On the Complexity of the Herding Attack and Some Related Attacks on Hash Functions

D.R. Stinson* and J. Upadhyay
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo Ontario N2L 3G1 Canada

January 18, 2010

Abstract

In this paper, we analyze the complexity of the construction of the 2^k-diamond structure proposed by Kelsey and Kohno [9]. We point out a flaw in their analysis and show that their construction may not produce the desired diamond structure. We then give a more rigorous and detailed complexity analysis of the construction of a diamond structure. For this, we appeal to random graph theory, which allows us to determine sharp necessary and sufficient conditions for the *message complexity* (i.e., the number of hash computations required to build the required structure). We also analyze the *computational complexity* for constructing a diamond structure, which has not been previously studied in the literature. Finally, we study the impact of our analysis on herding and other attacks that use the diamond structure as a subroutine. Precisely, our results shows the following:

1. The message complexity for the construction of a diamond structure is $\sqrt{k}$ times more than what was previously stated in literature.
2. The time complexity is n times the message complexity, where n is the size of hash value.

Due to above two results, the complexity of the herding attack [9] and the second preimage attack [3] on iterated hash functions have increased complexity. We also show that the message complexity of herding and second preimage attacks on "hash twice" is n times the complexity claimed by [2], by giving a more detailed analysis of the attack.

1 Introduction

In a recent paper, Kelsey and Kohno [9] proposed a new attack on Damgård-Merkle hash functions, called the *herding attack*. In such an attack, the adversary first commits to a hash value and is then provided with a prefix. The attacker is required to find a suitable suffix that "herds" to the previously committed hash value. That is, given x, h, and a hash function H, the adversary must find y such that $H(x \parallel y) = h$.

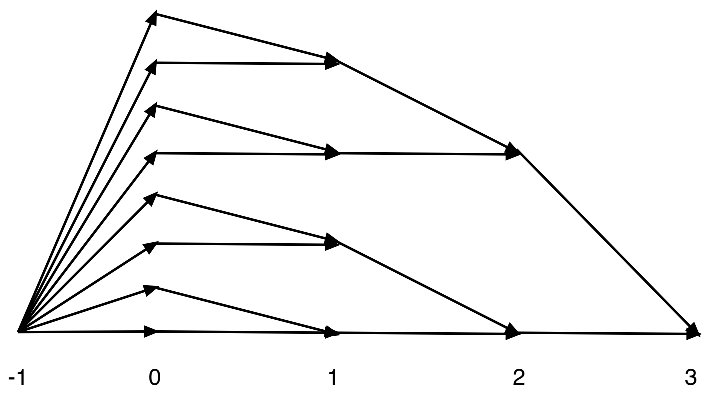
To launch the attack, [9] proposed a 2^k-*diamond structure*, which permits the construction of certain multi-collisions, but which is structurally different from Joux's multicollision attack [7].

A 2^k-diamond structure contains a complete binary tree of depth k. There are $2^{k-\ell}$ nodes at level ℓ, for $k \geq \ell \geq 0$. There is also a single node at level -1, which we will call the *source node*. The source node is joined to every node at level 0. The nodes at level 0 are called the *leaves* of the diamond structure and the node at level k is called the *root* of the tree.

Every edge e in the diamond structure is labeled by a string $\sigma(e)$. Typically, a string $\sigma(e)$ consists of one or more message blocks. We also assign a label $h(N)$ to every node N in the

*Research supported by NSERC grant 203114-06

Figure 1: A 2^3 diamond structure



structure at level at least 0, as follows. Consider the unique directed path P from the source node to the node N in the diamond structure. P will consist of some edges $e_0 e_1 \cdots e_\ell$, where N is at level ℓ in the tree. Then we define

$$h(N) = H(\sigma(e_0) \parallel \sigma(e_1) \parallel \cdots \parallel \sigma(e_\ell)).$$

At any level ℓ of the structure there are $2^{k-\ell}$ hash values. These must be paired up in such a way that, when the next message blocks are appended, $2^{k-\ell-1}$ collisions occur. Thus there are $2^{k-\ell-1}$ hash values at the next level, which is level $\ell + 1$. The entire structure yields a 2^k-multicollision, but it has other uses, as well.

A diagram of a 2^3 diamond structure is given in Figure 1.

The diamond structure has found applications in attacks on hash functions that resisted other techniques such as Joux's multicollision [7] and Kelsey-Schneier's expandable message attack [10]. Andreeva *et al* [3] used the diamond structure to launch a second-preimage attack on Rivest's dithered hash construction [15], which resisted the attack by Kelsey and Schneier [10]. They used the same dithering symbol for all the edges at the same level of the diamond structure. Using a similar technique, they also launched an attack on Shoup's domain extension algorithm for universal one-way hash functions [16]. In a recent paper, Andreeva *et al* [2] extended the application of diamond structure to launch herding attacks on various other variants of hash functions, such as hash twice, concatenated hash, zipper hash, and tree based hash, as well as second preimage attack on hash twice construction. They constructed diamond structures over multicollisions in the same vein as Joux's attack on concatenated hash functions.

All these attacks are based on the analysis of Kelsey and Kohno [9], which claims that construction of a diamond structure is not as expensive as a naive approach would suggest. They stated that for herding an n-bit hash function, one requires $2^{(n+k+4)/2}$ messages to construct a 2^k-diamond structure. In this paper, we show a problem with their analysis and present a corrected and more detailed analysis of the herding attack. We also perform the first analysis of the computational complexity for constructing a diamond structure. The main results are as follows:

1. The message complexity of constructing a 2^k-diamond structure is $\Theta(\sqrt{k} \times 2^{(n+k)/2})$ (Theorem 1).

2. If each hash computation takes unit time, the computational complexity of constructing a 2^k-diamond structure is $O(n \times \sqrt{k} \times 2^{(n+k)/2})$ (Theorem 2).

Using these result, we revisit the analysis of various attacks, such as [9, 3, 2], that are based on diamond structure.

The paper is organized as follows. In Section 2, we outline the definitions and notations that are used in the paper. In Section 3, we formally present the diamond construction and give an overview of the analysis of Kelsey and Kohno [9], pointing out the problem with their analysis. Also, in Section 3, we perform a detailed and rigorous analysis of the construction of a diamond structure. In Section 4, we revisit the complexity of the attacks that are based on diamond structure in the light of our analysis.

2 Preliminaries

In this section, we discuss some preliminaries that are required to understand this paper. We define the notation that we use in the paper, basics of hash functions, and variants of hash functions which are attacked using the diamond structure, as well as some concepts from the graph theory that we require in our analysis.

