Cryptanalyses on a Merkle-Damgård Based MAC — Almost Universal Forgery and Distinguishing-H Attacks

Yu Sasaki

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

Abstract. This paper presents two types of cryptanalysis on a Merkle-Damgård hash based MAC, which computes a MAC value of a message M by $\mathrm{Hash}(K\|\ell\|M)$ with a shared key K and the message length ℓ. This construction is often called LPMAC. Firstly, we present a distinguishing-H attack against LPMAC instantiating any narrow-pipe Merkle-Damgård hash function with $O(2^{n/2})$ queries, which indicates the incorrectness of the widely believed assumption that LPMAC instantiating a secure hash function should resist the distinguishing-H attack up to 2^n queries. In fact, all of the previous distinguishing-H attacks considered dedicated attacks depending on the underlying hash algorithm, and most of the cases, reduced rounds were attacked with a complexity between $2^{n/2}$ and 2^n. Because it works in generic, our attack updates these results, namely full rounds are attacked with $O(2^{n/2})$ complexity. Secondly, we show that an even stronger attack, which is a powerful form of an almost universal forgery attack, can be performed on LPMAC. In this setting, attackers can modify the first several message-blocks of a given message and aim to recover an internal state and forge the MAC value. For any narrow-pipe Merkle-Damgård hash function, our attack can be performed with $O(2^{n/2})$ queries. These results show that the length prepending scheme is not enough to achieve a secure MAC.

Keywords: LPMAC, distinguishing-H attack, almost universal forgery attack, multi-collision, diamond structure, prefix freeness

1 Introduction

Message Authentication Code (MAC) is a cryptographic technique which produces the integrity of the data and the authenticity of the communication player. MACs take a message and a key as input and compute a MAC value which is often called tag. Suppose that a sender and a receiver share a secret key K in advance. When the sender sends a message M to the receiver, he computes a tag σ and sends a pair of (M, σ). The receiver computes a tag by using the shared key K and the received M. If the result matches with the received σ, he knows that he received the correct message and it was surely sent by the sender.

MACs are often constructed by using block-ciphers or hash functions. There are three basic MAC constructions based on hash functions which were analyzed by Tsudik [1]. Let $\mathcal{H}$ be a hash function. A *secret-prefix* method computes a tag of a message M by $\mathcal{H}(K\|M)$. A *secret-suffix* method computes a tag by $\mathcal{H}(M\|K)$. A *hybrid* method computes a tag by $\mathcal{H}(K\|M\|K)$. Among the above three, the secret-prefix method is known to be vulnerable when $\mathcal{H}$ processes M block by block by iteratively applying a compression function h. This attack is called *padding attack* in [1] and *length-extension attack* in the recent SHA-3 competition [2]. Assume that the attacker obtains the tag σ for a message M. He then, without knowing the value of K and M, can compute a tag σ' for a message $M\|z$ for any z by computing $\sigma' \leftarrow h(\sigma, z)$.

LPMAC was suggested to avoid the vulnerability of the secret-prefix method [1][1]. In LPMAC, the length of the message to be hashed is prepended before the message is hashed, that is to say, $\sigma \leftarrow \mathcal{H}(K\|\ell\|M)$, where ℓ is the length of M. $K\|\ell$ is often padded to be a multiple of the block-length so that the computation of M can start from a new block. The length prepending scheme can be regarded as a concrete construction of the prefix freeness introduced by Bellare *et al.* [4]. Therefore, by [4], LPMAC was proven to be a secure pseudo-random function (PRF) up to $O(2^{n/2})$ queries.

The security of MAC is usually discussed with respect to the resistance against the following forgery attacks. An *existential forgery attack* creates a pair of valid (M, σ) for a message M which is not queried yet. A *selective forgery attack* creates a pair of valid (M, σ) where M is chosen by the attacker prior to the attack. A *universal forgery attack* creates a pair of valid (M, σ) where M can be any message chosen prior to the attack. Variants of these forgery attacks can also be considered. For example, Dunkelman *et al.* introduced an *almost universal forgery attack* [5] against the ALRED construction, which the attacker can modify one message block in M. In addition, the security against distinguishing attacks are also evaluated on MAC constructions. Kim *et al.* introduced two distinguishing attacks called distinguishing-R and distinguishing-H [6]. Let $\mathcal{R}$ and r be random functions which have the same domain as $\mathcal{H}$ and h, respectively. Moreover, we denote the hash function $\mathcal{H}$ instantiating a compression function f by $\mathcal{H}^f$. In the distinguishing-R attack, the attacker distinguishes a MAC $\mathcal{H}(K, M)$ from $\mathcal{R}(K, M)$. On the other hand, in the distinguishing-H attack, the attacker distinguishes $\mathcal{H}^h(K, M)$ from $\mathcal{H}^r(K, M)$.

Regarding the distinguishing-R attack, Preneel and van Oorschot [7] presented a generic attack against MACs with a Merkle-Damgård like iterative structure, which requires $O(2^{n/2})$ queries. In [7], it was explicitly mentioned that the same attack could be applied to LPMAC. Chang and Nandi later discussed its precise complexity when a long message is used [8]. The distinguishing-R attack is immediately converted to the existential forgery attack with the same complexity. On the other hand, no generic attack is known for the distinguishing-H attack, and it is widely believed that the complexity of the distinguishing-H at-

[1] The name "LPMAC" was given by Wang *et al.* [3].

Table 1. Comparison of distinguishing-H attacks against LPMAC

Attack Target	Size(n)	#Rounds	#Queries	Reference
SHA-1	160	43/80	$2^{124.5}$	[3]
SHA-1	160	61/80	$2^{154.5}$	[3]
SHA-1	160	65/80	$2^{80.9}$	[18]
SHA-256	256	39/64	$2^{184.5}$	[20]
RIPEMD	128	48/48 (full)	2^{66}	[19]
RIPEMD-256	256	58/64	$2^{163.5}$	[19]
RIPEMD-320	320	48/80	$2^{208.5}$	[19]
Generic narrow-pipe MD	n	full	$3 \cdot 2^{\frac{n}{2}}$	Ours

Our attack also requires $2^{n/2}$ offline computations and a memory to store $2^{n/2}$ tags. The attack can be memoryless with $6 \cdot 2^{n/2}$ queries and $2^{(n/2)+1}$ offline computations.

tack against a MAC instantiating a securely designed hash algorithm $\mathcal{H}^h$ should cost 2^n complexity.

