Forgery and Partial Key-Recovery Attacks on HMAC and NMAC Using Hash Collisions (Extended Version)

Scott Contini[1] and Yiqun Lisa Yin[2]

[1] Macquarie University, Centre for Advanced Computing – ACAC, NSW 2109, Australia
scontini@comp.mq.edu.au
[2] Independent Consultant, Greenwich CT, USA
yiqun@alum.mit.edu

Abstract. In this paper, we analyze the security of HMAC and NMAC, both of which are *hash-based* message authentication codes. We present *distinguishing, forgery, and partial key recovery attacks* on HMAC and NMAC using collisions of MD4, MD5, SHA-0, and reduced SHA-1. Our results demonstrate that the strength of a cryptographic scheme can be greatly weakened by the insecurity of the underlying hash function.

1 Introduction

Many cryptographic schemes use hash functions as a primitive. Various assumptions are made on the underlying hash function in order to prove the security of the scheme. For example, some proofs assume that the hash function behaves as a random oracle, while other proofs only assume collision resistance. With the continuing development in hash function research, especially several popular ones are no longer secure against collision attacks, a natural question is whether these attacks would have any impact on the security of existing *hash-based* cryptographic schemes.

In this paper, we focus our study on HMAC and NMAC, which are hash-based message authentication codes proposed by Bellare, Canetti and Krawczyk [2]. HMAC has been implemented in widely used security protocols including SSL, TLS, SSH, and IPsec. NMAC, although less known in the practical world, is the theoretical foundation of HMAC — existing security proofs [2, 1] were first given for NMAC and then extended to HMAC. It is commonly believed that the two schemes have identical security.

The constructions of HMAC and NMAC are based on a *keyed hash function* $F_k(m) = F(k, m)$, in which the IV of F is replaced with a secret key k. NMAC has the following nested structure: $\mathtt{NMAC}_{(k_1,k_2)}(m) = F_{k_1}(F_{k_2}(m))$, where $k = (k_1, k_2)$ is a pair of secret keys. HMAC is similar to NMAC, except that the key pair (k_1, k_2) is derived from a *single* secret key using the hash function. Hence, we can view HMAC as NMAC plus a key derivation function.

The security of HMAC and NMAC was carefully analyzed by its designers [2]. They showed that NMAC is a pseudorandom function family (PRF) under the two assumptions that (A1) the keyed compression function f_k of the hash function is a PRF, and (A2) the keyed hash function F_k is *weakly collision resistant*[3]. The proof for NMAC was then lifted to HMAC by further assuming that (A3) the key derivation function in HMAC is a PRF. The provable security of HMAC, besides its efficiency and elegancy, was an important factor for its wide deployment. However, recent collision attacks on hash functions [22, 25] imply that assumption (A2) in the original proof no longer holds when considering concrete constructions such as HMAC-MD5 and HMAC-SHA1. To fix this problem, Bellare recently showed [1] that NMAC is a PRF under the sole assumption that the keyed compression function f_k is a PRF. This implies that the security of HMAC now depends only on assumptions (A1) and (A3). The main advantage of the new analysis is that the proof assumptions do not seem to be refuted by existing attacks on hash functions.

The new security proofs are quite satisfying, especially since they are based on relatively weak assumptions of the underlying hash function. On the other hand, they have also raised interesting questions as whether the proof assumptions indeed hold for popular hash functions. In particular, does any existing collision attack on a hash function compromise the PRF assumption? And if so, does it lead to possible attacks on HMAC and NMAC?

1.1 Summary of main results

In this paper, we analyze the security of HMAC and NMAC. We answer the aforementioned questions in the affirmative by constructing various attacks on HMAC and NMAC based upon weaknesses of the underlying hash function.

Our analysis is based upon existing analyses of hash functions, especially the attacks on MD4, MD5, SHA-0, and reduced SHA-1 presented in [26, 9, 10, 7]. We first show that the *collision differential path* in these earlier attacks can be used to construct *distinguishing attacks* on the keyed compression function f_k. Hence, for MD4, MD5[4], SHA-0, and reduced SHA-1, f_k is *not* a PRF.

Building upon the above attacks, we show how to construct distinguishing, forgery, and partial key recovery attacks on HMAC and NMAC when the underlying hash functions are MD4, MD5, SHA-0, and reduced SHA-1. The complexity of our attacks is closely related to the total probability of the collision differential path, and in some cases it is less than the $2^{n/2}$ generic bound for birthday-type attacks. A summary of our main results is given in Table 1.

We remark that in our key recovery attack the adversary can retrieve the entire inner key k_2. This can greatly weaken the security of the scheme. In particular, when the *keyed* inner function is degraded to a hash function with a *known* IV, further attacks such as single-block forgeries become possible.

[3] Please refer to Section 3 for precise definitions of f_k and F_k. The notion of weakly collision resistant (WCR) was introduced in [2]. Roughly, F_k is WCR if it is computationally infeasible to find $m \neq m'$ s.t. $F_k(m) = F_k(m')$ for hidden k.

[4] In the case of MD5, f_k is not a PRF under *related-key attacks*.

Table 1. Result summary: number of queries in our attacks on HMAC/NMAC.

	hash function	distinguish & forgery attacks	key recovery attacks	comments
HMAC/NMAC	MD4	2^{58}	2^{63}	
NMAC	MD5	2^{47}	2^{47}	related-key attacks
HMAC/NMAC	SHA-0	2^{84}	2^{84}	
HMAC/NMAC	reduced SHA-1	2^{34}	2^{34}	inner function is 34 rounds

1.2 Use of hash collisions in our attacks

Our attacks on HMAC and NMAC are based on collisions of the keyed inner function F_{k_2}. The main reason that an adversary can observe such collisions is that in our scenario the outer function F_{k_1}, although hiding the output of the inner function, does not *hide* the occurrence of an inner collision.

In our key recovery attacks, each bit of collision information – whether or not a collision occurs from a set of properly chosen messages – roughly reveals one bit of the inner key. This is due to the fact that a collision holds information about the entire hash computation, and hence the secret key. Our techniques illustrate that collisions within a hash function can potentially be very dangerous to the security of the upper-layer cryptographic scheme.

1.3 Other results

`General framework for analyzing HMAC and NMAC.` We extend the approach in our attacks to provide a general framework for analyzing HMAC and NMAC. This framework also points to possible directions for hash function attacks that most likely lead to further improved attacks on HMAC and NMAC.

`Attacks on key derivation in HMAC-MD5.` We study the key derivation function in HMAC-MD5, which is essentially the MD5 compression function keyed through the *message input*. We describe distinguishing and second preimage attacks on the function with complexity much less than the theoretical bound.

`New modification technique.` We develop a new message modification technique in our key recovery analysis. In contrast with Wang's techniques [22, 23], our method does not require full knowledge of the internal hash computation process. We believe that our new technique may have other applications.