2.1 Notation

We let $\mathbb{N}$ denote the set of all natural numbers. For any integer $k \in \mathbb{N}$, denote by $[k]$ the set $\{1, 2, \ldots, k\}$. Let $n \in \mathbb{N}$, then $\{0,1\}^n$ denotes all the n-bit strings. For any $x \in \{0,1\}^*$, we denote the bit-length of x by $|x|$. We denote the concatenation of two strings x and y by $x\|y$. We denote the message blocks of any message M by $M_1\|M_2\|\cdots\|M_l$, where l denotes the number of message blocks. If $\mathcal{X}$ is a finite set, then $x \overset{R}{\leftarrow} \mathcal{X}$ denotes that we pick x uniformly at random from the set $\mathcal{X}$, and $x \overset{C}{\leftarrow} \mathcal{X}$ denotes that we commit to x from the set $\mathcal{X}$. We write $\log n$ for the logarithm of n to the base 2 and $\ln n$ for natural logarithm of n. We use the following asymptotic notation. If $f : \mathbb{N} \to \mathbb{R}$ and $g : \mathbb{N} \to \mathbb{R}$ are two functions such that $g(n) > 0$ for n sufficiently large, then we write:

$$\begin{aligned} f \ll g \quad &\text{if} \quad f(n)/g(n) \to 0 \text{ as } n \to \infty \\ f \gg g \quad &\text{if} \quad f(n)/g(n) \to \infty \text{ as } n \to \infty. \end{aligned}$$

2.2 Basic hash function constructions

Hash functions have been defined in two settings: traditional keyless hash function and as a family of hash functions. In a traditional keyless hash function, we have a single hash function H that maps an arbitrary length input to a fixed length output. We now give a formal definition of the keyed hash functions.

Definition 2.1. *For a finite* key space $\mathcal{K}$*, a space of* messages $\mathcal{M}$*, and a finite space of possible outputs called* message digests*,* $\mathcal{Y}$*, we define a* hash function family *as*

$$H : \mathcal{K} \times \mathcal{M} \to \mathcal{Y}.$$

We denote the family of hash functions by $\mathcal{H} := \{H_k\}_{k \in \mathcal{K}}$.

A key $k \in \mathcal{K}$ acts as an index which defines which hash function is chosen from the family. We usually denote the hash function $H(k, M)$ by $H_k(M)$ for $k \in \mathcal{K}$ and $M \in \mathcal{M}$. We even drop the subscript when the function is clear from the context. Note that a traditional keyless hash function is a hash family with $|\mathcal{K}| = 1$.

A hash function, h, with $\mathcal{Y} \in \{0,1\}^n$ is required to have three basic security properties [17, 18, 8]: collision resistance, preimage resistance, and second preimage resistance. Kelsey and Kohno [9] defined a new security property: the chosen-target-forced-prefix resistance. In this paper, we deal with second preimage resistance and chosen-target-forced-prefix resistance. These are defined informally as follows.

1. *Second preimage-resistance*: An attacker, when given one message M, should not be able to find another message M' such that $h(M) = h(M')$ with less than 2^n hash computations.

2. *Chosen-target-forced-prefix resistance*: An attacker commits to a hash value, h_c, and is then challenged with a *prefix* P. The attacker should not be able to find a *suffix* S such that $h(P\|S) = h_c$ with less than 2^n hash computations. The attack is also called the *herding attack*.

A second preimage attack or herding attack with less than 2^n work is considered to be a *break* of the hash function.

In this paper, we deal with a standard form of hash function, the *iterated hash function*. In an iterated hash function, we first define a function over a small domain, called the *compression function*, and then we extend the domain by using the compression function in an iterative manner.

Let $f : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$ be a compression function. We denote the n-bit output of every iteration by h and call them *chaining values*; they are the input to the next iteration. The hash function based on a compression function f is denoted by H^f. With a little abuse of notation, we drop the super-script if the compression function is clear from the context. For the rest of the paper, we assume that the message M is padded such that the padded message is a multiple of b. For this section, we assume that the padded message has a bit length bl and for simplicity, we denote the padded message by M. We represent the message in block form as $M = M_1\|M_2\| \cdots \|M_l$, where $|M_i| = b$ for all $i \in [l]$.

We next define some iterative hash functions that are susceptible to the herding attack and second preimage attack using the diamond structure.

MERKLE-DAMGARD.

The *Merkle-Damgård construction* is simple iteration, where

$$h_i = f(h_{i-1}, M_i) \ \forall i \in [l]$$

and h_0 is a publicly known initial hash value. The output of the computation is $\mathcal{MD}^f := h_l$.

HASH TWICE.

In *hash twice*, one hashes two consecutive copies of the message M. Formally, for a hash function H, it is defined as

$$\mathcal{HT} := H(H(IV, M), M)$$

where IV is the publicly known initial hash value.

DITHERED HASH FUNCTION

In the *dithered hash function*, every call to the compression function has three inputs: the *dithering sequence* which depends on the iteration, the chaining hash value, and the next message block. In the construction of Rivest [15], the author uses a dithering sequence that is abelian square free. We give a brief overview of such sequence to the generality required to understand the construction of dithered hash functions. For details, we refer the reader to reference [1].

Definition 2.2. *A* word *w is a sequence of letters over some finite alphabet. If a word w can be written as xyz (where y is non-empty and x and/or z can be empty), then y is called a* factor *of w. A word w is called* square free *if no factor of w can be represented in the form yy for any non-empty word y.*

Thus, square free words are strongly non-periodic and non-repeating. A much stronger notion of non-repeating word is *abelian square free word* which Rivest used in his construction.

Definition 2.3. *A word w is said to be* abelian square free *if it cannot be written in the form $w = xyy'z$, where y is a non-empty word and y' is a permutation of y.*

To capture the idea correctly, abcbca is square free, but not abelian square free as abc is followed by bca which is a permutation of abc.

Rivest construction. Let $\mathbf{z} = \{z_i\}_{i=0}^{\infty}$ be any abelian square free dithering sequence, then the construction of Rivest is

$$h_i = f(h_{i-1}, M_i, z_i) \text{ for all } i \in [l],$$

where $|M_i| + |z_i| = b$ and $\mathcal{DH}^f := h_l$.

2.3 Random graph theory

Our analysis uses some basic notions from graph theory (for more information, see a standard textbook such as Bondy and Murty [4]). A *graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a finite set of *vertices* (or *nodes*), $\mathcal{V}$, and a finite set of *edges*, $\mathcal{E}$, where an edge joins two distinct vertices. The edge joining two vertices u and v is denoted by the pair (u, v) or (v, u), and the vertices u and v are called *adjacent* vertices. A *path* is a finite set of vertices $(v_1, \ldots, v_k)$, such that v_i and v_{i+1} are adjacent for $1 \leq i \leq k-1$. If it is possible to reach any vertex from any other vertex in the graph via some finite path, then the graph is *connected.* If there exists a set of edges, no two of which share a vertex, then the set of edges is called a *matching.* M is a *maximum matching* in $\mathcal{G}$ if no matching in $\mathcal{G}$ contains more edges than M does. If a matching M contains every vertex of $\mathcal{G}$, then M is called a *perfect matching* or a *one-factor.*

All the above-defined characteristics of graph are preserved under any relabelling of the vertices. We term such a characteristics of a graph as a *property.* More formally, we have the following definition.

