# Full Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5

Pierre-Alain Fouque, Gaëtan Leurent, Phong Q. Nguyen

École Normale Supérieure – Département d'Informatique,
45 rue d'Ulm, 75230 Paris Cedex 05, France
{Pierre-Alain.Fouque,Gaetan.Leurent,Phong.Nguyen}@ens.fr

**Abstract.** At Crypto '06, Bellare presented new security proofs for HMAC and NMAC, under the assumption that the underlying compression function is a pseudo-random function family. Conversely, at Asiacrypt '06, Contini and Yin used collision techniques to obtain forgery and partial key-recovery attacks on HMAC and NMAC instantiated with MD4, MD5, SHA-0 and reduced SHA-1. In this paper, we present the first full key-recovery attacks on NMAC and HMAC instantiated with a real-life hash function, namely MD4. Our main result is an attack on HMAC/NMAC-MD4 which recovers the full MAC secret key after roughly $2^{88}$ MAC queries and $2^{95}$ MD4 computations. We also extend the partial key-recovery Contini-Yin attack on NMAC-MD5 (in the related-key setting) to a full key-recovery attack. The attacks are based on generalizations of collision attacks to recover a secret IV, using new differential paths for MD4.
**Key words:** NMAC, HMAC, key-recovery, MD4, MD5, collisions, differential path.

## 1 Introduction

Hash functions are fundamental primitives used in many cryptographic schemes and protocols. In a breakthrough work, Wang *et al.* discovered devastating collision attacks [20,22,23,21] on the main hash functions from the MD4 family, namely MD4 [20], RIPE-MD [20], MD5 [22], SHA-0 [23] and SHA-1 [21]. Such attacks can find collisions in much less time than the birthday paradox. However, their impact on the security of existing hash-based cryptographic schemes is unclear, for at least two reasons: the applications of hash functions rely on various security properties which may be much weaker than collision resistance (such as pseudorandomness); Wang *et al.*'s attacks are arguably still not completely understood.

This paper deals with key-recovery attacks on HMAC and NMAC using collision attacks. HMAC and NMAC are hash-based message authentication codes proposed by Bellare, Canetti and Krawczyk [3], which are very interesting to study for at least three reasons: HMAC is standardized (by ANSI, IETF, ISO and NIST) and widely deployed (*e.g.* SSL, TLS, SSH, Ipsec); both HMAC and

NMAC have security proofs [2,3]; and both are rather simple constructions. Let $H$ be an iterated Merkle-Damgård hash function. Its HMAC is defined by

$$\mathrm{HMAC}_k(M) = H(\bar{k} \oplus \mathrm{opad} \,||H(\bar{k} \oplus \mathrm{ipad} \,||M)),$$

where $M$ is the message, $k$ is the secret key, $\bar{k}$ its completion to a single block of the hash function, opad and ipad are two fixed one-block values. The security of HMAC is based on that of NMAC. Since $H$ is assumed to be based on the Merkle-Damgård paradigm, denote by $H_k$ the modification of $H$ where the public IV is replaced by the secret key $k$. Then NMAC with secret key $(k_1, k_2)$ is defined by:

$$\mathrm{NMAC}_{k_1,k_2}(M) = H_{k_1}(H_{k_2}(M)).$$

Thus, $\mathrm{HMAC}_k$ is essentially equivalent to $\mathrm{NMAC}_{H(k\oplus\mathrm{opad}),H(k\oplus\mathrm{ipad})}$[1]. Attacks on NMAC can usually be adapted to HMAC (pending few modifications), except in the related-key setting[2].

**HMAC/NMAC Security.** The security of a MAC algorithm is usually measured by the difficulty for an attacker having access to a MAC oracle to forge new valid MAC-message pairs. More precisely, we will consider two types of attack: the existential forgery where the adversary must produce a valid MAC for *a message of its choice*, and the universal forgery where the attacker must be able to compute the MAC of *any message*.

The security of HMAC and NMAC was carefully analyzed by its designers. It was first shown in [3] 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 $H_k$ is *weakly collision resistant*. The proof for NMAC was then lifted to HMAC by further assuming that (A3) the key derivation function in HMAC is a PRF. However, it was noticed that recent collision attacks [20,22,23,21] invalidate (A2) in the case of usual hash function like MD4 or MD5, because one can produce collisions for any public IV. This led Bellare [2] to present new security proofs for NMAC under (A1) only. As a result, the security of HMAC solely depends on (A1) and (A3). The security of NMAC as a PRF holds only if the adversary makes less than $2^{n/2}$ NMAC queries (where $n$ is the MAC size), since there is a generic forgery attack using the birthday paradox with $2^{n/2}$ queries.

Since recent collision attacks cast a doubt on the validity of (A1), one may wonder if it is possible to exploit collision search breakthroughs to attack HMAC and NMAC instantiated with real-life hash functions. In particular, MD4 is a very tempting target since it is by far the weakest real-life hash function with respect to collision resistance. It is not too difficult to apply collision attacks on MD4 [20,24], to obtain distinguishing and existential forgery attacks

[1] There is small difference in the padding: when we use $H(k||\cdot)$ instead of $H_k$, the length of the input of the hash function (which is included in the padding) is different.

[2] If we need an oracle $\mathrm{NMAC}_{k_1,k_2+\Delta}$, we can not emulate it with an related-key HMAC oracle.

on HMAC/NMAC-MD4: for instance, this was done independently by Kim *et al.* [10] and Contini and Yin [5]. The situation is more complex with MD5, because the differential path found in the celebrated MD5 collision attack [22] is not well-suited to HMAC/NMAC since it uses two blocks: Contini and Yin [5] turned instead to the much older MD5 pseudo-collisions of de Boer and Bosselaers [8] to obtain distinguishing and existential forgery attacks on NMAC-MD5 in the related-key setting. It is the use of pseudo-collisions (rather than full collisions) which weakens the attacks to the related-key setting.

Interestingly, universal forgery attacks on HMAC and NMAC seem much more difficult to find. So far, there are only two works in that direction. In [5], Contini and Yin extended the previous attacks to *partial* key-recovery attacks on HMAC/NMAC instantiated with MD4, SHA-0, and a step-reduced SHA-1, and related-key *partial* key-recovery attacks on NMAC-MD5. In [16], Rechberger and Rijmen improved the data complexity of Kim *et al.* [10] attacks, and extended them to a partial key recovery against NMAC-SHA-1. These attacks are only partial in the sense that the NMAC attacks only recover the second key $k_2$, which is not sufficient to compute new MACs of arbitrary messages; and the HMAC attacks only recover $H(k \oplus \text{ipad})$ where $k$ is the HMAC secret key, which again is not sufficient to compute new MACs of arbitrary messages, since it does not give the value of $k$ nor $H(k \oplus \text{opad})$. Note that recovering a single key of NMAC does not significantly improve the generic full key-recovery attack which recovers the keys one by one.

Very recently, Rechberger and Rijmen have proposed full key-recovery attacks against NMAC in the related-key setting in [17]. They extended the attack of [5] to a full key-recovery attack against NMAC-MD5, and introduced a full key-recovery attack against NMAC when used with SHA-1 reduced to 34 rounds.

**Our Results.** We present what seems to be the first universal forgery attack, *without related keys*, on HMAC and NMAC instantiated with a real-life hash function, namely MD4. Our main result is an attack on HMAC/NMAC-MD4 which recovers the full NMAC-MD4 secret key after $2^{88}$ MAC queries; for HMAC, we do not recover the HMAC-MD4 secret key $k$, instead we recover both $H(k \oplus \text{ipad})$ and $H(k \oplus \text{opad})$, which is sufficient to compute any MAC. We also obtain a full key-recovery attack on NMAC-MD5 in the related-key setting, by extending the attack of Contini and Yin [5]. This improvement was independently proposed in [16].

Our attacks have a complexity greater than the birthday paradox, so they are not covered by Bellare's proofs. Some MAC constructions have security proof against PRF-attacks, but are vulnerable to key-recovery attacks. For instance, the envelope method with a single key was proved to be secure, but a key-recovery attack using $2^{67}$ known text-MAC pairs, and $2^{13}$ chosen texts was found by Preneel and van Oorschot [14]. However, in the case of NMAC, we can prove that the security against universal forgery cannot be less than the security of the compression function against a PRF distinguisher (see Appendix A). This shows that NMAC offers good resistance beyond the birthday paradox: a universal

forgery attack will have a time complexity of $2^n$ if there is no weakness in the compression function. Conversely, there is a generic attack against any iterated stateless MAC using a collision in the inner hash function to guess the two subkeys $k_1$ and $k_2$ independently, in the case of NMAC it requires $2^{n/2}$ queries and $2^{n+1}$ hash computations [13,15].

Our attacks on MD4 and MD5 are rather different from each other, although both are based on IV-recovery attacks, which allow to recover the IV when one is given access to a hash function whose IV remains secret. Such IV-recovery attacks can be exploited to attack HMAC and NMAC because the oracle can in fact be very weak: we do not need the full output of the hash function; essentially, we only need to detect if two related messages collide under the hash function. The MD5 related-key attack closely follows the Contini-Yin attack [5]: the IV-recovery attack is more or less based on message modification techniques. The MD4 IV-recovery attack is based on a new technique: we use differential paths which depend on a condition in the IV. The advantage of this technique is that it can be used to recover the outer key quite efficiently, since we only need to control the *difference* of the inputs and not the *values* themselves. This part of the attack shares some similar ideas with [17]. To make this possible *without related keys*, we need a differential path with a message difference only active in the first input words. We found such IV-dependant paths using an automated tool described in [9]. To make this attack more efficient, we also introduce a method to construct cheaply lots of message pairs with a specific hash difference.