1.4 Implications

In practice, HMAC is mostly implemented with MD5 or SHA-1. To a much lesser extent, there is some deployment of HMAC-MD4 (for example, see [12]). We are not aware of any deployment of NMAC. The attacks presented in this paper do not imply any immediate practical threat to implementations of HMAC-MD5 or

HMAC-SHA1. However, our attacks on HMAC-MD4 may not be out of range of some adversaries, and therefore it should no longer be used in practice.

We emphasize that our results on HMAC complement, rather than contradict, the analysis in [2, 1]. While the designers proved that HMAC is secure under certain assumptions on the underlying hash function, we show that attacks are possible when these assumptions do not hold.

1.5 Organization of the paper

In Section 3, we provide brief descriptions of HMAC, NMAC and the MDx family. In Section 5, we present all three types of attacks on NMAC-MD5, which is based on the MD5 pseudo-collision (Section 4). The simplicity of the underlying differential path in this case facilitates our explanation, especially the technical details of our key recovery attack. For attacks on HMAC and NMAC using other underlying hash functions, the methods are similar and thus we just focus on what is different in each case in Section 6. In Section 7, we describe a general framework for analyzing HMAC and NMAC.

2 Related work

Our analysis on HMAC and NMAC is closely related to various attacks on hash functions, especially those in the MDx family. In addition, our work is also related to the rich literature on message authentication codes. Many early heuristic designs for MACs were broken, sometimes in ways that allowed forgery and key recovery [17–19]. These early analyses were the driving force behind proposals with formal security proofs, namely HMAC and NMAC [2]. Since their publication, most of the security analysis was provided by the designers. Recently, Coron *et al.* [11] studied the security of HMAC and NMAC in the setting of constructing iterative hash functions. After our submission to Asiacrypt'06, we learned that Kim *et al.* [15] did independent work on distinguishing and forgery attacks on HMAC and NMAC when the underlying functions are MD4, SHA-0, and reduced SHA-1. They did not consider key recovery attacks. Also recently, Rechberger and Rijmen [20] improved upon their results on HMAC-SHA1.

Some of our attacks are in the related-key setting. Related-key attacks were introduced by Biham [5] and Knudsen [14] to analyze block ciphers. A theoretical treatment of related-key attacks was given by Bellare and Kohno [4]. The relevance of related-key cryptanalysis is debated in the cryptographic community. For example, some suggest that the attacks are only practical in poorly implemented protocols. On the other hand, cryptographic primitives that resist such attacks are certainly more robust, and vulnerabilities can sometimes indicate weaknesses in the design. See the introduction to [13] for example settings in which related-key attacks can be applied. We note that the designers of HMAC and NMAC did not consider the related key setting in their security analysis.

3 Preliminaries

3.1 Hash functions and the MDx family

A cryptographic hash function is a mathematical transformation that takes an input message of arbitrary length and produces an output of fixed length, called the *hash value*. Formal treatment of cryptographic hash functions and their properties can be found in [21]. In practice, hash functions are constructed by iterating a *compression function* $f(cv, x)$ which takes fixed length inputs: a chaining variable cv of n bits and a message block x of b bits. The hash function F is defined as follows: First divide the input message m into $x_1, x_2, ..., x_s$ according to some preprocessing specification, where each x_i is of length b. Then set the first chaining variable cv_0 as the fixed IV, and compute $cv_i = f(cv_{i-1}, x_i)$ for $i = 1, 2, ..., s$. The final output cv_s of the iteration is the value of F.

The MDx family of hash functions includes MD4, MD5, SHA-0, SHA-1, and others with similar structure. Here we briefly describe the structure of MD5 and omit others. The compression function of MD5 takes a 128-bit chaining variable and a 512-bit message block. The chaining variable is split into four registers (A, B, C, D), and the message block is split into 16 message words $m_0, \ldots, m_{15}$. The compression function consists of 4 rounds of 16 steps each, for a total of 64 steps. In each step, the registers are updated according to one of the message words. The initial registers (A_0, B_0, C_0, D_0) are set to be some fixed IV. Each step t $(0 \leq t < 64)$ has the following general form[5]:

$$X_t \leftarrow (A_t + \phi(B_t, C_t, D_t) + w_t + K_t) \lll s_t$$
$$(A_{t+1}, B_{t+1}, C_{t+1}, D_{t+1}) \leftarrow (D_t, X_t + B_t, B_t, C_t)$$

In the above equation, ϕ is a round-dependent Boolean function, K_t is a step-dependent constant, and s_t is a step-dependent rotation amount. In each round, all 16 message words are applied in a different order, and so w_t is one of the 16 message words. After the 64 steps, the final output is computed as $(A_{64} + A_0, B_{64} + B_0, C_{64} + C_0, D_{64} + D_0)$.

3.2 Message authentication codes, HMAC and NMAC

A message authentication code is a mathematical transformation that takes as inputs a message and a secret key and produces an output called *authentication tag*. The most common attack on MACs is a *forgery attack*, in which the adversary can produce a valid message/tag pair without knowing the secret key. For MACs that are based on *iterative* hash functions, there is a birthday-type forgery attack [17, 3] that requires about $2^{n/2}$ MAC queries, where n is the length of the authentication tag.

HMAC and NMAC are both hash-based MACs. Let F be the underlying hash function and f be the compression function. The basic design approach

[5] We use a slightly different notation from previous work so that there is a unified description for all the steps.

for NMAC is to replace the fixed IV in F with a secret key (aka keyed via the IV). Following the notation in [2], we use $f_k(x) = f(k, x)$ to denote the keyed compression function and $F_k(x) = F(k, x)$ the keyed hash function. Let (k_1, k_2) be a pair of independent keys. The NMAC function, on input message m and secret key (k_1, k_2), is defined as:

$$\texttt{NMAC}_{(k_1,k_2)}(m) = F_{k_1}(F_{k_2}(m)).$$

The construction of HMAC was motivated by practical implementation needs. Since NMAC changes the fixed IV in F into a secret key, this requires a modification of existing implementations of the hash function. To avoid this problem, the designers introduced the fixed-IV variant HMAC. Let $\texttt{const}_1$ and $\texttt{const}_2$ be two fixed constants. The HMAC function, on input message m and a single secret key k, is defined as:

$$k_1 = f(IV, k \oplus \texttt{const}_1) \qquad (1)$$

$$k_2 = f(IV, k \oplus \texttt{const}_2) \qquad (2)$$

$$\texttt{HMAC}_k(m) = \texttt{NMAC}_{(k_1,k_2)}(m).$$

In the above description for HMAC, we can consider Equations (1) and (2) together as a key derivation function KDF which takes a single secret key k and outputs a pair of keys (k_1, k_2). That is, $(k_1, k_2) = \texttt{KDF}(k)$. Hence, HMAC is essentially "KDF + NMAC". We remark that the term "key derivation function" was not used in [2], but this view of the HMAC construction will be quite convenient for our later analysis.