Definition 2.4. *A* graph property *is defined to be a predicate that is preserved under all possible isomorphisms of a graph. A property of a graph is* monotone *if the property is preserved by addition of arbitrary new edges to the graph.*

Many natural properties of graphs are monotone properties, e.g., being connected, being two-connected, containing a Hamiltonian cycle, containing a perfect matching, containing a triangle, etc.

In this paper, we deal with random graphs. A *random graph* $\mathcal{G}(\nu, p)$, is a graph on ν labelled vertices, obtained by selecting each pair of vertices to be an edge randomly and independently with a fixed probability p. This model is commonly known as the *Erdös-Rényi model.*

The probability p has a very important role, as it can be seen as a parameter which determines the sparsity or density of the graph. As p ranges from 0 to 1, the graph becomes more and more dense on average. Moreover, many natural monotone graph-theoretic properties become true within a very small range of values of p. More precisely, given a monotone graph-theoretic property, there is typically a value of p (which will be a function of ν, the number of vertices) called the called *threshold function.* The given property holds in the model $\mathcal{G}(\nu, p)$ with probability close to 0 for p less than the threshold, and the property holds with probability close to 1 for p greater than the threshold. Formally, the notion of a threshold function is defined as follows:

Definition 2.5. *A function $t(\nu)$ is a* threshold function *for a monotone property P if, whenever $p \ll t(\nu)$, then $\mathcal{G}(\nu, p)$ does not satisfy P almost always, and whenever $p \gg t(\nu)$, then $\mathcal{G}(\nu, p)$ satisfies P almost always.*

In random graph theory, threshold functions play a very important role. However, it should be noted that a threshold function is not unique for a particular graph property. For example, $1/\nu + 1/\nu^2$ and $10^{10}\nu^{-1}$ are both threshold functions for the property of containing a triangle.

3 Analysis of the complexity of constructing a diamond structure

We begin by recalling the analysis provided by Kelsey and Kohno [9]. Before we begin, we note that this analysis implies that $n > k$. Kelsey and Kohno [9] argued as follows:

> *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 $2^{n/2+k/2+1/2}$ messages spread out from 2^k starting hash values (lines), we get $2^{n/2+k/2+1/2-k}$ messages per line, and thus between any pair of these starting hash values, we expect about $(2^{n/2+k/2+1/2-k})^2 \times 2^{-n} = 2^{n+k+1-2k-n} = 2^{-k+1}$ collisions. We thus expect about $2^{-k+k+1} = 2$ other hash values to collide with any given starting hash value.*

We agree with this conclusion. Unfortunately, we will prove that this line of reasoning does not imply that the 2^k nodes can be paired up in such a way that we get 2^{k-1} collisions. It is useful to think of this problem in a graph-theoretic setting. Suppose we label the nodes as $1, 2, \ldots, 2^k$. Then we construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in the following manner. The vertex set is $\mathcal{V} = \{v_1, \ldots, v_{2^k}\}$ and $(v_i, v_j) \in \mathcal{E}$ if the nodes i and j collide at the next level of the diamond structure. Based on the analysis given above, we see that the graph $\mathcal{G}$ is precisely a random graph in $\mathcal{G}(2^k, 2^{-k+1})$. Now, the question is if this random graph contains a perfect matching, as this is precisely what is required in order to be able to find the desired 2^{k-1} pairs of collisions.

Unfortunately, random graph theory tells us that it is very unlikely that there is a perfect matching in a random graph in $\mathcal{G}(2^k, 2^{-k+1})$. This is because Erdös and Rényi [6] proved that the threshold function for having a perfect matching in $\mathcal{G}(\nu, p)$ is roughly $\ln \nu/\nu$. The class of random graphs $\mathcal{G}(2^k, 2^{-k+1})$ has $p = 2/\nu$, which is well below the required threshold.

More precisely, the threshold function for having a perfect matching is any function having the form

$$t(\nu) = \frac{\ln \nu + f(\nu)}{\nu} \tag{1}$$

for any $f(\nu)$ such that $\lim_{\nu \to \infty} f(\nu) = \infty$.

Note also that the threshold function for not having an isolated vertex is $\ln \nu/\nu$ [5], and a graph that contains an isolated vertex obviously cannot contain a perfect matching.

3.1 Fixing the problem

In our analysis, in order to simplify the assumptions and algebra, we will assume that a random graph in $\mathcal{G}(\nu, \ln \nu/\nu)$ will have a perfect matching. This is perhaps a tiny bit optimistic in view of (1), but it is close enough for our purposes.[1]

We proceed in a similar manner to [9]. Suppose we construct $\nu = 2^k$ lists, each containing L messages. The probability that any two given messages collide under H is 2^{-n}. The probability that no message in one list collides with a message in another list can be estimated to be

$$\left(1 - \frac{1}{2^n}\right)^{L^2} \approx 1 - \frac{L^2}{2^n}.$$

[1]To be completely rigorous, we could construct a random graph in $\mathcal{G}(\nu, c \ln \nu/\nu)$, for any constant $c > 1$. However, this would have no impact on our asymptotic computations.

The probability (which we denote by p) that there is at least one collision between two given lists is

$$p \approx \frac{L^2}{2^n}. \tag{2}$$

In view of our analysis above, we want

$$p \approx \frac{\ln \nu}{\nu}.$$

Recalling that $\nu = 2^k$, it is clear that we need to take

$$L \approx \sqrt{k \ln 2} \times 2^{(n-k)/2} \approx 0.83 \times \sqrt{k} \times 2^{(n-k)/2}. \tag{3}$$

The *message complexity* (i.e., the number of hash computations) at level 0 is therefore

$$2^k L \approx 0.83 \times \sqrt{k} \times 2^{(n+k)/2}. \tag{4}$$

Ignoring constant factors, this is a factor of about $\sqrt{k}$ bigger than the estimate in [9].

Remark. If we take $L = 2^{(n-k+1)/2}$ in (2), then we obtain $p \approx 2^{-k+1}$, in agreement with [9].

