Making Large Hash Functions From Small Compression Functions

William R. Speirs and Ian Molloy

Center for Education and Research in Information Assurance and Security (CERIAS)
Department of Computer Sciences, Purdue University

Abstract. We explore the idea of creating a hash function that produces an s-bit digest from a compression function with an n-bit output, where $s > n$. This is accomplished by truncating a hash function with a digest size of ℓn-bits. Our work answers the question of how large ℓ can be while creating a digest of sn-bits securely. We prove that our construction is secure with respect to preimage resistance and collision resistance for $s \leq 2^{n/2}n$.

Keywords: Hash function, Merkle-Damgård construction, Double-Pipe construction.

1 Introduction

Current hash functions create a digest of a fixed size and yet are used with block ciphers and digital signature schemes that allow for different size keys. In some situations, such as AES and the SHA-2 family [5, 6], hash functions of specific sizes were created to be used in conjunction with a cipher that allows for different key sizes. In the case of the SHA-2 family, two different hash functions were constructed, SHA-256 and SHA-512. These hash functions are truncated[1] to provide two additional hash functions, SHA-224 and SHA-384, to match the key sizes of AES.

While these four functions work well with AES, they are not flexible enough to work with the protocols that allow for various key sizes. For example, they are not flexible enough to match the expected security of most digital signature schemes. Truncation can always be performed, but the problem of creating a larger digest is not as easy to solve. It is clear that larger digests are needed when the hash function is the weak link in the protocol. For example, if the hash-then-sign paradigm is used with RSA as the signing algorithm, then a 512-bit digest might not be sufficient for a 4096-bit RSA key. Currently there is no method for securely extending the size of a digest, assuming the hash function is constructed using an underlying compression function. This paper addresses this problem by providing a construction that securely extends the size of a digest.

[1] The initial values are also different.

1.1 Background

In 1989 Ralph Merkle and Ivan Damgård presented two independently written papers [4, 1] on a method for creating a hash function from a fixed domain function. This construction has been since appropriately named the Merkle-Damgård construction and is used by most hash functions. The construction works by extending the domain of the compression function from some binary strings of some fixed bit length to binary strings of arbitrary length. The construction works by dividing the input into blocks and iteratively applying the compression function to each input block. If f is a compression function, IV is some fixed known initial value, and $M = m_1 \parallel m_2 \parallel \cdots \parallel m_l$ is a message, then the Merkle-Damgård construction creates a digest $H(M)$ as follows:

$$H(M) = f(\cdots f(f(IV \parallel m_1) \parallel m_2) \cdots \parallel m_l).$$

To accommodate messages of any size, the message is padded to a multiple of the input size of the compression function. To strengthen the construction the length of the message, as a 64-bit integer, is always the final 64 bits of the padded message. The output size of the compression function is the same as the output size of the hash function created using the Merkle-Damgård construction.

1.2 Related Work

The most obvious method for creating a larger digest from smaller compression functions is to simply create two different hash functions using the Merkle-Damgård construction and concatenate their output. In [2] Joux demonstrated that such an approach does not achieve the desired security. His attack works by finding 2^k messages that all collide to the same digest by testing only an expected $k \cdot 2^{n/2}$ message blocks. When $k \geq n/2$ it is expected that one of the pairs of messages that collides one of the hash functions also collides the other. Therefore, a collision for a $2n$-bit hash function, built by concatenating the output of two n-bit hash functions, can be found by searching an expected $n/2 \cdot 2^{n/2}$ message blocks instead of the expected 2^n messages for an ideal $2n$-bit hash function.

In [3] Lucks described a construction that creates an n-bit hash function from an n bit compression function such that Joux's attack does not work. The construction, called a double-pipe construction, works by creating two interleaved *lines*, or *pipes*, of the Merkle-Damgård construction and compressing the lines together to create an n-bit digest. Joux's attack is not successful against this construction because the output of one line is concatenated with the message block being fed into the compression function of the other line. This mixing of outputs simulates a compression function that creates a $2n$-bit output. Therefore, when Joux's attack is applied to the construction $k \cdot 2^n$ work is required to find a collision.

While Lucks's construction was designed to thwart Joux's multi-collision attack, it also has the ability to create a hash function with a $2n$-bit output using a compression function with an n-bit output. Instead of applying a compression function to the two lines of Lucks's construction, the two lines can be concatenated together to form a $2n$-bit output. Because it requires 2^n work to find a collision for a single message block, this construction creates a secure hash function.