4 Pseudo-collisions of MD5

In [9], den Boer and Bosselaers analyzed the compression function of MD5 and found pseudo-collisions of the form $f(cv, m) = f(cv', m)$, where cv and cv' are two different IVs. Such pseudo-collisions of MD5 are the basis for our related-key attacks on NMAC-MD5. In this section, we discuss some properties of the pseudo-collisions under the framework of differential cryptanalysis.

Differential cryptanalysis was introduced by Biham and Shamir [8] to analyze the security of DES. The idea also applies to the analysis of hash functions. In a hash collision attack, we consider input pairs with an appropriately defined difference and analyze how the differences in the chaining variables evolve during the hash computation. The intermediate differences collectively are called a *differential path*, and its probability is defined to be the probability that the path holds when averaged over all input pairs satisfying the given difference.

For the MD5 pseudo-collisions in [9], the messages are the same and the input difference is only in the chaining variables. The pair of initial chaining variables (cv, cv') as well as all the intermediate values satisfy the following difference:

$$cv \oplus cv' = (\texttt{ 80000000 80000000 80000000 80000000 }) \stackrel{\text{def}}{=} \Delta^{\texttt{msb}}. \qquad (3)$$

Putting in concrete terms, the differences are only in the most significant bit (MSB) of each register A_t, B_t, C_t, D_t. This simple pattern propagates through all 64 steps of MD5. Because of the extra addition operation at the end, the difference disappears, yielding a pseudo-collision.

The differential path requires the following conditions on the IV:

$$\mathtt{MSB}(B_0) = \mathtt{MSB}(C_0) = \mathtt{MSB}(D_0) = b, \tag{4}$$

where $b = 0$ or 1. Moreover, the MSBs of the intermediate registers are the same for most of the first round. Namely, for $1 \leq t < 15$,

$$\mathtt{MSB}(A_t) = \mathtt{MSB}(B_t) = \mathtt{MSB}(C_t) = \mathtt{MSB}(D_t) = b.$$

The total probability of the differential is 2^{-46}.

5 Related-key attacks on NMAC-MD5

In this section, we present distinguishing, forgery, and partial key recovery attacks on NMAC-MD5 in the related-key setting. In this setting, the goal of the adversary is to break the MAC by obtaining input/output pairs of two MAC oracles whose keys are different but with a known relation.

As described in Section 4, the differential path for the MD5 pseudo-collision holds with probability 2^{-46}. Given the path, we can construct a related-key distinguishing attack on the keyed MD5 compression function that requires about 2^{47} queries. This distinguishing attack is the basis for all three types of attacks on NMAC-MD5.

Recall that in NMAC, the inner function F_{k_2} is keyed through the IV. Hence, in our related-key attacks, the difference in the *inner key* k_2 is set according to the input IV difference given by Equation (3). More specifically, we have the following setting for our related-key attacks on NMAC-MD5:

- There are two oracles $\mathtt{NMAC}_{(k_1,k_2)}$ and $\mathtt{NMAC}_{(k'_1,k'_2)}$. The relation between (k_1, k_2) and (k'_1, k'_2) is set as:

$$k_1 = k'_1 \quad \text{and} \quad k_2 \oplus k'_2 = \Delta^{\mathtt{msb}}. \tag{5}$$

- The adversary queries each oracle on input messages of its choice and is given the corresponding authentication tag.

5.1 Related-key distinguishing attack on keyed MD5 compression function

We present a related-key distinguishing attack on keyed MD5 compression based on pseudo collisions of MD5. Let f_k and $f_{k'}$ be two keyed MD5 compression functions such that $k \oplus k' = \Delta^{\mathtt{msb}}$. The adversary is given two oracles (O, O'), which can be either oracles for $(f_k, f_{k'})$ or oracles for a truly random function. In the attack, the adversary generates 2^{46} random messages and queries the two oracles. If a collision $O(m) = O'(m)$ is observed for any message m, it identifies the oracles as MD5 compression; otherwise, it identifies them as a truly random function.

The complexity of the above distinguishing attack is 2^{46} random queries to each oracle, for a total of 2^{47} queries. The attack succeeds if k satisfies the condition given by Equation (4), and so the success probability is 1/4. This attack shows that the keyed MD5 compression function is *not* pseudorandom against related-key attacks.

5.2 Related-key distinguishing attack on NMAC-MD5

The above distinguishing attack on MD5 compression can be easily converted into a distinguishing attack on NMAC-MD5. The adversary is given two oracles (O, O'), which can either be the two NMAC oracles as defined by Equation (5) or oracles for a truly random function. In the attack, the adversary generates 2^{46} random messages and queries the two oracles. If the adversary observes a collision $O(m) = O'(m)$ for any of the messages, it identifies the oracles as NMAC; otherwise, it identifies them as a truly random function.

The correctness of the attack is easy to see: After 2^{46} messages, a collision of the inner function is expected. That is, $F_{k_2}(m) = F_{k_2'}(m)$. Since the outer key k_1 is the same, the inner collision yields a collision for the two NMAC oracles. The complexity of the distinguishing attack is 2^{46} random queries to each oracle, for a total of 2^{47} queries. The attack succeeds if k_2 satisfies the condition given by Equation (4), and so the success probability is 1/4.

It is worth noticing that the outer function in NMAC, although making the output of the inner function hidden, does not *hide* the occurrence of an inner collision. This property is very useful for converting the distinguishing attack on the inner function (which is keyed MD5 compression) to a distinguishing attack on NMAC. Such a conversion also applies to HMAC.

5.3 Related-key MAC forgery attack on NMAC-MD5

The attack can be extended to a forgery attack as follows [17, 3]: Once a message m is found that causes a collision of the two NMAC oracles, the adversary queries the *first* oracle on $m||e$ for any extension e and obtains $tag = \mathtt{NMAC}_{(k_1,k_2)}(m||e)$. Then, it produces $(m||e, tag)$ as a forgery for the *second* oracle. Since $\mathtt{NMAC}_{(k_1,k_2)}(m||e) = \mathtt{NMAC}_{(k_1',k_2')}(m||e)$, the forged authentication tag is valid. The complexity is 2^{47} random queries plus one *chosen* query. Hence, the total number of queries is about 2^{47} and the success probability is 1/4.