Our results are summarized in the following table, together with previous attacks where "Data" means online queries and "Time" is offline computations:

| Attacks | | Data | Time | Mem | Remark |
|---|---|---|---|---|---|
| Generic | E-Forgery | $2^{n/2}$ | - | - | [15] Collision based |
| | U-Forgery | $2^{n/2}$ | $2^{n+1}$ | - | [15] Collision based |
| | | 1 | $2^{2n/3}$ | $2^{2n/3}$ | [1] TM tradeoff, $2^n$ precomputation |
| NMAC-MD4 HMAC-MD4 | E-Forgery | $2^{58}$ | - | - | [5] Complexity is actually lower [9] |
| | Partial-KR | $2^{63}$ | $2^{40}$ | - | [5] Only for NMAC |
| | U-Forgery | $2^{88}$ | $2^{95}$ | - | **New result** |
| NMAC-MD5 *Related keys* | E-Forgery | $2^{47}$ | - | - | [5] |
| | Partial-KR | $2^{47}$ | $2^{45}$ | - | [5] |
| | U-Forgery | $2^{51}$ | $2^{100}$ | - | **New result** – Same as [17] |

Like [5], we stress that our results on HMAC and NMAC do not contradict any security proof; on the contrary they show that when the hypotheses over the hash function are not met, an attack can be built.

**Road Map.** This paper is divided in five sections. In Section 2, we give background and notations on MD4, MD5 and collision attacks based on differential cryptanalysis. In Section 3, we explain the framework of our key-recovery attacks on HMAC and NMAC, by introducing IV-recovery attacks. In Section 4, we present key-recovery attacks on HMAC/NMAC-MD4. Finally, in Section 5, we present related-key key-recovery attacks on NMAC-MD5.

## 2 Background and notation

Unfortunately, there does not seem to be any standard notation in the hash function literature. Here, we will use a notation similar to that of Daum [7].

### 2.1 MD4 and MD5

MD4 and MD5 follow the Merkle-Damgård construction. Their compression function are designed to be very efficient using 32-bit words and operations implemented in hardware in most processors:

- rotation $\lll$;
- addition mod $2^{32}$ $\boxplus$;
- bitwise boolean operations $\Phi_i$. For MD4 and MD5, they are:
  - $\mathrm{IF}(x,y,z) = (x \wedge y) \vee (\neg x \wedge z)$
  - $\mathrm{MAJ}(x,y,z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$
  - $\mathrm{XOR}(x,y,z) = x \oplus y \oplus z$
  - $\mathrm{ONX}(x,y,z) = (x \vee \neg y) \oplus z$.

  MD4 uses $\mathrm{IF}(x,y,z)$, $\mathrm{MAJ}(x,y,z)$ and $\mathrm{XOR}(x,y,z)$, while MD5 uses $\mathrm{IF}(x,y,z)$, $\mathrm{IF}(z,x,y)$, $\mathrm{XOR}(x,y,z)$ and $\mathrm{ONX}(x,y,z)$.

The compression function cMD4 (resp. cMD5) of MD4 (resp. MD5) uses an internal state of four words, and updates them one by one in 48 (resp. 64) steps. Their input is 128 bits $\times$ 512 bits, and their output is 128 bits. Here, we will assign a name to every different value of these registers, following [7]: the value changed on step $i$ is called $Q_i$. Then the cMD4 compression function is defined by:

$$\begin{array}{|r|}\hline \text{Step update: } Q_i = (Q_{i-4} \boxplus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxplus m_i \boxplus k_i) \lll s_i \\ \text{Input: } Q_{-4}||Q_{-1}||Q_{-2}||Q_{-3} \\ \text{Output: } Q_{-4} \boxplus Q_{44}||Q_{-1} \boxplus Q_{47}||Q_{-2} \boxplus Q_{46}||Q_{-3} \boxplus Q_{45} \\ \hline \end{array}$$

And the cMD5 compression function is given by:

$$\begin{array}{|r|}\hline \text{Step update: } Q_i = Q_{i-1} \boxplus (Q_{i-4} \boxplus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxplus m_i \boxplus k_i) \lll s_i \\ \text{Input: } Q_{-4}||Q_{-1}||Q_{-2}||Q_{-3} \\ \text{Output: } Q_{-4} \boxplus Q_{60}||Q_{-1} \boxplus Q_{63}||Q_{-2} \boxplus Q_{62}||Q_{-3} \boxplus Q_{61} \\ \hline \end{array}$$

The security of the compression function was based on the fact that such operations are not "compatible" and mix the properties of the input.

We will also use $x^{[k]}$ to represent the $k+1$-th bit of $x$, that is $x^{[k]} = (x \ggg k) \bmod 2$ (note that we count bits and steps starting from 0).

### 2.2 Collision attacks based on differential cryptanalysis

It is natural to apply differential cryptanalysis to find collisions on hash functions based on block ciphers, like MD4 and MD5 (which both follow the Davies-Meyer

construction). The main idea is to follow the differences in the internal state $Q_i$ of the compression function, when the inputs (the IVs or the messages) have a special difference.

Our attacks on NMAC-MD5 are based on the MD5 pseudo-collision of de Boer and Bosselaers [8], like [5] (this is the only known attack against MD5 compression function's pseudo-randomness). Let $IV$ be a 128-bit value satisfying the so-called dBB condition: the most significant bit of the last three 32-bit words of $IV$ are all equal. Clearly, a randomly chosen $IV$ satisfies the dBB condition with probability $1/4$. It is shown in [8] that in such a case, a randomly chosen 512-bit message $M$ satisfies with heuristic probability $2^{-46}$:

$$\text{cMD5}(IV, M) = \text{cMD5}(IV', M),$$

where $IV'$ is the 128-bit value derived from $IV$ by flipping the most significant bit of each of the four 32-bit words of $IV$. The probability $2^{-46}$ is obtained by studying the most likely differences for the internal state $Q_i$, as is usual in differential cryptanalysis.

Our attacks on HMAC/NMAC-MD4 are based on recent collision search techniques for MD4 [20,24], which are organized as follows:

1. A precomputation phase:
   - choose a message difference $\Delta$
   - find a differential path
   - compute a set of sufficient conditions
2. Search for a message $M$ satisfying all the conditions for a given IV; then $\text{cMD4}(\text{IV}, M) = \text{cMD4}(\text{IV}, M \boxplus \Delta)$.

The differential path specifies how the computations of $\text{cMD4}(\text{IV}, M \boxplus \Delta)$ and $\text{cMD4}(\text{IV}, M)$ are related: it describes how the differences introduced in the message will evolve in the internal state $Q_i$. By choosing a special $\Delta$ with a low Hamming weight and extra properties, we can find differences in the $Q_i$ which are very likely. Then we look at each step of the compression function, and we can express a set of sufficient conditions that will make the $Q_i$'s follow the path. The conditions are on the $Q_i$'s, and their values depends of the $IV$ and the message $M$. For a given message $M$, we will have $\text{cMD4}(\text{IV}, M) = \text{cMD4}(\text{IV}, M \boxplus \Delta)$ if the conditions are satisfied; we expect this to happen with probability $2^{-c}$ for a random message if there are $c$ conditions. Wang introduced some further ideas to make the search for such message more efficient, but they can't be used in the context of NMAC because the IV is unknown.

## 3 Key-Recovery Attacks on HMAC and NMAC

In this section, we give a high-level overview of our key-recovery attacks on HMAC and NMAC instantiated with MD4 and MD5. Detailed attacks will be given in the next two sections: Section 4 for MD4 and Section 5 for MD5. We will assume that the attacker can request the MAC of messages of its choice, for

a fixed secret key, and the goal is to recover that secret key. In the related-key setting, we will assume like in [5] that the attacker can request the MAC of messages of its choice, for the fixed secret key as well as for other related secret keys (with a chosen relation). In fact, we will not even need the full output of MAC requests: we will only need to know if the two MACs of messages of our choice collide or not.

To simplify our exposition, we will concentrate on the NMAC case:

$$\mathrm{NMAC}_{k_1,k_2}(M) = H_{k_1}(H_{k_2}(M)).$$

The NMAC-MD4 attack can easily be extended to HMAC-MD4, pending minor modifications. Our NMAC attack will first recover $k_2$, then $k_1$. We will collect NMAC collisions of a special shape, in order to disclose hash collisions with first $k_2$ then $k_1$.

### 3.1 Extracting hash collisions from NMAC collisions

We will extract hash collisions from NMAC collisions, that is, pairs $(M_1, M_2)$ of messages such that:

$$\text{(C)} \qquad M_1 \neq M_2 \quad \text{and} \quad \mathrm{NMAC}_{k_1,k_2}(M_1) = \mathrm{NMAC}_{k_1,k_2}(M_2).$$

Our attacks are based on the elementary observation that $H$-collisions can leak through NMAC. More precisely, two messages $M_1$ and $M_2$ satisfy (C) if and only if they satisfy either (C1) or (C2):

**(C2)** $M_1 \neq M_2$ and $H_{k_2}(M_1) = H_{k_2}(M_2)$: we have a collision in the inner hash function;

**(C1)** $H_{k_2}(M_1) = N_1 \neq N_2 = H_{k_2}(M_2)$ and $H_{k_1}(N_1) = H_{k_1}(N_2)$: we have a collision in the outer hash function .