1.3 Contributions of This Paper

This paper investigates the problem of creating a hash function with a larger output than the underlying compression function securely. An ideal n-bit hash function requires an expected $2^{n/2}$ messages to be tested before a collision is found, and an expected 2^n messages to be tested before a preimage is found. The question this paper answers is if an s-bit hash function can be constructed using an n-bit compression function where $s > n$, such that an expected $2^{s/2}$ messages must be tested before finding a collision and 2^s messages for a preimage. We prove that our technique can securely create digests that are $2^{n/2}$ times larger than the output of an n-bit compression function.

2 The ℓ-Pipe Construction

The s-bit digest is the truncation of an ℓn-bit digest for $\ell = \lceil \frac{s}{n} \rceil$. We denote the s-bit digest as $H_s(\cdot) : \Sigma^* \rightarrow \Sigma^s$ since the number of lines, ℓ, is implicit. The rest of this paper will examine a hash function with an ℓn-bit digest and only comment on the s-bit truncation when necessary.

2.1 Notation

The notation used in this paper closely matches that which is most commonly seen. Let Σ denote the binary alphabet and Σ^n be the set of all binary strings of length n. Let Σ^* be the set of all binary strings. Let g be a compression function of the form $\Sigma^b \times \Sigma^n \rightarrow \Sigma^n$ where $b > n$. The function f is of the form $\Sigma^{b+n} \rightarrow \Sigma^n$ and is defined as $f(x \parallel y) = g(x, y)$. The Big-Oh notation is slightly abused to denote the expected number of something. For example, $\mathcal{O}(2^{n/2})$ means an expected $2^{n/2}$. Let the term *line* denote the Merkle-Damgård construction or a slight variation of it. Intermediate values are denoted by h and indexed via subscript by both iteration number and line number. For example, $h_{3,5}$ denotes the third iteration of line number five. Initial values are indexed by the line number with which they are used. The notation $\{ {x \atop x'} \}$ denotes one of two colliding message blocks, or different intermediate values. The function π_* is used to represent either π_1 or π_2.

2.2 General Construction

Expanding upon the idea of Lucks, one can generalize his construction to ℓ lines which, when omitting the final compression, produces an ℓn-bit digest from an n-bit compression function.

Each application of the compression function processes $(b - n)$ message-bits per iteration, and takes as input two chaining values from different lines. We first abstractly define the construction and then provide a concrete example. In Section 3.1 we show that our concrete construction is secure.

Let $M = m_0 \parallel m_1 \parallel \cdots \parallel m_k$ be a message broken into k blocks each of $(b - n)$-bits in length. Each block of the message is concatenated with the intermediate (or initial) value of line $\pi_2(j)$ and fed into the compression function where j is the current line. The function π_1 selects the line that will act as the "standard" chaining variable in a normal Merkle-Damgård construction and used as the first input to g. When each of the ℓ intermediate values are different, the construction behaves as though each line is governed by a different, unique compression function g_j. Equations (1) and (2) symbolically represent the construction.

$$h_{i,j} = g(h_{i-1,\pi_1(j)}, h_{i-1,\pi_2(j)} \parallel m_i) \tag{1}$$

$$H(M) = (h_{k,0} \parallel h_{k,1} \parallel \cdots \parallel h_{k,\ell-1}). \tag{2}$$

Figure 1 is a diagram of the ℓ-pipe construction where $\pi_1(j) = j$ and $\pi_2(j) = (j + 1) \bmod \ell$, without the final concatenation of the intermediate values, $h_{k,i}$. We will use these functions for π_1 and π_2 in future examples. Further comments on the selection of these particular functions for π_1 and π_2 are given in Section 3.1.

