Memoryless Unbalanced Meet-in-the-Middle Attacks: Impossible Results and Applications

Yu Sasaki

NTT Secure Platform Laboratories
3-9-11 Midori-cho, Musashino-shi, Tokyo 180-8585 Japan
sasaki.yu@lab.ntt.co.jp

Abstract. A meet-in-the-middle (MitM) attack is a popular tool for cryptanalysis. It independently computes two functions $\mathcal{F}$ and $\mathcal{G}$, and finds a match of their outputs. When the cost of computing $\mathcal{F}$ and $\mathcal{G}$ are different, the problem is called unbalanced MitM attack. It is known that, for the balanced case, the MitM attack can be performed only with a negligible memory size without significantly increasing the computational cost by using the Floyd's cycle-finding algorithm. It is also widely believed that the same technique can be applied to the unbalanced case, while no one has shown the evidence of its possibility yet. This paper contains two contributions. Firstly, we show an impossibility of the memoryless unbalanced MitM attack without significantly increasing the computational cost. The conversion to the memoryless attack with the Floyd's cycle-finding algorithm always requires additional computational cost. Secondly, we find applications of the memoryless unbalanced MitM attack to show that it is still meaningful even with some additional computational cost. It can be used to generate multi-collisions of hash functions by using a dedicated collision attack algorithm. Our method finds 3-collisions of SHA-1 with 2^{142} computations and negligible memory size, while the known best attack requires $2^{106.6}$ computations and $2^{53.3}$ memory size. The memoryless unbalanced MitM attack can also be applied to the limited-birthday distinguisher for hash functions.

Keywords: unbalanced meet-in-the-middle, memoryless attack, Floyd's cycle-finding algorithm, hash function, SHA-1, 3-collision, limited-birthday distinguisher.

1 Introduction

A meet-in-the-middle (MitM) attack is a tool for cryptanalysis on symmetric-key primitives. It was introduced by Diffie and Hellman [1]. Then, Chaum and Evertse applied it to the key recovery attack on reduced-round DES [2]. Since then, it has been applied to many block-ciphers, hash functions and MACs for various purposes. Showing all references is hard. Several examples are [3–13]. The MitM attack separates the target function to be analyzed into two independent subfunctions $\mathcal{F}$ and $\mathcal{G}$ so that the original function is represented by $\mathcal{G} \circ \mathcal{F}$. The goal of the attack is

I. Boureanu, P. Owesarski, and S. Vaudenay (Eds.): ACNS 2014, LNCS 8479, pp. 253–270, 2014.

finding a pair (x, y) such that $\mathcal{F}(x) = \mathcal{G}(y)$. Because $\mathcal{F}$ and $\mathcal{G}$ are independent, the attack can be efficiently performed.

Let n be the size of the two functions output, let N_F and N_G be two values satisfying $N_F \times N_G = 2^n$ and let C_F and C_G be the computational cost to compute $\mathcal{F}$ and $\mathcal{G}$, respectively. The attack is processed as follows.

1. $\mathcal{F}$ is computed for N_F distinct input values $x_1, x_2 \ldots, x_{N_F}$, and $(x_i, \mathcal{F}(x_i))$ for $i = 1, 2, \ldots, N_F$ are stored in a list L_F.
2. $\mathcal{G}$ is computed for N_G distinct input values $y_1, y_2 \ldots, y_{N_G}$, and $(y_j, \mathcal{G}(y_j))$ for $j = 1, 2, \ldots, N_G$ are stored in a list L_G.
3. Find a match between $\mathcal{F}(x_i)$ and $\mathcal{G}(y_j)$ stored in L_F and L_G.

If a match is found, the adversary can obtain some important information depending on the attack scenario, *i.e.*, secret-key candidates in key recovery attacks for block-ciphers or preimage candidates in preimage attacks for hash functions. With the simple method, the above procedure requires $N_F \times C_F + N_G \times C_G$ computations. Therefore, N_F and N_G are chosen so that $N_F \times C_F$ and $N_G \times C_G$ are balanced. The memory to store N_F pairs of $(x_i, \mathcal{F}(x_i))$ and N_G pairs of $(y_j, \mathcal{G}(y_j))$ is also required. Here, one of L_F and L_G can be omitted by checking the match online as soon as each pair is obtained. The attack is called a (balanced) MitM attack when the cost of computing $\mathcal{F}$ and $\mathcal{G}$ are the same, *i.e.* $C_F = C_G$, and called an unbalanced MitM attack when $C_F \neq C_G$. For the balanced case, the computational cost is $2^{n/2} \times C_F (= 2^{n/2} \times C_G)$, and the memory size is also $2^{n/2}$. For the unbalanced case, the computational cost is $N_F \times C_F (= N_G \times C_G)$, and the memory size is $\min\{N_F, N_G\}$. Note that the terminology of "MitM" is often used even if $\mathcal{F}$ and $\mathcal{G}$ are not subfunctions of the original attack target, but simply two independent functions. In this paper, we also use the terminology of "MitM" to describe a match of two independent function outputs.

It is well-known that the balanced case can be performed only with a negligible memory size by using the Floyd's cycle-finding algorithm [14] with keeping almost the same computational complexity as the sufficient memory case. The idea is computing $\mathcal{F}$ or $\mathcal{G}$ $2^{n/2}$ times in a sequential form to make a long chain so that the output value of the previous evaluation of $\mathcal{F}$ or $\mathcal{G}$ is used as an input value to the next evaluation of $\mathcal{F}$ or $\mathcal{G}$. In the below, we firstly explain the MitM attack with the Floyd's cycle-finding algorithm with $2^{n/2}$ memory size. The idea is also illustrated in Fig. 1.

1. Set a start value of the chain, v_0, to an arbitrary value.
2. For $i = 1, 2, 3, \cdots$ do as follows.

$$\begin{cases} v_i \leftarrow \mathcal{F}(v_{i-1}) & \text{if a selecting bit (\textit{e.g.} LSB) of } v_{i-1} \text{ is 0,} \\ v_i \leftarrow \mathcal{G}(v_{i-1}) & \text{if a selecting bit (\textit{e.g.} LSB) of } v_{i-1} \text{ is 1.} \end{cases}$$

   Store v_i in a list L.
3. If a match between v_i and previously stored $v_{i'}$ in L is found, check if v_i and $v_{i'}$ are computed with different choices of $\mathcal{F}$ and $\mathcal{G}$.

