SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

March 2019, Vol. 62 032112:1–032112:15
https://doi.org/10.1007/s11432-018-9568-2

Pseudo random oracle of Merkle-Damgård hash functions revisited

Kamel AMMOUR[1], Lei WANG[1,2*] & Dawu GU[1]

[1]*School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai* 200240*, China;*
[2]*Westone Cryptologic Research Center, Beijing* 100070*, China*

Received 8 June 2018/Accepted 21 August 2018/Published online 29 January 2019

Abstract Following the well-known random oracle Methodology, a cryptographic hash function is required to satisfy the property of pseudo-random oracle (PRO), that is indifferentiable from a random oracle. This paper revisits the PRO property of popular hash function modes purely from a theoretical point of view. Original Merkle-Damgård mode (sometimes referred to as Strengthened Merkle-Damgård) does not satisfy the PRO security due to the length-extension attack. To remedy it, a series of variants have been proposed with tweaks of either adopting a prefix-free padding or modifying the final primitive call. From these tweaks, we derive a common structural property named prefix-free computing. Indeed, all PRO-secure Merkle-Damgård variants published so far are prefix-free computing. Hence, an interesting question with respect to the nature of PRO security arises: is prefix-free computing a necessary condition for PRO-secure Merkle-Damgård hash function? This paper gives a negative answer. We investigate the difference between length-extension resistance and prefix-free computing, and find that length-extension resistance does not necessarily imply prefix-free computing. Consequently, we construct a dedicated Merkle-Damgård variant as a counterexample that is PRO-secure but not prefix-free computing.

Keywords Merkle-Damgård, random oracle, indifferentiability, prefix free, length extension attack

Citation Ammour K, Wang L, Gu D W. Pseudo random oracle of Merkle-Damgård hash functions revisited. Sci China Inf Sci, 2019, 62(3): 032112, https://doi.org/10.1007/s11432-018-9568-2

1 Introduction

Cryptographic hash functions (HFs) have become increasingly important and widely deployed in modern cryptography. The Merkle-Damgård (MD) [1,2] HF is one among the paradigms that have influenced the design of cryptographic HFs. It consists of an iteration of a fixed-input-length (FIL) compression function to handle strings of arbitrary length. These HFs are integrated in cryptosystems to provide services such as digital signatures, password protection, message integrity check, or message authentication code (MAC).

Security proofs of the most used cryptosytems (full domain hash (FDH) [3], optimal asymmetric encryption padding (OAEP) [4], etc.) are not rigorously provided in the standard model (through a reduction from a well-known security hypothesis) but are generally provided only in random oracle model (ROM) [5]. ROM [3] is an idealized model where the real HF is replaced by an idealized function called Rand Oracle $\mathcal{RO}$, which can be viewed as a magic box that returns a value y for a given input x [6]. This model was introduced in [3] "as a paradigm for designing efficient protocols", where all parties (even adversaries) have the right to access $\mathcal{RO}$. Theoretically speaking, a proof in ROM may indicate the non-existence of flaws in the structural design of the cryptosystem but provides no result when $\mathcal{RO}$ is

* Corresponding author (email: wanglei@cs.sjtu.edu.cn)

instantiated with a real cryptographic HF. In practice, cryptographers hope that the real cryptographic HF is "sufficiently good" at emulating $\mathcal{RO}$ [6]. Several studies [7–9] have questioned the relevance of such a proof by proposing secure (artificial) cryptosystems in ROM but they are no longer instantiated with a real HF. The following question arises: what is the main property of $\mathcal{RO}$ that makes a cryptosystem secure? Two approaches can be considered—either determining a concrete and sufficient property of $\mathcal{RO}$ or erasing its unneeded properties.

The indifferentiability notion, proposed by Maurer et al. [10], has been applied by Coron et al. [11] to HFs to show that the latter can be considered a good mimic of $\mathcal{RO}$ (although it is known that, in practice, the HF can never be $\mathcal{RO}$). Loosely speaking, the indifferentiability from $\mathcal{RO}$, also named the pseudo-random-oracle (PRO) property, indicates the analysis of the structural property of the Merkle-Damgård hash function (MDHF) using FIL-$\mathcal{RO}$ instead of a FIL compression function $\mathcal{CF}$. The resulting MDHF$^{\text{FIL-}\mathcal{RO}}$ is compared with a true $\mathcal{RO}$ [12]. The construction is considered to be without structural flaws, i.e., secure, if no attacker $\mathcal{A}$ can distinguish the HF with a FIL-$\mathcal{RO}$ from a real $\mathcal{RO}$.