2.3 Initial Value Creation

The ℓ-pipe construction requires ℓ initial values, one for each line. Similar to [3], the ℓ initial values are required to be unique to prevent reduced-line attacks. For some security applications it may be desirable to ensure an attacker cannot learn $H_{s'}(x)$ from $H_s(x)$ where $s' < s$ and $\left\lceil \frac{s'}{n} \right\rceil = \left\lceil \frac{s}{n} \right\rceil$. To prevent $H_{s'}(x)$ from being the truncation of $H_s(x)$, we use different initial values for each digest size s, similar to [6]. Due to the possibly large number of IVs required, it is desirable to define a single initial value IV and derive the line and digest size dependent initial values. The following equation provides a method for generating the required initial values.

$$h_{0,i} = g(IV, i \parallel s) \quad \text{for } i = 0, 1, \ldots, \ell - 1. \tag{3}$$

Assuming g is collision resistant, each of the initial values is unique.

2.4 Message Padding

Because hash functions can take messages of arbitrary length as input, they must be padded to an appropriate length. The padding scheme used is exactly

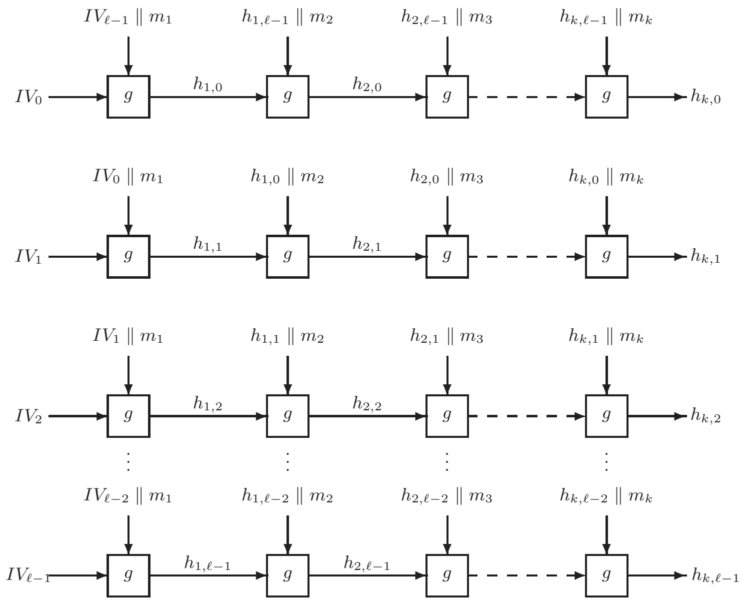

Fig. 1. The ℓ-pipe construction without the concatenation of the final values.

the same as with the strengthened Merkle-Damgård construction with a minor difference. Let l denote the length of the message. The suffix for all messages will be

$$10 \cdots 0 \parallel l \parallel i$$

where i is the line number. For all ℓ lines, every message block i is the same with the exception of the final message block, m_k. This padding method aids in proving the construction is secure with respect to preimage resistance, as shown in Theorem 2.

3 Security of the ℓ-Pipe Construction

The properties of preimage resistance and collision resistance are considered with respect to the security of the construction. In the rest of this section it is assumed that g is a random oracle of the form

$$\Sigma^n \times \Sigma^b \rightarrow \Sigma^n.$$

Definition 1. *An* r-way cross collision *occurs when a single message block causes a set* $\{a_1, a_2, \cdots, a_r\}$ *of* r *lines, where* $2 \leq r \leq \ell$, *to result in the same output:*

$$\begin{aligned} g\left(h_{i,\pi_1(a_1)}, m_i \parallel h_{i,\pi_2(a_1)}\right) &= g\left(h_{i,\pi_1(a_2)}, m_i \parallel h_{i,\pi_2(a_2)}\right) \\ &= \qquad \vdots \\ &= g\left(h_{i,\pi_1(a_r)}, m_i \parallel h_{i,\pi_2(a_r)}\right) \end{aligned}$$

Definition 2. *An* r-way strict collision *is caused when two different message blocks,* $\{m_i, m_i'\}$, *cause a set* $\{a_1, a_2, \cdots, a_r\}$ *of* r *lines, for* $r \leq \ell$, *to be same:*

$$\begin{aligned} g\left(h_{i,\pi_1(a_1)}, m_i \parallel h_{i,\pi_2(a_1)}\right) &= g\left(h_{i,\pi_1(a_1)}, m_i' \parallel h_{i,\pi_2(a_1)}\right) \\ \vdots \qquad &= \qquad \vdots \\ g\left(h_{i,\pi_1(a_r)}, m_i \parallel h_{i,\pi_2(a_r)}\right) &= g\left(h_{i,\pi_1(a_r)}, m_i' \parallel h_{i,\pi_2(a_r)}\right) \end{aligned}$$

Because the compression function g is assumed to be a random oracle with an output size of n bits, an expected 2^n messages must be tested to find a preimage and an expected $2^{n/2}$ messages must be tested to find a collision. These expected values are generalized and applied to cross collisions and strict collisions in the following Lemma.

