using the adapted recursion we tested the improved MWG algorithm on some parameters for k and n. We performed 100 experiments for each pair (k, n). The output of each experiment is

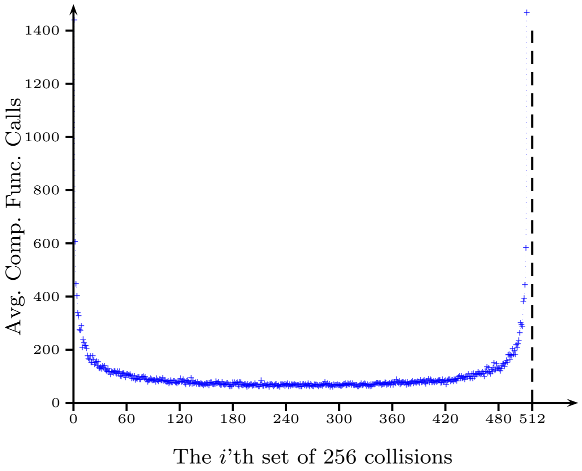

Fig. 4. Average number of compression functions calls needed for 256 collisions during the construction of the first layer of the diamond structure (for $k = 18, n = 28$)

the number of message blocks required to construct a diamond structure with 2^k leaves, using a compression function of n bits, and the diamond's length. We present here the average of these outputs from the 100 experiments and the sample standard deviation. Table 3 lists the number of message blocks ($\log_2$ of them) and the diamond's length.

In addition to the simulations which are based on the mathematical model, we also tested this improved algorithm on a diamond structure with 2^{18} leaves, using a 28-bit compression function (we used the 28 first bits of SHA1), i.e., $k = 18, n = 28$. We constructed a diamond structure 100 times with different initial values. The mean value of the number of required message blocks was $\mu = 42078721.93 \approx 2^{25.327}$, and the sample standard deviation was $\sigma = 58941.064 \approx 2^{15.847}$. Fig. 5 illustrates the distribution of the number of message blocks required to construct it (the numbers are represented by their $\log_2$). According to the t-Test, the data follows $Norm(\mu, \sigma^2)$ distribution, with statistical significance of $4.0967 \cdot 10^{-14}$.

We note that the mean value of the experiment is greater than those of the recursion by a few s.d. units. This difference is a subject for a future research.

$n\backslash k$			14	16	18	20	22
28	Blocks	Average	$2^{23.304}$	$2^{24.309}$	$2^{25.313}$	$2^{26.315}$	$2^{27.318}$
		S.D.	$2^{15.946}$	$2^{15.688}$	$2^{15.672}$	$2^{15.842}$	$2^{15.945}$
		Min	$2^{23.279}$	$2^{24.301}$	$2^{25.308}$	$2^{26.313}$	$2^{27.317}$
		Max	$2^{23.323}$	$2^{24.321}$	$2^{25.317}$	$2^{26.318}$	$2^{27.32}$
	Length	Average	15.01	17	19.01	21.07	23.11
		S.D.	0.1	0	0.1	0.256	0.314
		Min	15	17	19	21	23
		Max	16	17	20	22	24
32	Blocks	Average	$2^{25.304}$	$2^{26.309}$	$2^{27.312}$	$2^{28.314}$	$2^{29.315}$
		S.D.	$2^{17.667}$	$2^{17.627}$	$2^{17.678}$	$2^{17.776}$	$2^{17.865}$
		Min	$2^{25.282}$	$2^{26.3}$	$2^{27.306}$	$2^{28.311}$	$2^{29.314}$
		Max	$2^{25.323}$	$2^{26.318}$	$2^{27.315}$	$2^{28.317}$	$2^{29.316}$
	Length	Average	15.01	17.02	19.03	21.01	23.07
		S.D.	0.1	0.141	0.171	0.1	0.256
		Min	15	17	19	21	23
		Max	16	18	20	22	24
36	Blocks	Average	$2^{27.303}$	$2^{28.309}$	$2^{29.312}$	$2^{30.313}$	$2^{31.314}$
		S.D.	$2^{19.639}$	$2^{19.893}$	$2^{19.756}$	$2^{19.67}$	$2^{19.765}$
		Min	$2^{27.287}$	$2^{28.296}$	$2^{29.308}$	$2^{30.311}$	$2^{31.313}$
		Max	$2^{27.322}$	$2^{28.319}$	$2^{29.317}$	$2^{30.315}$	$2^{31.315}$
	Length	Average	15	17.03	19.02	21.02	23.1
		S.D.	0	0.171	0.141	0.141	0.302
		Min	15	17	19	21	23
		Max	15	18	20	22	24