If we select $M_1$ and $M_2$ uniformly at random, then (C) holds with probability $2^{-128}$ if NMAC is a random function. However, if we select many pairs $(M_1, M_2)$ in such a way that (C2) holds with a probability significantly higher than $2^{-128}$, then whenever $\mathrm{NMAC}_{k_1,k_2}(M_1) = \mathrm{NMAC}_{k_1,k_2}(M_2)$, it will be likely that we also have (C2). More precisely, we have (since $Ci \cap C = Ci$):

$$\frac{\Pr(C2|C)}{\Pr(C1|C)} = \frac{\Pr(C2)}{\Pr(C1)}$$

and we expect that $\Pr(C2) \gg \Pr(C1) \approx 2^{-128}$.

Note that the Merkle-Damgård construction used in $H$ leads to a simple heuristic way to distinguish both cases (without knowing the secret keys $k_1$ and $k_2$) if $M_1$ and $M_2$ have the same length, and therefore the same padding block $P$: if $H_{k_2}(M_1) = H_{k_2}(M_2)$, then for any $M$, we have $H_{k_2}(M_1||P||M) = H_{k_2}(M_2||P||M)$. In other words, the condition (C2) is preserved if we append $P||M$ to both $M_1$ and $M_2$ for a randomly chosen $M$, but that is unlikely for the condition (C1).

To illustrate our point, assume that we know a non-zero $\Delta$ such that for all keys $k_2$, a randomly chosen one-block message $M_1$ satisfies with probability $2^{-64}$ the condition $H_{k_2}(M_1) = H_{k_2}(M_2)$ where $M_2 = M_1 \boxplus \Delta$. If we select $2^{64}$ one-block messages $M_1$ uniformly at random and call the NMAC oracle on each $M_1$ and $M_1 \boxplus \Delta$, we are likely to find a pair $(M_1, M_2 = M_1 \boxplus \Delta)$ satisfying (C). By the previous reasoning, we expect that such a pair actually satisfies (C2).

Thus, the NMAC oracle allows us to detect collisions on $H_{k_2}$, if we are able to select messages which have a non-negligible probability of satisfying (C2). To detect collisions in $H_{k_1}$, we will use the values of $k_2$ (recovered using collisions in $H_{k_2}$): then, we can compute $H_{k_2}$ and directly check whether the NMAC collision come from (C1). We now explain how to use such collision detections to recover the secret keys $k_2$ and $k_1$.

### 3.2 IV-recovery attacks

The previous subsection suggests the following scenario. Assume that a fixed key $k$ is secret, but that one is given access to an oracle which on input $M_1$ and $M_2$, answers whether $H_k(M_1) = H_k(M_2)$ holds or not. Can one use such an oracle to recover the secret key $k$? If so, we have what we call an IV-recovery attack.

An IV-recovery attack would clearly reveal the second key $k_2$ of NMAC, because of (C2). But it is not clear why this would be relevant to recover the outer key $k_1$. To recover $k_1$ thanks to (C1), we would need the following variant of the problem. Namely, one would like to retrieve a secret key $k_1$ when given access to an oracle which on input $M_1$ and $M_2$, answers whether $H_{k_1}(H_{k_2}(M_1)) = H_{k_1}(H_{k_2}(M_2))$ holds or not, where $k_2$ is known. Since the messages are first processed through a hash function, the attacker no longer chooses the input messages of the keyed hash function, and this oracle is much harder to exploit than the previous one. We call such attacks composite IV-recovery attacks. In the attack on HMAC/NMAC-MD4, we will exploit the Merkle-Damgård structure of $H_{k_2}$ to efficiently extend the basic IV-recovery attacks into composite IV-recovery attacks.

We will present two types of IV-recovery attacks. The first type is due to Contini and Yin [5] and uses related messages, while the second type is novel, based on IV-dependent differential paths.

**Using related messages.** We present the first type of IV-recovery attacks. Assume that we know a specific differential path corresponding to a message difference $\Delta$ and with total probability $p$ much larger than $2^{-128}$. In other words, a randomly chosen message $M$ will satisfy with probability $p$:

$$H_k(M) = H_k(M \boxplus \Delta).$$

By making approximately $2/p$ queries to the $H_k$-oracle, we will obtain a message $M$ such that $H_k(M) = H_k(M \boxplus \Delta)$. Contini and Yin [5] then make the heuristic assumption that the pair $(M, M \boxplus \Delta)$ must follow the whole differential path, and not just the first and last steps. Since they do not justify that assumption,

let us say a few words about it. The assumption requires a strong property on our specific differential path: that there are no other differential paths with better (or comparable) probability. In some sense, differential cryptanalysis on block ciphers use similar assumptions, and to see how realistic that is, one makes experiments on reduced-round versions of the block cipher. However, one might argue that there are intuitively more paths in hash functions than in block ciphers because of the following facts:

- the message length of a compression function is much bigger than the length of the round key in a block cipher.
- a step of the compression function is usually much simpler than a block-cipher step.

Also, because the paths for hash functions have a different shape from those of block ciphers, experiments on reduced-round versions may not be as conclusive.

The paper [5] shows that for usual differential paths (like those of MD4), if $(M, M \boxplus \Delta)$ satisfies the whole path, then one can build plenty of messages $M^*$ closely related to $M$ such that:

- If a specific internal register $Q_i$ (during the computation of $H_k(M)$) satisfies certain conditions, then the pair $(M^*, M^* \boxplus \Delta)$ follows the whole path with probability $p$ or larger, in which case $H_k(M^*) = H_k(M^* \boxplus \Delta)$.
- Otherwise, the pair $(M^*, M^* \boxplus \Delta)$ will drift away from the path at some position, and the probability of $H_k(M^*) = H_k(M^* \boxplus \Delta)$ is heuristically $2^{-128}$.

Thus, by sending to the oracle many well-chosen pairs $(M', M' \boxplus \Delta)$, one can learn many bits of several internal register $Q_i$'s during the computation of $H_k(M)$. Applying exhaustive search on the remaining bits of such $Q_i$'s, one can guess the whole contents of four consecutive $Q_i$'s. By definition of cMD4 and cMD5, it is then possible to reverse the computation of $H_k(M)$, which discloses $k = (Q_{-4}, Q_{-3}, Q_{-2}, Q_{-1})$.

**Using IV-dependent differential paths.** We now present a new type of IV-recovery attacks, that we will apply against MD4. Assume again that we know a specific differential path corresponding to a message difference $\Delta$ and with total probability $p$ much larger than $2^{-128}$, but assume this time that the path is IV-dependent: it holds only if the IV satisfies a specific condition (SC). In other words, if $k$ satisfies (SC), then a randomly chosen message $M$ will satisfy with probability $p$:

$$H_k(M) = H_k(M \boxplus \Delta).$$

But if $k$ does not satisfy (SC), the pair $(M, M \boxplus \Delta)$ will drift away from the differential path from the first step, leading us to assume that $H_k(M) = H_k(M \boxplus \Delta)$ will hold with probability only $2^{-128}$.

This would lead to the following attack: we would submit approximately $2/p$ pairs $(M, M \boxplus \Delta)$ to the $H_k$-oracle, and conclude that $k$ satisfies (SC) if and

only if $H_k(M) = H_k(M \boxplus \Delta)$ for at least one $M$. When sufficient information on $k$ has been gathered, the rest of $k$ can be guessed by exhaustive search if we have at least one collision of the form $H_k(M) = H_k(M \boxplus \Delta)$.

Notice that in some sense the differential path of the MD5 pseudo-collision [8] is an example of IV-dependent path where (SC) is the dBB condition, but it does not disclose much information about the IV. We would need to find many IV-dependent paths. Our attack on HMAC/NMAC-MD4 will use 22 such paths, which were found by an automated search.

We note that such attacks require an assumption similar to the previous IV-recovery attack. Namely, we assume that for the same message difference $\Delta$, there is no differential paths with better (or comparable) probability, with or without conditions on the IV. To justify this assumption for our HMAC/NMAC-MD4 attack, we have performed experiments which will be explained in Section 4.

### 3.3 Subtleties between the inner and outer keys

Although the recovery of the inner key $k_2$ and the outer key $k_1$ both require IV-recovery attacks, we would like to point out subtle differences between the two cases. As mentioned previously, the recovery of $k_2$ only requires a basic IV-recovery attack, while the recovery of $k_1$ requires a composite IV-recovery attack. The composite IV-recovery attacks will be explained in Section 4 for MD4, and Section 5 for MD5.

When turning a basic IV-recovery attack into a composite IV-recovery, there are two important restrictions to consider:

- We have no direct control over the input of the outer hash function, it is the result of the inner hash function. So IV-recovery attacks using message modifications will become much less efficient when turned into composite IV-recovery. We will see this in Section 5 when extending the partial key-recovery from [5] into a full key-recovery.
- Since the input of $H_{k_1}$ is a hash, its length is only 128 bits. Any differential path using a message difference $\Delta$ with non-zero bits outside these 128 first bits will be useless. This means that the partial key-recovery attacks from [5] against MD4, SHA-0 and reduced SHA-1 can't be extended into a full key-recovery.
  Using related keys one can use a differential path with a difference in the IV and no message difference – such as the one from [8] – and try a given message with both keys. However, if we want to get rid of related keys, we need a differential path with no IV difference and a difference in the beginning of the message.

### 3.4 Summary

To summarize, our attacks will have essentially the following structure (the MD5 attack will be slightly different because of the related-key setting):

1. Apply an IV-recovery attack to retrieve $k_2$, repeating sufficiently many times:

(a) Select many one-block messages $M$ uniformly at random.
(b) Observe if $\mathrm{NMAC}_{k_1,k_2}(M) = \mathrm{NMAC}_{k_1,k_2}(M \boxplus \Delta_1)$ for some $M$ and a well-chosen $\Delta_1$.
(c) Deduce information on $k_2$.