5.4 Related-key key recovery attack on NMAC-MD5

We present a partial key recovery attack on NMAC-MD5, in which the adversary can retrieve the entire inner key k_2 in NMAC. This is the most technical part of the paper, so we start with a high level description of the key recovery algorithm consisting of four phases:

- **Phase 1.** The attacker generates random messages until it obtains a message m that causes a collision of the two NMAC oracles.
- **Phase 2.** The attacker modifies certain bits of m to create new messages m^* and observes whether any m^* causes a new collision. This collision information allows the attacker to recover many bits in the intermediate registers $S = (A_{14}, B_{14}, C_{14}, D_{14})$ in the computation of $F_{k_2}(m)$.
- **Phase 3.** Similar to Phase 2, the attacker recovers a few additional bits from other registers, and uses this information to determine more bits of S with a possible small additive error.
- **Phase 4.** The attacker guesses all remaining unknown bits of S and steps through the MD5 computation backwards to get (A_0, B_0, C_0, D_0) – a candidate for k_2. It verifies whether $F_{k_2}(m) = F_{k'_2}(m)$. If so, it outputs k_2 as the inner key; Otherwise, go back to Phase 1.

Phase 1 and Phase 4 of the key recovery algorithm are fairly straightforward, and so for the rest of the section we focus on Phase 2 and Phase 3. We first explain the main idea and then present detailed analysis.

Main idea For Phase 2 and Phase 3, the objective is to recover bits of some intermediate registers through collision information. To achieve this goal, we take a closer look at the collision differential paths and analyze what information can be derived from such paths. Let $\mathtt{DP}_m$ denote the differential path induced by m, i.e., all the intermediate differences in the computation of $F_{k_2}(m)$ and $F_{k'_2}(m)$. Since m yields a collision, we know that $\mathtt{DP}_m$ follows the differential path for the MD5 pseudo-collision. In particular, for the computation of $F_{k_2}(m)$, we have $\mathtt{MSB}(B_t) = b$ for $1 \le t < 15$. WLOG, we assume $b = 0$.

For a given step t in the first round, we introduce a new message m^* that is defined based on message m as follows:

$$m_j^* = \begin{cases} m_j & \text{if } 0 \le j < t \\ m_j + \Delta & \text{if } j = t \\ \text{random} & \text{if } t < j < 16 \end{cases} \qquad (6)$$

We next consider the differential path $\mathtt{DP}_{m^*}$, induced by m^*. Since m and m^* are the same up to Step $t-1$, the two paths $\mathtt{DP}_m$ and $\mathtt{DP}_{m^*}$ are the same until this step. For Step t, let B_{t+1}^* be the newly computed register by replacing m_t with $m_t^* = m_t + \Delta$. We know that B_{t+1}^* will be different from B_{t+1}. A key observation is that if $\mathtt{MSB}(B_{t+1}^*)$ changes from 0 to 1, then the path $\mathtt{DP}_{m^*}$ will *drift away* from the collision differential path, and hence the chance of it producing a collision after 64 steps is negligible. More precisely, we have the following lemma.

Lemma 1 *Let m^* be a message defined as in Equation (6), and let p^* be the probability that m^* causes a collision $F_{k_2}(m^*) = F_{k'_2}(m^*)$. If $\mathtt{MSB}(B_{t+1}^*) = 0$, then $p^* = 2^{t-45}$ when averaged over all random m_j^* $(j > t)$. If $\mathtt{MSB}(B_{t+1}^*) = 1$, then $p^* \approx 2^{-128}$.*

For a given value Δ, Lemma 1 can be used to detect the MSB of B^*_{t+1} as follows: generate about 2^{45-t} messages satisfying Equation (6) and query both NMAC oracles on these messages. If a collision is observed, then the MSB of B^*_{t+1} is 0; otherwise, the bit is 1.

In what follows, we show how to use the above collision information to recover B_{t+1}. To better illustrate the intuition, we consider a simplified step function where the rotate is *eliminated*. Hence Step t becomes $B_{t+1} = m_t + T$ and $B^*_{t+1} = m^*_t + T$, where the value T has been determined before Step t. To detect bit i of B_{t+1}, we set $m^*_t = m_t + 2^i$. This implies that

$$B^*_{t+1} = B_{t+1} + 2^i. \tag{7}$$

We consider the effect of the above increment, depending on whether bit i of B_{t+1} is 0 or 1:

- If bit i of B_{t+1} is 0, then the increment will not cause a carry. In this case, $\mathtt{MSB}(B^*_{t+1}) = \mathtt{MSB}(B_{t+1}) = 0$, and we will observe a collision in the expected number of queries.
- If bit i of B_{t+1} is 1, then the increment causes a carry. Furthermore, if we can set bits $[(i+1)..30]$ of B^*_{t+1} to be all 1, then the carry will go all the way to the MSB of B^*_{t+1} . In this case, $\mathtt{MSB}(B^*_{t+1}) = \mathtt{MSB}(B_{t+1}) + 1 = 1$, and we will *not* observe a collision.

  To ensure carry propagates to the MSB, we set $m^*_t = m_t + 2^i + d$, for an appropriate choice of d. So Equation (7) becomes $B^*_{t+1} = B_{t+1} + 2^i + d$.

The above analysis yields an algorithm for determining B_{t+1} one bit at a time, from bit 30 to bit 0. (Note that we already know bit 31 of B_{t+1} is 0 by assumption.) We refer to this algorithm as the *bit flipping algorithm*, and the complete description is given in Appendix A.

Detailed analysis The main idea described above generally applies to any register B_t for $0 \le t < 15$. In Phase 2, the registers to be recovered are

$$(B_{11}, B_{12}, B_{13}, B_{14}) = (A_{14}, D_{14}, C_{14}, B_{14}).$$

The reason why we choose later registers rather than earlier ones is to minimize the number of oracle queries, which is 2^{45-t} per oracle per bit computed of register B_{t+1}. We leave B_{15}, B_{16} free so that there is enough randomness for generating new collisions.

We now consider how to apply the bit flipping algorithm in the presence of rotation. We need to do $B^*_{t+1} = B_{t+1} + 2^i + d$ for $i = 30, 29, \ldots, 0$. However, we are not able to do so by just setting $m^*_t = m_t + 2^i + d$ because of the rotation operation $\lll s_t$. Instead, we use a *modified* bit flipping algorithm (see Appendix A for details). In this algorithm, we set $m^*_t = m_t + 2^{i'} + d'$ where

$$i' + s_t = i \bmod 32 \qquad \text{and} \qquad d' \lll s_t = d.$$

Note that if addition and rotation could commute, then setting m_t^* as above would have the same effect as $B_{t+1}^* = B_{t+1} + 2^i + d$. Since this is not the case, some error might occur when applying the modified algorithm. Fortunately, the error is manageable — we can show that the modified algorithm almost always succeeds for recovering the most significant $(32-s_t)$ bits of B_{t+1}. In other words, if it fails, it is almost always on the least significant s_t bits. More precisely, we have the following lemma which is proved in Appendix A.1.

