A New Algorithm for the Unbalanced Meet-in-the-Middle Problem

Ivica Nikolić[1] and Yu Sasaki[2]

[1] Nanyang Technological University, Singapore
[2] NTT Secure Platform Laboratories, Tokyo, Japan
`inikolic@ntu.edu.sg` `sasaki.yu@lab.ntt.co.jp`

Abstract. A collision search for a pair of n-bit unbalanced functions (one is R times more expensive than the other) is an instance of the meet-in-the-middle problem, solved with the familiar standard algorithm that follows the tradeoff $TM = N$, where T and M are time and memory complexities and $N = 2^n$. By combining two ideas, unbalanced interleaving and van Oorschot-Wiener parallel collision search, we construct an alternative algorithm that follows $T^2M = R^2N$, where $M \leq R$. Among others, the algorithm solves the well-known open problem: how to reduce the memory of unbalanced collision search.

Keywords: meet-in-the-middle, tradeoff, collision search

1 Introduction

Consider a collision search problem between two n-bit functions $f(x)$ and $g(x)$, in two similar scenarios. In the first case, assume $f(x)$ and $g(x)$ have the same cost (in terms of time complexity). In the second case, assume that $g(x)$ is only $2^{\frac{n}{10}}$ times more costly than $f(x)$. The state-of-the-art suggests we use two different time optimized algorithms for these two similar problems. For the first case we deploy Floyd's cycle finding algorithm [**?**] and produce a collision in $2^{\frac{n}{2}}$ time and *negligible memory*. For the second case, we store $2^{\frac{9n}{20}}$ images of $g(x)$, and with $2^{\frac{11n}{20}}$ evaluations of $f(x)$ find the collision – a process that requires a time equivalent[3] to $2^{\frac{11n}{20}}$ calls to $f(x)$ and a *memory of* $2^{\frac{9n}{20}}$. This sudden jump of memory from negligible to almost $2^{\frac{n}{2}}$, when the comparative cost of the functions has increased only by a small factor, indicates that the state-of-the-art algorithm is inefficient. We eliminate this inefficiency and show an alternative algorithm that relies on the more logical relation between the comparative cost of $g(x)$ to $f(x)$ and the memory: the smaller the comparative cost, the less memory is needed.

In the literature, the above second case is known as the meet-in-the-middle (MITM) problem, and it is solved with the described standard MITM algorithm. Many subproblems in cryptography can be modelled as MITM problems. In

[3] The $2^{\frac{9n}{20}}$ calls to $g(x)$ cost $2^{\frac{9n}{20}+\frac{n}{10}} = 2^{\frac{11n}{20}}$ calls to $f(x)$.

general, any collision search between two functions, which not necessary have the same domain and range, is a MITM problem. In such a form, this makes the MITM one of the most frequently occurring problems, and the MITM algorithms that solve the problems, one of the most widely used algorithms in cryptography.

The MITM problem has two instances. The first is the classical MITM as introduced by Diffie and Hellman [**?**] used for a key recovery in Double DES. It is a collision search problem between two functions with a range larger than a domain. The second instance aims at a collision search between two functions with a range not larger than a domain, but (usually[4]) with different weights. That is, one of the functions requires more time for execution. According to the previous naming convention, we call this instance an unbalanced MITM. In this paper we deal only with the unbalanced case. In the sequel, all references to the MITM problem implicitly assume the unbalanced MITM.

The algorithm that solves the unbalanced MITM allows a simple time-memory tradeoff. It is described with the curve $TM = N$, where $N = 2^n$, T is the time complexity measured in accumulative cost of calls to the functions $f(x)$ and $g(x)$, while M is the memory measured in blocks of certain size (comparable to n). By increasing time and reducing memory, solving certain MITM problems becomes feasible in practice, as usually, the memory is the bottleneck. Conversely, most theoretical applications require time optimized solutions, thus in these cases, the time is reduced and the memory is increased. Note, the time can be reduced only up to a certain bound, usually[5] defined as $\sqrt{N}$. If T goes below the bound and $f(x), g(x)$ are random mappings, then a collision may not be found as the total number of pairs is below N.

Our contribution. In our study of the unbalanced MITM problem, the MITM algorithms and the resulting tradeoffs, we include as a parameter the ratio R of costs of the two function (e.g. in the above first scenario $R = 1$, while in the second $R = 2^{\frac{n}{10}}$). This is essential because R defines how to balance the number of calls to $f(x)$ and to $g(x)$. In short, $f(x)$ can be evaluated R times more frequently than $g(x)$, while maintaining the same time complexity.

Our new MITM algorithm relies on a combination of two ideas, both well known, but never combined together. The first idea is based on a selection function (we call this method *interleaving*) from the memoryless collision search of two functions. Floyd's algorithm can be used to find a collision between the two functions, by interleaving the calls to $f(x)$ and $g(x)$ during the detection of the cycle. That is, Floyd's algorithm is run for a function $F(x)$ that, based on some selection function, evaluates either $f(x)$ or $g(x)$ with equal probability. Thus a collision for $F(x)$ is an actual collision for $f(x)$ and $g(x)$ with a probability $\frac{1}{2}$ and consequently, the search has to be repeated twice. *Unbalanced interleaving* happens when $F(x)$ evaluates one of the functions more frequently (e.g. R times

[4] If the functions are balanced, then a collision can be found trivially with Floyd's cycle finding algorithm.

[5] As shown further, the bound is not universal, but depends on the comparative cost of the two functions.

more) than the other. Then the collision search has to be repeated R times. The second idea relies on van Oorschot-Wiener [?] multiple/parallel collision search based on Hellman's table. First, Hellman's table is built by storing the first and the last points of multiple chains produced from iterative evaluations of a function $F(x)$. Then, to find one collision, another chain for $F(x)$ is built. With the right choice of parameters, one chain in Hellman's table will collide with the newly constructed chain, which can be detected by the end point. Multiple collisions can be built by repeating the same process.

We combine these two ideas by constructing Hellman's table for the function $F(x)$, which is produced by unbalanced interleaving of $f(x)$ and $g(x)$, such that $f(x)$ is called R times more often than $g(x)$. Then, the collision search for $F(x)$ is repeated R times in order to obtain a single collision between $f(x)$ and $g(x)$. (A full description of the algorithm in a pseudo code is given in Appendix A.) Our analysis reveals that this new algorithm relies on the tradeoff

$$T^2M = R^2N$$

where $M \leq R$. It follows that, when R tends to 1, then M tends to 1, and T tends to $2^{\frac{n}{2}}$. In other words, the closer the costs of the two functions, the less memory is required (and the time is closer to the case of balanced functions). In contrast, the standard MITM algorithm relies on the counterintuitive relation: the closer the costs of the two functions, the more memory is required.

We compare the new algorithm to the standard algorithm and show that the new is more memory effective and more time effective for certain values of R. In short, the new algorithm is more time effective when $MR^2 < N$, and more memory effective when $T > R^2$. A visual comparison of tradeoffs of the two algorithms is given in Appendix B.

We present a number of cases where the replacement of the standard algorithm with the new will lead either to a lower memory requirement or to a better time-memory tradeoff. In addition, we point out cases where such replacement will not work (e.g. known plaintext attacks on block ciphers). Finally, we show that some balanced collision search problems can be regarded as unbalanced, and thus with the use of the new algorithm, can be solved more efficiently (usually, will require less memory).