Lemma 1. *If* g *is an* n*-bit random oracle, then the expected number of messages that must be tested before finding a message block that causes an* r*-way cross collision is* $2^{(r-1)n}$. *The expected number of messages that must be tested before finding two message blocks that cause an* r*-way strict collision is* $2^{rn/2}$.

Proof. The expected number of messages for a cross collision is derived from the properties of a random oracle. The expected number of messages for a strict collision is a direct application of the birthday attack.

One should note that cross collisions can be built up through multiple iterations. In one iteration a cross collision is found for a subset of the lines. In the next iteration other lines are cross collided with the subset of lines that have already been collided. Figure 2 shows how cross collisions can be built up through multiple iterations, where capital letters are used to exemplify lines that mimic each other. In the first iteration, a cross collision is found for the first three lines. In the second iteration a cross collision is found for the top, bottom and one of the two middle lines. The other middle line will also collide because the input to the compression function for the two middle lines mimic each other. Building up a cross collision in this manner is advantageous because it reduces the overall amount of work required significantly.

3.1 Properties of π_*

There are two ways in which an attacker can cause a cross-collision. The first is by choice of message blocks m_i as discussed in Lemma 1. The second is to cause the compression functions g_j and $g_{j'}$, defined in Section 2.2, to be identical. This is done by causing $h_{i-1,\pi_1(j)} = h_{i-1,\pi_1(j')}$ and $h_{i-1,\pi_2(j)} = h_{i-1,\pi_2(j')}$, for $j \neq j'$. It should be noted that an attacker gains one or both of these collisions "for free" if $\pi_1(j) = \pi_1(j')$ or $\pi_2(j) = \pi_2(j')$. To prevent this form of simplified attack, we restrict π_* to be permutations over $\mathbb{Z}_\ell$.

We following definitions for π_1 and π_2 can be used when ℓ is odd and provides the expected level of security for the construction.

$$\pi_1(j) = 2j \bmod \ell \tag{4}$$
$$\pi_2(j) = 2j + 1 \bmod \ell \tag{5}$$

Theorem 1. *Given the above permutations, an r-way cross collision at one iteration with $r < \ell$ will produce at most an $(r-1)$-way cross collision at the next iteration if no additional work is performed.*

Proof. For any set $\mathcal{A} = \{a_1, \cdots, a_r\}$ of r cross colliding lines to cause an r-way cross collision given the next message block, there must exist a set $\mathcal{B} = \{b_1, \cdots, b_r\}$ of r lines such that

$$\forall i,\ \pi_1(b_i) \in \{a_1, \cdots, a_r\} \ \wedge\ \pi_2(b_i) \in \{a_1, \cdots, a_r\}\,.$$

We first note the inverse of the permutation of π_1, $\pi_1^{-1}(j) = j * (\lfloor \frac{\ell}{2} \rfloor + 1)$ and use π_1^{-1} along with π_1 and π_2 to derive the sets $\mathcal{A}, \mathcal{B}$.

If we start with a_1 and define $b_1 = \pi_1^{-1}(a_1)$, $a_2 = \pi_2(b_1)$, $b_i = \pi_1^{-1}(a_i)$, $a_i = \pi_2(b_{i-1})$, etc; we can define the entire chain of dependencies. In order for $\mathcal{B}$ to be an r-way cross collisions the chain must be cyclic. If not, then the r lines in $\mathcal{B}$ use chaining variables not in $\mathcal{A}$, and at least one cross collision is lost.

Next observe that $\pi_2(\pi_1^{-1}(j)) = j + 1$ $a_{i+1} = a_i + 1$. By induction, $a_r = a_1 + r - 1$, which is the fist input into b_r. The second input, $\pi_2(b_r) = a_r + 1$. Finally, $a_r + 1 \equiv a_1$ only when $r \equiv \ell$, otherwise we must loose at least one cross collision.