There are several cryptanalytic results on MAC constructions. Although several results are known for block-cipher based MACs e.g. [5, 9, 10], in this paper, we focus our attention on hash function based MACs. A notable work in this field is the one proposed by Contini and Yin, which presented distinguishing and key recovery attacks on HMAC/NMAC with several underlying hash functions [11]. After that, several improved results were published [12–16]. Another important work is the one proposed by Wang *et al.* [17], which presented the first distinguishing-H attack on HMAC-MD5 in the single-key setting. In this attack framework, the number of queries principally cannot be below $2^{n/2}$ because the birthday attack is used. With the techniques of [17], a series of distinguishing-H attacks on LPMAC were presented against SHA-1, SHA-256, and the RIPEMD-family [18, 19, 3, 20]. The attack results are summarized in Table 1. Note that the notion of the almost universal forgery attack was firstly mentioned by [5], while some of previous attacks e.g. [10] can directly be applied for this scenario.

Our Contributions

In this paper, we propose generic attacks on LPMAC instantiating any narrow-pipe Merkle-Damgård hash function. We firstly propose a distinguishing-H attack with $3 \cdot 2^{n/2}$ queries, $2^{n/2}$ offline computations, and a memory to store $2^{n/2}$ tags. Our attack updates the previous results, namely full rounds are attacked with $O(2^{n/2})$ complexity. Moreover, the attack can be memoryless by using the technique in [21, Remark9.93]. The complexity of our attack is listed in Table 1.

Our attack is based on a new technique which makes an internal collision starting from two different length-prepend values, and recovers an internal state value with queries of different lengths. This approach is completely different from previous attacks on LPMAC, which utilize the existence of a high-probability

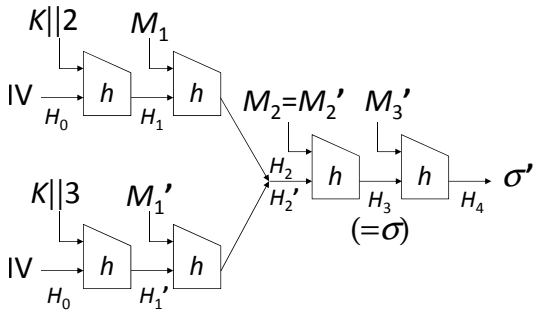

Fig. 1. Sketch of Distinguish-H Attack

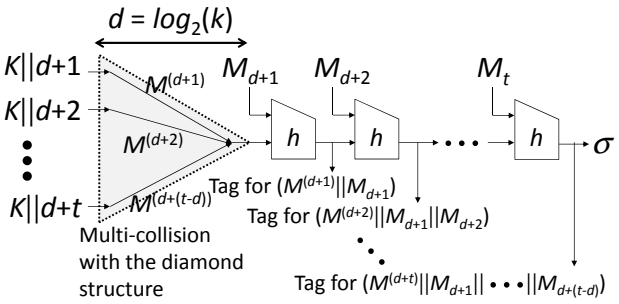

Fig. 2. Sketch of Almost Universal Forgery Attack

differential path of an underlying compression function. The idea of our technique is depicted in Fig. 1. The outline is as follows. We start from two length-prepend values; one is for 2-block messages and the other is for 3-block messages. With a very high probability, the internal states H_1 and H'_1 will be different values. Assume that we can easily obtain paired messages (M_1, M'_1) such that $h(H_1, M_1)$ and $h(H'_1, M'_1)$ form an internal collision. Then, we can obtain the value of $H_4(= \sigma')$, which is the output of the last compression function, by querying $M'_1 \| M'_2 \| M'_3$. In addition, we can obtain the value of $H_3(= \sigma)$, which is the input of the last compression function by querying $M_1 \| M_2$. Because we obtain all input and output information for the last compression function, we can judge whether h is the target algorithm or not by comparing $h(\sigma, M'_3)$ and σ'.

As mentioned before, the length prepending scheme is known to be prefix-free. However, an internal collision with different length-prepend values have the same effect as the prefix. In fact, in Fig. 1, $M_1 \| M_2$ can be regarded as a prefix of $M'_1 \| M'_2 \| M'_3$. This shows that the core of the security of LPMAC, which is the pre-fix freeness, can be totally broken with $O(2^{n/2})$ queries.

We then further extend the technique to mount an even stronger attack against LPMAC, which is called an almost universal forgery attack. In this at-

tack, for a given t-block message $M_1\|M_2\|\cdots\|M_t$, we assume the attacker's ability to modify the first d message blocks where $d = \lceil \log t \rceil$ into the value of his choice $M'_1\|M'_2\|\cdots\|M'_d$. Then, the attacker forges the tag for $M'_1\|\cdots M'_d\|M_{d+1}\|\cdots\|M_t$. Moreover, in our attack, the attacker can reveal the internal state value. The main idea is constructing a multi-collision starting from various different length-prepend values, which is depicted in Fig. 2. This enables the attacker to deal with various undetermined length-prepend values $(1,\ldots,t)$ in advance. To construct the multi-collision, we use the *diamond structure* proposed by Kelsey and Kohno for the herding attack [22].

Paper Outline

In Sect. 2, we introduce LPMAC and briefly summarize related work. In Sect. 3, we explain our generic distinguishing-H attack. In Sect. 4, we explain our generic almost universal forgery attack. In Sect. 5, we conclude this paper.

2 Related Work

2.1 LPMAC with Narrow-Pipe Merkle-Damgård Hash Functions

LPMAC is a hash function based MAC construction observed by Tsudik [1] to prevent the so called length-extension attack on the secret-prefix MAC. In LPMAC, the length of the message to be hashed is prepended before the message is hashed, that is to say, $\sigma \leftarrow \mathcal{H}(K\|\ell\|M)$, where ℓ is the length of M.