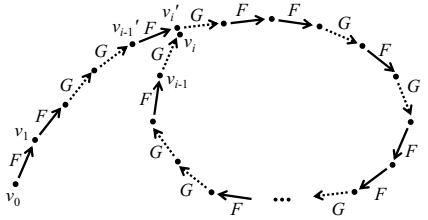

Fig. 1. MitM Attack with the Floyd's Cycle-Finding Algorithm ($\#\mathcal{F} : \#\mathcal{G} = 1 : 1$). v_{i-1} and v'_{i-1} are the solution

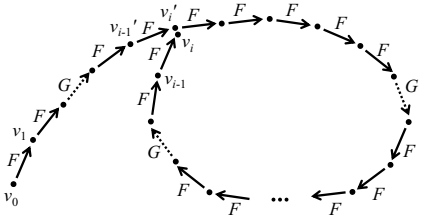

Fig. 2. Idea for the Unbalanced MitM Attack ($\#\mathcal{F} > \#\mathcal{G}$). The solution is not obtained from this cycle.

4. If so, output v_{i-1} and $v_{i'-1}$ as a solution of the MitM attack. Otherwise, go back to step 1, and repeat the attack until a solution is found.

Because the attack needs to find a match of different functions' outputs, the function must be switched between $\mathcal{F}$ and $\mathcal{G}$ with probability 1/2 when the cycle is constructed. At Step 2, a selecting bit is introduced for this purpose. Not only the value of LSB of v_{i-1}, but also any choice of an event that occurs with probability 1/2 can be used, as long as the selecting rule is fixed. At Step 4, the probability that v_i and $v_{i'}$ are computed with different choices of $\mathcal{F}$ and $\mathcal{G}$ is 1/2. Therefore, the cycle construction is iterated 2 times on average. Finally, a match between v_i and $v_{i'}$ is found with $O(2^{n/2})$ evaluations, and thus the size of L is also $O(2^{n/2})$. To perform the above procedure with a negligible memory size, instead of all v_i, only a very small fraction of v_i are stored in L. Due to the sequential computational structure, a match between v_i and $v_{i'}$ indicates that the chain becomes a cycle of size $O(2^{n/2})$. Therefore, even if a match between v_i and all previous $v_{i'}$ cannot be checked immediately, sooner or later, the computation reaches one of the values stored in L with at most $O(2^{n/2})$ additional computational cost. If a match is found, by recomputing the cycle from previously stored point, the match between v_{i-1} and $v_{i'-1}$ can be detected. The attack smartly reduces the memory size of the balanced MitM attack only with a small (constant time) increasing computational cost.

It raises a natural question: *can the unbalanced MitM attack also be performed with a negligible memory size only with a small increased computational cost?* Although no one has discussed its possibility in details yet, it is often believed to be possible.

Without losing the generality, throughout the paper, we suppose that C_G is much bigger than C_F, and thus N_G is much smaller than N_F. Obviously, by using the Floyd's cycle-finding algorithm in Fig. 1, *i.e.* by computing $\mathcal{F}$ and $\mathcal{G}$ for the same quantity, the attack can be memoryless. However, this requires $2^{n/2} \times C_G$ computational cost, which is much more than the standard attack (with a sufficient memory size) of $N_G \times C_G$. So far, no other attempt is known for the memoryless unbalanced MitM attack. Consequently, the possibility of the

memoryless attack with negligible additional computational cost from $N_G \times C_G$ is unknown.

Our Contributions. In this paper, we investigate the memoryless unbalanced MitM attack. The fact that the balanced case computes $\mathcal{F}$ and $\mathcal{G}$ in the same ratio seems to come from the fact that C_F and C_G are identical. Therefore, our approach is changing the ratio of computing $\mathcal{F}$ and $\mathcal{G}$ when the cycle is constructed. The idea is illustrated in Fig. 2. Because $C_F < C_G$, we compute $\mathcal{F}$ much more than $\mathcal{G}$. This raises a new difficulty: when a match between v_i and $v_{i'}$ are obtained, the probability that v_i and $v_{i'}$ are computed with different functions becomes lower. Both are likely to be generated by $\mathcal{F}$. Hence, the number of iterations of the cycle construction will increase.

In this paper, we begin with summarizing the computational cost of the unbalanced MitM attack with a sufficient memory size when the cost of two functions are given in variables C_F and C_G. Most of previous work analyzed the case that $C_F = 1$. We extend it to a two-variable case.

Then, we evaluate the computational cost of the memoryless unbalanced MitM attack described in Fig. 2, and show that improving $2^{n/2} \times C_G$, which is the simple application of the Floyd's cycle-finding algorithm, is impossible for any values of C_F and C_G, and any choice of the ratio of computing $\mathcal{F}$ and $\mathcal{G}$.

Finally, we show an application of the memoryless unbalanced MitM attack. That is to say, the simple application of the Floyd's cycle-finding algorithm is still meaningful. As the first application, we show that it can be used to generate 3-collisions of hash functions by using a dedicated collision attack algorithm. The current best generic 3-collision attack against an n-bit hash function is the one proposed by Joux and Lucks [17], which requires a computational cost of $O(2^{2n/3})$ and a memory size of $O(2^{n/3})$. Although the computational cost of this attack matches the information theoretic lower-bound, preparing a memory of $O(2^{n/3})$ size is hard for a large n. For example, SHA-1 [18] produces 160-bit hash digests, and the generic attack by [17] requires $2^{106.6}$ computational cost and $2^{53.3}$ memory size. We point out that if a collision attack exists for a hash function, it can be converted to a memoryless[1] 3-collision attack with some additional computational cost. For SHA-1, a (memoryless) collision attack was proposed by Wang *et al.* [19] which claimed 2^{69} computational cost. Although many papers claim the improved computational cost [21–26], the current best complexity is unclear. Because our purpose is showing a generic conversion framework, the current exact computational cost is not a main issue. Suppose that collisions of SHA-1 can be generated with 2^{61} computational cost [25]. Then, our conversion method can find 3-collisions of SHA-1 with 2^{142} computational cost and negligible memory size. We do not claim that this is better than the generic attack by [17]. However, we believe that the possibility of memoryless 3-collision attack is worth noting, and if the attack is measured by a product of a computational cost and a memory size, our result ($2^{142} \times 1 = 2^{142}$) becomes better than [17]

[1] Here, we suppose that the collision attack itself is memoryless. In fact, many collision attacks based on the ones by Wang *et al.* [19, 20] require few memory.