Theorem 1. *The total message complexity required to construct a 2^k-diamond structure is $\Theta(\sqrt{k} \times 2^{(n+k)/2})$.*

Proof. The entire diamond structure requires finding collisions at levels $0, 1, \ldots, k-1$. The above analysis shows that, at level ℓ, the message complexity is about $0.83 \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2}$. Therefore the total message complexity is

$$\begin{aligned}
\sum_{\ell=0}^{k-1} 0.83 \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2} &= \sum_{i=1}^{k} 0.83 \times \sqrt{i} \times 2^{(n+i)/2} \\
&= 0.83 \times 2^{n/2} \sum_{i=1}^{k} \sqrt{i} \times 2^{i/2} \\
&< 0.83 \times \sqrt{k} \times 2^{n/2} \sum_{i=1}^{k} 2^{i/2} \\
&= 0.83 \times \sqrt{k} \times 2^{n/2} \times \frac{2^{1/2}(2^{(k+1)/2} - 1)}{2^{1/2} - 1} \\
&= O(\sqrt{k} \times 2^{(n+k)/2}). \qquad (5)
\end{aligned}$$

For the lower bound, we have

$$\begin{aligned}
\sum_{\ell=0}^{k-1} 0.83 \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2} &= \sum_{i=1}^{k} 0.83 \times \sqrt{i} \times 2^{(n+i)/2} \\
&= 0.83 \times 2^{n/2} \sum_{i=1}^{k} \sqrt{i} \times 2^{i/2} \\
&> 0.83 \times \sqrt{\frac{k}{2}} \times 2^{n/2} \times \sum_{i=k/2}^{k} 2^{i/2} \\
&= 0.83 \times \sqrt{\frac{k}{2}} \times 2^{n/2} \times \frac{2^{k/4}(2^{(k/2+1)/2} - 1)}{\sqrt{2} - 1} \\
&= \Omega(\sqrt{k} \times 2^{(n+k)/2}).
\end{aligned}$$

The result follows. □

3.2 Computational complexity

So far, we have only considered the message complexity of constructing the diamond structure. In this section, we look at the computational complexity. The computational complexity of constructing a diamond structure has not previously been considered in the literature. However, this clearly an important and relevant issue if these structures are ever going to be implemented in a real attack.

There are three main steps required to proceed from one level of the diamond structure to the next. As before, we start by analyzing the work done to go from level 0 to level 1 (the work done at other levels can be analyzed in the same way).

1. Compute $L = 0.83 \times \sqrt{k} \times 2^{(n-k)/2}$ hash values for each of the 2^k lists.
2. Construct the associated graph, i.e., for each pair of lists, determine if there is a common hash value.
3. Determine if the associated graph contains a perfect matching.

Under the assumption that each hash computation takes unit time, the complexity of step 1 is just the message complexity, which we have already computed to be $\Theta(2^k L)$ in (4).

In step 2, we have to search every pair of lists for a repeated value. Various solutions are possible. Asymptotically, we cannot do better than concatenating all the lists, sorting them, and then performing a single pass through the sorted list to detect duplicates. The total time is therefore

$$O(2^k L \log(2^k L)).$$

In step 3, we need to find a perfect matching in a graph on 2^k vertices. Finding a maximum matching in a graph on ν vertices, with m edges and with average degree at least $\ln \nu$ can be done in time $O\left((m \log \nu)/(\log\log \nu)\right)$ [14][2]. In our case, we have a graph that almost surely contains a perfect matching and the expected number of edges is $k \times 2^k$, so an algorithm that finds a maximum matching will in fact find a perfect matching. This will take time

$$O\left(\frac{k^2 2^k}{\log k}\right).$$

Combining the three steps, we see that the total time required at level 0 is

$$O\left(2^k L + 2^k L \log(2^k L) + \frac{k^2 2^k}{\log k}\right) = O\left(2^k L \log(2^k L) + \frac{k^2 2^k}{\log k}\right).$$

Recall from (3) that $L = \Theta(\sqrt{k} \times 2^{(n-k)/2})$. Then we have

$$\begin{aligned} 2^k L \log(2^k L) &= \Theta(\sqrt{k} \times 2^{(n+k)/2} \times \log(\sqrt{k} \times 2^{(n+k)/2})) \\ &= \Theta\left(\sqrt{k} \times 2^{(n+k)/2} \times \left(\frac{n+k}{2} + \frac{1}{2}\log k\right)\right) \\ &= \Theta(\sqrt{k} \times 2^{(n+k)/2} \times n), \end{aligned} \tag{6}$$

since $k < n$.

[2]Note that the algorithm of Motwani [14] is a randomized algorithm that finds a maximal matching with high probability. The best known algorithm under worst case analysis is due to Micali and Vazirani [13]; the running time is $O(m\sqrt{\nu})$. However, for our purposes, the randomized algorithm of Motwani suffices.

The total computation time at level 0 is thus

$$O\left(\sqrt{k} \times 2^{(n+k)/2} \times n + \frac{k^2 2^k}{\log k}\right). \tag{7}$$

Finally, we determine the total computation time over all k levels. From (7), this total is seen to be

$$O\left(\sum_{\ell=0}^{k-1} n \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2} + \frac{(k-\ell)^2 2^{k-\ell}}{\log(k-\ell)}\right). \tag{8}$$

The first part of the sum is just the message complexity multiplied by n. For the second part, note that

$$\begin{aligned} \sum_{\ell=0}^{k-1} \frac{(k-\ell)^2 \times 2^{k-\ell}}{\log(k-\ell)} &< \sum_{\ell=0}^{k-1} \left((k-\ell)^2 \times 2^{k-\ell}\right) \\ &= \underbrace{\sum_{i=1}^{k} \left(i^2 \times 2^i\right)}_{S}. \end{aligned} \tag{9}$$

We now evaluate the sum S

$$\begin{aligned} S &= 1^2 \times 2 + 2^2 \times 2^2 + 3^2 \times 2^3 + \cdots k^2 \times 2^k & (10) \\ 2S &= 1^2 \times 2^2 + 2^2 \times 2^3 + 3^2 \times 2^4 \cdots (k-1)^2 \times 2^k + k^2 \times 2^{k+1} & (11) \end{aligned}$$

(11)−(10) yields

$$S = k^2 \times 2^{k+1} - \sum_{i=1}^{k} (i^2 - (i-1)^2) \times 2^i = O(k^2 \times 2^k) \tag{12}$$

Combining equations (8) and (12), we have the total computation time as,

$$O(n \times \sqrt{k} \times 2^{(n+k)/2} + k^2 \times 2^k) = O(n \times \sqrt{k} \times 2^{(n+k)/2}),$$

since $n > k$. We can summarize it as the following theorem.

Theorem 2. *If each hash computation takes unit time, the computational complexity of constructing a 2^k-diamond structure is*

$$O(n \times \sqrt{k} \times 2^{(n+k)/2}). \tag{13}$$

Therefore, the computational complexity is n times the message complexity.