Related work. The unbalanced MITM has been mentioned as a subproblem in a large number of papers. A few result provide an actual memoryless solution to the problem, for instance, Dunkelman et al. in [?]. The most extensive analysis in this direction has been done by Sasaki [?], who even considers unbalanced interleaving and comes to a conclusion that the time complexity of the memoryless unbalanced MITM is invariant of the interleaving factor. In short, all of the currently proposed memoryless algorithms for the unbalanced MITM provide the same time complexity, which is actually the precise point of our tradeoff curve with $M = 1$ and thus $T = R\sqrt{N}$.

Van Oorschot-Wiener multiple collision search [?] has been a fundamental tool in many research papers as well. Among the latest applications of this technique, we single out the memory efficient multicollision search by Joux and Lucks [?], the technique of dissection by Dinur et al. [?], the tradeoffs for the generalized birthday problem by Nikolić-Sasaki [?] and Khovratovich-Biryukov [?], the multi-user collisions by Fouque et al. [?], and others.

2 Preliminaries

2.1 Basics

Let n be a positive integer, and $N = 2^n$. Let $f(x), g(x) : \{0,1\}^n \to \{0,1\}^n$ be two random functions (the range can be smaller than the domain, without affecting the presented analysis). Assume that the time T_f required to compute $f(x)$ is not more than the time T_g required to compute $g(x)$. Let $R = 2^\rho$ be the ratio of the costs of $g(x)$ to $f(x)$, that is, $R = 2^\rho = \frac{T_g}{T_f}$. Obviously, $R \geq 1$. We measure the time complexity of an algorithm in the number of equivalent calls/evaluations to $f(x)$. For instance, if an algorithm makes u calls to $f(x)$ and v calls to $g(x)$, then the time complexity is $u + R \cdot v$.

The MITM problem for $f(x), g(x)$, also known as the collision search problem between $f(x)$ and $g(x)$, consists in finding two n-bit values a and b such that $f(a) = g(b)$. This problem can be solved with the use of the MITM algorithm, referred further as the standard MITM algorithm or $\texttt{MITM}^{\texttt{STD}}$. The algorithm works in two phases. First, in a hash table L it stores 2^m pairs $(g(b_i), b_i)$ indexed by $g(b_i)$, where $b_i, i = 1, \ldots, 2^m$ are random values. Then, it keeps generating pairs $(a_j, f(a_j))$, where a_j are random values, until for some j the value of $f(a_j)$ collides with some $g(b_i)$ from the table L. As $f(x), g(x)$ are random, a collision will occur after around 2^{n-m} values of $f(a_j)$ have been generated.

The memory complexity of the standard algorithm is $M = 2^m$. It makes 2^m calls to $g(x)$ to create L, and 2^{n-m} calls to $f(x)$ to find the collision. According to the above notation, the time complexity T of the algorithm is $T = R \cdot 2^m + 2^{n-m} = 2^{m+\rho} + 2^{n-m}$. For convenience, assume that $T = \max(2^{m+\rho}, 2^{n-m})$, as this reduces the actual time at most by a factor of two.

Let us focus on possible time-memory tradeoffs. When, $m + \rho \leq n - m$, then $T = 2^{n-m}$, and thus the standard algorithm allows the tradeoff $TM = 2^{n-m}2^m = 2^n = N$. On the other hand, when $m + \rho > n - m$, then $T = 2^{m+\rho}$, and thus $TM = 2^{m+\rho}2^m > 2^{n-m}2^m = N$. Obviously this option is worse and therefore further we assume that the memory satisfies $m + \rho \leq n - m$ or equivalently $RM^2 \leq N$ and focus on the tradeoff

$$TM = N. \tag{1}$$

From $RM^2 \leq N$, it follows that $N \geq RM^2 = R(\frac{N}{T})^2$, which leads to $T \geq \sqrt{RN}$.

2.2 Collisions Search with Interleaving

Let us consider the collision search problem between two n-bit functions $f(x)$ and $g(x)$. A memoryless approach to this problem is based on alteration of the well-known Floyd's cycle-finding algorithm that finds collisions for a single function, i.e. finds (a, b) such that $f(a) = f(b)$.

In the case of a single function $f(x)$, Floyd's algorithm picks a random starting point u, assigns $v_0 = w_0 = u$, and iteratively produces values $v_i = f(v_{i-1}), w_i = f(f(w_{i-1}))$ until a collision between v_i and w_i is reached. This colliding value belongs to a cycle, and if the random point u was chosen to be outside the cycle, then with an additional effort, the two colliding values a and b for $f(x)$ can be found: a will be the value that turns the iteration into a cycle, while b the value of the cycle. From the properties of random mappings, it follows that length of the cycle and the length of a chain that leads to a cycle is around $2^{\frac{n}{2}}$, thus the whole algorithm has a time complexity of around $2^{\frac{n}{2}}$ evaluations of $f(x)$ and it uses a negligible memory.

In the case of two functions $f(x), g(x)$, Floyd's algorithm still works and requires a small alteration. The trick is to *interleave* the evaluations of $f(x)$ and $g(x)$ with the use of a selection function $\sigma(x)$ which maps n-bit values to a single bit in a random fashion. That is, $\sigma(x)$ outputs 0 or 1, randomly and with equal probability. Define a function $F(x)$ as follows:

$$F(x) = \begin{cases} f(x) & \text{if } \sigma(x) = 0 \\ g(x) & \text{if } \sigma(x) = 1 \end{cases}$$

Then, with Floyd's algorithm find a colliding pair (a, b) for $F(x)$. Obviously if $\sigma(a) \neq \sigma(b)$, then this translates to a collision between $f(x)$ and $g(x)$. Otherwise, repeat the collision search with another starting value. As a result, a colliding pair (a, b) for $f(x), g(x)$ is found with around $2^{\frac{n}{2}}$ evaluations of both $f(x)$ and $g(x)$, and it requires a negligible memory.

We have assumed above that the cost of the two functions is the same, i.e. $R = 1$. However, if $g(x)$ is more costly than $f(x)$, then the time complexity of the above Floyd's algorithm is around $R \cdot 2^{\frac{n}{2}}$. An alternative way to find a collision between two unbalanced function is to use *unbalanced interleaving* as suggested by Sasaki [**?**]. That is, the selection function $\sigma(x)$ outputs 0 around R times more often than 1. In such a case, a collision for $F(x)$ can be found in around $2^{\frac{n}{2}}$ calls to $f(x)$ and $\frac{2^{\frac{n}{2}}}{R}$ calls to $g(x)$, thus in time equivalent to around $2^{\frac{n}{2}}$ calls to $f(x)$ (recall that we measure the time complexity in calls to $f(x)$). However, a collision for $F(x)$ is an actual collision between $f(x)$ and $g(x)$ only with a probability of $\frac{1}{R}$, thus the collision search has to be repeated around R times. This brings the total time complexity of producing a collision between $f(x)$ and $g(x)$ to $R \cdot 2^{\frac{n}{2}}$.