($2^{106.6} \times 2^{53.3} = 2^{160}$). We also apply the memoryless 3-collision attack to hash function HAVAL [27] by exploiting the existing collision attack presented by [28]. The comparison of the complexity is given in Table 1.

Table 1. Comparison of 3-collision Attacks

Target	Attack Method	Computational Cost	Memory Size	Reference
SHA-1	Generic Attack	$2^{106.6}$	$2^{106.6}$	[29]
	Improved Generic Attack	$2^{106.6}$	$2^{53.3}$	[17]
	Memoryless Attack	2^{142}	negl.	Sect. 4.3
4-pass HAVAL	Generic Attack	$2^{170.6}$	$2^{170.6}$	[29]
	Improved Generic Attack	$2^{170.6}$	$2^{85.3}$	[17]
	Memoryless Attack	2^{165}	negl.	Sect. 4.3
5-pass HAVAL	Generic Attack	$2^{170.6}$	$2^{170.6}$	[29]
	Improved Generic Attack	$2^{170.6}$	$2^{85.3}$	[17]
	Memoryless Attack	2^{252}	negl.	Sect. 4.3

As the second application, we show that the memoryless unbalanced MitM can be applied to the limited-birthday distinguisher on a hash function which has been recently proposed by Iwamoto *et al.* [30].

Paper Outline. The organization of this paper is as follows. In Sect. 2, we generalize the cost of the unbalanced MitM attack in which the computational cost of $\mathcal{F}$ and $\mathcal{G}$ are given in variables. In Sect. 3, we show the impossibility of improving the simple application of the Floyd's cycle-finding algorithm for the memoryless unbalanced MitM attack. In Sect. 4, we show the application of the memoryless unbalanced MitM attack. Finally, we conclude this paper in Sect. 5.

2 Generalizing the Computational Cost of Unbalanced MitM Attacks

In this section, we evaluate the cost of the unbalanced MitM attack when a sufficient memory size is given. The goal of the attack is finding a match between two functions of n-bit output $\mathcal{F}$ and $\mathcal{G}$, where one execution of $\mathcal{F}$ and $\mathcal{G}$ require C_F and C_G computational cost, respectively. To make the discussion easy, we discuss the cost in exponential forms. Hence, we suppose that $C_F = 2^\alpha$ and $C^G = 2^\beta$. Without losing the generality, we suppose that $\alpha < \beta$.

2.1 Previous Work for $C_F = 1$

There are several previous work, *e.g.* [3, 31], which performs the unbalanced MitM attack when the cost for the cheaper function is 1, *i.e.* $C_F = 1$ and $C_G = 2^\beta$. The attack procedure for this case is described as follows.

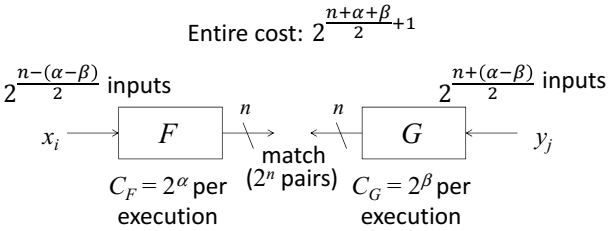

Fig. 3. Sketch of the Unbalanced MitM Attack with Sufficient Memory

1. Set $N_G \leftarrow 2^{(n-\beta)/2}$. For $j = 1, 2, \cdots, N_G$, choose a value of y_j, compute $\mathcal{G}(y_j)$, and store the result in a list L where the data is indexed by $\mathcal{G}(y_j)$.
2. Set $N_F \leftarrow 2^{(n+\beta)/2}$. For $i = 1, 2, \cdots, N_F$, choose a value of x_i, compute $\mathcal{F}(x_i)$, and search for a match with $\mathcal{G}(y_j)$ in the list L.

Because $N_F \times N_G = 2^n$, one match is expected on average. The memory size for Step 1 is N_G, which is $2^{(n-\beta)/2}$. N_F and N_G are chosen so that the computational cost for Step 1 and Step 2 are balanced. The computational cost for Step 1 is $2^{(n-\beta)/2} \times 2^\beta = 2^{(n+\beta)/2}$. The computational cost for Step 2 is $2^{(n+\beta)/2} \times 1 = 2^{(n+\beta)/2}$. In the end, the attack is performed with $2^{((n+\beta)/2)+1}$ computational cost and $2^{(n-\beta)/2}$ memory size.

2.2 Generalization for $C_F = 2^\alpha$

We generalize the attack in Sect. 2.1 so that the cost of computing $\mathcal{F}$ is given by 2^α. The only difference from Sect. 2.1 is the choice of N_F and N_G. The attack procedure is as follows, which is also depicted in Fig. 3.

1. Set $N_G \leftarrow 2^{(n+(\alpha-\beta))/2}$. For $j = 1, 2, \cdots, N_G$, choose a value of y_j, compute $\mathcal{G}(y_j)$, and store the result in a list L where the data is indexed by $\mathcal{G}(y_j)$.
2. Set $N_F \leftarrow 2^{(n-(\alpha-\beta))/2}$. For $i = 1, 2, \cdots, N_F$, choose a value of x_i, compute $\mathcal{F}(x_i)$, and search for a match with $\mathcal{G}(y_j)$ in the list L.

Because $N_F \times N_G = 2^n$, one match is expected on average. The memory size for Step 1 is N_G, which is $2^{(n+(\alpha-\beta))/2}$. The computational cost for Step 1 is $N_G \times 2^\beta = 2^{(n+\alpha+\beta)/2}$. The computational cost for Step 2 is $N_F \times 2^\alpha = 2^{(n+\alpha+\beta)/2}$. In summary, the unbalanced MitM attack can be performed with $2^{((n+\alpha+\beta)/2)+1}$ computational cost and $2^{(n+(\alpha-\beta))/2}$ memory size.

Note that by setting $C_F = 1$, *i.e.* $\alpha = 0$, the complexity becomes $2^{((n+\beta)/2)+1}$ computational cost and $2^{(n-\beta)/2}$ memory size, which matches in Sect. 2.1.