4 Revised analysis of the other attacks

The diamond structure has been used in second preimage attacks [3, 2] and in herding attacks [9, 2]. The steps followed in both attacks are the same except that they occur in a different order. In case of a herding attack, the attacker first computes the hash of the prefix, then finds a linking message that links the hash of prefix to one of the intermediate hash values of the diamond structure. Finally, she appends the messages on the path inside the diamond structure from the intermediate node to the root in front of the prefix and the linking message. The suffix is the message formed by appending the linking message with the message on the path.

In the case of finding a second preimage, the attacker first finds a linking message that links the root of the diamond structure to one of the intermediate hash value of the original message.

Then she finds a prefix that hashes to one of the nodes at level 0 of the diamond structure. She then appends the messages associated with the edges on the path inside the diamond structure that leads to the root of the diamond structure. The output is the message in the proper order.

We analyze the complexity of some known attacks in light of our revised analysis of the diamond structure. We use the term *offline phase* for the steps carried out by adversary before she is given the challenge and the term *online phase* for the steps followed by adversary after she is provided with the challenge.

4.1 Herding attack on Merkle-Damgård construction

Kelsey and Kohno [9] proposed the construction of diamond structure and the herding attack on Mekle-Damgård construction using the diamond structure. According to their analysis, the message complexity of the attack is $O(2^{(n+k)/2} + 2^{n-k})$. We revisit the attack in the light of our correction of their analysis.

Message complexity

We proceed step by step. We already analyzed the complexity of the construction of the diamond structure. In the online phase, the attacker has to find a linking message that links the hash of the prefix to one of the leaves of the diamond structure. The complexity of finding the linking message is $O(2^{n-k})$. Therefore, from Theorem 1, we can calculate the message complexity of the attack as

$$O(\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k}).$$

From the arithmetic-geometric mean inequality, we can find the minimum of the message complexity by finding the value of k satisfying the following equation

$$n - 3k - \log k = 0$$

We do not know a method to solve this equation exactly in k. However, if we see the trend of first few values by using Newton-Raphson's method, we see that the root of the above equation is approximately $n/3$. For lower values of k, the message complexity is dominated by $O(2^{n-k})$. Therefore, we see the same trend as in [9]. However, for larger values of k, the message complexity is dominated by the construction of diamond structure. In this case, the major contribution is from the term $O(\sqrt{k} \times 2^{(n+k)/2})$ and we see the factor of $\sqrt{k}$ coming in to the picture.

Time complexity

The time complexity of the offline phase is the same as given by Theorem 2. However, in the online phase, we need to find the linking message and then actually find the hash value at the leaves to which it maps. We can safely assume that the nodes at level 0 are sorted by their hash values. This is because, if the nodes are not sorted by their hash values, we can perform the sorting in the offline phase, which takes $O(k \times 2^k)$ time. However, this has no impact in asymptotic analysis as it get subsumed by the complexity of construction of diamond structure (we will use this fact in the sequel and never explicitly mention it). Hence, the time complexity to hash the required number of messages and then to find if it links to any of the leaves is $O(k \times 2^{n-k})$. Therefore, the time complexity comes out to be

$$O(n \times \sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k}).$$

The minimum occurs at the solution of the equation

$$n - 3k - 2\log n + \log k = 0.$$

Again, calculating the first few value using the Newton-Raphson's method, we see that the root can be approximated as $k = n/3$. When k is larger than $n/3$, then the time complexity is n times the message complexity and it is k times the message complexity for smaller values of k.

We tabulate the complexity for some fixed values of n and compare our result with the analysis of [9] for certain values of k in Table 1.

4.2 Second preimage attack on dithered hash

The dithered hash construction was proposed by Rivest [15] to counter the second preimage attack of Kelsey and Schenier [10]. However, recently, Andreeva *et al.* [3] proposed an attack based on the diamond construction. For a challenge message M of block length 2^l, the adversary first finds a linking message that links the root of the diamond structure to one of intermediate hash values in the hash computation of M. She then finds a linking message that hashes to one of the intermediate hash values (say N_0) in the diamond structure. The adversary then finds the path inside the diamond structure that leads from N_0 to the root.

The success of the attack depends on the fact that although dithering sequences are non-periodic and non-repeating, they have large size factors. The attack uses such a factor with the messages that are used in the construction of diamond structure. If $Fact_{\mathbf{z}}(\ell)$ denote the number of factors of size ℓ in the sequence $\mathbf{z}$, then Andreeva *et al.* evaluated the message complexity of their attack as

$$O(Fact_{\mathbf{z}}(\ell) \times 2^{n-l} + 2^{n-k} + 2^{(n+k)/2})$$

They also stated following lemmas about two dithering sequences that were advocated by Rivest.

Lemma 3. *For $\ell \leq 85$, for the Keränen sequence* [11, 12] *$\boldsymbol{z}$, we have*

$$Fact_{\boldsymbol{z}}(\ell) \leq 8l + 332.$$

Lemma 4. *Let $\boldsymbol{c}$ denote the sequence obtained by diluting the Keränen sequence $\boldsymbol{z}$ with a 13-bit counter* [15]*. Then for every $\ell \in [0, 2^{13}]$, we have*

$$Fact_{\boldsymbol{c}}(\ell) = 8l + 32760.$$

In other words, if we represent the above dithering sequence by $\mathbf{z}$, the number of factors of size ℓ is bounded by

$$Fact_{\mathbf{z}}(\ell) = O(\ell)$$

Using the Lemma 3 and Lemma 4, the message complexity evaluated by [3] for the second preimage of a 2^l-block message, using a 2^k-diamond structure is

$$O(k \times 2^{n-l} + 2^{n-k} + 2^{(n+k)/2}).$$

n	Example	$k = \lfloor n/3 \rfloor$			$k = \lfloor 2n/5 \rfloor$		
		Message Complexity		Time Complexity	Message Complexity		Time Complexity
		[9]	Actual		[9]	Actual	
128	MD5	2^{85}	2^{88}	2^{96}	2^{89}	2^{93}	2^{100}
160	SHA-1	2^{106}	2^{110}	2^{117}	2^{112}	2^{117}	2^{123}
192	Tiger	2^{128}	2^{131}	2^{139}	2^{134}	2^{138}	2^{145}
256	SHA-256	2^{170}	2^{174}	2^{182}	2^{179}	2^{183}	2^{191}
512	Whirlpool	2^{341}	2^{345}	2^{354}	2^{358}	2^{362}	2^{371}

Table 1: Comparison of complexity of herding attack (all calculations are done without considering the constants.)

We reconsider this analysis in the light of our analysis of diamond structure.