Most of widely used hash functions adopt the Merkle-Damgård domain extension with the narrow-pipe structure and the MD-strengthening. In this scheme, the input message M is first padded to be a multiple of the block-length by the padding procedure. Roughly speaking, a single bit '1' is appended to M, and then a necessary number of '0's are appended. Finally, the length of M is appended. Note that this padding scheme is not only the one, and replacing it with another padding scheme e.g. split padding [23] is possible. However, we only use it in this paper because it is the most common. The padded message is divided into message blocks $M_0, M_1, \ldots, M_{t-1}$ with the block size of b bits. Then, the hash value of size n is computed by iteratively updating the chaining variable of size n bits with the compression function h defined as $\{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$;

$$H_0 \leftarrow \mathrm{IV}, \qquad H_{i+1} \leftarrow h(H_i, M_i) \text{ for } i = 0, 1, \ldots, t-1, \tag{1}$$

where IV is an n-bit pre-specified value. Finally, H_t is the hash value of M.

In this paper, for simplicity, we assume that $K\|\ell$ is always padded to be 1-block long. Note that all of our attacks can work without this assumption by fixing remaining message bits in the first block to some constant value, say 0.

2.2 Summary of Previous Analyses on MAC Algorithms

To distinguish the target compression function h from a random function r, previous distinguishing-H attacks exploited a high probability differential path on h. Assume that a good near-collision path exists, namely there exists (Δ, Δ') such that $h(c, m) \oplus h(c, m \oplus \Delta) = \Delta'$ holds with probability p where $p > 2^{-n}$ for a randomly chosen c and m. Then, h is distinguished by querying $1/p$ paired messages with difference Δ and checking whether the output difference is Δ'. To construct a high probability differential path, we usually need the difference of the input chaining variable (pseudo-near collision). However, because the MAC computation starts from the secret information, it is hard to generate a specific difference on intermediate chaining variables, and is also hard to detect it only from the tag values. Wang *et al.* [17] solved these problems by using the birthday attack to generate a specific difference of an intermediate chaining variables and efficiently detect it only by changing the next message block. Previous distinguishing-H attacks on LPMAC [18, 19, 3, 20] used the similar idea as [17]. As long as the birthday attack is used to generate an intermediate difference, the attack complexity is between $2^{n/2}$ and 2^n.

2.3 Multi-Collision Attack

The naive method to construct a multi-collision is too expensive. For narrow-pipe Merkle-Damgård hash functions, several generic attacks to construct a multi-collision are known; collisions of sequential blocks [24], collisions with a fixed point [25], *an expandable message* [26], *a diamond structure* [22], and *multi-pipe diamonds* [27]. These are the methods to attack hash functions, which no secret exists in the computation. Because our attack is targeting MAC with secret information, we should choose the most suitable construction carefully.

3 Generic Distinguishing-H Attack on LPMAC

In this section, we present a generic distinguishing-H attack on a narrow-pipe Merkle-Damgård hash function. The attack complexity is $3 \cdot 2^{\frac{n}{2}}$ queries, $2^{\frac{n}{2}}$ offline computations, and a memory to store $2^{\frac{n}{2}}(n + b)$-bit information. Moreover, the attack becomes memoryless with $6 \cdot 2^{\frac{n}{2}}$ queries and $2^{\frac{n}{2}+1}$ offline computations. The results indicate that the hardness of the distinguishing-H attack on LPMAC is almost the same level as the distinguishing-R attack when a narrow-pipe Merkle-Damgård hash function is used.

3.1 Main Idea

Our attack is based on a new technique which makes an internal collision starting from two different length-prepend values t_1 and t_2 where $t_1 < t_2$. The idea is illustrated in Fig. 3. First the attacker generates $2^{n/2}$ t_2-block messages which start from the length-prepend value t_2 and the last $t_2 - t_1$ blocks are fixed to

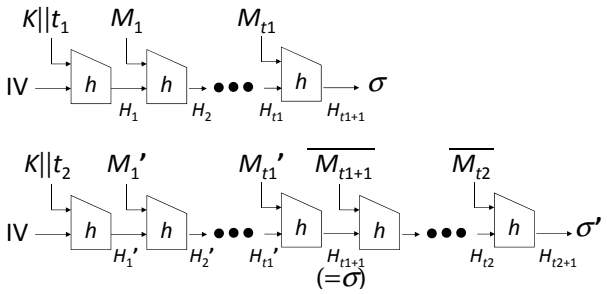

Fig. 3. Internal Collision with Different Length-Prepend Values

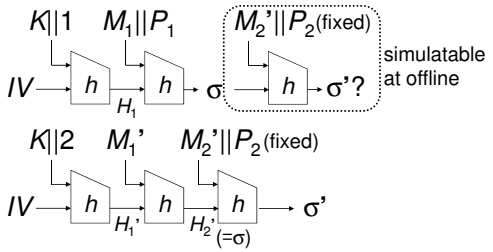

Fig. 4. Procedure of Distinguishing-H Attack

some value. Then he generates $2^{n/2}$ t_1-block messages starting from the length-prepend value t_1 and further computes $t_2 - t_1$ blocks at offline with the fixed value. By checking their match, the inner collision at H_{t_1+1} can be detected with a good probability. The attacker can know the values of H_{t_1+1} and H_{t_2+1} by querying the colliding t_1-block message and t_2-block message, respectively. Therefore, by simulating the last $t_2 - t_1$ blocks at offline, which of h or r is instantiated can be detected.

3.2 Attack Procedure

We assume that the MD-strengthening is used as the padding procedure of the underlying hash function, because it is adopted by most of widely used hash functions. The attack procedure is described in Alg. 1, which is also depicted in Fig. 4. Note that the length-prepend value can be chosen by the distinguisher. The attack procedure returns a bit 1 if the underlying compression function is h and a bit 0 if the underlying compression function is r.

Attack Evaluation. Alg. 1 correctly returns a bit 1 only if the compression function is h.