The generalization in this section is quite straight-forward. We showed the generalization as a tool for the future usage. Indeed, the discussion from the next section uses this result.

3 Impossibility of Efficient Memoryless Unbalanced MitM Attacks

In this section, we aim to convert the unbalanced MitM attack in Sect. 2 to the memoryless attack by using the Floyd's cycle-finding algorithm. Firstly, we explain the simple application of the Floyd's cycle-finding algorithm in Sect. 3.1. We then explain that it is impossible to improve the computational cost by changing the ratio of computing $\mathcal{F}$ and $\mathcal{G}$ in Sect. 3.2. We give some remarks in Sect. 3.3.

3.1 Simple Application of the Floyd's Cycle-Finding Algorithm

In this section, we simply apply the memoryless attack on the balanced case even though the cost of two functions $\mathcal{F}$ and $\mathcal{G}$ are unbalanced. We again suppose that $C_F = 2^\alpha$, $C^G = 2^\beta$ and $\alpha < \beta$. Because $\mathcal{F}$ and $\mathcal{G}$ are computed in the same ratio, we choose an event of probability $1/2$ as a selection rule of the choice of $\mathcal{F}$ and $\mathcal{G}$. Here, we simply use the LSB of previous chaining value.

To find a match, both $\mathcal{F}$ and $\mathcal{G}$ are computed about $2^{n/2}$ times, which requires the computational cost of $2^{\alpha+(n/2)}$ and $2^{\beta+(n/2)}$, respectively. If a match between v_i and v'_i is found, we check if they are generated from different choices of $\mathcal{F}$ and $\mathcal{G}$. Because $\mathcal{F}$ and $\mathcal{G}$ are switched with probability $1/2$, they are generated from different functions with probability $1/2$. Therefore, we need to iterate the attack 2 times on average, which requires the computational cost of $2^{\alpha+(n/2)+1} + 2^{\beta+(n/2)+1}$. Considering that $\alpha < \beta$, the entire computational cost is

$$2^{\beta+(n/2)+1}. \tag{1}$$

The evaluation is summarized in Table 2.

Table 2. Complexity for the simple application of the Floyd's cycle-finding algorithm

Functions	#computed times per match	Computational cost per match	#iterations of cycle constructions	Dominant computational cost
$\mathcal{F}$	$2^{n/2}$	$2^{\alpha+(n/2)}$	2	$2^{\beta+(n/2)+1}$
$\mathcal{G}$	$2^{n/2}$	$2^{\beta+(n/2)}$		

Note that the computational cost with a sufficient memory size is $2^{((n+\alpha+\beta)/2)+1}$ as shown in Sect. 2.2. Compared to it, the simple application of the Floyd's cycle-finding algorithm increases the computational cost by a factor of $2^{(\beta-\alpha)/2}$.

3.2 Unbalanced Selecting Bits: Changing the Ratio of $\mathcal{F}$ to $\mathcal{G}$

To make the computational cost of computing $\mathcal{F}$ and $\mathcal{G}$ balanced, we use an event with probability $2^{-(\beta-\alpha)}$ as a selection rule of the choice of $\mathcal{F}$ and $\mathcal{G}$. Therefore,

if the $(\beta - \alpha)$ LSBs of v_{i-1} are all 0, we compute $v_i \leftarrow \mathcal{G}(v_{i-1})$. Otherwise, we compute $v_i \leftarrow \mathcal{F}(v_{i-1})$. Then, the ratio of $\mathcal{F}$ to $\mathcal{G}$ becomes $2^{\beta-\alpha}$ to 1. To be more precise, the chain is computed as follows.

$$\begin{cases} v_i \leftarrow \mathcal{F}(v_{i-1}) & \text{if } (\beta-\alpha) \text{ LSBs of } v_{i-1} \text{ are all 0,} \\ v_i \leftarrow \mathcal{G}(v_{i-1}) & \text{otherwise.} \end{cases}$$

The number of computations of $\mathcal{F}$ and $\mathcal{G}$ become $2^{n/2+(\beta-\alpha)/2}$ and $2^{n/2-(\beta-\alpha)/2}$ respectively. The corresponding computational cost is obtained by multiplying 2^{α} and 2^{β} respectively, which result in $2^{(n+\alpha+\beta)/2}$ for both. Because $\mathcal{F}$ and $\mathcal{G}$ is computed in the ratio $2^{\beta-\alpha}$ to 1, the probability that a match is obtained from different choice of $\mathcal{F}$ and $\mathcal{G}$ is about $2^{-(\beta-\alpha)}$. Therefore, the cycle construction must be iterated $2^{\beta-\alpha}$ times on average. The entire computational cost becomes $2^{(n+3\beta-\alpha)/2+1}$. The evaluation is summarized in Table 3.

Table 3. Complexity using the unbalanced ratio of $\mathcal{F}$ to $\mathcal{G}$

Functions	#computed times per match	Computational cost per match	#iterations of cycle constructions	Dominant computational cost
$\mathcal{F}$	$2^{n/2+(\beta-\alpha)/2}$	$2^{(n+\alpha+\beta)/2}$	$2^{\beta-\alpha}$	$2^{(n+\alpha+\beta)/2+1} \times 2^{\beta-\alpha}$
$\mathcal{G}$	$2^{n/2-(\beta-\alpha)/2}$	$2^{(n+\alpha+\beta)/2}$		$= 2^{(n+3\beta-\alpha)/2+1}$

Let us compare the computational cost of this case with the one in Sect. 3.1. The condition that the computational cost of this attack can be smaller than the one in Sect. 3.1 is

$$2^{\beta+(n/2)+1} > 2^{(n+3\beta-\alpha)/2+1},$$

which is converted into

$$\alpha > \beta.$$

This clearly contradicts to the assumption of $\alpha < \beta$, thus regardless of the values of α and β, the attack in Sect. 3.1 is better.

Results do not change even if we consider the other ratio. Let us set the ratio of the number of $\mathcal{F}$ and $\mathcal{G}$ to 2^z to 1. The evaluation is similar. $\mathcal{F}$ is computed $2^{n/2+z/2}$ times in a cycle, and its computational cost is $2^{n/2+z/2+\alpha}$. $\mathcal{G}$ is computed $2^{n/2-z/2}$ times in a cycle, and its computational cost is $2^{n/2-z/2+\beta}$. The probability that a match is obtained from different choices of $\mathcal{F}$ and $\mathcal{G}$ is about 2^{-z}. Therefore, the cycle construction will be iterated 2^z times on average. Hence, the total computational cost is $2^{n/2+z/2+\alpha} + 2^{n/2+z/2+\beta}$. The results are summarized in Table 4.