4.2.1 Message complexity

In the attack on the dithered hash, Andreeva *et al* used the same dithering sequence for all edges at the same depth of the diamond structure, and one dithered sequence to connect the root of diamond structure to M. Thus, for a 2^k-diamond structure, the size of dithering sequence they need on the diamond structure is exactly $k+1$. In the worst case, when all the factors have same size in the dithering sequence, the probability that a randomly chosen factor of size k in the sequence $\mathbf{z}$ is the one used in the construction of diamond structure is $(Fact_{\mathbf{z}}(k))^{-1}$. Therefore, the message complexity to connect the root of the diamond structure to M is

$$O(Fact_{\mathbf{z}}(k+1) \times 2^{n-l}) = O(k \times 2^{n-l}). \tag{14}$$

Suppose the root gets linked to the i^{th} iteration of the hashing of M.

We already calculated the message complexity of the construction of diamond structure in Theorem 1. To find the complexity of the attack, note that the attacker needs to hash a message (let us say $M^{'}$) of block length $i-2-k$, where k is the depth of diamond structure, in order to defy the Merkle-Damgård strengthening. This is upper bounded by 2^k hash computations. Since $k < n$, we see that this has no impact on the asymptotic calculation as it get subsumed by the construction of the diamond structure (we use this fact again in sequel and never explicitly mention it). The message complexity to find a linking message that links $H(M^{'})$ to one of the leaves of the diamond structure is $O(2^{n-k})$. We calculate the message complexity of the second preimage attack on dithered hash function as

$$O\left(2^k + \sqrt{k} \times 2^{(n+k)/2} + 2^{n-k} + k \times 2^{n-l}\right) = O\left(\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k} + k \times 2^{n-l}\right),$$

Under the assumption that $l \approx k$, it can be further simplified to

$$O\left(\sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k}\right).$$

4.2.2 Computational complexity

We note that the attack uses the simple diamond structure as constructed in Section 3. Hence, the total time required to construct the diamond structure is as stated in Theorem 2. For the time required in the online phase, note that we need to find the intermediate hash value to which we link the root of the diamond structure. Moreover, we can safely assume that the intermediate hash values in the hash computation of M are sorted. If not, then we first sort them in the online phase, which takes $O(l \times 2^l)$ time. Note that it does not effect the computational complexity, because it get subsumed by the construction of the diamond structure (we implicitly use this fact again in the analysis of second preimage attack on hash twice construction). Hence, it takes a factor of l time more than what is required in message complexity (equation (14)) resulting in total time $O(l \times k \times 2^{n-l})$. Also, as in herding attack, we need to find the leaf to which the linking message maps. Thus, the time complexity to find that leaf is $O(k \times 2^{n-k})$. Therefore, the total time required for the attack is given by

$$T(k,l) = O\left(k \times 2^{n-k} + k \times l \times 2^{n-l} + n \times \sqrt{k} \times 2^{(n+k)/2}\right).$$

Under the assumption that $k \approx l$, we have

$$\begin{aligned} M(k,l) &= O\left(\sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k}\right). \\ T(k,l) &= O\left(k^2 \times 2^{n-k} + n \times \sqrt{k} \times 2^{(n+k)/2}\right). \end{aligned}$$

It is easy to check that the message complexity is minimized when k is the solution of following equation,

$$n - 3k + \log k = 0,$$

which is roughly $n/3$. Following the argument in Section 4.1, we can say that the message complexity of the attack is dominated by $O(k \times 2^{n-k})$ for $k < n/3$ and therefore it is the same as in [3]. However, for large values of k, a factor of $\sqrt{k}$ comes into the picture.

Also, the time complexity is minimized when k satisfies the following equation:

$$n - 3k + 3\log k - \log n = 0$$

We present a comparison of the computational complexity of the attack with the message complexity and compare it with the analysis of [3] in Table 2.

4.3 Herding attack and second preimage attack on hash twice function

Andreeva *et al* [2] proposed a herding attack and a second preimage attack on the hash twice function by building a diamond structure on multi-collision. For a hash twice function of the form defined in Section 2.2, both the attacks constructed Joux's multi-collision on first pass of hash function, and then used the messages that caused the multi-collision to build a diamond structure for the second pass. In total, the attack requires the construction of two diamond structures, one on each of the two passes. In fact, the adversary performs the following steps during the offline phase:

1. Construct a 2^k-diamond structure (let us say $\mathcal{D}_1$) for the first pass.
2. Construct 2^{n-k+r}-multicollision (let us say $\mathcal{M}$) in front of $\mathcal{D}_1$, where the value of r is to be calculated later. Let the hash value at the end of multi-collision be h_1.
3. Construct a diamond structure (let us say $\mathcal{D}_2$) on the second pass using the messages from the last r blocks of messages in $\mathcal{M}$.

n	Example	$k = \lfloor n/3 \rfloor$			$k = \lfloor 2n/5 \rfloor$		
		Message Complexity		Time Complexity	Message Complexity		Time Complexity
		[3]	Actual		[3]	Actual	
128	MD5	2^{89}	2^{91}	2^{95}	2^{90}	2^{93}	2^{100}
160	SHA-1	2^{111}	2^{113}	2^{117}	2^{112}	2^{115}	2^{122}
192	Tiger	2^{132}	2^{134}	2^{139}	2^{134}	2^{138}	2^{145}
256	SHA-256	2^{175}	2^{177}	2^{182}	2^{179}	2^{183}	2^{191}
512	Whirlpool	2^{347}	2^{349}	2^{354}	2^{358}	2^{362}	2^{371}

Table 2: Comparison of message and computational complexity of second preimage attack on dithered hash (all calculations are done without considering the constants).

4.3.1 Herding attack on hash twice

After the offline phase, the adversary commits to the hash value at the root of $\mathcal{D}_2$. In the online phase, the adversary is given a prefix P. She calculates $H(P)$ and then finds a linking message, m that links $H(P)$ to one of the leaves of $\mathcal{D}_1$. Using h_1 as inital value, the adversary hashes $P\|m$. She uses the first 2^{n-k}-multicollisions of $\mathcal{M}$ to find the linking message to $\mathcal{D}_2$. Finally, she finds a path inside $\mathcal{D}_2$ that leads to the root. Andreeva *et al* [2] calculated the total message complexity of the attack to be $O(2^{n-k}+2^{(n+k)/2})$. We next perform a more detailed analysis of the attack and compute the message complexity and the computational complexity of the attack.