2.3 Multiple Collision Search

Consider the problem of finding multiple collisions for a function $f(x)$, i.e. pairs $(a_1, b_1), \ldots, (a_s, b_s)$ such that $f(a_i) = f(b_i)$ for $i = 1, \ldots, s$. By running Floyd's cycle finding algorithm s times (each with a different starting point and a different reduction function), the required s collisions are found in $s \cdot 2^{\frac{n}{2}}$ evaluations of

$f(x)$ and with negligible memory. However, if s is sufficiently large, then the parallel collisions search algorithm by Van Oorschot and Wiener [?] has favourable time complexity, but it requires non-negligible memory.

Let $M = 2^m$ be the available amount of memory. Van Oorschot-Wiener algorithm (given in a pseudo code in Alg. 1 in Appendix A) starts by building a hash table L_m that resembles Hellman's table from the well known time-memory tradeoffs [?]. Each entry in the table consists of two values: a random starting value v_s, and a value v_e produced after $2^{\frac{n-m}{2}}$ iterative applications of $f(x)$ to v_s (i.e. $v_0 = v_s, v_{i+1} = f(v_i), v_e = v_{2^{\frac{n-m}{2}}}$). The table L_m has 2^m such entries[6] indexed by the values v_e, and thus it requires 2^m memory. It is built in $2^m 2^{\frac{n-m}{2}} = 2^{\frac{n+m}{2}}$ time. Note, collisions between iterations are prevented by the so-called matrix stopping rule[7]. It guarantees that if $M \cdot l^2 \leq 2^n$, where l is the length of an iteration, then the number of collisions is negligible. In the above case $l = 2^{\frac{n-m}{2}}$, hence $M \cdot l^2 = 2^n$, thus the condition is fulfilled.

To find one collision for $f(x)$ with the use of L_m, choose a random value w_0 and build a chain composed of values w_i where $w_{i+1} = f(w_i)$ (refer to Alg. 2 in Appendix A). Each time a new value of w_{i+1} is computed, check in L_m if it coincides with one of v_e. If it does, then pick the corresponding starting value v_s and the value w_0 and find the colliding pair. The table L_m covers $2^{\frac{n+m}{2}}$ values. Hence, if the length of the chain is around $2^{n-\frac{n+m}{2}} = 2^{\frac{n-m}{2}}$, we can expect that some value of the chain will hit a value produced during the construction of the table. Obviously, such a hit can be detected once one of the consecutive points of the chain has coincided with some v_e from L_m. As mentioned earlier, the average length of the chain at the moment of hit is $2^{\frac{n-m}{2}}$ and with an additional effort of not more than $2^{\frac{n-m}{2}}$ evaluations[8] of $f(x)$ such hit can be detected. Therefore, a collision can be found in around $2^{\frac{n-m}{2}}$ evaluations of $f(x)$. The procedure of generating L_m and finding a collision is illustrated in Fig 1.

To produce s collisions, van Oorschot-Wiener algorithm requires $T = 2^{\frac{n+m}{2}} + s \cdot 2^{\frac{n-m}{2}}$ evaluations of $f(x)$. Floyd's algorithm needs $T = s \cdot 2^{\frac{n}{2}}$. Thus, roughly when the required number of collisions s satisfies $s > \sqrt{M}$, then van Oorschot-Wiener algorithm has a lower time complexity than Floyd's algorithm. Conversely, if s collisions are required, then it suffices to use $M < s^2$ memory in order to achieve better algorithm in terms of time complexity. For instance, when $s = 2^{\frac{n}{3}}$ and $M = 2^{\frac{n}{3}}$, then van Oorschot-Wiener algorithm requires only $2^{\frac{2n}{3}}$ time, while Floyd's algorithm requires $2^{\frac{n}{3}+\frac{n}{2}} = 2^{\frac{5n}{6}}$ time.

[6] Each built by starting from a different point v_s.

[7] The term *matrix stopping rule* has been introduced by Biryukov-Shamir [?]. In the original Hellman's paper [?] on TMTO, this rule was given without any particular name (see page 403, Remark 1).

[8] Because v_e is produced from v_s in $2^{\frac{n-m}{2}}$ iterations.

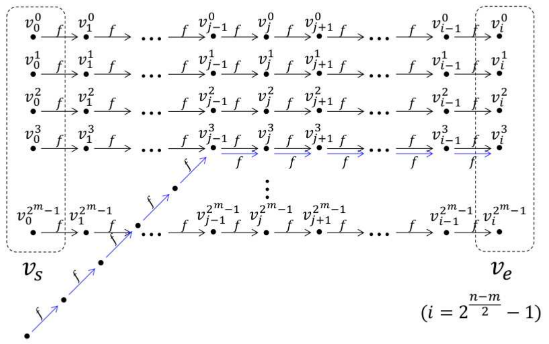

Fig. 1. L_m generation and collision detection when 2^m memory is available. Only v_s and v_e are stored in L_m. Blue lines describe a collision detection performed after L_m is generated. It will hit an intermediate value of one of the chains. By continuing the computation, it reaches v_e, thus the collided chain is identified. The exact colliding value can be detected after some re-computation of the two chains (refer to Alg. 2 in Appendix A).

3 A New Meet-in-the-Middle Algorithm

3.1 The Algorithm

To construct our new meet-in-the-middle algorithm, we combine the concepts of unbalanced interleaving and multiple collisions search.

Specification of the algorithm. A description of the complete algorithm in a pseudo code is given in Alg. 4 of the Appendix. In short, the algorithm can be defined as follows:

1. **Unbalanced interleaving:** Define a function $F(x)$ as

$$F(x) = \begin{cases} f(x) & \text{if } \sigma(x) = 0 \\ g(x) & \text{if } \sigma(x) = 1 \end{cases}$$

where the selection function $\sigma(x) : \{0,1\}^n \rightarrow \{0,1\}$ outputs 0 around 2^ρ times more frequently than 1. For instance, $\sigma(x)$ can be defined as[9]

$$\sigma(x) = \begin{cases} 1 & \text{if } \rho \text{ least significant bits of } x \text{ are zero} \\ 0 & \text{otherwise} \end{cases}$$

Hence, $F(x)$ evaluates $f(x)$ around 2^ρ times more frequently than $g(x)$.

[9] With such a definition, we assume that $f(x), g(x)$ are random.

2. **Collision table:** Based on van Oorschot-Wiener algorithm, create a table L_m with $M = 2^m$ entries for the function $F(x)$.

3. **Multiple collision search:** With the use of L_m, keep producing collisions for $F(x)$, until actual collision between $f(x)$ and $g(x)$ occurs.

After around 2^ρ collisions for $F(x)$, the required collision between $f(x)$ and $g(x)$ will appear. Indeed, from the definition of $F(x)$ it follows that the probability that a collision for $F(x)$ is an actual collision between $f(x)$ and $g(x)$ is $2^{-\rho}$.

Time-memory tradeoff. Let us find the time complexity of the above algorithm. As stated in the previous section, van Oorschot-Wiener algorithm requires $T_1 = 2^{\frac{n+m}{2}}$ evaluations of $F(x)$ to construct the table and $T_2 = 2^\rho \cdot 2^{\frac{n-m}{2}} = 2^{\rho+\frac{n-m}{2}}$ evaluations to find 2^ρ collisions. Hence, the time complexity of our algorithm is $T = T_1 + T_2$. To simplify the analysis we assume that $T = \max(T_1, T_2)$ (we ignore the constant factor of 2). The required memory is $M = 2^m$.