Table 4. Complexity evaluation by setting the ratio of $\mathcal{F}$ to $\mathcal{G}$ as a variable

Func.	#computed times per match	Computational cost per match	#iterations of cycle constructions	Dominant computational cost
$\mathcal{F}$	$2^{n/2+z/2}$	$2^{n/2+z/2+\alpha}$	2^z	$2^{n/2+3z/2+\alpha} + 2^{n/2+z/2+\beta}$
$\mathcal{G}$	$2^{n/2-z/2}$	$2^{n/2-z/2+\beta}$		

Compared to $2^{\beta+(n/2)+1}$ for the simple application in Sect. 3.1, the second term of the dominant computational cost in Table 4, which is $2^{n/2+z/2+\beta}$, is always higher by a factor of $2^{z/2-1}$.

3.3 Summary and Remarks

We showed that as long as the Floyd's cycle-finding algorithm is used, the simple application that computes $\mathcal{F}$ and $\mathcal{G}$ in the same ratio is the best, and the attack increases the computational cost by a factor of $2^{(\beta-\alpha)/2}$ compared to the one with a sufficient memory size.

We like to note that without using the Floyd's cycle-finding algorithm, the trivial time-memory tradeoff exists for both balanced and unbalanced MitM attacks. In the MitM attack, N_G outputs are stored for $\mathcal{G}$, and N_F outputs are generated online for $\mathcal{F}$, where $N_G \times N_F = 2^n$. Usually, N_G and N_F are chosen so that the computational cost $N_G \times C_G$ and $N_F \times C_F$ are balanced. In order to reduce the memory size, instead of storing N_G outputs for $\mathcal{G}$, we only store $\mathcal{G}/w$ results. When N_F are computed later, we generate $w \times N_F$ results so that a match is still expected. This reduces a memory size by a factor of w and increases the computational cost by a factor of w. This is a tradeoff such that Time $\times$ Memory = constant. If the memory size needs to be reduced only by a small factor, using this trivial time-memory tradeoff may be more convenient than using the fully memoryless cycle-finding algorithm which always increases the computational cost by a factor of $2^{(\beta-\alpha)/2}$.

4 Applications of the Memoryless Unbalanced MitM Attacks

In the previous section, we showed a negative result, *i.e.* the computational cost of the memoryless unbalanced MitM attack cannot be faster than the simple application in Sect. 3.1. In this section, we show that the memoryless unbalanced MitM attack in Sect. 3.1 is still meaningful. Firstly in Sect. 4.1, we explain the generic condition that the memoryless unbalanced MitM attack can be applied. Secondly in Sect. 4.2, we explain that several claims of the previous memoryless MitM preimage attacks, *e.g.* [15], is incorrect. Thirdly in Sect. 4.3, as a concrete example, we show that it can be used to generate 3-collisions for hash functions by exploiting a dedicated collision attack algorithm. Finally in Sect. 4.4, as another example, we show the applications to the limited-birthday distinguisher recently proposed by Iwamoto *et al.* [30].

4.1 Conditions to Apply Memoryless Unbalanced MitM Attack

From eq. (1), the computational cost of the memoryless unbalanced MitM is given by $2^{\beta+(n/2)+1}$, where 2^{β} is the cost to execute the heavier function $\mathcal{G}$. In order to be faster than 2^n computational cost, we obtain the condition $2^{\beta+(n/2)+1} < 2^n$, which is converted to

$$C_G = 2^{\beta} < 2^{(n/2)-1}. \tag{2}$$

Therefore, to be converted to the memoryless attack, the computational cost of $\mathcal{G}$ must be smaller than the birthday attack complexity.

4.2 Incorrectness of Previous Memoryless MitM Preimage Attack

Recently, various preimage attacks based on the MitM attack have been proposed against narrow-pipe Merkle-Damgård hash functions [7, 11, 13, 15, 16]. In those hash functions, the hash digest is computed by iteratively computing the compression function $H_i \leftarrow \mathrm{CF}(H_{i-1}, M_{i-1})$, where M_i is the message value and H_0 is an initial value defined in the hash function specification.

Those attacks, for a given hash digest, aim to generate a preimage consisting of two message blocks. The attack is depicted in Fig. 4. It firstly generates pseudo-preimages, which are a pair of (H_1, M_1) such that $H_1 \neq \mathrm{IV}$ and the corresponding compression function output, H_2, is a given digest. In many cases, the cost of the pseudo-preimage generation is much higher than $2^{n/2}$. After that, generated pseudo-preimages are converted into a preimage by applying the unbalanced MitM attack for the first message block. In details, the pseudo-preimage attack is regarded as function $\mathcal{G}$ and the random message generation for the first message block is regarded as function $\mathcal{F}$ whose computational cost is 1 per execution.

Several (but not all) previous work claim that the unbalanced MitM part can be performed with negligible memory size by using the Floyd's cycle-finding algorithm, *e.g.* [15, Section 4.5].

However, the conversion from pseudo-preimages to preimages is an example that the unbalanced MitM attack part cannot be memoryless by using the Floyd's cycle-finding algorithm. If the conversion is applied, the computational cost of the memoryless preimage attack becomes more than 2^n, which is worse than the generic attack. This result immediately indicates the incorrectness of the claim in previous work about the memoryless preimage attack. We would like to stress that, with a sufficient amount of memory, the previous attack can work correctly.

4.3 Application to 3-Collisions

The first application of the memoryless unbalanced MitM attack is a 3-collision attack on hash functions. A 3-collision on a hash function $\mathcal{H}$ is a triplet of distinct input values (I_1, I_2, I_3) such that $\mathcal{H}(I_1) = \mathcal{H}(I_2) = \mathcal{H}(I_3)$.