Let us evaluate the probability that Alg. 1 returns a bit 1 when the compression function is h. In the above procedure, we generate $2^{n/2}$ values for $H_2(= \sigma)$

Algorithm 1 Distinguishing-H Attack

Input: a compression function algorithm h to be distinguished
Output: a determining bit 0 or 1

1: Randomly choose the value of M_2' so that the padding string P_2 is included in the same block.
2: **for** $2^{n/2}$ different values of M_1' **do**
3: Query $M_1'\|M_2'$ to obtain the corresponding tag σ'
4: Store the pair of (M_1', σ') in a table T.
5: **end for**
6: **for** $2^{n/2}$ different values but the same length of M_1 whose padding string P_1 is included in the same block **do**
7: Query M_1 and obtain the corresponding tag σ.
8: Compute $temp \leftarrow h(\sigma, M_2'\|P_2)$ at offline.
9: Check if the same value as $temp$ exists in T.
10: **if** the same value exists **then**
11: Replace M_2' with $\overline{M_2'}$ such that $M_2' \neq \overline{M_2'}$ and $|M_2'| = |\overline{M_2'}|$.
12: Query $M_1'\|\overline{M_2'}$ to obtain the corresponding tag $\overline{\sigma'}$.
13: Compute $\overline{temp} \leftarrow h(\sigma, \overline{M_2'}\|P_2)$ at offline.
14: **if** $\overline{\sigma'} = \overline{temp}$ **then**
15: Return a bit 1.
16: **end if**
17: **end if**
18: **end for**
19: Return a bit 0.

and $2^{n/2}$ values for H_2', and thus we have 2^n pairs of (H_2, H_2'). Hence, we have a collision $(H_2 = H_2')$ with probability $1 - e^{-1}$. If $H_2 = H_2'$, the simulated value $temp$ and σ' always become a collision due to the identical message in the last block. Note that we happen to have a collision at Step 9 even if $H_2 \neq H_2'$ with probability $1 - e^{-1}$. This collision is noise in order to detect the internal collision. However, with the additional check at Step 14, the internal collision pair $(H_2 = H_2')$ also generates a collision for $\overline{M_2'}$ with probability 1, whereas the noise collision only produces another collision with probability 2^{-n}. Overall, Alg. 1 returns a bit 1 with a probability of $1-e^{-1}$ and this is a right pair making an internal collision with probability $1 - 2^{-n} \approx 1$.

Let us evaluate the probability that Alg. 1 returns a bit 1 when the compression function is r. An internal collision $H_2 = H_2'$ occurs with probability $1 - e^{-1}$. However, the collision between H_2 and H_2' is not preserved between $temp$ and σ' at Step 9. This is because σ' is obtained by a query and thus $\sigma' = r(H_2, M_2'\|P_2)$, whereas, $temp$ is computed based on the algorithm of h at offline and thus $temp = h(H_2, M_2'\|P_2)$. The probability that $r(H_2, M_2'\|P_2)$ collides with $h(H_2, M_2'\|P_2)$ is 2^{-n}. As a result, we expect to obtain only one collision at Step 9, and the probability that this pair generates another collision at Step 14 is 2^{-n}. Overall, Alg. 1 returns 1 only with probability 2^{-n}. This is significantly smaller than the probability for the case of h.

Complexity Evaluation. The iteration at Step 2 requires to query $2^{n/2}$ 2-block messages and to store $2^{n/2}$ tag values and corresponding messages. Hence, the query complexity is $2^{(n/2)+1}$ message blocks and the memory complexity is $2^{n/2}(n+b)$ bits, where b is the block size. The iteration at Step 6 requires to query $2^{n/2}$ 1-block messages and to compute h for each of them. Hence, the query complexity is $2^{n/2}$ message blocks and the time complexity is $2^{n/2}$ compression function computations. Overall, the total attack cost is $3 \cdot 2^{n/2}$ message blocks in query, $2^{n/2}$ h computations in time, and $2^{n/2}(n+b)$ bits in memory.

Memoryless Attack. The core of the attack is a collision finding problem on an internal state, thus it can be memoryless with the well-known technique using the cycle structure. Because it needs a collision starting from two different length-prepend values, the problem is a memoryless meet-in-the-middle attack [21, Remark9.93] rather than a memoryless collision attack. To make the cycle, we define the computation from an internal state value s_i to s_{i+1} as follows.

- If the LSB of s_i is 0, apply procedure $\mathcal{F}$. If the LSB of s_i is 1, apply $\mathcal{G}$.

$\mathcal{F}$: Convert an n-bit string s_i into a b-bit string $s_i^{\mathcal{F}}$ e.g. by appending 0s. Query $s_i^{\mathcal{F}} \| M_2'$ to obtain σ' and set $s_{i+1} \leftarrow \sigma'$.

$\mathcal{G}$: Convert an n-bit string s_i into a string $s_i^{\mathcal{G}}$ by some appropriate method so that the padding string P_1 is included in the same block. Query $s_i^{\mathcal{G}}$ to obtain σ and compute $s_{i+1} \leftarrow h(\sigma, M_2' \| P_2)$ at offline.

Finally, the attack can be memoryless with double of the query and time complexities, which are queries of $6 \cdot 2^{n/2}$ message blocks and $2^{(n/2)+1}$ computations of h.

Impact of the Attack. The goal of the attack presented in this section is the distinguishing-H attack. However, the attack not only distinguishes h from r, but also achieves a much stronger result, which is the recovery of the internal state. The knowledge of the internal state leads to much stronger attacks.

The first application is the length-extension attack. We use Fig. 3 to explain the attack. As explained in Sect. 3.1, our attack first recovers the internal state value h_{t_1+1}. That is, we know that any t_2-block message whose first t_1 blocks are fixed to $M_1' \| M_2' \| \cdots \| M_{t_1}'$ will result in the known collision value at h_{t_1+1}. Therefore, for any $(t_2 - t_1)$-block message $\overline{M_{t_1+1}} \| \cdots \| \overline{M_{t_2}}$, the attacker can compute the tag for $M_1' \| \cdots \| M_{t_1}' \| \overline{M_{t_1+1}} \| \cdots \| \overline{M_{t_2}}$ only with one offline computation (of $t_2 - t_1$ blocks) without knowing the value of K. In other words, the length-extension attack can be performed once an internal collision with different length-prepend values is detected.