2. Apply a composite IV-recovery attack to retrieve $k_1$, repeating sufficiently many times:
(a) Construct carefully many pairs $(M_1, M_2)$.
(b) Observe if $\mathrm{NMAC}_{k_1,k_2}(M_1) = \mathrm{NMAC}_{k_1,k_2}(M_2)$ for some pair $(M_1, M_2)$.
(c) Deduce information on $k_1$.

# 4 Attacking HMAC/NMAC-MD4

## 4.1 Our IV-recovery Attack against MD4

In order to find differential paths which leak information about the key, we consider differential paths with a message difference in the first word (eg. $\delta m_0 = 1$). Then in the first steps of the compression function, we have:

$$Q_0 = (Q_{-4} \boxplus \mathrm{IF}(Q_{-1}, Q_{-2}, Q_{-3}) \boxplus m_0) \lll 3$$
$$Q'_0 = (Q_{-4} \boxplus \mathrm{IF}(Q_{-1}, Q_{-2}, Q_{-3}) \boxplus m_0 \boxplus 1) \lll 3$$

So $Q_0^{[3]} \neq Q_0'^{[3]}$. Then

$$Q_1 = (Q_{-3} \boxplus \mathrm{IF}(Q_0, Q_{-1}, Q_{-2}) \boxplus m_1) \lll 7$$
$$Q'_1 = (Q_{-3} \boxplus \mathrm{IF}(Q'_0, Q_{-1}, Q_{-2}) \boxplus m_1) \lll 7$$

Thus, if $Q_{-1}^{[3]} \neq Q_{-2}^{[3]}$, we will have $\mathrm{IF}(Q_0, Q_{-1}, Q_{-2}) \neq \mathrm{IF}(Q'_0, Q_{-1}, Q_{-2})$ and $Q_1 \neq Q'_1$. On the other hand, if $Q_{-1}^{[3]} = Q_{-2}^{[3]}$ and there is no carry when going from $Q_0$ to $Q'_0$, then $Q_1 = Q'_1$. Therefore, collision paths where $Q_{-1}^{[3]} = Q_{-2}^{[3]}$ will be significantly different from collision paths where $Q_{-1}^{[3]} \neq Q_{-2}^{[3]}$. This suggests that the collision probability will be correlated with the condition (SC) : $Q_{-1}^{[3]} = Q_{-2}^{[3]}$, and we expect to be able to detect the bias. More precisely we believe that the case $Q_{-1}^{[3]} \neq Q_{-2}^{[3]}$ will give a much smaller collision probability, since it means that an extra difference is introduced in step 1.

To check this intuition experimentally, we ran one cMD4 round (16 steps) with a random $IV$ on message pairs $(M, M \boxplus 1)$ where $M$ was also picked at random, and we looked for pseudo-collisions in $Q_{12}...Q_{15}$ with the following properties:

- The weight of the non-adjacent form of the difference is lower or equal to 4.
- There is no difference on $Q_{12}^{[12]}$.

The second condition is required to eliminate the paths which simply keep the difference introduced in $Q_0^{[3]}$ without modifying it. We ran this with $5 \cdot 10^{11}$ random messages and IVs and found 45624 collisions out of which 45515 respected

the condition: this gives a ratio of about 420. This does not prove that we will have such a bias for collisions in the full MD4, but it is a strong evidence.

The same arguments apply when we introduce the message difference in another bit $k$ (*ie.* $\delta m_0 = 2^k$): we expect to find more collisions if $Q_{-1}^{[k \boxplus s_0]} = Q_{-2}^{[k \boxplus s_0]}$.

We ran a differential path search algorithm to find such paths, and we did find 22 paths for different values of $k$ with $Q_{-1}^{[k \boxplus s_0]} = Q_{-2}^{[k \boxplus s_0]}$. The path for $k = 0$ is given in Appendix D, and the other paths are just a rotation of this one. The corresponding set of sufficient conditions contains 79 conditions on the internal variables $Q_i$, so we expect that for a random message $M$:

$$\begin{aligned} \Pr\left[\mathrm{MD4}(M) = \mathrm{MD4}(M + \Delta)\right] = p \geq 2^{-79} \qquad & \text{if } Q_{-1}^{[k \boxplus s_0]} = Q_{-2}^{[k \boxplus s_0]} \\ \ll p \qquad & \text{if } Q_{-1}^{[k \boxplus s_0]} \neq Q_{-2}^{[k \boxplus s_0]} \end{aligned}$$

If we try $2^{82}$ message pairs per path, we will find a collision for every path whose condition is fulfilled with a probability[3] of more than 99%. Then we know 22 bits of the IV ($Q_{-1}^{[k \boxplus s_0]} = Q_{-2}^{[k \boxplus s_0]}$ or $Q_{-1}^{[k \boxplus s_0]} \neq Q_{-2}^{[k \boxplus s_0]}$), which leaves only $2^{106}$ IV candidates. To check if a given IV is the correct one, we just check whether it gives a collision on the pairs colliding with the real IV, so we expect to find the IV after computing $2^{105}$ pairs of hashes in an offline phase.

We show in Appendix B.2 how to reduce the search space to $2^{94}$ keys by extracting more than one bit of information when a collision is found. This gives an IV-recovery attack against MD4 with a data complexity of $2^{88}$ MD4 oracle queries, and a time complexity of $2^{94}$ MD4 evaluations.

### 4.2 Deriving a Composite IV-recovery Attack against MD4

To turn this into a composite IV-recovery attack, we need to efficiently compute message pairs $M$, $M'$ such that $H_{k_2}(M) = H_{k_2}(M') \boxplus \Delta$ (we will need $2^{82}$ such pairs). As we know $k_2$ (in the HMAC attack, we first recover it using the basic IV-recovery attack), we can compute $H_{k_2}(M)$ and find such pairs offline. If we do this naively using the birthday paradox, we need to hash about $2^{106}$ random messages to have all the pairs we need[4]. Then we can use the IV-recovery attack to get $k_1$.

Actually, we can do much better: we use the birthday paradox to find one pair of one-block messages $(R, R')$ such that $H_{k_2}(R') = H_{k_2}(R) \boxplus \Delta$, and then we extend it to a family of two-block message pairs such that $H_{k_2}(R'||Q') = H_{k_2}(R||Q) \boxplus \Delta$ with very little extra computation. In the end, the cost to generate the messages with $H_{k_2}(M) = H_{k_2}(M') \boxplus \Delta$ will be negligible and the composite IV-recovery attack is as efficient as the basic one. This is the most important part of our work: thanks to our new path, we only need a low level of control on the input of the hash function to extract the IV.

[3] We have $\left(1 - \left(1 - 2^{-79}\right)^{2^{82}}\right)^{22} > 0.992$

[4] This gives $2^{210}$ pairs of messages, and each pair has a probability of $2^{-128}$ to have the correct difference.

**Extending a Pair of Good Messages into a Family of Pairs.** Figure 1 shows how we will create many message pairs with $H_{k_2}(M) = H_{k_2}(M') \boxplus \Delta$. We use a pair of one-block message $(R, R')$ such that $H_{k_2}(R') = H_{k_2}(R) \boxplus \Delta$. Then we will generate a second block pair $(Q, Q')$ such that $H_{k_2}(R'||Q') \boxminus H_{k_2}(R||Q) = \Delta$. Thanks to the Davies-Meyer construction of the compression function, all that we need is a difference path which starts with a $\Delta$ difference and ends with a zero difference in the internal state; then the feed-forward of $H$ will keep $\delta H = \Delta$. This path can be found by a differential path search algorithm, or created by hand by slightly modifying a collision path.

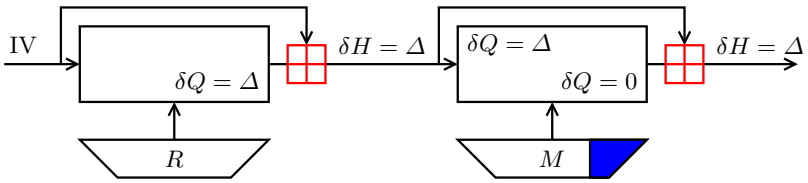


Fig. 1: Generating many pairs of message with a fixed hash difference

In this step, we also have to take care of the padding in MD4. Usually we ignore it because a collision at the end of a block is still a collision after an extra block of padding, but here we want a specific non-zero difference, and this will be broken by an extra block. So we have to adapt our collision finding algorithm to produce a block with a 55-byte message $M$ and the last 9 bytes fixed by the MD4 padding. This can be done with nearly the same complexity as unconstrained MD4 collisions (about 4 MD4 computations per collision) using the technique of Leurent [12]. Thus, the cost of the message generation in the composite IV-recovery attack drops from $2^{106}$ using the birthday paradox to $2^{90}$ and becomes negligible in the full attack.

### 4.3 MD4 Attack Summary

This attack uses the same IV-recovery attack for the inner key and the outer key, with a complexity of $2^{88}$ online queries and $2^{94}$ offline computations. We manage to keep the complexity of the composite IV-recovery as low as the basic IV-recovery because we only need to control the hash differences, and we introduce a trick to generate many messages with a fixed hash difference.

In Appendix B.1 we show how to reduce a little bit the query complexity of the attack, and in the end the NMAC full key-recovery attack requires $2^{88}$ requests to the oracle, and $2 \times 2^{94}$ offline computations.

## 5 Attacking NMAC-MD5

In this section, we will describe the attack of Contini and Yin [5], and we extend it to a full key recovery. This improved attack was independently found by Rechberger and Rijmen in [17].