Lemma 2 *For step t, let p_t be the probability that the modified bit flipping algorithm correctly recovers the most significant $(32-s_t)$ bits of B_{t+1}, when averaged over all possible input messages m. Then $p_t \geq 1 - 2^{-s_t} - 2^{-s_t-1}$.*

For the four steps $t = 10, 11, 12, 13$, the rotation amounts are $s_t = 17, 22, 7, 12$. Hence, we can use the modified bit flipping algorithm to determine the following bits of the registers:

$$
\begin{aligned}
A_{14} = B_{11} &: \text{ most significant 15 bits} \\
D_{14} = B_{12} &: \text{ most significant 10 bits} \\
C_{14} = B_{13} &: \text{ most significant 25 bits} \\
B_{14} = B_{14} &: \text{ most significant 20 bits}
\end{aligned}
$$

In total we already recover 70 bits of the registers. We could proceed to Phase 4 and guess the remaining 58 bits. This would yield a key recovery algorithm with query complexity 2^{47} and time complexity equal to about 2^{58} MD5 operations, which is much less than exhaustive key search. Instead, with refined analysis we can reduce the workload complexity further by doing an insignificant number of additional queries in Phase 3, which is described in Appendix A.2.

Our analysis shows that the total number of queries of the algorithm is dominated by that of Phase 1, which is 2^{47}. The computing time is less than 2^{45} MD5 operations. Detailed calculations are given in Appendix A.3. Similar to the distinguishing and forgery attack, the key recovery attack succeeds with probability 1/4.

Implementation results We have implemented the key recovery attack on NMAC-MD5. In our implementation, we used a reduced-round version of MD5, in which the last round (16 steps) is omitted. Since the attack only depends on properties of the first round, the reduction in rounds does not affect the analysis except that the query complexity is reduced from 2^{47} to 2^{31}. In our experiment, the algorithm correctly recovered the inner key bits.

Remarks on message modification techniques In the key recovery analysis, we use information about the collision differential paths to derive information about the intermediate registers. To generate useful paths, we developed a new message modification technique that works even when the internal hash computation is unknown due to the presence of the *secret key.*

It is worth comparing our modification techniques with Wang's original message modification techniques [22, 23], which deals with the situation where the entire hash computation is known since there is no secret for a keyless hash function. Note that the objective of the modification is also different for collision attacks and our key recovery attacks: the goal for the former is to modify messages so that collisions can occur with high probability; the goal for the latter is to modify messages so that certain collisions may or may *not* occur, depending upon the value of the secret key.

5.5 Attacks on the KDF in HMAC-MD5

Given our related-key attacks on NMAC-MD5, an immediate question is whether they are applicable to HMAC-MD5. Since the difference between HMAC and NMAC is the extra key derivation function `KDF`, we analyze properties of `KDF` in HMAC-MD5, which consists of two functions of the form $k_i = f(IV, k \oplus \mathtt{const_i})$. Here the MD5 compression function f is used as $f(x, K)$, where $x \in \{0,1\}^{128}$ and the key $K \in \{0,1\}^{512}$. For ease of reference, we denote $f(x, K)$ by $g_K(x)$. So $\{g_K\}_{K \in \{0,1\}^{512}}$ is a family of functions indexed by K.

As noted in Section 5.4 of [1], Rijmen observed that it seems possible to extend the pseudo-collision of MD5 [9] to a distinguishing attack on $\{g_K\}$. Here, we describe the details of such an attack: The adversary generates 2^{46} random pairs (x, x') such that $x \oplus x' = \Delta^{\mathtt{msb}}$, and queries an oracle, which is either g_K or a truly random function. If the adversary observes a collision for any pair, then it identifies the oracle as g_K; otherwise, it identifies the oracle as a truly random function. The complexity of the attack is 2^{47} queries.

Recall that the HMAC security proofs [1, 2] require `KDF` to be a PRF. However, the above distinguishing attack implies that the `KDF` in HMAC-MD5 is *not* a PRF. Despite the non-pseudorandomness, its presence does help HMAC-MD5 to resist our related-key attacks for the following reason. In order to apply the attacks to HMAC-MD5, we would need to set appropriate differences in the single key k and hope that $(k_1, k_2) = \mathtt{KDF}(k)$ would yield the required difference for k_2 while keeping k_1 the same (see Equation (5)). However, this appears to be very difficult, since any differences in k would almost certainly cause differences in both k_1 and k_2, thus making the attacks impossible.

Of independent interest, we present a *second preimage attack* on g_K, also based on [9]. Here the key K can be either secret or known. The attack works as follows: For a given random input $x \in \{0,1\}^{128}$, the adversary sets x' such that $x \oplus x' = \Delta^{\mathtt{msb}}$, and outputs x' as a second preimage of x. The success probability is about 2^{-48}, since the probability that x satisfies Equation (4) is 2^{-2}, and the probability that the pair (x, x') then follows the differential path to produce a collision is 2^{-46} (meaning x' is a second preimage of x). Hence, the above attack requires $O(1)$ workload, no queries, and succeeds with probability 2^{-48}, which is much higher than the 2^{-128} theoretical bound.

6 Attacks on HMAC/NMAC with other hash functions

The basis for our attacks on NMAC-MD5 is a collision differential path for the keyed MD5 compression function that holds with *relatively large* probability. The same ideas and techniques also apply to other underlying hash functions such as MD4, SHA-0, and reduced SHA-1. In this section, we present three types of attacks on HMAC and NMAC for these underlying hash functions, all in the *standard* setting.

6.1 Attacks on HMAC/NMAC-MD4

MD4 has long been known to be insecure, but it was an open question whether HMAC-MD4 can still be used as a PRF or a secure MAC. We answer the question in the negative by presenting attacks on HMAC/NMAC-MD4.

Our attacks are based upon the second preimage attack on MD4 by Yu *et al.* [26]. Table 3 of [26] gives a differential path that leads to a collision with probability 2^{-62}. The details that are most relevant to our attacks are the message difference: there is only a one-bit difference in one of the message words, namely, $m_4 \oplus m'_4 = 2^i$, and the path holds for any i $(0 \leq i < 32)$, for a total of 32 possible paths.

Distinguishing attack on keyed MD4 compression Using any of the 32 differential paths, we can mount a distinguishing attack on the keyed MD4 compression function with about 2^{63} queries. This implies that keyed MD4 compression is *not* a PRF.

Distinguishing and forgery attacks on HMAC-MD4 In the distinguishing attack, there is a single oracle O, which can be either $\texttt{HMAC}_k$ or a truly random function. The adversary generates about 2^{62} message pairs (m, m') such that $m_4 \oplus m'_4 = 2^i$ for some i, queries the oracle, and observes whether a collision $O(m) = O(m')$ occurs. If so, it identifies the oracle as HMAC; otherwise, it identifies it as a truly random function. The expected query complexity is 2^{63}, and the success probability is one. From the collision, a forgery attack easily follows (similar to Section 5.1) which requires an additional chosen query.