Let us express the values of T_1 in terms of calls to $f(x)$ (recall that we measure the time cost in terms of the lighter function $f(x)$). In T_1, there are a total of $2^{\frac{n+m}{2}}$ evaluations of $F(x)$, out of which, around $2^{\frac{n+m}{2}}$ are to the function $f(x)$ and $2^{\frac{n+m}{2}}/2^\rho = 2^{\frac{n+m}{2}-\rho}$ to $g(x)$ which in turn are equivalent to $2^\rho \cdot 2^{\frac{n+m}{2}-\rho} = 2^{\frac{n+m}{2}}$ calls to $f(x)$. Thus $T_1 = 2^{\frac{n+m}{2}}$ calls to $f(x)$. In T_2, there are $2^{\rho+\frac{n-m}{2}}$ evaluations of $F(x)$, out of which around $2^{\rho+\frac{n-m}{2}}$ are to $f(x)$ and $2^{\rho+\frac{n-m}{2}}/2^\rho = 2^{\frac{n-m}{2}}$ to $g(x)$, which is equivalent to $2^{\rho+\frac{n-m}{2}}$ calls to $f(x)$. As a result, $T_2 = 2^{\rho+\frac{n-m}{2}}$ calls to $f(x)$. Therefore, the total time complexity T expressed above as number of calls to $F(x)$ can be replaced with calls to $f(x)$.

Further we focus on $T = \max(T_1, T_2) = \max(2^{\frac{n+m}{2}}, 2^{\rho+\frac{n-m}{2}})$ and analyze the two cases:

1. Assume $2^{\frac{n+m}{2}} \le 2^{\rho+\frac{n-m}{2}}$ and thus $T = 2^{\rho+\frac{n-m}{2}}$. In this case, we obtain that
$$T^2 M = 2^{2\rho+n-m}2^m = 2^{2\rho}2^n 2^{-m}2^m = R^2 N \tag{2}$$

2. Assume $2^{\frac{n+m}{2}} \ge 2^{\rho+\frac{n-m}{2}}$ and thus $T = 2^{\frac{n+m}{2}}$. Similarly, we end up with the tradeoff
$$T^2 = 2^{n+m} = N \cdot M \tag{3}$$

At the point $2^{\frac{n+m}{2}} = 2^{\rho+\frac{n-m}{2}}$ the tradeoffs switch. This point is defined as

$$\frac{n+m}{2} = \rho + \frac{n-m}{2} \tag{4}$$
$$m = \rho \tag{5}$$

Hence, when the available memory M is not more than R, the time T and memory M complexity of our meet-in-the-middle follows the tradeoff $T^2M = R^2N$. On the other hand, when $M \ge R$, then our tradeoff follows the curve $T^2 = N \cdot M$. In this case, we can see that when the memory increases, the time increases as well. Therefore this tradeoff is not beneficial and thus further in our discussion we focus only on the tradeoff $T^2M = R^2N$, where $M \leq R$. In addition, the time is limited to $T = \sqrt{\frac{R^2N}{M}} \geq \sqrt{\frac{R^2N}{2^\rho}} = \sqrt{RN}$.

3.2 Comparison of Tradeoffs

Let us compare the new meet-in-the-middle algorithm `MITM` $^{\texttt{NEW}}$ to the standard meet-in-the-middle algorithm `MITM` $^{\texttt{STD}}$ in terms of time and memory complexities. A graphical comparison of the two tradeoffs is given in Appendix B.

Time comparison of the tradeoffs. Assume `MITM` $^{\texttt{NEW}}$ and `MITM` $^{\texttt{STD}}$ use the same amount of memory M and we want to find the case when our algorithm has a lower time complexity than the standard. When $M \leq R$, then the time complexity of `MITM` $^{\texttt{NEW}}$ is $T_1 = R\sqrt{\frac{N}{M}}$, while of `MITM` $^{\texttt{STD}}$ is $T_2 = \frac{N}{M}$ and thus

$$R\frac{N^{\frac{1}{2}}}{M^{\frac{1}{2}}} = T_1 < T_2 = \frac{N}{M} \tag{6}$$

$$RM^{\frac{1}{2}} < N^{\frac{1}{2}} \tag{7}$$

$$R^2M < N \tag{8}$$

From (8) and $M \leq R$ we can conclude that

Fact 1 *Let R be the ratio of costs of $g(x)$ to $f(x)$, M be the available memory, and let $M \leq R$. Then `MITM` $^{\texttt{NEW}}$ has a lower time complexity than `MITM` $^{\texttt{STD}}$ when*

$$M < \frac{N}{R^2}. \tag{9}$$

Remark 1 (Necessary condition). From (9) it follows that the new algorithm may have a better time complexity only if $R < N^{\frac{1}{2}}$.

Memory comparison of the tradeoffs. Similarly, let us compare the memory complexities of the two algorithms when they use the same amount of time T. Assume $M \leq R$. Then the memory complexity of `MITM` $^{\texttt{NEW}}$ is $M_1 = \frac{R^2N}{T^2}$, while of the `MITM` $^{\texttt{STD}}$ is $M_2 = \frac{N}{T}$, thus

$$\frac{R^2N}{T^2} = M_1 < M_2 = \frac{N}{T} \tag{10}$$

$$T > R^2 \tag{11}$$

The condition $R \geq M_1 = \frac{R^2N}{T^2}$ is equivalent to $T \geq \sqrt{RN}$. As a result we get

Fact 2 *Let R be the ratio of costs of $g(x)$ to $f(x)$, T be the available time, and let $T \geq \sqrt{NR}$. Then `MITM` $^{\texttt{NEW}}$ has a lower memory complexity than `MITM` $^{\texttt{STD}}$ when*

$$T > R^2. \tag{12}$$

Remark 2 (Necessary condition). From (9) and $T < N$ it follows that the new algorithm may have a better memory complexity only if $R < N^{\frac{1}{2}}$.

When used in analysis, often the parameters of the tradeoff are chosen in a way to minimize the time complexity. That is, the most used point of the curve in the tradeoff of the standard meet-in-the-middle algorithm is the one where the time complexity reaches the minimum. As mentioned in Sect. 2.1, this point is defined as $T = 2^{\frac{n+\rho}{2}} = \sqrt{NR}$ and $M = 2^{\frac{n-\rho}{2}} = \sqrt{\frac{N}{R}}$. As the condition $T \geq \sqrt{NR}$ of Fact 2 is satisfied, it follows that our $\texttt{MITM}^{\texttt{NEW}}$ will always use less memory than $\texttt{MITM}^{\texttt{STD}}$ as long as $T > R^2 = \frac{T^4}{N^2}$ or equivalently, $T < N^{\frac{2}{3}}$. This leads to

Fact 3 *Let $T < N^{\frac{2}{3}}$ be the minimal time complexity of $\texttt{MITM}^{\texttt{STD}}$, that uses $M_2 = \frac{N}{T}$ memory. Then, with the use of $\texttt{MITM}^{\texttt{NEW}}$, the memory complexity can be reduced to $M_1 = \frac{T^2}{N}$.*

Proof. From $T < N^{\frac{2}{3}}$ it follows that $R = \frac{T^2}{N} < N^{\frac{1}{3}}$ and $M_2 = \frac{N}{T} > N^{\frac{1}{3}}$. We choose $M_1 = R$, and use our $\texttt{MITM}^{\texttt{NEW}}$ to achieve $M_1 = \frac{R^2N}{T^2} = \frac{T^4N}{N^2T^2} = \frac{T^2}{N} < \frac{N^{\frac{4}{3}}}{N} = N^{\frac{1}{3}}$. $\square$

3.3 Practical Confirmation

We confirm the correctness of the new algorithm and the resulting tradeoff by implementing it and by running a series of computer experiments. In the experiments, the value of N is in the range of 2^{32} to 2^{40}, and the values of R and M vary (but comply to $M \leq R$). For each particular N, R, and M, we run 100 experiments, each with different $f(x)$ and $g(x)$, and measure the time complexity required to produce a collision between $f(x)$ and $g(x)$.