In general, it is known that a t-collision can be generated with a computational cost of $O(2^{(t-1)n/t})$ and $O(2^{(t-1)n/t})$ memory size [29]. For $n = 3$, the

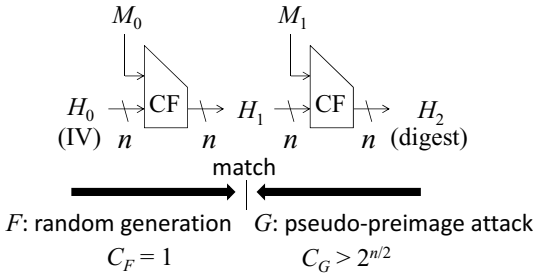

Fig. 4. Previous preimage attacks using unbalanced MitM attack

computational cost is $O(2^{2n/3})$ and the memory size is $O(2^{2n/3})$. At Asiacrypt 2009, Joux and Lucks showed a generic 3-collision attack on an n-bit narrow-pipe Merkle-Damgård hash function [17], which requires a computational cost of $O(2^{2n/3})$ and a memory size of $O(2^{n/3})$. One drawback of this attack is a memory size of $O(2^{n/3})$. For a relatively large n, preparing a memory of size $O(2^{n/3})$ is infeasible.

We point out that a memoryless 3-collision attack on $\mathcal{H}$ can be achieved from a (memoryless) collision attack on $\mathcal{H}$. In short, the strategy is as follows, which is also illustrated in Fig. 5. A collision attack produces a pair of input messages (I_1, I_2) such that $\mathcal{H}(I_1) = \mathcal{H}(I_2)$ with a computational cost of $C_G = 2^\beta$, where $\beta < n/2$. This operation is regarded as function $\mathcal{G}$, namely, $\mathcal{G}$ produces an n-bit value $\mathcal{H}(I_1) = \mathcal{H}(I_2)$ at a computational cost of $C_G = 2^\beta$. To find the third colliding input message I_3, we simply test randomly generated messages. Therefore, $\mathcal{F}$ produces an n-bit value $\mathcal{H}(I_3)$ at a computational cost of $C_F = 1$. A 3-collision is generated by observing a match of the output values between $\mathcal{F}$ and $\mathcal{G}$. This is exactly the unbalanced MitM attack. Hence the 3-collision attack

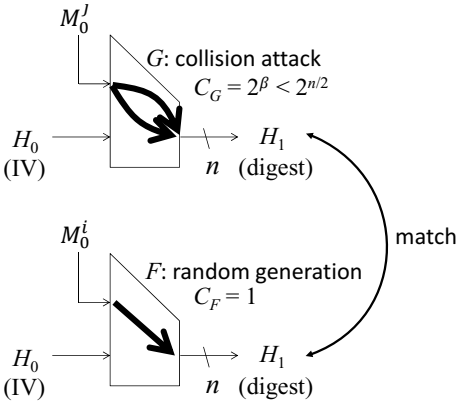

Fig. 5. Memoryless 3-collision Attack with Memoryless Unbalanced MitM Attack

can be memoryless by following the strategy in Sect. 3.1, which results in the computational cost of $2^{\beta+(n/2)+1}$ and a negligible memory size.

For example, SHA-1 [18] is a 160-bit narrow-pipe hash function. On one hand, the generic attack by [17] requires $2^{106.6}$ computational cost and $2^{53.3}$ memory size to find a 3-collision. On the other hand, [25] showed that collisions of SHA-1 can be generated with 2^{61} computational cost and negligible memory size. Then, the conversion in Sect. 3.1 can find 3-collisions of SHA-1 with $2^{142}(= 2^{61+(160+/2)+1})$ computational cost and negligible memory size. Other two examples are 4-pass HAVAL and 5-pass HAVAL [27], each is a 256-bit narrow-pipe hash function. A collision attack was proposed by Yu *et al.*, which finds collisions of 4-pass HAVAL with 2^{36} computational cost and negligible memory and collisions of 5-pass HAVAL with 2^{123} computational cost and negligible memory [28]. The generic 3-collision attack by [17] requires $2^{170.6}$ computational cost and $2^{85.3}$ memory size for both of 4-pass and 5-pass HAVAL. Preparing a memory of size $2^{85.3}$ seems almost infeasible. The conversion in Sect. 3.1 can find 3-collisions of 4-pass HAVAL with $2^{165}(= 2^{36+(256/2)+1})$ computational cost and negligible memory size and 3-collisions of 5-pass HAVAL with $2^{252}(= 2^{123+(256/2)+1})$ computational cost and negligible memory size.

Strictly speaking, we need to be more careful about the details of collision finding algorithm $\mathcal{G}$ to generate a Floyd's cycle. The collision attack on SHA-1 [19, 25], to achieve the claimed computational cost, involves various analytic techniques such as message modification technique and early aborting technique. To make the cycle, $\mathcal{G}$ must be solely dependent on the value of v_{i-1}. Besides, for the same v_{i-1} received in different timings, $\mathcal{G}$ must reproduce the same output value v_i. In short, this can be achieved by fixing the search rule *i.e.* in which order the freedom degrees are used and the conditions to apply the techniques. We show the detailed analysis specific to the SHA-1 collision search in Appendix.

4.4 Application to Limited-Birthday Distinguisher

The limited-birthday distinguisher was firstly mentioned by Gilbert and Peyrin at FSE 2010 [32] to distinguish a target function $\mathcal{H}$ from an ideal one. In this framework, the distinguisher is firstly given a pair of an input (truncated) difference Δ_{IN} and an output (truncated) difference Δ_{OUT}. The goal of the distinguisher is finding a value x satisfying both the input and output differences, *i.e.*, $\mathcal{H}(x) \oplus \mathcal{H}(x \oplus \Delta_{IN}) = \Delta_{OUT}$. If such a value can be found for $\mathcal{H}$ faster than for an ideal function, $\mathcal{H}$ is said to be non-ideal.

Then, Iwamoto *et al.* showed that, for a narrow-pipe hash function, the limited-birthday distinguisher for the entire construction can be constructed from a semi-free-start collision attack on the compression function [30]. Here, a semi-free-start collision attack is a kind of collision attack on the compression function CF, which finds a triplet of a previous chaining variable H_{i-1}, a message x, and a message difference δ_{IN} such that $\mathrm{CF}(H_{i-1}, x) = \mathrm{CF}(H_{i-1}, x \oplus \delta_{IN})$. Iwamoto *et al.* point out that in many of previous semi-free-start collision attacks, δ_{IN} can be fixed before x is searched. Then, their framework of the

conversion to the limited-birthday distinguisher on $\mathcal{H}$ is as follows, which is also illustrated in Fig. 6.