Table 3. The number of required message blocks (represented by their $\log_2$), and the diamond's length, using the improved MWG algorithm.

6 Summary

In this paper we showed a time complexity optimization for the construction of *diamond structures*. We presented two ideas to optimize the construction:

1. Messages-Layers Trade-off: We generate less message blocks in each layer in exchange for more layers in the construction.
2. Match While Generate (MWG): We generate the message blocks one by one and look for collision after every generation.

We also showed how to combine these two ideas together to improve the MWG algorithm. Using the improved MWG we got the best results to date with respect to time complexity.

For comparison, we present in Table 4 the number of required message blocks using the previous methods and using our improved MWG algorithm, for each (k, n).

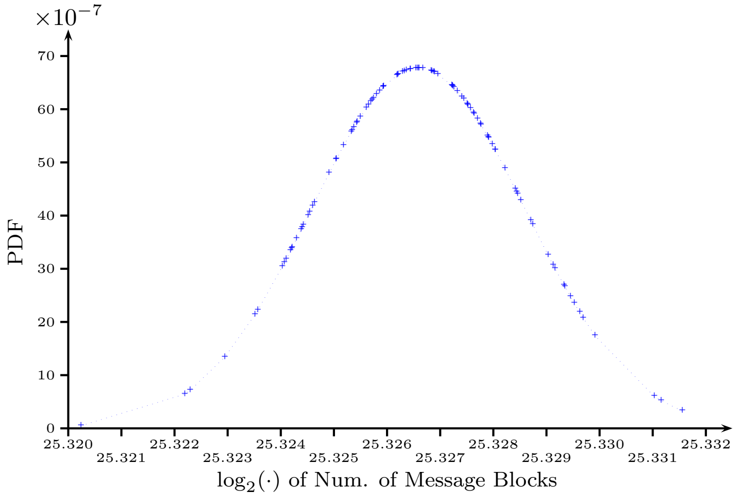

Fig. 5. Sampled distribution of the number of message blocks (represented by its $\log_2$) required to construct a diamond structure with 2^{18} leaves, and compression function of 28 bits, using the improved MWG algorithm.

Method	Time Complexity
Kelsey-Kohno [16] [11]	$2^{\frac{n+k}{2}+2}$
Blackburn et al. [7]	$\sqrt{k} \cdot 2^{\frac{n+k}{2}+2}$
Kortelainen-Kortelainen [21]	$\frac{1+\frac{1}{\sqrt{2}}+2\frac{e}{e-1}\left(1+\frac{1}{\sqrt{2}}\right)^2}{4} \cdot 2^{\frac{n+k}{2}+2} \approx 2.732 \cdot 2^{\frac{n+k}{2}+2}$
Improved MWG	$1.254 \cdot 2^{\frac{n+k}{2}+2}$

Table 4. Comparing the time complexity of the different methods.

References

1. Andreeva, E., Bouillaguet, C., Dunkelman, O., Fouque, P., Hoch, J.J., Kelsey, J., Shamir, A., Zimmer, S.: New second-preimage attacks on hash functions. J. Cryptology 29(4), 657–696 (2016)
2. Andreeva, E., Bouillaguet, C., Fouque, P.A., Hoch, J.J., Kelsey, J., Shamir, A., Zimmer, S.: Second preimage attacks on dithered hash functions. In: Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4965, pp. 270–288. Springer (2008)

[11] We remind the reader that Kelsey-Kohno's analysis is inaccurate.