In Tbl. 1 we report the measured time as the average of the 100 experiments. It is evident that the experimental time is very close to the expected time and differs roughly by a factor of four.

3.4 Additional Cases

Besides for the unbalanced MITM, $\texttt{MITM}^{\texttt{NEW}}$ can be used as well to solve a few other collision problems between balanced functions. Further we describe two potential applications. In Section 4 we provide concrete examples of these applications.

- **Reducing calls to one of the functions.** In certain applications, even though the costs of the two functions are the same ($R = 1$), it may be beneficial to reduce the number of calls to one of them. For instance, if $g(x)$ depends on a secret key k thus is written as $g(k, x)$, then it has to be queried to get the result. Thus the number of calls to $g(x)$ corresponds to the data complexity D. If reducing D is the priority, then the collision search becomes unbalanced.

Table 1. Experimental verification of the new tradeoff.

MITM space	**Ratio**	**Memory**	**Expected time**	**Experimental time**
N	R	M	$T = \sqrt{R^2N/M}$	T
2^{32}	2^{8}	2^{6}	2^{21}	$2^{22.6}$
2^{32}	2^{4}	2^{4}	2^{18}	$2^{20.0}$
2^{32}	2^{12}	2^{10}	2^{23}	$2^{25.3}$
2^{32}	2^{12}	2^{12}	2^{22}	$2^{24.0}$
2^{36}	2^{10}	2^{8}	2^{24}	$2^{26.2}$
2^{36}	2^{10}	2^{10}	2^{23}	$2^{24.7}$
2^{36}	2^{12}	2^{12}	2^{24}	$2^{25.9}$
2^{40}	2^{6}	2^{4}	2^{24}	$2^{26.1}$
2^{40}	2^{6}	2^{6}	2^{23}	$2^{24.9}$
2^{40}	2^{8}	2^{8}	2^{24}	$2^{25.7}$

– **Reduced domain of one of the functions.** So far, we have assumed that the ranges of the two functions are not larger than their domain. If one of the balanced functions has a domain smaller than the range, then MITM $^{\texttt{NEW}}$ can be used to find a collision. That is, a collision between $f(x) : \{0,1\}^n \rightarrow \{0,1\}^n$ and $g(x) : \{0,1\}^m \rightarrow \{0,1\}^n$, where $m < n$ and f, g are balanced, can be found with the proposed algorithm.

3.5 Degenerate Cases

MITM $^{\texttt{NEW}}$ in an alternative to the MITM $^{\texttt{STD}}$, but in some cases it may not be applied or it may not follow the expected time-memory tradeoff curve. Let us take a closer look at such degenerate cases.

– **The ratio R depends on the available memory.** An implicit assumption used in the above analysis is that the ratio R of costs of the two function is fixed and invariant of the available memory. This may not always be the case, and one of the functions (most likely $g(x)$), may have execution time that depends on the available memory (the larger the memory, the shorter the time). In such a situation, the ratio R becomes a function of the memory M, i.e. $R = R(M)$, and the curve becomes $T^2M = R(M)^2N$. This may limit the flexibility of choosing M, lead to another tradeoff, or even make the entire tradeoff invalid (recall that it is valid when $M \leq R$, which becomes $M \leq R(M)$ – this condition may not have a solution for $M > 0$).

– **Sets instead of functions.** MITM $^{\texttt{NEW}}$ makes calls to both $f(x)$ and $g(x)$, thus the functions must be computable. If one of the function is given as a set, then the algorithm will not function properly. Note, the naive idea of storing the set only leads to MITM $^{\texttt{STD}}$.

– **Known plaintext attack.** MITM $^{\texttt{NEW}}$ makes adaptive chosen queries to both $f(x)$ and $g(x)$. Thus attacks on block ciphers that are based on MITM $^{\texttt{NEW}}$ cannot be known plaintext attacks.

4 Applications

Further we show applications of the `MITM` $^{\texttt{NEW}}$ in three different cases: the first is the standard unbalanced MITM, while the remaining two are for the additional cases mentioned in Sect. 3.4.

4.1 The Case of Unbalanced Functions

Prior to presenting concrete applications of the `MITM` $^{\texttt{NEW}}$, we emphasize two points. First, `MITM` $^{\texttt{NEW}}$ can be used to achieve better tradeoffs (for certain values of M and T) in a lot of cases where `MITM` $^{\texttt{STD}}$ has been applied. There are numerous such cases – listing and analyzing them is too tedious, and therefore we do not mention them. Second, when the amount of memory is not limited, then both `MITM` $^{\texttt{NEW}}$ and `MITM` $^{\texttt{STD}}$ have the same time complexity (both achieve the minimal possible theoretical time $T = \sqrt{RN}$). Hence, if the user is not concerned about the memory, then he/she can use either `MITM` $^{\texttt{STD}}$ or `MITM` $^{\texttt{NEW}}$. We are ready now to proceed with concrete applications.

Iwamoto et al. [?] show that in narrow-pipe Merkle-Damgård hash functions, a collision attack for the compression function can be converted into a limited-birthday-distinguisher for the corresponding hash function. Recall that a collision[10] for a compression function $CF(h, m)$, where h is the chaining value and m is the message block, is a tuple (h^*, m, m') such that $CF(h^*, m) = CF(h^*, m')$. On the other hand, a limited-birthday distinguisher for a hash function $H(M)$ is the following problem: given two sets I, O, find a message M^* such that $H(M^* \oplus \delta_{in}) \oplus H(M^*) = \delta_{out}$, where $\delta_{in} \in I, \delta_{out} \in O$. It can be seen as a problem of finding a message that follows a certain truncated differential (I, O are the truncated differences at the input and at the output, respectively).

Iwamoto et al. convert the collision into a limited-birthday distinguisher by placing the collision at the second block (refer to Fig. 2). That is, they first find multiple collisions (h_1, m_1, m'_1) for the compression function, store all h_1, and from the initial chaining value h_0, find a message m_0 that will produce a match with one of the stored h_1, i.e. find m_0 such that $CF(h_0, m_0) = h_1$. The complexity of the limited-birthday distinguisher in part depends on the complexity of producing collisions for the compression function. Thus, it is a classical example of an unbalanced MITM problem. Iwamoto et al. essentially use `MITM` $^{\texttt{STD}}$ while we will switch to `MITM` $^{\texttt{NEW}}$.

Application to LANE-256. LANE-256 is a SHA-3 candidate hash function designed by Indesteege et al. [?] that has 256-bit state. A collision attack on the full compression function has been presented by Matusiewicz et al. [?]. Naya-Plasencia [?] has improved the attack – her collision search requires 2^{80} calls to the compression function and 2^{66} memory.