Lemma 2. *The most optimal way to find a cross collision for $r \geq 3$ lines is by constructing the cross collision iteration-by-iteration, which requires testing an expected $(r-2)2^{2n}$ messages.*

Proof. First it is proved that an expected $(r-2)2^{2n}$ messages must be tested to find a cross collision for $r \geq 3$ lines. By application of Theorem 1, the smallest r can be so that progress is still made towards combining lines is $r = 3$. When $r = 3$ and the three lines are adjacent to each other, two of the lines will be the same in the next iteration. A cross collision is then found for one of the two lines that still collide and two additional lines. This results in four lines that cross collide, or three lines that still cross collide in the next iteration. Repeating this process and adding up the number of expected messages until all r lines cross collide results in the stated $(r-2)2^{2n}$ messages by application of Theorem 1 and Lemma 1.

Any attack that is more efficient than the one described must do less work per iteration or the same amount of work, but in fewer iterations. If less work is performed in each iteration, then only two lines can cross collide. This will result in all lines being different in the next iteration by Theorem 1. Therefore, no less than 2^{2n} work can be done in each iteration. Any attack that works in fewer steps must cross collide more than one additional line in each iteration. This cannot be done in the same amount of work as cross colliding three lines by Lemma 1. Therefore, the attack described is the most optimal way to cross collide $r \geq 3$ lines.

3.2 Restrictions on b

Applying the attack in Lemma 2 and the work required in Lemma 1, it follows that the ℓ-pipe construction is secure for $\ell \leq 4$. This attack is only successful for $\ell > 4$ when an attacker is able to test enough message blocks m_i such that an r-way cross-collision can be found. From Theorem 1 for an attacker to make progress, a 3-way cross-collision must be found.

By restricting $b < 3n$ we can reduce number of possible messages an attacker can try to less than the expected number required for the attack to succeed. This restriction requires that more than a single message block to be tested for an r-way cross-collision to be found when $r > 2$. Combined with Theorem 1, this prevents the attack described in Lemma 2 from succeeding. The restriction and its ability to prevent attacks is discussed further in Section 3.4.

3.3 Preimage Resistance

To find the expected number of messages that must be tested to find a preimage of any digest is a straightforward calculation. Lemma 3 gives the expected number of messages that must be tested when all of the outputs are different. As expected, this scenario is the most costly with respect to testing messages. To find a conservative estimate in all other cases, it is assumed that whenever all of the outputs are not different no work is needed.

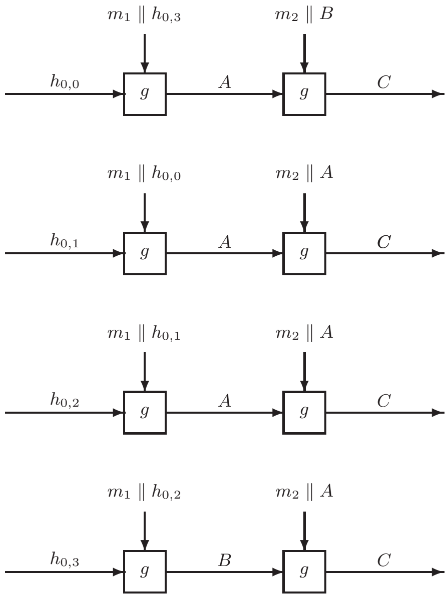

Fig. 2. Cross collisions built up through multiple iterations.

Lemma 3. *If all ℓ output lines of the ℓ-pipe construction are different and the compression function g is a random oracle, then an expected $2^{\ell n}$ messages must be tested before finding a preimage for the entire hash function.*

Proof. By definition of the ℓ-pipe construction, the inputs to the final application of the compression function are different for each line because the line number is part of the last input. An expected 2^n messages must be tested to find a preimage for a single line of the construction by the definition of a random oracle. For ℓ lines, the expected number of messages that must be tested is $2^{\ell n}$, because the inputs to the final compression function are all independent.

Theorem 2. *If the compression function g is a random oracle, then an expected $2^{\ell n}$ messages must be tested before a preimage is found for the ℓ-pipe construction where $\ell \leq 2^{n/2}$.*

Proof. When the ℓ output lines are all different, the expected number of messages that need to be tested is $2^{\ell n}$ by Lemma 3. It is possible that some attack exists where less work is needed when some of the lines have the same output. Certainly no more work is needed. Assume that no work is needed in this case. It is shown that this assumption does not affect the expected number of messages that must be tested to find a preimage.