3. Aronson, J., Frieze, A., Pittel, B.G.: Maximum matchings in sparse random graphs: Karp-Sipser re-visited. Random Structures & Algorithms 12, 111–178 (1998)
4. Barham, M., Dunkelman, O., Lucks, S., Stevens, M.: New Second Preimage Attacks on Dithered Hash Functions with Low Memory Complexity. In: Proceedings of Selected Areas in Cryptology 2016. Springer, to appear in Lecture Notes in Computer Science
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak sponge function family main document. Submission to NIST (Round 2) 3, 30 (2009)
6. Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., Jalby, W.: Collisions of SHA-0 and Reduced SHA-1. In: Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3494, pp. 36–57. Springer (2005)
7. Blackburn, S.R., Stinson, D.R., Upadhyay, J.: On the complexity of the herding attack and some related attacks on hash functions. Designs, Codes and Cryptography 64(1-2), 171–193 (2012)
8. Damgård, I.: A design principle for hash functions. In: Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. Lecture Notes in Computer Science, vol. 435, pp. 416–427. Springer (1989)
9. Dean, R.D.: Formal aspects of mobile code security. Ph.D. thesis, Princeton University, Princeton (1999)
10. Erdös, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci 5, 17–61 (1960)
11. Erdös, P., Rényi, A.: On the strength of connectedness of a random graph. Acta Mathematica Hungarica 12(1-2), 261–267 (1961)
12. Erdös, P., Rényi, A.: On the existence of a factor of degree one of a connected random graph. Acta Mathematica Hungarica 17(3-4), 359–368 (1966)
13. Hoch, Y.Z.: Security analysis of generic iterated hash functions. Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel (2009)
14. Joux, A.: Multicollisions in iterated hash functions. application to cascaded constructions. In: Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3152, pp. 306–316. Springer (2004)
15. Karp, R.M., Sipser, M.: Maximum Matchings in Sparse Random Graphs. In: 22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981. pp. 364–375. IEEE Computer Society (1981), https://doi.org/10.1109/SFCS.1981.21
16. Kelsey, J., Kohno, T.: Herding hash functions and the Nostradamus attack. In: Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings, Lecture Notes in Computer Science, vol. 4004, pp. 183–200. Springer (2006)
17. Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2n work. In: Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3494, pp. 474–490. Springer (2005)
18. Klima, V.: Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications. Cryptology ePrint Archive, Report 2005/102 (2005)

19. Knuth, D.E.: The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1997)
20. Kortelainen, T.: On iteration-based security flaws in modern hash functions. Ph.D. thesis, University of Oulu, Finland (2014)
21. Kortelainen, T., Kortelainen, J.: On diamond structures and Trojan message attacks. In: Advances in Cryptology - ASIACRYPT 2013: 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II. Lecture Notes in Computer Science, vol. 8270, pp. 524–539. Springer (2013)
22. Merkle, R.C.: One way hash functions and DES. In: Conference on the Theory and Application of Cryptology–CRYPTO 1989. Lecture Notes in Computer Science, vol. 435, pp. 428–446. Springer (1989)
23. Nivasch, G.: Cycle Detection Using a Stack. Inf. Process. Lett. 90(3), 135–140 (May 2004)
24. van Oorschot, P.C., Wiener, M.J.: Parallel Collision Search with Cryptanalytic Applications. Journal of Cryptology 12(1), 1–28 (1999)
25. Rivest, R.L.: Abelian Square-Free Dithering for Iterated Hash Functions. Presented at ECrypt Hash Function Workshop, June 21, 2005, Cracow, and at the Cryptographic Hash workshop, November 1, 2005, Gaithersburg, Maryland (August 2005)
26. Sasaki, Y., Aoki, K.: Finding preimages in full MD5 faster than exhaustive search. In: Advances in Cryptology - EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5479, pp. 134–152. Springer (2009)
27. Stevens, M.: Attacks on hash functions and applications. Ph.D. thesis, Leiden University (2012)
28. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A.K., Molnar, D., Osvik, D.A., de Weger, B.: Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In: Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference. Lecture Notes in Computer Science, vol. 5677, pp. 55–69. Springer (2009)
29. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In: Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3494, pp. 1–18. Springer (2005)
30. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3621, pp. 17–36. Springer (2005)
31. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3494, pp. 19–35. Springer (2005)