[10] Sometimes, it is called a semi-free-start collision for the compression function.

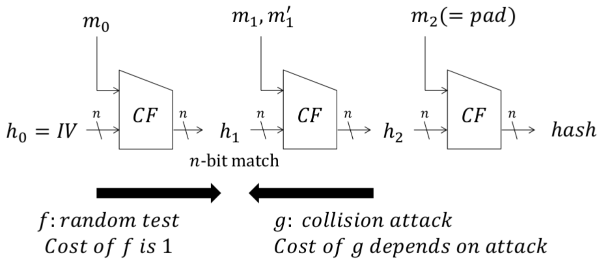

Fig. 2. Conversion from a collision attack on the compression function into a limited-birthday-distinguisher on hash function as shown in [?]. The third message block deals with padding. As the collision occurs on h_2, the third block preserves collision.

We use Iwamoto et al. conversion of the collision attack for the compression function into a limited-birthday distinguisher for the hash function. The resulting unbalanced MITM (on which the limited-birthday distinguishers relies on) consists of the two functions $f(x)$ and $g(x)$, such that $f(x)$ is equivalent to one compression function call (with a random message block), while $g(x)$ is equivalent to one collision for the compression function (according to Naya-Plasencia equivalent to 2^{80} calls and 2^{66} memory). Therefore, the ratio of costs is $R = 2^{80}$.

Iwamoto et al. [?] use MITM $^{\text{STD}}$ to find the complexity of the limited-birthday distinguisher for LANE-256. We use MITM $^{\text{NEW}}$ and show its advantage. From $N = 2^{256}$ and $R = 2^{80}$, it follows that MITM $^{\text{NEW}}$ can be described as $T^2M = 2^{2*80+256} = 2^{416}$. For example, if we set the time complexity to be identical to [?], i.e. $T = 2^{169}$, then the memory complexity M is reduced to $2^{416-2*169} = 2^{78}$, which improves the previous 2^{88} by a factor of 2^{10}. If we set the memory complexity to the lowest possible $M = 2^{66}$ (Naya-Plasencia collision attack requires this much memory), then the previous MITM $^{\text{STD}}$ requires $T = 2^{190}$ (the tradeoff is $TM = 2^{256}$), while our MITM $^{\text{NEW}}$ requires $T = 2^{175}$.

Application to AES-Miyaguchi-Preneel. Iwamoto et al. show as well that in 2^{48} time they can find a collision for the compression function built upon 6-round AES in Miyaguchi-Preneel mode. Therefore, a limited-birthday distinguisher for the corresponding Merkle-Damgård hash function, is equivalent to an unbalanced MITM, where $R = 2^{48}$.

According to Fact 1, with parameters $N = 128$ and $R = 48$, MITM $^{\text{NEW}}$ has lower complexity than MITM $^{\text{STD}}$ if $M < \frac{N}{R^2} = 2^{32}, M \leq R = 2^{48}$, which reduces to $M < 2^{32}$. The time complexity of Iwamoto et al. result with $M = 2^{48}$ cannot be improved. However, 2^{48} memory may be too costly and it may be beneficial to reduce the time, when the available memory is much smaller. As suggested by the above condition, when the memory is limited up to 2^{32}, MITM $^{\text{NEW}}$ gives better time than MITM $^{\text{STD}}$.

4.2 The Case of Reduced Calls

Consider a MITM attack between two balanced functions $f(x)$ and $g(x)$, where $f(x)$ can be computed offline, while $g(x)$ requires oracle queries. Calls to $f(x)$ are counted as a time complexity, while to $g(x)$ as a data complexity. In practice, we often want to keep the data complexity low, which results in an unbalanced MITM. In addition, some schemes (for instance, Chaskey [?]) limit the number of online queries to less than the birthday bound, and thus are able to prove beyond-birthday-bound security.

The best example that illustrates the importance of `MITM` $^{\texttt{NEW}}$ to these cases would be to use it to answer Dunkelman et al. [?] open problem about memoryless attack on Even-Mansour with T time and $D = \frac{N}{T}$ data. However, this problem already has been solved partially by Fouque et al. [?]. They provide a solution that uses M memory and D data, such that $M < D$ and $MD^2 = N$. Interestingly, their approach also relies on van Oorschot-Wiener algorithm, but they do not use unbalanced interleaving. With `MITM` $^{\texttt{NEW}}$, we can obtain the same solution (thus we omit it from the paper). However, our approach is more generic than [?] – we show this by applying `MITM` $^{\texttt{NEW}}$ to key recovery attacks on tweakable block cipher constructions[11].

Tweakable Block Cipher Mode-of-Operation. The first example is a *Tweak-dependent Rekeying (TDR)* mode-of-operation proposed by Minematsu [?]. Let E_K be a block cipher with n-bit state and n-bit key, and let E_K^t, where $t < \frac{n}{2}$, be a construction in which the first $n - t$ bits of the plaintext for E_K are fixed to 0, namely the plaintext space is limited to t bits. The TDR mode converts E_K into a tweakable block cipher (uses t-bit tweak) with two E_K calls: the first encrypts a tweak Tw with E_K^t, used in the second call as a key:

$$K' \leftarrow E_K^t(Tw),$$
$$C \leftarrow E_{K'}(P).$$

Minematsu proves that the TDR mode achieves $O(\frac{2^n}{2^t})$ security. As $t < \frac{n}{2}$, the TDR mode achieves beyond-birthday-bound security. This bound is tight, as shown by the following attack that uses `MITM` $^{\texttt{STD}}$:

1. Fix P to a randomly chosen value.
2. Choose D random values of Tw, query (P, Tw) to obtain the corresponding C, and store all C in a table L.
3. Make $2^n/D$ guesses of K', compute $C \leftarrow E_{K'}(P)$ and look for a match in L.

A match suggests a candidate for K'. With a negligibly small additional cost, the correct K' can be verified. As the analysis relies on `MITM` $^{\texttt{STD}}$, it follows the tradeoff $TD = 2^n$. The required memory is identical to the data, i.e. $M = D$.

[11] In our understanding, these problems cannot be solved with the algorithm from [?].

To find a collision between steps 2) and 3), we can use `MITM` $^{\texttt{NEW}}$– as in the above analysis[12], such collision will exist as long as $TD = 2^n$. The memory, however, can be reduced with `MITM` $^{\texttt{NEW}}$. The unbalanced MITM will make T calls to $f(x)$ and D calls to $g(x)$, if we set $R = \frac{T}{D}$. In such a case, the tradeoff becomes $T^2M = \left(\frac{T}{D}\right)^2 2^n$, which is equivalent to $MD^2 = 2^n$. Thus, when the data D satisfies $D > 2^{\frac{n}{3}}$, the new approach will require less memory. For instance, if $D = 2^{\frac{3n}{7}}$, then the standard (as given above in steps 2),3)) will require $M = 2^{\frac{3n}{7}}$, while the new only $M = 2^{\frac{n}{7}}$ memory.