When $\ell \leq 2^{2/n}$, the number of digests in which all ℓ output lines are different is,

$$(2^n)(2^n - 1) \cdots (2^n - \ell + 1) = 2^{\ell n} - (\ell(\ell-1)/2)2^{(\ell-1)n} + \mathcal{O}(2^{(\ell-2)n}).$$

Therefore, the expected number of messages that must be tested to find a preimage for any digest is

$$\begin{aligned}\left(\frac{2^{\ell n} - (\ell(\ell-1)/2)2^{(\ell-1)n} + \mathcal{O}(2^{(\ell-2)n})}{2^{\ell n}}\right) 2^{\ell n} + \\ \left(\frac{(\ell(\ell-1)/2)2^{(\ell-1)n} + \mathcal{O}(2^{(\ell-2)n})}{2^{\ell n}}\right) 0 = \\ 2^{\ell n} - (\ell(\ell-1)/2)2^{(\ell-1)n} + \mathcal{O}(2^{(\ell-2)n}) \approx 2^{\ell n}.\end{aligned}$$

3.4 Collision Resistance

All attacks against collision resistance that reduce the amount of work required to find a collision have one of two forms. The first method for attacking this construction is to find cross collisions so that multiple lines can be treated as a single line, and then find a strict collision for the subset of lines. The second method is the opposite, finding a strict collision for a subset of lines and then find cross collisions to combine them together. It should be noted that causing a strict collision in more than one message block does not help to efficiently attack this construction. Theorem 3 proves this fact.

Theorem 3. *The most efficient attack, excluding the birthday attack, that causes a collision in the ℓ-pipe construction produces two messages that differ in only one message block, that is there is just one r-way strict collision.*

Proof. Without loss of generality, assume that for two messages $M \neq M'$ and $H(M, s) = H(M', s)$, that the i^{th} block in each message is the first message block that is different in the two messages. There are two cases.

Case 1: The i^{th} message blocks in each message cause a strict collision for the entire construction. In this case searching for another pair of message blocks that causes a strict collision requires additional, unnecessary, work. Note that this is the birthday attack.

Case 2: The i^{th} message blocks in each message cause a strict collision in a subset of the lines. Let $h_i \neq h'_i$ denote the two outputs of some g in a line not part of the strict collision. If a cross collision is not found for the two outputs

h_i and h'_i, then those differences will propagate to additional lines with each iteration that a cross collision is not found. Letting the two different outputs propagate requires twice as much work to cause a full collision.

If additional strict collisions are used to collide all of the lines, those lines that were not collided and those that were must be considered. Those lines that were not collided count twice in the calculation of the expected number of messages because the inputs for either previous message block must be considered. Let r be the number of messages that do not collide by the strict collision. Each line that does not collide results in two different inputs to the compression function in the next iteration. These two different inputs require additional work to cause a collision. In this scenario the strict collision causes the first $\ell - r$ lines to collide. This results in $(2r+2) + (\ell - (r+1))$ inputs that must be considered for the next strict collision. The expected number of messages that must be tested to find the two strict collisions is:

$$2^{(\ell-r)n/2} + 2^{((2r+2)+(\ell-(r+1)))n/2}.$$

For two strict collisions to be more efficient than finding cross collisions first, $((2r+2) + (\ell - (r+1)))n$ has to be less than $\log_2((\ell - r) - 2) + 2n$. This can never happen with integer values for ℓ, r and n.

Therefore, any attack that finds more than one strict collision requires more work than finding a single strict collision.

To find an attack more efficient than the birthday attack, by application of Theorem 3 only a single strict collision should be found for t of the ℓ lines, where $t < \ell$. To find a strict collision for t lines, $2^{tn/2}$ messages must be tested, by application of Lemma 1. One should note that each line that does not collide by the strict collision results in $2(\ell - t)$ "lines" that must be collided with a cross collision. This is because there will be two different outputs (one for each message) for each line not collided during the strict collision which are inputs to the next iteration. These different inputs require twice as much work to cross collide the line. Therefore, it is more beneficial to cross collide lines before a strict collision than after.