Message complexity

Step 1 is the simple diamond construction for which we gave a detailed analysis in Section 3. To find the complexity of Step 2, we need to calculate the value of r, the number of blocks of messages required to construct the second diamond structure. Since, we need $0.83 \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2}$ messages to herd from level ℓ to level $(\ell+1)$, the total number of multicollisions required to construct the diamond structure is

$$C = \sum_{\ell=0}^{k-1} 0.83 \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2} = \Theta(\sqrt{k} \times 2^{(n+k)/2}).$$

The last equality is due to equation (5). Hence, the message complexity to find a 2^{n-k+r}-multicollision is

$$\Theta\left(((n-k)+\log C)\, 2^{n/2}\right) = \Theta(n \times 2^{n/2}) \tag{15}$$

For Step 3, we find the message complexity to go from level 0 to level 1 (the rest of them can be done analogously). Note that every message that needs to be hashed is in fact of block length $\log 2^k L$ and there are $2^k L$ such messages. From equation (6), the total hash computation is,

$$2^k L \log 2^k L = \Theta(n \times \sqrt{k} \times 2^{(n+k)/2}).$$

Therefore, the message complexity to construct $\mathcal{D}_2$ is

$$\Theta\left(\sum_{\ell=0}^{k-1} \sqrt{k-\ell} \times 2^{(n+k-\ell)/2} \times n\right) = \Theta(n \times \sqrt{k} \times 2^{(n+k)/2}). \tag{16}$$

For the online phase, we comment that the total time is the time required to find the two linking messages (one to $\mathcal{D}_1$ and another one to $\mathcal{D}_2$). The work done to find the linking message to $\mathcal{D}_1$ is simply 2^{n-k}. However, the message complexity to find the linking message to $\mathcal{D}_2$ is $O((n-k)2^{n-k})$, because each message is $n-k$ blocks long. Therefore, the total message complexity in the online phase is

$$O((n-k)2^{n-k}) = O(n \times 2^{n-k}) \tag{17}$$

Using equation (5) and (15)$-$(17), the message complexity for the attack is

$$O\left(\sqrt{k} \times 2^{(n+k)/2} + n \times \left(2^{n/2} + \sqrt{k} \times 2^{(n+k)/2} + 2^{n-k}\right)\right) = O\left(n \times \left(\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k}\right)\right).$$

Computational complexity

Since $\mathcal{D}_1$ is a simple diamond structure, we note that the time required to construct $\mathcal{D}_1$ is the same as equation (13). Also, Step 2 is just finding the required number of multicollisions. Therefore, its time complexity equals the message complexity of finding the required number of multicollisions, which is precisely equal to equation (15). For the final part of the attack, we analyze each step of Section 3.2 for the construction of $\mathcal{D}_2$ for level 0 (the other levels are analyzed analogously). Recall that to proceed from one level to the next in a diamond structure, we first hash certain lists of messages, construct an associated graph by creating an edge when a pair of lists has a common hash value, and then find a perfect matching on that graph.

For the construction of $\mathcal{D}_2$, we have L lists of messages for each of 2^k hash values, but now all the messages are $\log(2^k L)$ blocks long. Using equation (6), we calculate the total time required to hash the $2^k L$ lists as

$$2^k L \log(2^k L) = \Theta(\sqrt{k} \times 2^{(n+k)/2} \times n).$$

Now, we comment that the time complexity to construct the associated graph and then to find a perfect matching in that graph for $\mathcal{D}_2$ is same as in the construction of a simple diamond structure. This is because we need to sort the same number of hash values (which defines the complexity of the construction of the graph) and we have the same number of vertices, the same expected number of edges, and the same average degree (which determines the complexity of finding a perfect matching). Therefore, the total time required at level 0 is,

$$O\left(2^k L \log(2^k L) + 2^k L \log(2^k L) + \frac{k^2 2^k}{\log k}\right) = O\left(2^k L \log(2^k L) + \frac{k^2 2^k}{\log k}\right).$$

Note that this is same as in the construction of simple diamond structure. Hence, the time complexity of the construction of $\mathcal{D}_2$ is the same as equation (13).

Arguing in a similar fashion as in the previous cases for the online phase, we comment that the total time is k times the message complexity to link the message. Therefore, the total time complexity of the attack (pre-computation and online phase) is

$$O\left(n \times \left(2^{n/2} + \sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k}\right)\right) = O\left(n \times \left(\sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k}\right)\right).$$

Therefore, the message complexity of the attack on hash twice is just n times the message complexity of the herding attack of Kelsey and Kohno (see Section 4.1 and compare Table 1 and Table 3).

4.3.2 Second preimage attack on hash twice

Let M be a 2^l blocks long challenge message for the adversary. The steps followed by the adversary are similar to the herding attack. The only additional step for the adversary is in the online phase when she creates the prefix of the required block length that links to one of the leaves of $\mathcal{D}_1$ and has to find a linking message that links the root of $\mathcal{D}_2$ to one of the intermediate hash values in the second pass of M. The analysis of [2] showed the message complexity of the attack is $O(2^{n-k} + 2^{(n+k)/2} + 2^{n-l})$. We perform a more detailed analysis and tabulate the effect of the difference in the Table 3.

Let M be a 2^l block challenge message for the adversary. The steps followed by the adversary are similar to the herding attack. She creates two diamond structures $\mathcal{D}_1$ and $\mathcal{D}_2$ in a similar manner as in the herding attack. The only additional step for the adversary is in the online phase when she creates the prefix of the required block length that links to one of the leaves of $\mathcal{D}_1$ and has to find a linking message that links the root of $\mathcal{D}_2$ to one of the intermediate hash values in the second pass of M. Hence, the total hash computation is $O(2^{n-k})$ for finding the linking message to $\mathcal{D}_1$ and $O(2^{n-l})$ to find the linking message from the root of $\mathcal{D}_2$ to one of the intermediate hash values in the second pass of M. However, every message that the adversary uses to link to $\mathcal{D}_2$ is $n - k$ blocks long. Therefore, the number of hash computations the adversary has to perform is

$O((n-k)2^{n-k})$. Thus the total message complexity of the online phase is

$$O(2^{n-l} + (n-k) \times 2^{n-k} + 2^{n-k}) = O(2^{n-l} + n \times 2^{n-k}).$$

We can now estimate the message complexity as

$$O\left(n \times (2^{n-k} + \sqrt{k} \times 2^{(n+k)/2}) + 2^{n-l}\right).$$

Arguing in a similar line for online phase as in Section 4.2.2, we can say that the time complexity for linking the message to the diamond structure is k times the message complexity, and the time complexity for linking the root of $\mathcal{D}_2$ to one of the intermediate hash value in the hash computation of M is l times its message complexity. Therefore, the time complexity comes out to be

$$O\left(n \times (\sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k}) + l \times 2^{n-l}\right)$$

Remark. We can analyze the time and the message complexity of the herding attack on concatenated hash functions of the form

$$H^{f_1}(M) \| H^{f_2}(M),$$

where f_1 and f_2 can be the same or different compression functions, in a similar fashion as in Section 4.3. This is because the basic construction is the same as for the hash twice function. The only difference lies in the commitment stage, where the adversary commits to the $h_1 \| h_2$, where h_1 is the hash value as in Step 2 of the herding attack on hash twice, and h_2 is the hash value of the root of $\mathcal{D}_2$ constructed in the attack.