We can reduce the query complexity to 2^{58} by using a structure, which is a common trick in differential cryptanalysis. The idea is to take advantage of the multiple differential paths by generating input pairs (m, m') in a more compact way as follows: First, generate 2^{26} random m_3 (it can actually be any message word m_j as long as $j \neq 4$). Second, for each m_3, generate all 2^{32} possible values for m_4. Hence, the total number of messages is 2^{58}. It is easy to show that the 2^{58} messages collectively create 2^{62} pairs of (m, m') for which $m_4 \oplus m'_4 = 2^i$ for some i. One of the pairs is expected to produce a collision.

Partial key recovery attack on HMAC-MD4 We can construct a partial key recovery attack on HMAC-MD4 following similar phases as that of NMAC-MD5. Given the particular form of the 32 differential paths and their associated conditions, it is best to use only a single path ($\Delta m_4 = 2^{22}$). The intermediate registers that we try to recover is

$$(B_{11}, B_{12}, B_{13}, B_{14}).$$

Using the conditions given in Table 4 of [26], we can count the number of bits in these registers that can be recovered using the modified bit flipping algorithm, and the results are listed in Table 2. We only list the condition on $B_{t,J}$ for which J is the maximum index among all the conditions on B_t. Following similar analysis as in the case of NMAC-MD5, we know that the total number of bits in B_t that can be recovered is $J + 1 - s_t$, where s_t is the rotation amount in Step t. (Note that for the differential path we used to attack NMAC-MD5, J is always 31.)

Table 2. Number of bits in register B_t that can be recovered in our key recovery attack on HMAC-MD4.

Step	Output	Conditions on relevant bits	rotation s_t	No. of recovered bits
10	B_{11}	$B_{11,29} = 1$	11	$29 + 1 - 11 = 19$
11	B_{12}	$B_{12,31} = B_{11,31}$	19	$31 + 1 - 19 = 13$
12	B_{13}	$B_{13,31} = 0$	3	$31 + 1 - 3 = 29$
13	B_{14}	$B_{14,31} = 0$	7	$31 + 1 - 7 = 25$

Hence the total number of recovered bits is $19 + 13 + 29 + 25 = 86$, leaving 42 bits for exhaustive search. Using techniques such as early stopping (see Appendix A.3), the workload can be made much less than 2^{40} MD4 operations.

The reason why we chose the particular differential path with $\Delta m_4 = 2^{22}$ is that for other paths (i.e., $\Delta m_4 = 2^i$), J is different, and more bits would be left for exhaustive search. Since we use only one path, the number of queries for finding the initial collision is 2^{63}, rather than 2^{58}. Hence, the query complexity for the key recovery attack is 2^{63}.

6.2 Attacks on HMAC/NMAC-SHA0

Chabaud and Joux [10] presented the first collision attack on SHA-0 with complexity 2^{61}. Their analysis also introduced important concepts such as local collisions and disturbance vectors, which prove to be the basis for all subsequent attacks on SHA-0 and SHA-1. The differential path used in their attack holds with probability $p = 2^{-83}$ (see Table 4 in [10] for detailed calculation).

We can use the above differential path to construct distinguish and forgery attack on HMAC-SHA0, and the query complexity is about 2^{84}. There is one subtle issue for SHA-0 and SHA-1 that is different from the case of MD4 and

MD5. For the collision differential path to hold, some extra conditions need to be set on the message bits. Hence, we should generate message pairs so that they not only satisfy the required conditions on the *message difference* but also extra conditions on certain *message bits*.

A partial key recovery attack on HMAC-SHA0 can also be constructed. In fact, the analysis would be much simpler than that of NMAC-MD5 due to the particular form of the SHA-0 (and SHA-1) step function, which is

$$A_i = (A_{i-1} \lll 5) + f_i(B_{i-1}, C_{i-1}, D_{i-1}) + E_{i-1} + m_{i-1} + k_i.$$

Since there is no rotation associated with the message word m_{i-1}, we can use the bit flipping algorithm directly (rather than the modified version) to recover bits in register A_i. Also, the absence of rotation means that more bits can be recovered for each A_i. Our analysis shows that the query complexity for the key recovery attack on HMAC-SHA0 is about 2^{84}.

6.3 Attacks on reduced-round variants of HMAC/NMAC-SHA1

Biham *et al.* [7] presented collision attacks on several reduced-round variants of SHA-1. Their attack on 34-round SHA-1 used a disturbance vector with very low Hamming weight (see Table 1 of [7]). Based on this vector, we can derive the differential path which consists of 6 local collisions in Rounds 1-20 and 2 local collisions in Rounds 21-34. This translates to $6 \times 5 + 2 \times 2 = 34$ conditions for the differential path, one of which is in the IV. Hence, the probability of the differential path is 2^{-33}, and it holds for half of the randomly chosen IVs.

We can use the above differential path to construct a distinguishing attack on the keyed SHA-1 compression function reduced to 34 rounds, which implies that the function is not a PRF.

Using our techniques developed earlier, we can construct all three types of attacks on HMAC-SHA1 when the inner function is reduced to 34 rounds. The query complexity is about 2^{34} and the success probability is 1/2 for a randomly chosen key.

6.4 Further improvements

It is possible to further improve the complexity of our attacks. Krawczyk [16] pointed out a useful tradeoff between query complexity and the success probability of the attacks. More specifically, we can construct new attacks with 2^t queries and success probability 2^{t-q}, where 2^q is the number of queries in our original attacks and $1 \leq t \leq q$. Biham [6] suggested that attacks on HMAC can be extended to 40-round SHA-1 using results in [7].

7 A general framework for analyzing HMAC/NMAC

In this section we extend the approach in our attacks to provide a general framework for analyzing HMAC/NMAC. Let DP be a collision differential path for the compression function f, and let $\Delta = (\Delta cv, \Delta m)$ be the required input difference for the path. Suppose that the path holds with probability at least $P_0 = 2^{-w}$ for a fraction q of all randomly chosen inputs (cv, cv') and (m, m') satisfying Δ. We consider two cases depending on Δcv:

- $\Delta cv = 0$. In this case, the path DP yields a real collision. The attacks to be considered are in the standard setting and apply to both HMAC and NMAC.
- $\Delta cv \neq 0$. In this case, the path DP yields a pseudo-collision. The attacks to be considered are in the related-key setting and apply only to NMAC.

There are three types of possible attacks, all having success probability q.

1. *Distinguishing attack.* The complexity is about $O(2^{w+1})$ queries.
2. *Forgery attack.* If the hash function F is iterative, the distinguishing attack implies a forgery attack with one additional chosen query.
3. *Key recovery attack.* If F has similar step functions as MDx, the collision path may allow the recovery of the inner key in HMAC and NMAC. The query complexity is $O(2^{w+1})$, and the time complexity depends on the form of the collision path.