As for MD4, the IV-recovery attack is based on a specific differential path, and assumes that when a collision is found with the given message difference, the $Q_i$'s follow the path. This gives some bits of the internal state already, and a kind of message modification technique to disclose more bits is proposed in [5].

If the path depends on the value of a bit $Q_t^{[k]}$ in the step $t$, then we can extract some extra bits from the register $Q_t$: set $\Delta = 2^{k-1}$ and modify the message $M$ into a message $M^*$:

$$m_j^* = \begin{cases} m_j & \text{if } j < t \\ m_j + \Delta & \text{if } j = t \\ \text{random} & \text{if } j > t \end{cases}$$

Then, using MD4 or MD5 step update without the rotation, we have:

$$Q_j^* = \begin{cases} Q_j & \text{if } j < t \\ Q_j + \Delta & \text{if } j = t \\ \text{random} & \text{if } j > t \end{cases}$$

We call the oracle with enough such messages (with a different random part) to distinguish if $Q_t^*$ still follows the path. If it does, this means that $Q_t^{*[k]} = Q_t^{[k]}$: there was no carry in $Q_t + \Delta$, therefore $Q_t^{[k-1]} = 0$. On the other hand, if it does not follow the path anymore, then $Q_t^{[k-1]} = 1$.

We can find $Q_t^{[k-2]}$ if we set $\Delta$ so that there is a carry up to the bit $k$ if and only if $Q_t^{[k-2]} = 1$:

$$\Delta = \begin{cases} 2^{k-2} & \text{if } Q_j^{[k-1]} = 1 \\ 2^{k-2} + 2^{k-1} & \text{if } Q_j^{[k-1]} = 0 \end{cases}$$

and we can repeat this to get the bits $Q_t^{[k-1]}...Q_t^{[0]}$.

The rotation will have little effect over this simplified explanation, it will mainly limit the number of bits we can recover (see [5] for details).

### 5.1 The IV-recovery Attack against MD5

The IV-recovery attack on MD5 is the same as the one presented in [5]. It uses the related-message technique with the pseudo-collision path of de Boer and Bosselaers [8]. Since the differences are in the IV and not in the message, the IV-recovery needs an oracle that answers whether $\text{MD5}_{\text{IV}}(M) = \text{MD5}_{\text{IV}'}(M)$, instead of the standard oracle that answers whether $\text{MD5}_{\text{IV}}(M) = \text{MD5}_{\text{IV}}(M')$.

To apply this to an HMAC key-recovery, we will have to use the related-key model: we need an oracle for $\text{NMAC}_{k_1,k_2}$, $\text{NMAC}_{k'_1,k_2}$ and $\text{NMAC}_{k_1,k'_2}$.

The IV-recovery attack in the related-key setting requires $2^{47}$ queries and $2^{45}$ hash computations, and this translates into a partial key-recovery (we will recover $k_2$) against NMAC-MD5 in the related-key model with the same complexity.

### 5.2 Deriving a Composite IV-recovery against MD5

To extend this to a composite IV-recovery attack, we run into the problem previously mentioned; to use this attack we need to create many inputs $N^*$ of the hash function related to one input $N$, but these inputs are the outputs of a first hash function, and we cannot choose them freely: $N = \text{MD5}_{k_2}(M)$. However, we know $k_2$, so we can compute many $N_R = H_{k_2}(R)$ for random messages $R$ and select those that are related to a particular $N$; if we want to recover bits of $Q_t$ we will have to choose $32(t+1)$ bits of $N_R$. We also run into the problem that any $N_R$ is only 128 bits long; the last 384 bits will be fixed by the padding and there are the same for all messages. Therefore, we can only use the related-message technique to recover bits of the internal state of in the very first steps, whereas in the simple IV-recovery it is more efficient to recover the internal state of later steps (Contini and Yin used step 11 to 14). If we want to recover bits of $Q_0$ (due to the rotation we can only recover 25 bits of them), we need to produce $24 \times 2^{45}$ messages $N^*$ with the first 32 bits chosen; this will cost $24 \times 2^{45} \times 2^{32} \approx 2^{82}$ hash computations. Then, we know 25 bits of $Q_0$, plus the most significant bit of $Q_1$, $Q_2$, and $Q_3$; we still have 100 bits to guess. Thus, we have a related-key composite IV-recovery attack against MD5 with $2 \times 24 \times 2^{45} \approx 2^{51}$ oracle queries and $2^{100}$ MD5 evaluations.

If we try to guess bits in $Q_1$, we have to select at least $2^{44}$ hashes with 64 chosen bits; this costs about $2^{108}$ MD5, so it does not improve the attack.

### 5.3 MD5 Attack Summary

Thus, the Contini-Yin NMAC-MD5 attack can be extended into a full key-recovery attack in the related-key setting, with a query complexity of $2^{51}$, a time complexity of $2^{100}$ MD5 operations, and success rate of $2^{-4}$ (due to the dBB condition for $k_1$ and $k_2$).

It is a very simple extension of the attack from Contini and Yin: we apply their technique to recover the outer key, but since we cannot choose the value of $H_{k_2}(M)$, we compute it for many random messages until we find a good one. This requires to change the step in which we extract internal bits, and the complexity become much higher.

## Acknowledgement

Part of this work is supported by the Commission of the European Communities through the IST program under contract IST-2002-507932 ECRYPT, and by the French government through the Saphir RNRT project.

## References

1. Amirazizi, H.R., Hellman, M.E.: Time-memory-processor trade-offs. IEEE Transactions on Information Theory **34**(3) (1988) 505–512
2. Bellare, M.: New Proofs for NMAC and HMAC: Security Without Collision Resistance. In Dwork, C., ed.: CRYPTO. Volume 4117 of Lecture Notes in Computer Science., Springer (2006) 602–619
3. Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication. In Koblitz, N., ed.: CRYPTO. Volume 1109 of Lecture Notes in Computer Science., Springer (1996) 1–15
4. Cannière, C.D., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications . [11]
5. Contini, S., Yin, Y.L.: Forgery and Partial Key-Recovery Attacks on HMAC and NMAC Using Hash Collisions. [11]
6. Cramer, R., ed.: Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of Lecture Notes in Computer Science., Springer (2005)
7. Daum, M.: Cryptanalysis of Hash Functions of the MD4-Family. PhD thesis, Ruhr-University of Bochum (2005)
8. den Boer, B., Bosselaers, A.: Collisions for the Compression Function of MD5. In: Proc. EUROCRYPT '93. (1993) 293–304
9. Fouque, P.A., Leurent, G., Nguyen, P.: Automatic Search of Differential Path in MD4. ECRYPT Hash Worshop – Cryptology ePrint Archive, Report 2007/206 (2007) `http://eprint.iacr.org/`.
10. Kim, J., Biryukov, A., Preneel, B., Hong, S.: On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1. In Prisco, R.D., Yung, M., eds.: SCN. Volume 4116 of Lecture Notes in Computer Science., Springer (2006) 242–256
11. Lai, X., Chen, K., eds.: 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006. . In Lai, X., Chen, K., eds.: ASIACRYPT. Volume 4284 of Lecture Notes in Computer Science., Springer (2006)
12. Leurent, G.: Message Freedom in MD4 and MD5: Application to APOP Security. In Biryukov, A., ed.: FSE. To appear in LNCS, Springer (2007)
13. Preneel, B., van Oorschot, P.C.: MDx-MAC and Building Fast MACs from Hash Functions. In Coppersmith, D., ed.: CRYPTO. Volume 963 of Lecture Notes in Computer Science., Springer (1995) 1–14
14. Preneel, B., van Oorschot, P.C.: On the Security of Two MAC Algorithms. In: EUROCRYPT. (1996) 19–32
15. Preneel, B., van Oorschot, P.C.: On the Security of Iterated Message Authentication Codes. IEEE Transactions on Information Theory **45**(1) (1999) 188–199
16. Rechberger, C., Rijmen, V.: Note on Distinguishing, Forgery, and Second Preimage Attacks on HMAC-SHA-1 and a Method to Reduce the Key Entropy of NMAC. Cryptology ePrint Archive, Report 2006/290 (2006) `http://eprint.iacr.org/`.

17. Rechberger, C., Rijmen, V.: On Authentication with HMAC and Non-Random Properties. In Dietrich, S., ed.: Financial Cryptography. To appear in LNCS, Springer (2007)
18. Schläffer, M., Oswald, E.: Searching for Differential Paths in MD4. In Robshaw, M., ed.: FSE. Volume 4047 of Lecture Notes in Computer Science., Springer (2006) 242–261
19. Shoup, V., ed.: Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005)
20. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. [6] 1–18
21. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. [19] 17–36
22. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. [6] 19–35
23. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. [19] 1–16
24. Yu, H., Wang, G., Zhang, G., Wang, X.: The Second-Preimage Attack on MD4. In Desmedt, Y., Wang, H., Mu, Y., Li, Y., eds.: CANS. Volume 3810 of Lecture Notes in Computer Science., Springer (2005) 1–12

## A NMAC Security beyond the Birthday Paradox

In this section we study the security of NMAC when the attacker can make enough queries to use the birthday paradox. There is a generic existential forgery attack, but we will show that universal forgery against NMAC requires more resources. Actually, we will prove a much stronger statement: NMAC is secure against short forgery (less than one block) if the underlying compression function is a PRF.

We define the 1-block MAC advantage of a MAC-adversary as (using the same notations as [2], $B$ is the set of 1-block messages):

$$\mathbf{Adv}_f^{\text{1-MAC}}(A) = \Pr\left[\mathtt{pad}(M) \in B,\ (M,x) \text{ is a forgery}:\ (M,x) \leftarrow A^{f(K,\cdot),VF_f(K,\cdot,\cdot)}; K \xleftarrow{\$} Keys\right]$$

Note that the attacker is allowed to make any query of any length to the MAC oracle, we only limit the length of the forgery. We require that the padded message fit in one block, so the actual message has to be somewhat smaller; for instance, in the case of MD4/MD5, the forged message has to be 447 bits or less.