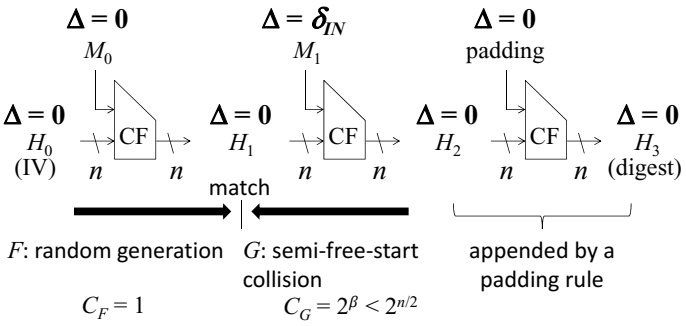

Fig. 6. Framework of the Limited-birthday Distinguisher on $\mathcal{H}$ from a Semi-free-start Collision Attack on CF. An input difference is fixed to $(0\|\delta_{IN})$, and an output difference is fixed to 0.

1. Find a semi-free-start-collision attack on CF, which can work for a pre-specified message difference δ_{IN}. Set an input difference for $\mathcal{H}$ to $(0\|\delta_{IN})$ and set an output difference of $\mathcal{H}$ to 0.
2. For the second message block, perform the semi-free-start-collision attack to find many triplets of $(H_1, M_1, M_1 \oplus \delta_{IN})$. The results are stored in a list L.
3. For the first message block, compute $H_1 \leftarrow (H_0, M_0)$ for many randomly generated messages M_0, and find a match with one of the H_1 in L.
4. For a matched $M_0\|M_1$ and $M_0\|M_1 \oplus \delta_{IN}$, a padding block is often appended inside the computation of $\mathcal{H}$. Because all input information for the third message block has no difference, the output difference is ensured to be 0.

The match between the first message block and the second message block is an unbalanced MitM attack. Suppose that the cost of semi-free-start collision attack is 2^β, and $\beta < n/2$. It generates $2^{(n-\beta)/2}$ semi-free-start collisions with a computational cost of $2^{(n-\beta)/2} \times 2^\beta = 2^{(n+\beta)/2}$ and the results are stored in a list L with $2^{(n-\beta)/2}$ memory size. It also tests $2^{(n+\beta)/2}$ random messages in the first message block. In the end, the total computational cost is $2^{((n+\beta)/2)+1}$.

As discussed in Sect. 3.1, the attack can be memoryless, which results in the computational cost of $2^{\beta+(n/2)+1}$ and negligible memory size.

5 Concluding Remarks

In this paper, we studied the memoryless unbalanced MitM attack with the Floyd's cycle-finding algorithm. On the contrary to the previous belief, the unbalanced MitM attack cannot be memoryless without significantly increasing the computational cost compared to the sufficient-memory case. We showed

that among any ratio of the numbers of computing $\mathcal{F}$ and $\mathcal{G}$, 1 to 1 leads to the best computational cost. We then searched for the applications in which the memoryless unbalanced MitM attack with the Floyd's cycle-finding algorithm is still useful. The condition to apply the Floyd's cycle-finding algorithm is that the computational cost for the heavier function $\mathcal{G}$ must be below $2^{(n/2)-1}$. We showed that the application to the MitM preimage attack is impossible. We also showed that a 3-collision attack and a limited-birthday distinguisher on a hash function are good examples of applications.

Our research is the first-step to discuss the unbalanced memoryless MitM attack. A possible future research direction is finding alternatives of the Floyd's cycle-finding algorithm to reduce the memory size. In particular, it is interesting to investigate the memoryless attack when the computational cost of the heavier function $\mathcal{G}$ is bigger than the birthday attack cost.

Acknowledgments. The author would like to appreciate Takanori Isobe for insightful discussions.

References

1. Diffie, W., Hellman, M.E.: Exhaustive Cryptanalysis of the NBS Data Encryption Standard. Computer Issue 6(10) (1977)
2. Chaum, D., Evertse, J.-H.: Crytanalysis of DES with a Reduced Number of Rounds: Sequences of Linear Factors in Block Ciphers. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 192–211. Springer, Heidelberg (1986)
3. Aoki, K., Sasaki, Y.: Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 103–119. Springer, Heidelberg (2009)
4. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
5. Bogdanov, A., Rechberger, C.: A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 229–240. Springer, Heidelberg (2011)
6. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-Middle: Improved MITM Attacks. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 222–240. Springer, Heidelberg (2013)
7. Guo, J., Ling, S., Rechberger, C., Wang, H.: Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 56–75. Springer, Heidelberg (2010)
8. Isobe, T.: A Single-Key Attack on the Full GOST Block Cipher. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 290–305. Springer, Heidelberg (2011)
9. Isobe, T., Shibutani, K.: All Subkeys Recovery Attack on Block Ciphers: Extending Meet-in-the-Middle Approach. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 202–221. Springer, Heidelberg (2013)

10. Isobe, T., Shibutani, K.: Generic Key Recovery Attack on Feistel Scheme. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 464–485. Springer, Heidelberg (2013)
11. Knellwolf, S., Khovratovich, D.: New Preimage Attacks against Reduced SHA-1. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 367–383. Springer, Heidelberg (2012)
12. Khovratovich, D., Nikolić, I., Weinmann, R.P.: Meet-in-the-Middle Attacks on SHA-3 Candidates. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 228–245. Springer, Heidelberg (2009)
13. Sasaki, Y., Aoki, K.: Finding Preimages in Full MD5 Faster Than Exhaustive Search. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 134–152. Springer, Heidelberg (2009)
14. Floyd, R.W.: Nondeterministic Algorithms. Journal of the ACM 14(4), 636–644 (1967)
15. Aoki, K., Sasaki, Y.: Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 70–89. Springer, Heidelberg (2009)
16. Sasaki, Y., Wang, L., Wu, S., Wu, W.: Investigating Fundamental Security Requirements on Whirlpool: Improved Preimage and Collision Attacks. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 562–579. Springer, Heidelberg (2012)
17. Joux, A., Lucks, S.: Improved Generic Algorithms for 3-Collisions. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 347–363. Springer, Heidelberg (2009)
18. U.S. Department of Commerce, National Institute of Standards and Technology: Secure Hash Standard (SHS) (Federal Information Processing Standards Publication 180-3) (2008), http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf
19. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
20. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 1–16. Springer, Heidelberg (2005)
21. Chen, R.: New Techniques for Cryptanalysis of Cryptographic Hash Functions. Ph.D. thesis, Technion (2011)
22. Cochran, M.: Notes on the Wang et al. 2^{63} SHA-1 Differential Path. Cryptology ePrint Archive, Report 2007/474 (2007)
23. Joux, A., Peyrin, T.: Hash Functions and the (Amplified) Boomerang Attack. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 244–263. Springer, Heidelberg (2007)
24. Rijmen, V., Oswald, E.: Update on SHA-1. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 58–71. Springer, Heidelberg (2005)
25. Stevens, M.: New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 245–261. Springer, Heidelberg (2013)
26. Wang, X.: Cryptanalysis of SHA-1 Hash Function. Keynote Speech at The First Cryptographic Hash Workshop conducted by NIST (2005), http://csrc.nist.gov/groups/ST/hash/first_workshop.html