To beat the generic birthday-type forgery attack, we need to find a collision differential path such that $P_0 > 2^{-n/2}$, and to beat the exhaustive key search attack, we need $P_0 > 2^{-n}$. Hence, the above general framework reduces the problem of attacking HMAC/NMAC to the problem of finding a "good" collision differential path for the underlying compression function.

Finding suitable differential paths There have been many collision attacks on hash functions, each relying on a specific differential path. One important point is that a differential path that works best for finding collisions may not be the best for the purpose of attacking HMAC and NMAC. To better explain this, we introduce a variable P_r, which is the probability of the differential path from Step r to the last step.

- For collision attacks, we should select a path such that P_r is minimized, assuming message modification techniques can apply up to Step r-1 of the hash function.
- For attacks on HMAC and NMAC, we should select a path such that P_0 is minimized.

For example, for the purpose of analyzing HMAC-SHA0, Chabaud and Joux's attack offers a better differential path than the improved collision attack in [24], since the probability P_0 associated with the differential path in the former attack is much larger than the latter.

To break HMAC-MD5, we would need to find differential paths that hold with large enough probability P_0 and lead to real collisions. The differential path in Wang's MD5 attack [22] was constructed to minimize P_{17} ($\approx 2^{-37}$) so that it works best with modification techniques. The total probability P_0 of the path is only about 2^{-300}. So far, improvements to the MD5 attack were all due to refined modification techniques: nobody has discovered new differential paths. An open question is whether differential paths for MD5 with $P_0 > 2^{-128}$ can be found. New automated search methods may provide promising ways for finding such differential paths.

Acknowledgements We thank Mihir Bellare and Hugo Krawczyk for valuable suggestions on an early draft of this work. We thank Eli Biham for enlightening discussions. We also thank Lily Chen, Antoine Joux, Josef Pieprzyk, and the Asiacrypt reviewers for helpful comments. The first author was supported by ARC grant DP0663452.

References

1. M. Bellare. *New Proofs for NMAC and HMAC: Security without Collision-Resistance*. ePrint archive, May 2006. To appear in CRYPTO 2006.
2. M. Bellare, R. Canetti and H. Krawczyk. *Keyed Hash Functions for Message Authentication*. CRYPTO 1996.
3. M. Bellare, R. Canetti and H. Krawczyk. *Pseudorandom Functions Revisited: the Cascade Construction*. FOCS 1996.
4. M. Bellare and T. Kohno. *A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications*. EUROCRYPT 2003.
5. E. Biham. *New Types of Cryptanalytic Attacks Using Related Keys*. EUROCRYPT 1993.
6. E. Biham. *Personal communication*. August 2006.
7. E. Biham, R. Chen, A. Joux, P. Carribault, W. Jalby and C. Lemuet. *Collisions in SHA-0 and Reduced SHA-1*. EUROCRYPT 2005.
8. E. Biham and A. Shamir. *Differential Cryptanalysis of DES-like Cryptosystems*. CRYPTO 1990.
9. B. den Boer and A. Bosselaers. *Collisions for the Compression Function of MD5*. EUROCRYPT 1993.
10. F. Chabaud and A. Joux. *Differential Collisions in SHA-0*. CRYPTO 1998.
11. J.-S. Coron, Y. Dodis, C. Malinaud and P. Puniya. *Merkle-Damgard Revisited : how to Construct a Hash Function*. CRYPTO 2005.
12. The GNU Crypto project. http://www.gnu.org/software/gnu-crypto/.
13. J. Kelsey, B. Schneier, and D. Wagner. *Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA*. ICICS 1997.
14. L. Knudsen. *Cryptanalysis of LOKI91*. AusCrypt 1992.
15. J. Kim, A. Biryukov, B. Preneel and S. Lee. *On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1*. SCN 2006. Also available at http://eprint.iacr.org/2006/187.
16. H. Krawczyk. *Personal communication*. June 2006.
17. B. Preneel and P.C. van Oorschot. *MDx-MAC and Building Fast MACs from Hash Functions*. CRYPTO 1995.
18. B. Preneel and P.C. van Oorschot. *On the Security of Two MAC Algorithms*. EUROCRYPT 1996.
19. B. Preneel and P. C. van Oorschot. *A key recovery attack on the ANSI X9.19 retail MAC*. Electronics Letters 32(17), 1996.

20. C. Rechberger and V. Rijmen *Note on Distinguishing, Forgery, and Second Preimage Attacks on HMAC-SHA-1 and a Method to Reduce the Key Entropy of NMAC.* Crypto eprint archive, http://eprint.iacr.org/2006/290.
21. P. Rogaway and T. Shrimpton. *Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance.* FSE 2004.
22. X. Wang and H. Yu. *How to Break MD5 and Other Hash Functions.* EUROCRYPT 2005.
23. X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. *Cryptanalysis of the Hash Functions MD4 and RIPEMD.* EUROCRYPT 2005.
24. X. Wang, H. Yu, and Y.L. Yin. *Efficient Collision Search Attacks on SHA-0.* CRYPTO 2005.
25. X. Wang, Y.L. Yin, and H. Yu. *Finding Collisions in the Full SHA-1.* CRYPTO 2005.
26. H. Yu, G. Wang, G. Zhang, and X. Wang. *The Second-Preimage Attack on MD4.* CANS 2005. Available on Springerlink web site.

A The bit flipping algorithms

We first give the bit flipping algorithm in Figure 1. This is for the simplified MD5 step function where the rotation is eliminated.

```
For j = 0,...,t-1, set m*_j = m_j
Set d = 0                        (a)
For i = 30 downto 0 do           (b)
{
    Set m*_t = m_t + 2^i + d     (c)
    Repeat order 2^(46-t) times
    {
        Choose m*_{t+1},...,m*_15 at random.
        /* now all 16 words of m* have been set */
        Query the two nmac oracles on m*
        If there is a collision, then
        {
            Bit i of B_{t+1} is 0
            Set d = d + 2^i      (d)
            break;
        }
    }
    If no collision found, then bit i of B_{t+1} is 1
}
```

Figure 1: Bit flipping algorithm for computing B_{t+1}.