Cryptanalysis on McOE-X. At FSE 2009, Fleischmann et al.[**?**] propose a family of online authenticated encryption called McOE. Let $E_{K,Tw}$ be a tweakable block cipher under a key K and a tweak Tw. Then, the ciphertext C_i of the i-th message block P_i of McOE is defined as follows:

$$\begin{aligned} t_i &\leftarrow P_{i-1} \oplus C_{i-1}, \\ C_i &\leftarrow E_{K,t_i}(P_i). \end{aligned}$$

McOE-X is an instance of the McOE family, such that $E_{K,Tw} = E_{K \oplus Tw}$.

Mendel at al. [**?**] show that the key of McOE-X can be recovered in $O(2^{\frac{n}{2}})$ time and data, or more general, in T time and $D = \frac{2^n}{T}$ data, with `MITM` $^{\texttt{STD}}$.

1. Fix the message for the second block P_1 to a randomly chosen value.
2. Choose D random values of the first message block P_0, query $P_0 \| P_1$ to obtain the corresponding $C_0 \| C_1$, and store them in a table L along with $P_0 \oplus C_0$.
3. Make $2^n/D$ guesses of $K \oplus t_1$, denoted by K', and compute $C_1 \leftarrow E_{K'}(P_1)$. Check for a match with L.

A match suggests that the K can be computed as $P_0 \oplus C_0 \oplus K'$.

As in the case of TDR, with the use of `MITM` $^{\texttt{NEW}}$ we can reduce the memory requirement of Mendel et al. attack (which currently is $M = D$), while maintaining the same time T and data D. Fleischmann et al. instantiate McOE-X with AES-128 as an underlying block cipher. Thus, according to Fact 3, Mendel et al. attack will have a lower memory complexity if $T < 2^{85.3}$ and if it relies on `MITM` $^{\texttt{NEW}}$ (rather than `MITM` $^{\texttt{STD}}$). (Considering that accessing D data requires some computational cost of about D, limiting $T > D$ is reasonable. Then the range of T becomes $2^{64} < T < 2^{85.3}$.) For instance, if $T = 2^{70}$, then $D = 2^{58}$, and thus Mendel et al. attack will require 2^{58} memory if it uses `MITM` $^{\texttt{STD}}$, and only 2^{12} memory if it relies on `MITM` $^{\texttt{NEW}}$. However, note that `MITM` $^{\texttt{NEW}}$ overweights `MITM` $^{\texttt{STD}}$ only if $D > 2^{42.7}$.

4.3 The Case of Reduced Domain

Let us apply `MITM` $^{\texttt{NEW}}$ to the case of a reduced domain. To do so, we focus on triple encryption $\overline{E_{k_1,k_2,k_3}}(P) = E_{k_3}(E_{k_2}(E_{k_1}(P))) = C$, where $E_k(P)$ is an

[12] We stress out that we are not showing a weakness of the TDR-mode, but a possible improvement in the memory requirement of the analysis that matches the proved security bound.

n-bit cipher with n-bit key k, and provide a key recovery given three pairs of known plaintext-ciphertext $(P_i, C_i), i = 1, 2, 3$.

First, let us reduce the key recovery to a collision search problem. For this purpose, we define two functions (below, $||$ denotes concatenation)

$$\begin{aligned} F(k_1, k_2) &= E_{k_2}(E_{k_1}(P_1))||E_{k_2}(E_{k_1}(P_2)), \\ G(k_3) &= E^{-1}_{k_3}(C_1)||E^{-1}_{k_3}(C_2) \end{aligned}$$

Obviously, $F : \{0, 1\}^{2n} \to \{0, 1\}^{2n}$ and $G : \{0, 1\}^n \to \{0, 1\}^{2n}$, that is, G *has a reduced domain.* A collision between F and G corresponds to a triplet of keys (k_1, k_2, k_3) such that $\overline{E_{k_1,k_2,k_3}}(P_1) = C_1$ and $\overline{E_{k_1,k_2,k_3}}(P_2) = C_2$. We need to produce 2^n such collisions to get the final $\overline{E_{k_1,k_2,k_3}}(P_3) = C_3$, as on average there is only a single triplet of keys that encrypts the three plaintexts P_1, P_2, P_3, into the three ciphertexts C_1, C_2, C_3.

To find a single collision on $2n$ bits, we use MITM $^{\text{NEW}}$ with $R = 2^{\frac{n}{2}}$. This value is chosen to avoid collisions of chains in the Hellman's table. Recall that chains have length $\sqrt{\frac{2^{2n}}{M}}$. This ensures that the matrix stopping rule is fulfilled for points on which F is evaluated: dimension of the domain of F is $2n$, each chain has at most $\sqrt{\frac{2^{2n}}{M}}$ evaluations of F and thus, $M \cdot \left(\sqrt{\frac{2^{2n}}{M}}\right)^2 \le 2^{2n}$. When $R = 2^{\frac{n}{2}}$, then each chain has $\sqrt{\frac{2^{2n}}{M}}/2^{\frac{n}{2}}$ evaluations of G, thus the matrix stopping rule for G (with domain of dimension n) is fulfilled as well because $M \cdot \left(\sqrt{\frac{2^{2n}}{M}}/2^{\frac{n}{2}}\right)^2 \le 2^n$. Therefore, the number of colliding chains is negligible. Note, as the ranges of the two functions are of dimensions $2n$ each, while the domain of G has a dimension of only n, when building the chains, we need to use a reduction function for the inputs of G, which can be defined simply as a truncation of the $2n$-bit value to n bits.

According to the tradeoff curve of MITM $^{\text{NEW}}$, we can produce a collision with complexities that follow $T^2M = R^2 2^{2n} = 2^{3n}$. The condition of the tradeoff dictates that $M \le R$, hence given a memory $M = 2^{\frac{n}{2}}$, we can get a collision in time $T_1 = 2^{\frac{5n}{4}}$. To get 2^n collisions we repeat 2^n times the whole collision search (rebuild a new Hellman's table with different reduction function). As a result, we can recover the whole $3n$-bit key in time $T = 2^{\frac{9n}{4}}$ and memory $M = 2^{\frac{n}{2}}$.

The standard MITM algorithm on triple encryption by Merkle and Hellman [?] follows $TM = 2^{3n}$, thus for $M = 2^{\frac{n}{2}}$ it requires time $T = 2^{\frac{5n}{2}}$, which is larger than our time. In addition, the dissection by Dinur et al.[?] used for attacks on multiple encryption, applies only when the number of encryption is at least four. Therefore, MITM $^{\text{NEW}}$ leads to the lowest time complexity attack on triple encryption with $M = 2^{\frac{n}{2}}$.

5 Conclusion

We have shown that one of the most common subproblems in cryptanalysis, the unbalanced meet-in-the-middle problem, can be solved with an alternative algorithm. The new algorithm relies on combination of two ideas: unbalanced interleaving and van Oorschot-Wiener multiple collision search. It follows the tradeoff $T^2M = R^2N$, where R is the ratio of costs of the two functions. It outperforms the standard algorithm (with the tradeoff $TM = N$) in terms of time when $MR^2 < N$, and in terms of memory when $T > R^2$ (in both of the cases, assume that $M \leq R$).

The new algorithm follows a more intuitive relation between the ratio R and the required memory M: the lower the ratio, the less memory is required. In fact, the complexity of the balanced collision search between two functions (solved with the Floyd's algorithm), can be described as a point of the tradeoff curve of the new algorithm ($R = 1, M = 1$ and thus $T^2 = N$). This is not the case with the standard algorithm ($M = 1$ will lead to $T = N$).