27. Zheng, Y., Pieprzyk, J., Seberry, J.: HAVAL — One-Way Hashing Algorithm with Variable Length of Output. In Seberry, J., Zheng, Y., eds.: AUSCRYPT'92. In: Zheng, Y., Seberry, J. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 83–104. Springer, Heidelberg (1993)
28. Yu, H., Wang, X., Yun, A., Park, S.: Cryptanalysis of the Full HAVAL with 4 and 5 Passes. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 89–110. Springer, Heidelberg (2006)
29. Suzuki, K., Tonien, D., Kurosawa, K., Toyota, K.: Birthday Paradox for Multi-Collisions. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E91-A(1), 39–45 (2008)
30. Iwamoto, M., Peyrin, T., Sasaki, Y.: Limited-birthday Distinguishers for Hash Functions: Collisions Beyond the Birthday Bound can be Meaningful. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 504–523. Springer, Heidelberg (2013)
31. Leurent, G.: MD4 is Not One-Way. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 412–428. Springer, Heidelberg (2008)
32. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 365–383. Springer, Heidelberg (2010)
33. De Cannière, C., Mendel, F., Rechberger, C.: Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 56–73. Springer, Heidelberg (2007)
34. Grechnikov, E.A.: Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics. Cryptology ePrint Archive, Report 2010/413 (2010)
35. Grechnikov, E., Adinetz, A.: Collision for 75-step SHA-1: Intensive Parallelization with GPU. Cryptology ePrint Archive, Report 2011/641 (2011)

A Cycle Construction with SHA-1 Collision Attack

The core idea of the Floyd's cycle-finding algorithm is that, for a previous chain value v_{i-1}, a function $\mathcal{G}$ (and $\mathcal{F}$) must reproduce an identical value v_i as the next chain value. Because the collision attack requires a complicated attack procedure, the detailed operations in $\mathcal{G}$ must be carefully determined. In this section, we explain the following three points that require special attention.

- How to ensure sufficient freedom degrees to find a collision of SHA-1.
- How to reproduce the same colliding value for the same input v_{i-1} received in different timings.
- How to apply advanced collision-search techniques by ensuring the reproduction of the same colliding value.

Ensuring Sufficient Freedom Degrees. Collision attack on SHA-1 [19] generates a collision of 2-blocks long. From several experimental researches on reduced rounds [33–35], we can see that the available freedom degrees for the collision search within the first message block may not be sufficient. This is because most

of the message bits must be fixed to control the differential propagation. Therefore, embedding a previous 160-bit chain value v_{i-1} inside the first message block gives a critically bad impact.

In our method, $\mathcal{G}$ generates a collision consisting of four message blocks, where the size of each message block is 512 bits. The overview is given in Fig. 7. We

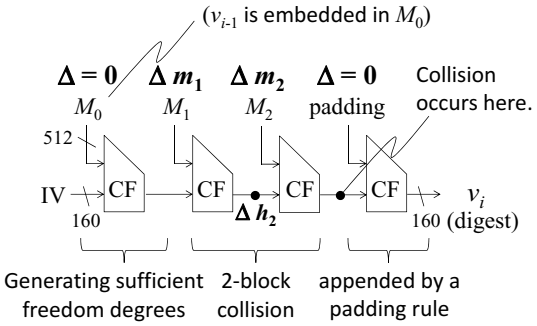

Fig. 7. Construction of $\mathcal{G}$ with SHA-1 Collision Attack

add another message block M_0 before the 2-block collision. The size of M_0 is 512 bits. We set 160 bits of M_0 to v_{i-1}. The other 352 bits can be fixed to any value as long as the rule is uniquely fixed for the reproduction. The simplest way is fixing the other 352 bits to 0. This reproduces the same output value of the first message blocks for the same v_{i-1}. Then, the 2-block collision is located in the second and third message blocks. Note that the third message blocks are also heavily fixed to control the differential propagation. Hence, we cannot embed the padding string inside the third message block. This is the reason why we need the fourth message block.

Because no limitation exists for the second and third message blocks, sufficient freedom degrees can be ensured for generating a 2-block collision.

Reproducing the Same Colliding Value. Collision search algorithm is usually a random algorithm. Messages to be tested are generated randomly from uniformly distributed space. However, this way cannot be used in our case due to the problem of reproduction.

The problem can be simply avoided by stopping using the random algorithm but choose messages to be tested in a specific rule. An example is pre-determining the message-bit positions to be modified during the collision search. If 2^β messages need to be tested, we can choose particular β-bit position. Whenever we change the message, we take the message by modifying the chosen β-bit position. In addition, we set the rule of the order of the modification. For example, we modify the message from the least significant bit. The rule enables us to reproduce the same message when the same situation occurs. Note that not only inside M_1 but also the 352 bits of M_0 can be modified as long as the modification rule is uniquely fixed.

Application of Advanced Collision-Search Techniques. Complicated collision-search techniques such as the message modification technique and the early aborting technique can also be applied with ensuring the reproductivity by pre-determining the application rule. The important thing is that the collision-search algorithm must behave in the same way when the same situation occurs. Therefore, by setting the condition to apply the message modification or early aborting technique, reproducing the same result is possible.