One may suspect that this length extension attack is the same as the existential forgery because it requires $O(2^{n/2})$ queries. However, obtaining the knowledge of the internal state is actually stronger than the distinguish-R attack [7] because it forges the tag of a message of t_2 blocks long without any more query.

It may be interesting to see our attack from a different viewpoint. The security proof of LPMAC by Bellare *et al.* assumed the prefix-freeness of the construction, where the prefix-freeness means that any message is not the prefix of other messages. Due to the length-prepend values, LPMAC satisfies the prefix-freeness. However, an internal collision starting from different length-prepend values has the same effect as the prefix. In fact in the above explanation, $M_1\|\cdots\|M_{t_1}$ can be regarded as a prefix of $M'_1\|\cdots\|M'_{t_1}\|\overline{M_{t_1+1}}\|\cdots\|\overline{M_{t_2}}$, and thus the length-extension attack is applied. Note that our attack requires $3\cdot 2^{n/2}$ queries and thus does not contradict with the security proof by Bellare *et al.* where LPMAC is a secure PRF up to $2^{n/2}$ queries.

One limitation of this length extension attack is that it only can forge the tag for messages of t_2 blocks long. We remove this limitation in the next section.

4 Generic Almost Universal Forgery Attack on LPMAC

The almost universal forgery attack was mentioned by Dunkelman *et al.* [5]. The original explanation by [5] is as follows.

> *we can find in linear time and space the tag of essentially any desired message m chosen in advance, after performing a onetime precomputation in which we query the MAC on $2^{n/2}$ messages which are completely unrelated to m. The only sense in which this is not a universal forgery attack is that we need the ability to modify one message block in an easy to compute way.*

In the attack by [5], the first message block of the given message is modified by applying the XOR with two precomputed values.

In our attack, the attacker first determines parameter t, which is the limitation of the block-length of a message to be forged. For parameter t, the block length of the target message must be longer than d blocks and shorter than or equal to $d+t$ blocks, where $d=\lceil \log_2\ t\rceil$. We perform a onetime precomputation; query tags for $(t-1)\cdot 2^{n/2}$ messages of at most $d+t$ blocks long (in total $O(2t^2\cdot 2^{n/2})$ message blocks) which are completely unrelated to a target message M, and perform $t\cdot 2^{n/2}$ offline computations of h. At the online stage, we replace the first d blocks of $M(M_1\|M_2\|\cdots\|M_d)$ with the precomputed values $M'_1\|M'_2\|\cdots\|M'_d$. Then, the attacker generates the tag for the modified message only with one offline computation (without any query).

4.1 Easiness and Hardness of Almost Universal Forgery Attack

Easiness of Almost Universal Forgery Attack on Various MACs. First of all, we point out that the almost universal forgery attack is trivial for various MACs with an iterated structure such as HMAC if the precomputation that queries $O(2^{n/2})$ messages is allowed. To achieve this, we can simply run the distinguishing-R attack by [7]. In details, we generate a pair of one-block messages (X_1, X'_1) which forms an internal collision. Then, for any given target

message $M_1\|M_2\|\cdots\|M_t$, we replace M_1 with X_1 and query $X_1\|M_2\|\cdots\|M_t$. The corresponding tag is also the valid tag for $X_1'\|M_2\|\cdots\|M_t$.

Hardness of Almost Universal Forgery Attack on Prefix-Free MACs. The above attack cannot be applied for prefix-free MACs such as LPMAC in an easy manner. Regarding LPMAC, if the length (not value) of the target message is given to the attacker, the almost universal forgery attack can be performed with the attack presented in Sect. 3.2. However, if the length is not given, performing the precomputation on LPMAC is hard. This is because, in LPMAC, the length-prepend value changes depending on the length of the message to be processed. Hence, without the knowledge of the length of the target message, performing the precomputation seems hard.

A simple method to deal with various length-prepend values in advance is performing the internal state recovery attack in Sect. 3.2 many times. Suppose that you apply the almost universal forgery attack for a message of ℓ blocks where $1 \leq \ell \leq t$, but the exact value of ℓ is unknown during the precomputation stage. You first assume $\ell = 1$ and apply the internal state recovery attack with the $O(2^{n/2})$ complexity. Then, the value of ℓ is changed into $2, 3, \ldots, t$ and the internal state recovery attack is performed for each of ℓ. Finally, the almost universal forgery attack can be performed for any ℓ-block message where $1 \leq \ell \leq t$ with t times $O(2^{n/2})$ queries where each query consists of at most t-blocks.

In the following part, we show another approach with the same complexity as the simple method at the order level, but seems to have more applications.

4.2 Overall Strategy

Our idea is constructing an internal multi-collision starting from various length-prepend values as shown in Fig. 5. Assume that a one-block message X makes a multi-collision for $\ell = 2, 3, \ldots, 9$, that is, $h(h(\mathrm{IV}, K\|2), X) = h(h(\mathrm{IV}, K\|3), X) = \cdots = h(h(\mathrm{IV}, K\|9), X)$. Also assume that the attacker knows the collision value denoted by Y. Then, for any message with the block length 2 to 9, we know that replacing the first block with X results in the chaining variable Y, and thus the computation for the remaining message blocks can be simulated at offline.

Finding a multi-collision within one block is very inefficient. For the Merkle-Damgård structure, several constructions of the multi-collision are known as explained in Sect. 2.3. Considering that our multi-collision needs to start from different values due to different length-prepend values, the diamond structure [22] and multi-pipe diamonds [27] are suitable. The diamond structure was proposed for the herding attack and the multi-pipe diamonds were proposed for the herding attack on more complicated structures such as the cascaded construction or zipper hash. So far, we have not discovered the way to utilize the multi-pipe diamonds. We thus construct the multi-collision based on the diamond structure. The construction is described in Fig. 6.