The new algorithm outperforms the standard algorithm in terms of time when $M \leq R$, $M \leq \sqrt{\frac{N}{R}}$ and $M < \frac{N}{R^2}$, and in terms of memory when $T \geq \sqrt{RN}$ and $T > R^2$.

In applications where minimizing the time complexity is the only concern, both the new and the standard algorithm behave the same ($T = \sqrt{RN}$). However, once the focus expands to memory as well as time, the new algorithm may provide significant advantage over the standard. As a general rule of the thumb, the new algorithm should be considered as the first choice in unbalanced meet-in-the-middle problems with $R < N^{\frac{1}{3}}$.

A Pseudo Code of Algorithms

Algorithm 1 Construction of table L_m

```
procedure CONSTRUCTL($f(x), n, m$)
    $L_m \leftarrow \emptyset$
    for $i$=1 to $2^m$ do
        $v_s \overset{\$}{\leftarrow} \{0,1\}^n$                          ▷ Generate random value
        $v_e \leftarrow v_s$
        for $j$=1 to $2^{\frac{n-m}{2}}$ do                        ▷ Iteratively apply $f(x)$
            $v_e \leftarrow f(v_e)$
        end for
        $L_m \leftarrow L_m \cup (v_s, v_e)$                      ▷ Store $(v_s, v_e)$ in $L_m$
    end for
    return $L_m$
end procedure
```

Algorithm 2 Find collision with L_m

```
procedure FINDCOLLISION($L_m, f(x), n, m$)
    $w_0 \overset{\$}{\leftarrow} \{0,1\}^n$                              ▷ Generate random value
    $w_i \leftarrow w_0$
    length $\leftarrow 0$
    do
        length $\leftarrow$ length $+ 1$
        $w_i \leftarrow f(w_i)$
        $v_s \leftarrow$ Find($L_m, w_i$)                        ▷ Check if $w_i$ is in $L_m$
    while $v_s = \emptyset$
    for $i$=1 to $2^{\frac{n-m}{2}}$ $-$ length do                  ▷ Align the two chains
        $v_s \leftarrow f(v_s)$
    end for
    while $f(v_s) \neq f(w_0)$ do                             ▷ Find the colliding pair
        $v_s \leftarrow f(v_s)$
        $w_0 \leftarrow f(w_0)$
    end while
    return $(v_s, w_0)$
end procedure
```

Algorithm 3 Definition of $F(x)$

```
procedure F($f, g, \rho$)
    if $x$ % $2^\rho = 0$ then                ▷ Least $\rho$ bits of $x$ are zeros
        return $g(x)$                          ▷ $F(x) = g(x)$
    else
        return $f(x)$                          ▷ $F(x) = f(x)$
    end if
end procedure
```

Algorithm 4 New MITM Algorithm

```
procedure MITM($n, m, \rho$)
    $L_m \leftarrow$ ConstructL(F($f, g, \rho$), $n, m$)
    do
        $(a, b) \leftarrow$ FindCollision($L_m$, F($f, g, \rho$), $n, m$))
    while $a$ % $2^\rho > 0$ and $b$ % $2^\rho > 0$
    return $(a, b)$
end procedure
```

B Graphical Comparison of the Tradeoffs

A comparison of our tradeoff $T^2M = R^2N$ to the standard tradeoff $TM = N$ is given in Figures 3,4, and 5.

In Fig. 3, we can see that as long as $R < 2^{\frac{n}{2}}$, there is a range of values of M, where the time of `MITM` $^{\texttt{NEW}}$ is lower than the time of `MITM` $^{\texttt{STD}}$. For `MITM` $^{\texttt{NEW}}$,

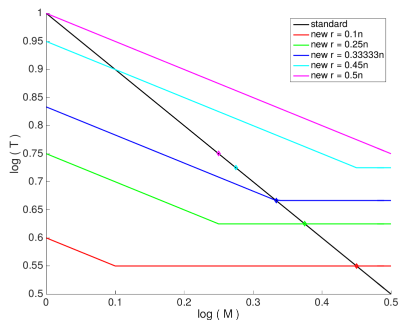

Fig. 3. Dependency of time on memory between `MITM` $^{\texttt{STD}}$ (in black) and `MITM` $^{\texttt{NEW}}$ (coloured), when $2^{\frac{n}{10}} \leq R \leq 2^{\frac{n}{2}}$.

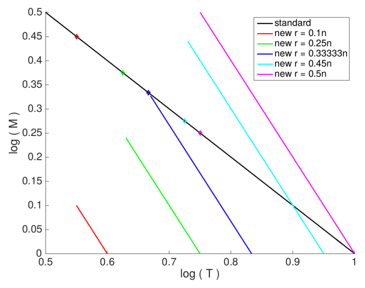

Fig. 4. Dependency of memory on time between `MITM` $^{\texttt{STD}}$ (in black) and `MITM` $^{\texttt{NEW}}$ (coloured), when $2^{\frac{n}{10}} \leq R \leq 2^{\frac{n}{2}}$.

when $M > R$, the time remains the same as for the point $M = R$ (recall the tradeoff is valid as long as $M \leq R$). Note, a similar is true for `MITM` $^{\texttt{STD}}$ and is denoted with coloured dots on the black line. For instance, when $R = 2^{\frac{n}{4}}$

(denoted in green), `MITM` $^{\texttt{STD}}$ is valid as long as $M \leq 2^{\frac{3n}{8}}$. For larger values of M, the standard tradeoff does not actually follow the black line (the time does not reduce), but the time remains the same as in the point $M = 2^{\frac{3n}{8}}$.

Similarly, in Fig. 4, we can see a range of values T for which `MITM` $^{\texttt{NEW}}$ outperforms `MITM` $^{\texttt{STD}}$ in terms of memory. Note, both of the algorithms require minimal time of $T = \sqrt{RN}$. Therefore, the lines start from the point $T = \sqrt{RN}$.

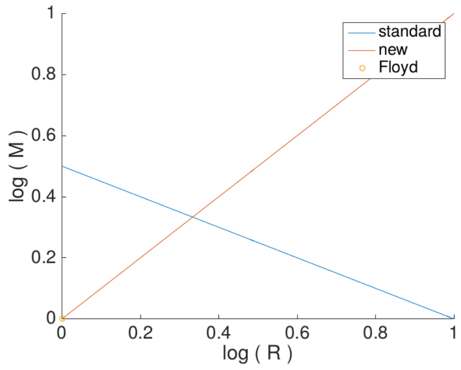

Fig. 5. Dependency of the memory on the ratio, when the time is minimal ($T = \sqrt{RN}$), between `MITM` $^{\texttt{NEW}}$ (in red) and `MITM` $^{\texttt{STD}}$ (in blue).

Finally, in Fig. 5, we show the dependency on the memory of the comparative cost of the two functions, when the time is set to minimal, that is, when $T = \sqrt{RN}$. When $R = 1$, then the Floyd's algorithm requires no memory to find the collision (denoted with a yellow circle at the point (0,0)). However, once $R > 1$, the memory requirement of `MITM` $^{\texttt{STD}}$ immediately jumps to almost $2^{\frac{n}{2}}$ (in blue), whereas the memory of `MITM` $^{\texttt{NEW}}$ increases gradually (in red).