The only generic short forgery attack we are aware of is the key-recovery attack using the birthday paradox which require $2^{n/2}$ queries and a time of $2^{n+1}$ (the generic forgery attack against iterated MAC produces at least a two-block forgery). On the other hand, short message forgeries are possible if the compression function is weak: using the partial key-recoveries attacks of Contini and Yin to find $k_2$, one can compute a pair of short messages $(M_1, M_2)$ such that $H_{k_2}(M_1) = H_{k_2}(M_2)$ and forge a MAC for one of them. This pair can be found using the birthday paradox in time $2^{n/2}$, or in the case of MD4, in time about $2^9$ using the results of [12] to include the padding in the one-block message.

Obviously, this kind of attack is stronger than a PRF-attack against HMAC, and weaker than a universal forgery or a key-recovery. The following theorem proves that such short forgeries are as hard as a PRF attack against the compression function, *ie.* it requires a time of the order of $2^n$ if there is no weakness in the compression function.

**Theorem 1.** *Let $A_{\mathrm{NMAC}}$ be a 1-block MAC-adversary against NMAC instantiated with the compression function $h$. Then, there exist PRF-adversaries $A_1$ and $A_2$ against $h$ such that:*

$$\mathbf{Adv}^{\text{1-MAC}}_{\mathrm{NMAC}}(A_{\mathrm{NMAC}}) \leq \mathbf{Adv}^{\mathrm{prf}}_{h}(A_1) + \mathbf{Adv}^{\mathrm{prf}}_{h}(A_2) + 2^{-n+1}$$

*Furthermore, $A_1$ and $A_2$ have the same time complexity as $A_{\mathrm{NMAC}}$, and just make one extra oracle query.*

*Proof.* We will build the adversary $A_1$ from $A_{\mathrm{NMAC}}$ by simulating the NMAC oracle. We are given a function $g$, and we try to guess if $g$ is a random function or $h(K,\cdot)$ for some $K$. First we choose a random $k_2$, and we will then answer NMAC queries with $g(H_{k_2}(\cdot))$. $A_{\mathrm{NMAC}}$ will output some message $M$ and a tag $x$, and we guess that $g$ is a $f(K,\cdot)$ iff $x = g(H_{k_2}(M))$. Thus, we have:

- If $g$ was a random function, $x = g(H_{k_2}(M))$ holds with a very low probability:
  - If we did not query the oracle $g$ with the input $H_{k_2}(M)$ yet, then $g(H_{k_2}(M))$ is random and it will be equal to $x$ with probability $2^{-n}$.
  - If $M$ collides under $H_{k_2}$ with one of the $q$ NMAC queries made by $A_{\mathrm{NMAC}}$, this means we can build a PRF-attacker $A_2$ against $h$ from $A_{\mathrm{NMAC}}$: we answer NMAC queries using a random oracle for $H_{k_1}$ and the given $g'$ to compute the inner hash function (it will require one call to the oracle $g'$ and some computations of $f$), then we output 1 iff $H_{k_2}(M) \in Q$, where $Q$ denotes the set of $H_{k_2}$ outputs computed by $A_2$. If $g$ was a random function, there is a negligible probability that we answer 1, and if $g$ if some $h(K,\cdot)$ we will recognize it whenever $A_{\mathrm{NMAC}}$ succeeds to forge. Hence:

$$\begin{aligned}
\Pr\left[A_2^{\$} \Rightarrow 1\right] &\leq 2^{-n} \\
\Pr\left[A_2^{f(K,\cdot)} \Rightarrow 1\right] &\geq \Pr\left[A_1^{\$} \Rightarrow 1, f(K,M) \in Q\right] \\
\Pr\left[A_1^{\$} \Rightarrow 1, f(K,M) \in Q\right] &\leq \mathbf{Adv}^{\mathrm{prf}}_{h}(A_2) + 2^{-n}
\end{aligned}$$

  By combining these two cases, we have:

$$\begin{aligned}
\Pr\left[A_1^{\$} \Rightarrow 1\right] &= \Pr\left[A_1^{\$} \Rightarrow 1, f(K,M) \in Q\right] + \Pr\left[A_1^{\$} \Rightarrow 1, f(K,M) \notin Q\right] \\
&\leq \mathbf{Adv}^{\mathrm{prf}}_{h}(A_2) + 2^{-n} + 2^{-n}
\end{aligned}$$

- If $g$ is $h(K,\cdot)$, we will correctly output 1 if $A_{\mathrm{NMAC}}$ makes a correct forgery, and we finish the proof by combining the two cases:

$$\begin{aligned}
\Pr\left[A_1^{f(K,\cdot)} \Rightarrow 1\right] &\geq \mathbf{Adv}^{\text{1-MAC}}_{\mathrm{NMAC}}(A_{\mathrm{NMAC}}) \\
\mathbf{Adv}^{\text{1-MAC}}_{\mathrm{NMAC}}(A_{\mathrm{NMAC}}) &\leq \mathbf{Adv}^{\mathrm{prf}}_{h}(A_1) + \mathbf{Adv}^{\mathrm{prf}}_{h}(A_2) + 2^{-n+1}
\end{aligned}$$

## B Improving the MD4 IV-recovery

### B.1 Reducing the Online Cost

First, we can easily lower the number of calls to the NMAC-oracle in the first phase of the IV-recovery. Instead of trying $22 \times 2^{82}$ random message pairs, we will choose the messages more cleverly so that each message belongs to 22 pairs: we first choose 490 bits of the message at random and then use every possibility for the 22 remaining bits. Thus, we only need $2^{83}$ calls to the oracle instead of $22 \times 2^{83}$.

Note that we cannot use this trick in the composite IV-recovery attack, so the number of queries for the full key-recovery will only be halved (the queries for the basic IV-recovery for $k_2$ become negligible compared to the queries for the composite IV-recovery that will reveal $k_2$).

### B.2 Reducing the Offline Cost

We may also lower the computational cost of the attack, by getting more than one bit of the IV once a collision has been found. This will require the extra assumption the colliding messages follow the differential path in step 1 (previously we only needed step 0), but this seems quite reasonable, for the same reasons. Out of the 22 paths used to learn IV bits, let $p$ be the number of paths for which the condition holds, and a collision is actually found. From each message that collides following the differential path, we can also extract some conditions on the internal states $Q_0$ and $Q_1$. These states are not part of the IV, but since we know the message used, we can use these conditions to learn something on the IV. If we have a message pair that collides with $M' \boxminus M = 2^k$, we will call them $M^{(k)}$ and $M'^{(k)}$ and the condition gives us $Q_0^{[k \boxplus s_0]}(M^{(k)})$ and $Q_1^{[k \boxplus s_0]}(M^{(k)})$. The idea is the following (the symbol '➤' summarizes the number of bits to guess at each step):

1. We guess $Q_{-1}$. Let $n = 32 - |Q_{-1}|$ be the number of `0` bits in $Q_{-1}$ (we use $|x|$ to denote the Hamming weight of $x$).
2. We compute 22 bits of $Q_{-2}$ using the conditions on the IV, and we guess the others ➤ 10 bits.
3. We guess the bits of $Q_{-3}$ used to compute $Q_0$. Since we have $Q_0 = (Q_{-4} \boxplus \mathrm{IF}(Q_{-1}, Q_{-2}, Q_{-3}) \boxplus k_0 \boxplus m_0) \lll s_0$, we only need $Q_{-3}^{[i]}$ when $Q_{-1}^{[i]} = 0$ ➤ $n$ bits.
4. We have $Q_{-4} = (Q_0 \ggg s_0) \boxminus \mathrm{IF}(Q_{-1}, Q_{-2}, Q_{-3}) \boxminus m_0 \boxminus k_0$. If we use it with the message $M^{(0)}$ and take the equation modulo 2, it becomes: $Q_{-4}^{[0]} = Q_0^{[s_0]}(M^{(0)}) \boxminus (\mathrm{IF}(Q_{-1}, Q_{-2}, Q_{-3}) \boxminus m_0^{(0)} \boxminus k_0) \bmod 2$, and it gives us $Q_{-4}^{[0]}$. Then, if we write the equation with $M^{(1)}$ and take it modulo 2, we will learn $Q_0^{[s_0]}(M^{(1)})$ from $Q_{-4}^{[0]}$. Since we know $(Q_0(M^{(1)}) \lll s_0) \bmod 4$ from $Q_0^{[s_0]}(M^{(1)})$ and $Q_0^{[1 \boxplus s_0]}(M^{(1)})$, we can take the equation modulo 4 to learn $Q_{-4}^{[1]}$.

By repeating this process, we learn the full $Q_{-4}$, but we need to guess the bit $i$ when we don't have a message pair $M^{(i)}, M'^{(i)}$ ➤ $32-p$ bits.

5. We apply the same process to compute the remaining bits of $Q_{-3}$. We already know $n$ bits and we expect to be able to compute a ratio of $p/32$ of the missing ones. ➤ $\frac{(32-n)(32-p)}{32}$ bits.