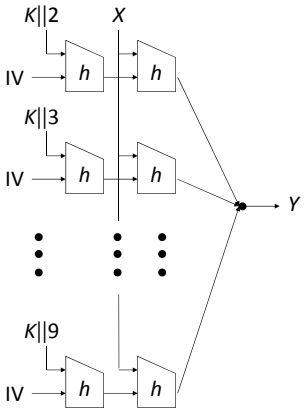

Fig. 5. Length Adjustment with Multi-collision

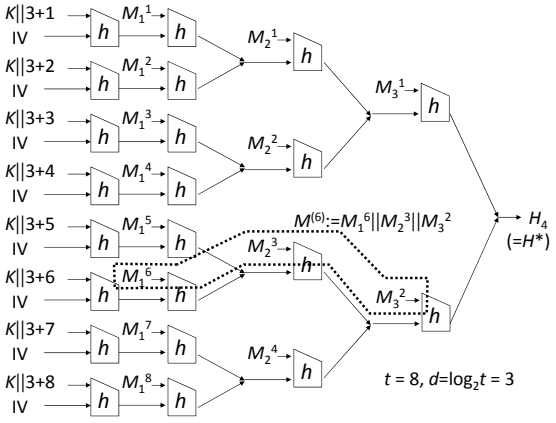

Fig. 6. Multi-Collision with Diamond Structure for $t = 8$

4.3 Multi-Collision with Diamond Structure

We explain how to construct a multi-collision with the diamond structure. Because our attack target is a MAC with secret information and the length prepending scheme, we need to detect an internal collision only with queries and tag values. In Alg. 2, we show the procedure to generate a collision starting from $\ell = t$ and $\ell = t + 1$, which is also described in Fig. 7. It is easy to see that if a collision can be generated, the entire diamond can be constructed by iteratively generating collisions.

In this attack we fix all message words but M_1 and M_1' to identical value. Therefore, the collision generated by M_1 and M_1' can be observed as a collision

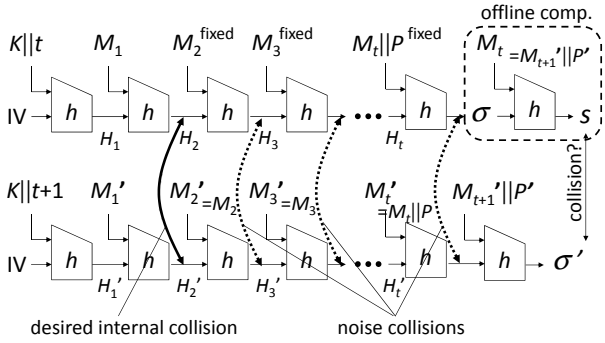

Fig. 7. Construction of an Internal Collision for $\ell = t$ and $t + 1$

Algorithm 2 Construction of an Internal Collision for $\ell = t$ and $t+1$

Input: $\ell = t$ (and $t+1$)

Output: a pair of message M_1^t and M_1^{t+1} such that $h(h(\mathrm{IV}, K\|t), M_1^t) = h(h(\mathrm{IV}, K\|t+1), M_1^{t+1})$

1: Fix the values of $M_2, M_3, \ldots, M_t$ so that the padding string P is included in the same block as M_t.
2: Set $M'_i \leftarrow M_i$ for $i = 2, 3, \ldots, t-1$. Set $M'_t \leftarrow M_t\|P$.
3: Fix the value of M'_{t+1} so that the padding string P' is included in the same block as M'_{t+1}.
4: **for** $2^{n/2}$ different values of M'_1 **do**
5: Query $M'_1\|M'_2\|\cdots\|M'_{t+1}$ to obtain the corresponding tag σ'.
6: Store the pair of (M'_1, σ') in a table T.
7: **end for**
8: **for** $2^{n/2}$ different values of M_1 **do**
9: Query each $M_1\|M_2\|\cdots\|M_t$ and obtain the corresponding tag σ.
10: Compute $temp \leftarrow h(\sigma, M'_{t+1}\|P')$ at offline.
11: Check if the same value as $temp$ exists in T.
12: **if** the same value exists **then**
13: Choose a value of $\overline{M_2}$ such that $M_2 \neq \overline{M_2}$.
14: Set $\overline{M'_2} \leftarrow \overline{M_2}$.
15: Query $M'_1\|\overline{M'_2}\|M'_3\|\cdots\|M'_{t+1}$ to obtain the corresponding tag $\overline{\sigma'}$.
16: Query $M_1\|\overline{M_2}\|M_3\|\cdots\|M_t$ to obtain the corresponding tag $\overline{\sigma}$, and compute $\overline{temp} \leftarrow h(\overline{\sigma}, M'_{t+1}\|P')$.
17: **if** $\overline{\sigma'} = \overline{temp}$ **then**
18: Return M_1 and M'_1.
19: **end if**
20: **end if**
21: **end for**

of the tag. Because we try $2^{n/2}$ different M_1 at Step 8 and $2^{n/2}$ different M'_1 at Step 4, we will obtain a collision $H_2 = H'_2$ with probability $1 - e^{-1}$. Note that even if $H_2 \neq H'_2$, we have other opportunities of obtaining collisions $H_3 = H'_3$, $H_4 = H'_4$, and so on. These are noise to obtain a collision at H_2, and thus need to be filtered out. For this purpose, for all tag collisions, we replace M_2 and M'_2 with another fixed value and check if another collision is generated (Step 17).

The complexity of the attack is as follows. At Step 4, it queries $(t+1) \cdot 2^{n/2}$ message blocks and requires a memory to store $2^{n/2}$ tags. At Step 8, it queries $t \cdot 2^{n/2}$ message blocks and computes h at offline $2^{n/2}$ times. We expect to obtain one desired internal collision and t noise collisions. Therefore Steps 15 and 16 are computed $t+1$ times and it requires to query $2(t+1)^2$ message blocks and $2(t+1)^2$ offline computations. Overall, the cost of Alg. 2 is approximately $(2t+1) \cdot 2^{n/2}$ queries and $2^{n/2}$ offline computations and a memory for $2^{n/2}$ tags. Note that the attack can be memoryless as discussed in Sect. 3.