The modified bit flipping algorithm is similar, except the following four steps:

- Step (a) $\Rightarrow$ `Set` $d' = 0$
- Step (b) $\Rightarrow$ `For` i' `=` $30 - s_t$ `downto 0 do`
- Step (c) $\Rightarrow$ `Set` $m_t^* = m_t + 2^{i'} + d'$
- Step (d) $\Rightarrow$ `Set` $d' = d' + 2^{i'}$

A.1 Proof of Lemma 2

For ease of discussion, we introduce a few more variables and rewrite Step t of MD5 for both m_t and m_t^* as follows:

$$\begin{aligned} Y_t &= m_t + Z \\ X_t &= Y_t \lll s_t \\ B_{t+1} &= X_t + B_t \end{aligned}$$

$$\begin{aligned} Y_t^* &= m_t^* + Z \\ X_t^* &= Y_t^* \lll s_t \\ B_{t+1}^* &= X_t^* + B_t \end{aligned}$$

In the modified bit flipping algorithm, we set $m_t^* = m_t + \Delta$, where $\Delta = 2^{i'} + d'$. Given the choice of i' and d', we know that $\Delta = 2^{i'} + d' < 2^{31-s_t}$ and its rotated value is $2^{s_t}\Delta = 2^i + d$.

To prove the lemma, we need to analyze the difference between B_{t+1}^*, the value that is computed by the algorithm, and $B_{t+1} + 2^{s_t}\Delta$, the desired value that would make the algorithm work *correctly*. The difference between the two values is the same as the difference between X_t^* and $X_t + 2^{s_t}\Delta$, which we will analyze below. Because of the rotate, these two quantities are computed in two different parts: the high order $(32 - s_t)$ bits which consists of bits $[s_t..31]$, and the low order s_t bits which consists of bits $[0..s_t - 1]$.

We first observe that the high order bits of X_t^* and $X_t + 2^{s_t}\Delta$ ar e always the same, namely,

$$(2^{s_t} Y_t + 2^{s_t}\Delta) \bmod 2^{32}.$$

We next consider the low order bits of X_t^* and $X_t + 2^{s_t}\Delta$. Let `Carry` be the event that the addition $Y_t^* = Y_t + \Delta$ causes a carry into bit $(32 - s_t)$ of Y_t^*. If `Carry` does not happen, then the low order bits of the two values are also the same, and hence there is no difference between B_{t+1}^* and $B_{t+1} + 2^{s_t}\Delta$. If `Carry` happens, then there are two cases where it can affect the MSB of B_{t+1}^* for some iteration of the algorithm. Below, we bound the probability of each case.

The first case is when the carry propagates all the way to bit 32. This can happen only if bits $[31 - s_t..31]$ of Y_t are all 1's, so the probability of this event is $\leq 2^{-s_t-1}$. The second case is when the carry goes into bit $32 - s_t$ of Y_t^* but does not go over bit 31. In this case, we have $X_t^* = X_t + 2^{s_t}\Delta + 1$, where the extra 1 is the result of carry after it is rotated to the least significant bits of X_t^*. So we now have $B_{t+1}^* = B_{t+1} + 2^{s_t}\Delta + 1$, and the extra 1 can affect the high order bits of B_{t+1}^* only if B_{t+1} has all 1's in bits $[0..(s_t - 1)]$. The probability of this event is 2^{-s_t}. In fact, when this does happen, the modified bit flipping algorithm will indeed fail because by construction we are setting the high order bits of B_{t+1}^* to all 1's, so the extra 1 causes a carry to propagate all the way through. Thus, the probability of failure for the second case is exactly 2^{-s_t}.

So the total failure probability is at most $2^{-s_t} + 2^{-s_t-1}$, and we can conclude that the success probability is $p_t \geq 1 - 2^{-s_t} - 2^{-s_t-1}$. QED

A.2 Phase 3

In Phase 3, we will try to determine more bits of the states $(A_{14}, D_{14}, C_{14}, B_{14})$ with additional queries. Let us consider the computation of B_{14} in Step 13:

$$B_{14} = (A_{13} + \phi(B_{13}, C_{13}, D_{13}) + M_{13} + K_{13}) \lll 12 + B_{13}.$$

We first use the modified bit flipping algorithm to learn the most significant 20 bits of B_{10}. Together with information obtained from Phase 2 (see Section 5.4), we already know 70 bits of the states on the RHS of the above equation. These bits are:

$$\begin{aligned} A_{13} = B_{10} &: \text{ most significant 20 bits} \\ D_{13} = B_{11} &: \text{ most significant 15 bits} \\ C_{13} = B_{12} &: \text{ most significant 10 bits} \\ B_{13} = B_{13} &: \text{ most significant 25 bits} \end{aligned}$$

We next set the unknown bits on the RHS to 0 and trace through the computation in this step. We can compute the most significant 10 bits of $\phi(B_{13}, C_{13}, D_{13})$. Since we also know the same bits from A_{13}, we can for sure compute bits [22..31] of $A_{13} + \phi(B_{13}, C_{13}, D_{13}) + M_{13} + K_{13}$, with a possible additive error of a 1 in bit 22 (i.e. additive error of 2^{22}) which may have come from a carry. The rotation by 12 is then applied, which means we have computed bits [2..11], with a possible additive error of 2^2. Then we add on B_{13}, for which we know bits [7..31]. The net result is that we have determined bits [7..11] of B_{14}, with a possible additive error of 2^7. In Phase 2, we already know bits [12..31] of B_{14}. So now we know in total bits [7..31] of B_{14}, with a possible additive error of 2^7. In other words, we can recover 5 more bits of B_{12} with a possible additive error.

Similarly, we can learn more bits of B_8, B_9 and trace through the computation from Step 11. We can recover 5 more bits of $B_{12} = D_{14}$ with a possible additive error. That gives us four possibilities that determine a total of 80 bits for the state $(A_{14}, D_{14}, C_{14}, B_{14})$ (The "four" accounts for the possible additive errors). Hence, the remaining search effort is at most $4 \times 2^{128-80} = 2^{50}$.

A.3 Complexity analysis

¿From Lemma 2, the modified flipping algorithm succeeds with probability $\geq 1 - 2^{s_t} - 2^{-s_t-1}$ in Step t for recovering $(31 - s_t)$ (note that the MSB is assumed) of B_{t+1}. Thus, the probability of success is at least $\prod_{t=7}^{13} 1 - 2^{-s_t} - 2^{-s_t-1} > 0.97$. So one iteration of the four phases is usually enough to recover the correct inner key.

Finding the original collision in Phase 1 requires order 2^{46} queries for each oracle. For Phases 2 and 3, the number of queries is order $2 \times \sum_{t=7}^{13} (31 - s_t) 2^{45-t} < 2^{44}$. Thus, the total number of queries for the key recovery algorithm is order 2^{47}. In other words, the number of oracle queries is dominated by that for finding the original collision.

The remaining work is order 2^{50} MD5 operations. There are a variety of tricks that can be used to reduce this further. For example, if the resulting A_{13} does not match the expected value when taking the first step backwards in the MD5 computation for a guessed key, then we can do an early abort and go to the next guess. This reduces the workload by a factor of 2^5, down to 2^{45} MD5 operations. One can easily find other short cuts, but 2^{45} workload is already do-able with moderate computing resources.