5 Conclusion

In this paper, we pointed out that the analysis of diamond structure proposed by Kelsey-Kohno is not complete and may not yield the desired structure. We also gave a rigorous analysis of the construction of the diamond structure using concepts from random graph theory. There are some consequences of this, as enumerated below:

1. Our analysis showed that the message complexity of the construction of 2^k-diamond structure is about $\sqrt{k}$ times more than what [9] claimed (Theorem 1).

2. We also showed that the computational complexity of the construction of a diamond structure is n times the message complexity (Theorem 2).

n	Example	$k = \lfloor n/3 \rfloor$			$k = \lfloor 2n/5 \rfloor$		
		Message Complexity		Time Complexity	Message Complexity		Time Complexity
		[2]	Actual		[2]	Actual	
128	MD5	2^{85}	2^{95}	2^{98}	2^{90}	2^{100}	2^{102}
160	SHA-1	2^{107}	2^{117}	2^{120}	2^{112}	2^{122}	2^{125}
192	Tiger	2^{128}	2^{139}	2^{142}	2^{134}	2^{145}	2^{148}
256	SHA-256	2^{171}	2^{182}	2^{185}	2^{179}	2^{191}	2^{194}
512	Whirlpool	2^{341}	2^{354}	2^{358}	2^{358}	2^{371}	2^{375}

Table 3: Comparison of message complexity of second preimage attack (assuming $l \approx k$) on hash twice construction (all calculations are done without considering the constants).

3. For the Merkle-Damgård construction, the ratio of time complexity and the message complexity of the attacks is k if we choose the value of k to be strictly less than $n/3$ and it is n if we choose k to be larger than $n/3$.

4. For the hash twice construction and concatenated hash functions, the ratio of computational and message complexity is linear in k when k is strictly less than $n/3$, and it is constant for larger values of k.

We summarize the values of k to minimize message complexity in Table 4 and to minimize computational complexity in Table 5 for different attacks. We summarize our results on the message complexity in Table 6 and time complexity in Table 7.

n	Example	Herding Attack	Second Preimage on Dithered Hash	Herding and Second Preimage on Hash Twice
128	MD5	41	45	41
160	SHA-1	52	55	52
192	Tiger	62	66	62
256	SHA-256	83	88	83
512	Whirlphool	168	173	168

Table 4: Values of k to get minimum message complexity (assuming $l \approx k$)

n	Example	Herding Attack	Second Preimage on Dithered Hash	Herding and Second Preimage on Hash Twice
128	MD5	40	43	45
160	SHA-1	48	54	55
192	Tiger	61	65	66
256	SHA-256	82	87	87
512	Whirlphool	166	172	173

Table 5: Values of k to get minimum time complexity (assuming $l \approx k$)

	Message Complexity
Herding attack on Merkle-Damgård	$O(\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k})$
Second Preimage on Dithered Hash	$O(\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k} + k \times 2^{n-l})$
Herding attack on Hash Twice	$O(n \times (\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k}))$
Second Preimage on Hash Twice	$O(n \times (\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k}) + 2^{n-l})$

Table 6: Message complexity for various attacks

References

[1] J. P. Allouche and J. Shallit. *Automatic Sequences: Theory, Applications, Generalizations.* Cambridge University Press, 2003.

[2] E. Andreeva, C. Bouillaguet, O. Dunkelman, and J. Kelsey. Herding, second preimage and Trojan message attacks beyond Merkle-Damgård. In *Selected Areas in Cryptography*, pages 393–414, 2009.

	Time Complexity
Herding attack on Merkle-Damgård	$O(n \times \sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k})$
Second Preimage on Dithered Hash	$O(n \times \sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k} + l \times k \times 2^{n-l})$
Herding attack on Hash Twice	$O(n \times (k \times 2^{n-k} + \sqrt{k} \times 2^{(n+k)/2}))$
Second Preimage on Hash Twice	$O(n \times (k \times 2^{n-k} + \sqrt{k} \times 2^{(n+k)/2}) + l \times 2^{n-l})$

Table 7: Time complexity for various attacks

[3] E. Andreeva, C. Bouillaguet, P. A. Fouque, J. J. Hoch, J. Kelsey, A. Shamir, and S. Zimmer. Second preimage attacks on dithered hash functions. In *EUROCRYPT*, pages 270–288, 2008.

[4] J. A. Bondy and U. S. R. Murty. *Graph Theory.* Springer, 2008.

[5] P. Erdös and A. Renyi. On the evolution of random graphs. In *Proceedings of the Hungarian Academy of Sciences*, volume 5, pages 17–61, 1960.

[6] P. Erdös and A. Rényi. On the existence of a factor of degree one of a connected random graph. *Acta Mathematica Academiae Scientiarum Hungaricae Tomus*, 17(3–4):359–368, 1966.

[7] A. Joux. Multicollisions in iterated hash functions. Application to cascaded constructions. In *CRYPTO*, pages 306–316, 2004.

[8] J. Katz and Y. Lindell. *Introduction to Modern Cryptography.* Chapman and Hall, CRC Press, 2007.

[9] J. Kelsey and T. Kohno. Herding hash functions and the Nostradamus attack. In *EUROCRYPT*, pages 183–200, 2006.

[10] J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *EUROCRYPT*, pages 474–490, 2005.

[11] V. Keränen. Abelian squares are avoidable on 4 letters. In *ICALP*, pages 41–52, 1992.

[12] V. Keränen. On abeliean square-free DT0L-languages over 4 letters. In *Proceedings of Conference on Combinatorics on Words*, pages 41–52, 2003.

[13] Silvio Micali and Vijay V. Vazirani. An O($m\sqrt{n}$) algorithm for finding maximum matching in general graphs. In *FOCS*, pages 17–27, 1980.

[14] R. Motwani. Average-case analysis of algorithms for matchings and related problems. *J. ACM*, 41(6):1329–1356, 1994.

[15] R. Rivest. Abelian square-free dithering for iterated hash functions. 2005.

[16] V. Shoup. A composition theorem for universal one-way hash functions. In *EUROCRYPT*, pages 445–452, 2000.

[17] W. Stallings. *Cryptography and Network Security.* Prentice Hall, 2006.

[18] D. Stinson. *Cryptography: Theory and Practice.* Chapman & Hall/CRC Press Inc., 2006.