Hereafter we denote the value of the multi-collision at H_{d+1} by H^*. We also denote the d-block message starting from $\ell = i$ and reaching H^* by $M^{(i)}$. For example, in Fig. 6, $M^{(1)} := M_1^1\|M_2^1\|M_3^1$ and $M^{(6)} := M_1^6\|M_2^3\|M_3^2$.

Algorithm 3 Forging Procedure

Output: a pair of message in which the first d blocks are modified and valid tag

Offline phase

1: Construct the diamond structure which can be used to forge the message whose length is longer than d blocks and shorter than or equal to $d+t$ blocks with $2t^2 \cdot 2^{n/2}$ queries and $t \cdot 2^{n/2}$ offline computations.

Online phase

2: Receive a target message M^* whose length is ℓ blocks where $d < \ell \leq d+t$, that is, $M^* = M_1^* \| M_2^* \| \cdots \| M_\ell^*$.
3: Replace the first d blocks of M^* with $M^{(\ell)}$.
4: Compute the tag value by using $H_{d+1} (= H^*)$ and $M_{d+1}^* \| M_{d+2}^* \| \cdots \| M_\ell^*$, which is denoted by σ^*.
5: **return** a pair of message and valid tag $(M^{(\ell)} \| M_{d+1}^* \| M_{d+2}^* \| \cdots \| M_\ell^*, \sigma^*)$.

Complexity for Entire Diamond Structure. By using Alg. 2, we evaluate the complexity for constructing the entire structure. Assume that we generate a diamond structure which can be used to forge the message whose length is longer than d blocks and shorter than or equal to $d+t$ blocks where $d = \log_2 t$.

The complexity of generating one collision starting from two different ℓ is determined by the bigger value of ℓ. In this example, the biggest value of ℓ is $d+t$ and thus the complexity for generating collisions with other ℓ is smaller than this case. According to Alg. 2, the complexity for the case $\ell = d+t$ is $(2(d+t)+1) \cdot 2^{n/2}$ queries and $2^{n/2}$ offline computations. The number of the leaf in the diamond structure is t, and thus we need to generate a collision $t-1$ times. Hence, the total complexity is less than $(t-1)(2t+2d+1) \cdot 2^{n/2}$ queries and $(t-1) \cdot 2^{n/2}$ offline computations. If only the head term is considered, the complexity is $2t^2 \cdot 2^{n/2}$ queries and $t \cdot 2^{n/2}$ offline computations.

4.4 Forging Procedure

Finally, we show how to produce a forged tag in Alg. 3. The construction of the diamond structure is completely independent of the online phase. As long as the block length ℓ of the given message M^* is in the valid range, by replacing the first $d (= \log_2 t)$ blocks of M^* with $M^{(\ell)}$, the tag for $M^{(\ell)} \| M_{d+1}^* \| M_{d+2}^* \| \cdots \| M_\ell^*$ is computed only with one t-block offline computation.

4.5 Comparison between Simple Method and Diamond Structure

We compare the simple method in Sect. 4.1 (applying the internal state recovery attack t times) and the diamond structure. As long as only the almost universal forgery attack is considered, the simple method is better than the diamond structure. The simple method does not require the offline computation in the precomputation phase, and the number of message blocks we need to replace is only 1 which is shorter than $d = \log_2 t$.

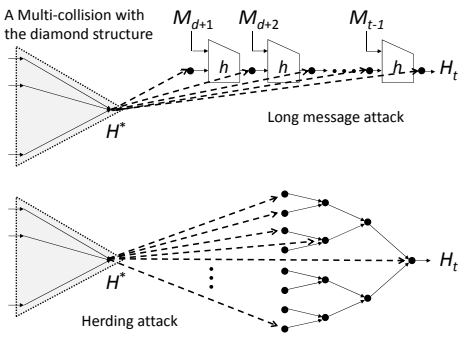

Fig. 8. Examples of potential applications of the diamond structure. The application is finding a message which connects the multi-collision H^* to one of chaining variables. The length information cannot be adjusted in advance.

On the other hand, the diamond structure has a unique property; it achieves a common internal state value for various length-prepend values. So far, good applications of this property have not been discovered. However, we show an example that gives some intuition to use the property. Let us consider the connection problem; the goal is finding a message block for the compression function h, which the input chaining variable is fixed to a given value and there are several target output values with different message lengths. The long-message second preimage attack and herding attack in Fig. 8 are such problems, though these problems are for the key-less situations. Finding suitable applications is an open problem.

5 Concluding Remarks

In this paper, we presented two cryptanalyses on LPMAC. Our first result was a generic distinguishing-H attack on LPMAC instantiating a narrow-pipe Merkle-Damgård hash function with a complexity of $O(2^{n/2})$. This showed that the widely believed assumption that a secure hash function should have the n-bit security against the distinguishing-H attack was not correct. Note that although previous results were updated by our generic attack with respect to the distinguish-H attack, the approach was very different. Finding a new problem which the previous differential distinguishers can work faster than a generic attack is an open problem. Our attacks are based on the new technique which generates an internal collision starting from different length-prepend values. One of such colliding messages can be regarded as the prefix of the other colliding message, and thus the core of the security of LPMAC, which is the prefix-freeness, is completely broken.

Our second result was an almost universal forgery attack on LPMAC. With the precomputation of $2t^2 \cdot 2^{n/2}$ queries and $t \cdot 2^{n/2}$ offline computations, we constructed the diamond structure which could realize an identical intermediate

value from t different length-prepend values. Hence, by modifying the first $\log_2 t$ message blocks into the value of the attacker's choice, the internal state was recovered and the valid tag of the modified message was forged with only 1 offline computation. Our results show that the security of the length-prepending structure can be totally broken with $O(2^{n/2})$ queries, and thus it is not enough to achieve a secure MAC.