So, for each choice of $Q_{-1}$ in step 1, we have to try a number of choices for the other bits that depends on the Hamming weight $32-n$ of $Q_{-1}$. In the end, the number of keys to try is:

$$\sum_{Q_{-1}} 2^{10+n+32-p+(32-n)(32-p)/32} = 2^{74-2p}\left(1+2^{p/32}\right)^{32}$$

With $p=11$, this becomes a little less than $2^{90}$, but the complexity depends on the number of conditions fulfilled by the key. If we assume that every condition has a probability of one half to hold, we can compute the average number of trials depending on the keys, and we will have to try half of them:

$$\frac{1}{\#k}\sum_{k} 2^{74-2p}\left(1+2^{p/32}\right)^{32} < 2^{93.8}$$

Hence, we have an IV-recovery attack requiring less than $2^{88}$ queries to the NMAC oracle, and less than $2^{94}$ offline hash computations. See the full version of this paper for a detailed complexity analysis.

### B.3 Complexity Details

First, we will have to compute sums of Hamming weight power:

$$\begin{aligned}
S_n(\alpha) &= \sum_{x\in\mathbb{Z}_2^n} \alpha^{|x|} \\
S_1(\alpha) &= \alpha^0+\alpha^1 && = 1+\alpha \\
S_{n+1}(\alpha) &= S_n(\alpha)+\alpha S_n(\alpha) && = (1+\alpha)S_n(\alpha) \\
S_n(\alpha) &= (1+\alpha)^n
\end{aligned}$$

This allows us to compute the total complexity for every $Q_{-1}$:

$$\begin{aligned}
\sum_{Q_{-1}} 2^{10+n+32-p+(32-n)(32-p)/32} &= 2^{74-2p}\sum_{Q_{-1}} 2^{np/32} \\
&= 2^{74-2p}S_{32}\left(2^{p/32}\right) \\
&= 2^{74-2p}\left(1+2^{p/32}\right)^{32}
\end{aligned}$$

At the end, we need to compute the average complexity over the keys: we will reduce it to a sum only over the 22 bits of $k$ related to the conditions counted

in $p$.

$$\begin{aligned}
\frac{1}{\#k}\sum_k 2^{74-2p}\left(1+2^{p/32}\right)^{32} &= \frac{2^{74}}{2^{22}}\sum_{k'\in\mathbb{Z}_{2^{22}}} 2^{-2|k'|}\left(1+2^{|k'|/32}\right)^{32} \\
&= 2^{52}\sum_{k'\in\mathbb{Z}_{2^{22}}} 2^{-2|k'|}\sum_{i=0}^{32}\binom{32}{i}2^{|k'|i/32} \\
&= 2^{52}\sum_{i=0}^{32}\binom{32}{i}S_{22}\left(2^{i/32-2}\right) \\
&= 2^{52}\sum_{i=0}^{32}\binom{32}{i}\left(1+2^{i/32-2}\right)^{22} \\
&< 2^{93.8}
\end{aligned}$$

# C The Differential Path Search Algorithm

In this section we give a quick description of our differential path search algorithm. This section is only intended for the interested reader, it is absolutely not required to read it in order to understand our attack against HMAC/NMAC. This algorithm was only used to find the path given in Appendix D.

Automated search techniques for differential paths have appeared before: see Schläffer and Oswald [18] for MD4, and De Cannière and Rechberger [4] for SHA-1. However, such techniques were targetting improved collision search. Our work confirm the claim of [5] that automated search methods can lead to better attacks on HMAC and NMAC.

## C.1 Notations and basic idea of the algorithm

Here, we use $\delta(x,y) = y \boxminus x$ to denote the modular difference and $\partial(x,y) = \langle y^{[31]} - x^{[31]}, y^{[30]} - x^{[30]}, ...y^{[1]} - x^{[1]}, y^{[0]} - x^{[0]}\rangle$ to denote Wang's difference. We will use ▲ and ▼ to represent $+1$ and $-1$, and we will give a compact representation by omitting the zeroes, and grouping the bits, eg. $\langle ▲^{[0]}, ▼▲^{[3,4]}, ▲▲^{[30,31]}\rangle$.

We will consider two messages $M$ and $M'$, and we use a prime to represent any variable related to the message $M'$ (*eg.* $Q'_i$, $m'_i$). As a shortcut, we will sometimes use $\delta X$ (resp. $\partial X$) to represent $\delta(X,X')$ (resp. $\partial(X,X')$), and $\Phi_i$ for $\Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3})$.

When we are given a differential path, we will call it $\partial_i$, and a message follows the path if $\partial Q_i = \partial_i$ holds for every step $i$. We will also use $\delta_i$ as the desired value of $\delta Q_i$.

Our algorithm is based on the sufficient conditions (SC) algorithm. The basic idea is to run the SC computation, but since we do not know $\delta_i$ nor $\delta\Phi_i$, we will assume that $\delta\Phi_i = 0$, which gives $\delta_i = \delta_{i+4}^{\ggg} \boxminus \Delta_{i+4}$: the differences will only appear every 4 turns, and will not propagate in between. This is possible in the

first two rounds, because the boolean functions IF and MAJ can absorb one input difference. Using this basic idea, we find a path with a non-zero difference in $Q_{-4}...Q_{-1}$, that is, a path leading to pseudo-collisions (this initial path is called $\epsilon$ in the algorithm).

Then we will run another pass of the algorithm, but we will try to modify the path so as to lower the number of differences in the IV. In fact, we will have a set of paths $\mathcal{P}$, and every run will select a path, try to improve it, and insert new paths in this set. This basic structure is described in Algorithm 1: we will make an extensive use of recursivity to explore the path space. This algorithm will be referred to as the DP algorithm.

**Algorithm 1** Overview of the differential path search algorithm

```
1: function PathFind
2:     P ← {ε}                                   ▷ ε is the path with δΦ_i = 0
3:     loop
4:         extract P from P
5:         PathStep(P,ε,48)                      ▷ start search from last step
6: function PathStep(P_0,P,i)                    ▷ Extend path P to step i, following P_0
7:     if i < 0 then
8:         add P to P
9:     else
10:        for all possible choice P' do
11:            PatchTarget(P_0,P',i)
12: function PatchTarget(P_0,P,i)                ▷ Modify P to fix IV differences in the end
13:    for all possible choice P' do
14:        PatchCarries(P_0,P',i)
15: function PatchCarries(P_0,P,i)               ▷ Extend some carries to help the next steps
16:    for all possible choice P' do
17:        PathStep(P_0,P',i − 1)
```

### C.2 Path representation.

During the computation of a path, we represent the path as $\langle \partial Q_i \rangle_{i=0}^{48}$, where each $\partial Q_i$ is given as 32 values in $\{-1, 0, +1\}$. However, between two passes, this representation is almost useless: when we apply a local modification to a $\partial Q_i$, the $\partial Q_j$'s for the rest of the path will become quite different.

Therefore we propose a new representation of the path: we will store $\langle \delta\Phi_i \rangle_{i=0}^{48}$. The $\partial Q_i$'s can be efficiently computed from the $\delta\Phi_i$'s, even if there is a little loss of information: a given $\langle \delta\Phi_i \rangle_{i=0}^{48}$ can correspond to many $\langle \partial_i \rangle_{i=0}^{48}$ (for instance using different carry extensions), but the algorithm quickly find a good one. The main advantage of this representation is that a local modification of $\delta\Phi_i$ will not modify the other $\delta\Phi_j$, and we recompute the full path $\langle \partial Q_i \rangle_{i=0}^{48}$. In fact, since $\partial\Phi_i = 0$ most of the time, this is a much better description of the path: it tells us where we have to do something unusual.

### C.3 Overview of the algorithm.

The function PATHSTEP will extend the path one step further, using the same ideas as the SC algorithm at step $i+4$. It assumes the $\partial Q_j$'s and $\delta\Phi_j$'s are chosen for $j > i$. Then, for every possible choice of $\delta^{\lll}_{i+4}$, it will compute $\delta Q_i$ from $\partial Q_{i+4}$ and $\partial\Phi_{i+4}$ and add the $\lll$-conditions and $\Phi$-conditions. It will have to choose a $\partial\Phi_{i+4}$ matching $\delta\Phi_{i+4}$ that is feasible given $\partial Q_{i+1}$, $\partial Q_{i+2}$, and $\partial Q_{i+3}$; if none is available, this branch of the search is aborted. Here we will also set $\delta\Phi_i$ to the value it had in the path $P_0$, so that the new path is similar the old one.

The function PATCHTARGET will then modify $\partial\Phi_i$ so as to remove some unwanted differences in the IV (trying to turn a pseudo-collision path into a collision path).

To finish the step $i$, the function PATCHCARRIES will select a $\partial Q_i$ corresponding to $\delta Q_i$, and will extend some carries according to the values $\delta\Phi_{i+1}$, $\delta\Phi_{i+2}$ and $\delta\Phi_{i+3}$. This step is important because we need a non-zero bit in a $\partial Q_{j-1}$, $\partial Q_{j-2}$ or $\partial Q_{j-3}$ for every non-zero bit in $\partial\Phi_j$. Then it will add the $\partial$-conditions.

### C.4 Correcting Differences.

The critical part of the algorithm is the computation of the bits to modify in step $i$ so as to correct a difference in the IV. To change directly a bit $Q_{i_0}^{[k]}$, we will set a non-zero difference in $\Phi_{i_0}^{[k\boxminus s_{i_0}]}$. However, we detect the differences in the IV, and we can't fix them here; we will have to act on a different step and see how the difference evolves. The simplest way to do so is to keep $\delta\Phi_i$ unmodified in the rest of the path, which is possible if the difference is absorbed by the $\Phi_i$'s. So we will try to use bit $Q_{i_0+4}^{[k\boxplus s_{i_0}]}$ to modify bit $Q_{i_0}^{[k]}$, and so on until we find a bit of $Q_{i_0+4k}$ which can be changed using $\Phi$.