Lemma 4. *Any efficient attack other than the birthday attack that finds a collision for the entire construction uses cross collisions before the strict collision.*

Proof. By application of Theorem 3 any optimal attack other than the birthday attack against the construction uses one strict collision. Let t be the number of lines that collide after the strict collision. For the $\ell - t$ lines that do not collide after the strict collision, the cross collision(s) must consider $2(\ell - t)$ "lines" by definition of the construction. If the $\ell - t$ lines were collided via a cross collision before the strict collision, less work would be required for the overall attack, by application of Lemma 1. To collide $\ell - t$ lines before the strict collision, $((\ell-t)-2)2^{2n}$ messages must be tested. After the strict collision, $(2(\ell-t)-2)2^{2n}$ messages must be tested. Therefore, finding cross collisions before the strict collision is always more efficient.

Because of Lemma 4, the only efficient attack against this construction is performed by finding a cross collision for a subset of the lines, and then a strict collision for the remaining lines. However the attack in Lemma 2 cannot be used to construction the cross collisions because of the restriction on the block size. It is assumed that $b < 3n$ which limits the number of possible message blocks that can be tested by an attacker to find a cross collision. Instead multiple message blocks must be used to construct the r-way cross collision. However, the expected number of messages that must be tested is dictated by Lemma 1 instead of Lemma 2. The following theorem states the overall work required to find a collision.

Theorem 4. *If r is the number of lines collided by the cross collision(s), then an expected $2^{(r-1)n} + 2^{(\ell-r+1)n/2}$ messages must be tested before a collision is found for the entire construction.*

Proof. First, r cross collisions are found requiring $2^{(r-1)n}$ messages to be tested from a direct application of Lemma 1. Then a strict collision for the remaining lines is found by testing an expected $2^{(\ell-r+1)n/2}$ messages, which is derived from Lemma 1 and the fact that even if all the lines are the same at the iteration where the r-way cross collision occurs, a single strict collision must still be found. It is that fact that accounts for the additional $n/2$ term in the exponent. Therefore, these two pieces added together results in the stated expected number of messages that must be tested to find a collision.

The expected number of messages that must be tested is minimized when $r \approx \ell/3$. Unfortunately, only three lines can be found in a single strict collision because of the restriction that $b < 3n$. Therefore, the total amount of work is $2^{(\ell-2)n} + 2^{3n/2}$.

Theorem 5. *If the compression function g is a random oracle, then an expected $2^{\ell n/2}$ messages must be tested before a collision is found for the ℓ-pipe construction where $\ell \leq 2^{n/2}$.*

Proof. To determine when this attack is successful, each piece can be examined to ensure that it is less than $2^{\ell n/2}$, the amount of work for the birthday attack. For $2^{(\ell-2)n} < 2^{\ell n/2}$, it is required that $\ell < 4$. For $2^{3n/2} < 2^{\ell n/2}$ it is required that $\ell > 3$.

These two restrictions upon ℓ are contradictory. Therefore, no attack against the construction is more efficient than the birthday attack when $b < 3n$.

4 Conclusion and Future Work

In this paper we present a construction that creates a hash function that produces an s-bit digest from an n-bit compression function, where $n < s < 2^{n/2}$. We provide a abstract description of the construction, and two concrete instantiations of the construction. Both of the concrete instantiations are secure, with respect to preimage resistance and collision resistance.

References

1. Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference*, Lecture Notes in Computer Science, pages 416–427, Santa Barbara, California, USA, August 1989. Springer.
2. Antoine Joux. Multicollisions in iterated hash functions. Application to cascaded constructions. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316, Santa Barbara, California, USA, August 2004. Springer.
3. Stefan Lucks. A failure-friendly design principle for hash functions. In Bimal K. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494, Chennai, India, December 2005. Springer.
4. Ralph C. Merkle. One way hash functions and DES. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference*, Lecture Notes in Computer Science, pages 428–446, Santa Barbara, California, USA, August 1989. Springer.
5. NIST. FIPS PUB 197: Advanced encryption standard (AES). Technical report, National Institute for Standards and Technology, Gaithersburg, MD, USA, November 2001.
6. NIST. FIPS PUB 180-2: Secure hash standard. Technical report, National Institute for Standards and Technology, Gaithersburg, MD, USA, May 2002.
7. William Speirs. *Dynamic Cryptographic Hash Functions*. PhD thesis, Purdue University, May 2007.