Acknowledgements

The author would like to thank Lei Wang for his comments about the simple method of the almost universal forgery attack on LPMAC. The author also would like to thank the participants of Dagstuhl-Seminar (Jan. 2012), especially Orr Dunkelman and Adi Shamir for their comments on the memoryless meet-in-the-middle attack. Finally, the author would like to thank anonymous referees of Eurocrypt2012 for their helpful comments and suggestions.

References

1. Tsudik, G.: Message authentication with one-way hash functions. In: ACM SIGCOMM Computer Communication Review. Volume 22(5)., ACM (1992) 29–38
2. U.S. Department of Commerce, National Institute of Standards and Technology: Federal Register /Vol. 72, No. 212/Friday, November 2, 2007/Notices. (2007) `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.
3. Wang, X., Wang, W., Jia, K., Wang, M.: New distinguishing attack on MAC using secret-prefix method. In Dunkelman, O., ed.: Fast Software Encryption 2009. Volume 5665 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2009) 363–374
4. Bellare, M., Canetti, R., Krawczyk, H.: Pseudorandom functions revisited: The cascade construction and its concrete security. In: FOCS. (1996) 514–523
5. Dunkelman, O., Keller, N., Shamir, A.: ALRED blues: New attacks on AES-based MACs. Cryptology ePrint Archive, Report 2011/095 (2011) `http://eprint.iacr.org/2011/095`.
6. 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.: Security in Communication Networks SCN 2006. Volume 4116 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (2006) 242–256
7. Preneel, B., van Oorschot, P.C.: MDx-MAC and building fast MACs from hash functions. In Coppersmith, D., ed.: Advances in Cryptology — CRYPTO 1995. Volume 963 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (1995) 1–14
8. Chang, D., Nandi, M.: General distinguishing attacks on NMAC and HMAC with birthday attack complexity. Cryptology ePrint Archive, Report 2006/441 (2006) `http://eprint.iacr.org/2006/441`.
9. Jia, K., Wang, X., Yuan, Z., Xu, G.: Distinguishing and second-preimage attacks on CBC-like MACs. In Garay, J.A., Miyaji, A., Otsuka, A., eds.: Cryptology and Network Security, CANS 2009. Volume 5888 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2009) 349–361

10. Yuan, Z., Wang, W., Jia, K., Xu, G., Wang, X.: New birthday attacks on some MACs based on block ciphers. In Halevi, S., ed.: Advances in Cryptology — CRYPTO 2009. Volume 5677 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2009) 209–230
11. Contini, S., Yin, Y.L.: Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions. In Lai, X., Chen, K., eds.: Advances in Cryptology — ASIACRYPT 2006. Volume 4284 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (2006) 37–53
12. Fouque, P.A., Leurent, G., Nguyen, P.: Full key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In Menezes, A., ed.: Advances in Cryptology — CRYPTO 2007. Volume 4622 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2007) 15–30
13. Lee, E., Kim, J., Chang, D., Sung, J., Hong, S.: Second preimage attack on 3-pass HAVAL and partial key-recovery attacks on NMAC/HMAC-3-pass HAVAL. In Nyberg, K., ed.: Fast Software Encryption — 15th International Workshop, FSE 2008. Volume 5086 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2008) 189–206
14. Rechberger, C., Rijmen, V.: On authentication with HMAC and non-random properties. In Dietrich, S., Dhamija, R., eds.: Financial Cryptography 2007. Volume 4886 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2007) 119–133
15. Rechberger, C., Rijmen, V.: New results on NMAC/HMAC when instantiated with popular hash functions. Journal of Universal Computer Science **14**(3) (2008) 347–376
16. Wang, L., Ohta, K., Kunihiro, N.: New key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In Smart, N.P., ed.: Advances in Cryptology — EUROCRYPT 2008. Volume 4965 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (2008) 237–253
17. Wang, X., Yu, H., Wang, W., Zhang, H., Zhan, T.: Cryptanalysis on HMAC/NMAC-MD5 and MD5-MAC. In Joux, A., ed.: Advances in Cryptology — EUROCRYPT 2009. Volume 5479 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (2009) 121–133
18. Qiao, S., Wang, W., Jia, K.: Distinguishing attack on secret prefix MAC instantiated with reduced SHA-1. In Lee, D., Hong, S., eds.: ICISC 2009. Volume 5984 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (2010) 349–361
19. Wang, G.: Distinguishing attacks on LPMAC based on the full RIPEMD and reduced-step RIPEMD-{256, 320}. In Lai, X., Yung, M., Lin, D., eds.: Inscrypt 2010. Volume 6584 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2011) 199–217
20. Yu, H., Wang, X.: Distinguishing attack on the secret-prefix MAC based on the 39-step SHA-256. In Boyd, C., Nieto, J.M.G., eds.: Information Security and Privacy, 14th Australasian Conference, ACISP 2009. Volume 5594 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2009) 185–201
21. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography. CRC Press (1997)
22. Kelsey, J., Kohno, T.: Herding hash functions and the Nostradamus attack. In Vaudenay, S., ed.: Advances in Cryptology — EUROCRYPT 2006. Volume 4004 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2006) 183–200

23. Yasuda, K.: How to fill up Merkle-Damgård hash functions. In Pieprzyk, J., ed.: Advances in Cryptology — Aasiacrypt 2008. Volume 5350 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (2008) 272–289
24. Joux, A.: Multicollisions in iterated hash functions. Application to cascaded constructions. In Franklin, M., ed.: Advances in Cryptology — CRYPTO 2004. Volume 3152 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2004) 306–316
25. Dean, R.D.: Formal aspects of mobile code security. Ph.D Dissertation, Princeton University (1999)
26. Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2^n work. In Cramer, R., ed.: Advances in Cryptology — EUROCRYPT 2005. Volume 3494 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (2005) 474–490
27. Andreeva, E., Bouillaguet, C., Dunkelman, O., Kelsey, J.: Herding, second preimage and Trojan message attacks beyond Merkle-Damgård. In Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R., eds.: Selected Areas in Cryptography SAC 2009. Volume 5867 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2009) 393–414