When such a modification succeeds, it will remove one difference in the IV. This simple correction method is already useful: it finds the path from [24], but not the one from [20].

### C.5 Indirect Correction.

While searching for more complex paths, we will have some differences in the IV which cannot be dealt this way. So we will introduce a difference which will not directly cancel the difference in the IV, but which will allow us to remove the target difference using the previous method. More precisely, to fix $Q_{i_0}^{[k]}$, we want a difference in some $Q_{i_0+4k}$, but we need a difference in the inputs of $\Phi_{i_0+4k}$; so we will try to introduce a difference in $Q_{i_0+4k+a}$, where $a \in \{1, 2, 3\}$, and this will use $\Phi_{i_0+4k+a+4k'}$. See Algorithm 2 for a pseudo-code description.

When this succeeds, it removes the target difference, but it introduces a new unwanted difference. Hopefully, we may remove this new difference without indirect modifications... This method works rather well, and finds many paths using the message difference from [20].

**Algorithm 2** Details on the bit correcting part of the algorithm

```
1: function PatchTarget(P_0,P,i)
2:     for all Q_{i_0}^{[k]} bit to fix in P_0 do          ▷ we try every difference, one by one
3:         PatchTargetBit(P_0,P,i,i_0,k,η_0)
4: function PatchTargetBit(P_0,P,i,i_0,k,η)              ▷ η indirect modifications allowed
5:     if i < i_0 then return
6:     else if i = i_0 then
7:         modify P on bit k of step i
8:         PatchCarries(P_0, P, i)                       ▷ next step of the algorithm
9:     else
10:        PatchTargetBit(P_0, P, i, i_0 + 4, k + s_{i_0} mod 32, η)   ▷ Direct correction
11:    if η > 0 then
12:        modify P_0 on bit k of step i_0               ▷ Indirect correction
13:        for a ∈ {1, 2, 3} do PatchTargetBit(P_0,P,i,i_0 + a,k,η − 1)
```

### C.6 Impossible paths.

As we compute the differential path and the sufficient conditions at the same time, we do not have to deal with impossible path, during the execution of the algorithm: if a modification of the paths leads to an impossibility, we abort the search and look for other modifications. However, if the path with $\delta\Phi_i = 0$ is impossible – and this is the case if there are some differences in the third round[5] – the first pass of the algorithm will abort with an incomplete path. Therefore we also add incomplete paths to the set $\mathcal{P}$, and we correct their errors in the same ways we correct differences in the IV.

## D IV-dependent Differential Path

Here is one of the 22 IV-dependent paths we found in MD4. The 22 paths can be deduced from this one by rotating all the bit differences and bit conditions: it works on bit positions 0, 1, 3, 4, 6-8, 12-17, 19-24, 26, 27, and 29, and fails on other positions due to carry expansions.

This path was found using an automated differential paths search algorithm described in [9].

[5] for Wang's EUROCRYPT path, the differences in the third round form a local collisions, so we can as well run the algorithm only for the first two rounds.

| step | $s_i$ | $\delta m_i$ | $\partial\Phi_i$ | $\partial Q_i$ | $\Phi$-conditions and $\lll$-conditions |
|---|---|---|---|---|---|
| 0 | 3 | $\langle \blacktriangle^{[0]} \rangle$ | | $\langle \blacktriangle^{[3]} \rangle$ | |
| 1 | 7 | | | | $Q_{-1}^{[3]} = Q_{-2}^{[3]}$ |
| 2 | 11 | | | | $Q_{1}^{[3]} = 0$ |
| 3 | 19 | | | | $Q_{2}^{[3]} = 1$ |
| 4 | 3 | | | $\langle \blacktriangledown\blacktriangle^{[6,7]} \rangle$ | |
| 5 | 7 | | | | $Q_{3}^{[6]} = Q_{2}^{[6]}$, $Q_{3}^{[7]} = Q_{2}^{[7]}$ |
| 6 | 11 | | | | $Q_{5}^{[6]} = 0$, $Q_{5}^{[7]} = 0$ |
| 7 | 19 | | $\langle \blacktriangle^{[7]} \rangle$ | $\langle \blacktriangle^{[26]} \rangle$ | $Q_{6}^{[6]} = 1$, $Q_{6}^{[7]} = 0$ |
| 8 | 3 | | $\langle \blacktriangledown^{[26]} \rangle$ | $\langle \blacktriangle^{[9]}, \blacktriangledown^{[29]} \rangle$ | $Q_{5}^{[26]} = 1$, $Q_{6}^{[26]} = 0$ |
| 9 | 7 | | | | $Q_{7}^{[9]} = Q_{6}^{[9]}$, $Q_{8}^{[26]} = 0$, $Q_{7}^{[29]} = Q_{6}^{[29]}$ |
| 10 | 11 | | | | $Q_{9}^{[9]} = 0$, $Q_{9}^{[26]} = 1$, $Q_{9}^{[29]} = 0$ |
| 11 | 19 | | | $\langle \blacktriangle^{[13]} \rangle$ | $Q_{10}^{[9]} = 1$, $Q_{10}^{[29]} = 1$ |
| 12 | 3 | | | $\langle \blacktriangledown^{[0]}, \blacktriangle^{[12]} \rangle$ | $Q_{10}^{[13]} = Q_{9}^{[13]}$ |
| 13 | 7 | | | | $Q_{11}^{[0]} = Q_{10}^{[0]}$, $Q_{11}^{[12]} = Q_{10}^{[12]}$, $Q_{12}^{[13]} = 0$ |
| 14 | 11 | | $\langle \blacktriangledown^{[0]} \rangle$ | $\langle \blacktriangle\blacktriangledown^{[11\ldots13]} \rangle$ | $Q_{13}^{[0]} = 1$, $Q_{13}^{[12]} = 0$, $Q_{13}^{[13]} = 1$ |
| 15 | 19 | | $\langle \blacktriangledown^{[13]} \rangle$ | | $Q_{14}^{[0]} = 1$, $Q_{13}^{[11]} = Q_{12}^{[11]}$, $Q_{13}^{[12]} = 0$, $Q_{13}^{[13]} = 1$, $Q_{12}^{[13]} = 0$ |
| 16 | 3 | $\langle \blacktriangle^{[0]} \rangle$ | $\langle \blacktriangle\blacktriangledown^{[12,13]} \rangle$ | | $Q_{15}^{[11]} = Q_{13}^{[11]}$, $Q_{15}^{[12]} \neq Q_{13}^{[12]}$, $Q_{15}^{[13]} \neq Q_{13}^{[13]}$ |
| 17 | 5 | | | | $Q_{16}^{[11]} = Q_{15}^{[11]}$, $Q_{16}^{[12]} = Q_{15}^{[12]}$, $Q_{16}^{[13]} = Q_{15}^{[13]}$ |
| 18 | 9 | | | $\langle \blacktriangle\blacktriangle\blacktriangle\blacktriangledown^{[20\ldots23]} \rangle$ | |
| 19 | 13 | | | | $Q_{17}^{[20]} = Q_{16}^{[20]}$, $Q_{17}^{[21]} = Q_{16}^{[21]}$, $Q_{17}^{[22]} = Q_{16}^{[22]}$, $Q_{17}^{[23]} = Q_{16}^{[23]}$ |
| 20 | 3 | | $\langle \blacktriangledown^{[23]} \rangle$ | $\langle \blacktriangledown^{[26]} \rangle$ | $Q_{19}^{[20]} = Q_{17}^{[20]}$, $Q_{19}^{[21]} = Q_{17}^{[21]}$, $Q_{19}^{[22]} = Q_{17}^{[22]}$, $Q_{19}^{[23]} \neq Q_{17}^{[23]}$ |
| 21 | 5 | | | | $Q_{20}^{[20]} = Q_{19}^{[20]}$, $Q_{20}^{[21]} = Q_{19}^{[21]}$, $Q_{20}^{[22]} = Q_{19}^{[22]}$, $Q_{20}^{[23]} = Q_{19}^{[23]}$, $Q_{19}^{[26]} = Q_{18}^{[26]}$ |
| 22 | 9 | | | $\langle \blacktriangledown^{[29]} \rangle$ | $Q_{21}^{[26]} = Q_{19}^{[26]}$ |
| 23 | 13 | | | | $Q_{22}^{[26]} = Q_{21}^{[26]}$, $Q_{21}^{[29]} = Q_{20}^{[29]}$ |
| 24 | 3 | | | $\langle \blacktriangle\blacktriangledown^{[29,30]} \rangle$ | $Q_{23}^{[29]} = Q_{21}^{[29]}$ |
| 25 | 5 | | | | $Q_{23}^{[30]} = Q_{22}^{[30]}$ |
| 26 | 9 | | $\langle \blacktriangle^{[29]} \rangle$ | | $Q_{25}^{[29]} \neq Q_{23}^{[29]}$, $Q_{25}^{[30]} = Q_{23}^{[30]}$ |
| 27 | 13 | | | | $Q_{26}^{[29]} = Q_{25}^{[29]}$, $Q_{26}^{[30]} = Q_{25}^{[30]}$ |
| 28 | 3 | | | $\langle \blacktriangledown^{[0]} \rangle$ | |
| 29 | 5 | | | | $Q_{27}^{[0]} = Q_{26}^{[0]}$ |
| 30 | 9 | | | | $Q_{29}^{[0]} = Q_{27}^{[0]}$ |
| 31 | 13 | | | | $Q_{30}^{[0]} = Q_{29}^{[0]}$ |
| 32 | 3 | $\langle \blacktriangle^{[0]} \rangle$ | | | |

Path 1: A path with the message difference on the first word.