This notion is an appropriate framework to express the aforementioned process rigorously. Under this property, any cryptographic protocol proved secure in ROM will remain so, once the $\mathcal{RO}$ is replaced by an indifferentiable MDHF from a $\mathcal{RO}$ ($\mathcal{PRO}$-MDHF) [13]. Thus, this notion fills the existing gap in the proofs of ROM and has become a de facto requirement for the security of HFs [14]. Coron et al. [11] showed that the plain MD scheme is not indifferentiable from $\mathcal{RO}$ even if the underlying compression function is FIL-$\mathcal{RO}$. They advocated adjustments in order to obtain indifferentiable $\mathcal{RO}$ schemes. These adjustments, which are sufficient conditions, are summarized as follows: (a) chopping some bits from the hash value (Chop MD); (b) making an extra call to a compression function (hash-based MAC (HMAC) and nested MAC (NMAC)); (c) adopting a prefix-free padding (prefix-free encoding).

Subsequent researches [15–19] improved the analysis by Coron et al. and provided a series of variants that have been proven to be $\mathcal{PRO}$ when the compression function ($\mathcal{CF}$) is an FIL-RO. Bellare and Ristenpart [20] presented an enveloped MD (EMD) construction involving the use of extra calls to $\mathcal{CF}$ to envelope the MD iteration. Hirose et al. [21] provided an MD construction with a permutation (MDP), which is inserted before the last iteration of MDHF. In [22], Hirose presented an MDHF construction with minimum padding, which is $\mathcal{PRO}$ and secure against a length extension attack (LEA). Bagheri et al. [15] proposed a generic iterated HF called suffix-free-prefix-free (SFPF) construction. To ensure both prefix-free and suffix-free properties, it uses three distinct $\mathcal{CF}$s. They provide a formal definition for prefix-free HF and claim its necessity for a $\mathcal{PRO}$-MDHF [15].

Our goals. In this paper, we would like to answer the following question: what makes these MDHF variants PRO? In other words, we intend to determine the common property shared by these constructions and provide its formal definition. The definition that we seek should be general and applicable to all variants of MDHF. It is an interesting and non-trivial task, especially if we intend it to be natural and easy to use. Moreover, we intend to show the importance of this shared property for a $\mathcal{PRO}$-MDHF construction—i.e., someone can automatically deduce (or not) the $\mathcal{PRO}$ property of a new MDHF construction variant if it has the shared property (or not). Along with this conclusion, it will be possible either to emphasize the necessity of this shared property for a $\mathcal{PRO}$-MDHF construction or introduce a new method, not yet explored, to construct a $\mathcal{PRO}$-MDHF.

It can be recalled further that the indifferentiability notion as defined by Coron et al. [11] allows the capture of security of the (iterated) function against generic attacks in general and LEAs in particular. Although LEA is the most important known weakness of MDHF $\mathcal{H}$, so far, no formal definition has been provided. Roughly speaking, LEA uses the hash value $z = \mathcal{H}^{\mathcal{CF}}(M)$ of an unknown message M to compute $z' = \mathcal{H}^{\mathcal{CF}}(M||m)$, where m is known. Owing to the iterative structure of the MDHF, it is easy to compute $\mathcal{H}(M||m)$ when it is impossible to compute $\mathcal{RO}(M||m)$. From this attack, it is easy to differentiate between $\mathcal{RO}$ and any iterative MDHF. The following question thus arises: is there any relation between the LEA and the shared property that we defined, i.e., can we consider that an MDHF construction is secure against LEA if it enjoys the shared property?

Our results. First, from the tweaks applied to ensure the $\mathcal{PRO}$-MDHF, we derive a common structural property named prefix-free computing ($\mathcal{PFC}$) and provide a satisfactory definition. Second, we claim the non-necessity of prefix-free computing ($\mathcal{PFC}$) for $\mathcal{PRO}$ MDHF by providing a counterexample. We construct an indifferentiable MDHF assuming the compression function to be ideal. Mainly, we tweak the padding algorithm. Our padding algorithm is defined as $\texttt{Pad}(\mathrm{M}) := \texttt{Pad}_1(M)\|\texttt{Pad}_2(M)$, where $\|$ is concatenation, and $\texttt{Pad}_1(\cdot)$ and $\texttt{Pad}_2(\cdot)$ are two suffix-free padding algorithms (refer to Section 4 for details). The hash process is the same as strengthened MD (SMD). Note that each message block is hashed twice. Further, Liskov et al. [23] proposed an iterated MDHF that processes each block twice. Our counterexample is not prefix-free computing, i.e., for two messages M and $M' = \texttt{Pad}(M)$, $\texttt{Pad}(M)$ is a prefix of $\texttt{Pad}(M')$. We provide a definition of the simulator that permits the maintenance of consistency with $\mathcal{RO}$ in order to prove the indifferentiability of our construction.

Finally, we observed that there is no relationship between LEA and the $\mathcal{PFC}$ property. Indeed, from our non-$\mathcal{PFC}$ counterexample given above, which is $\mathcal{PRO}$, we deduce that an LEA-resistant MDHF need not be a prefix-free computing algorithm. Furthermore, we provide a simple $\mathcal{PFC}$ MDHF construction that uses a permutation at its end. We show that this construction is not secure against LEA. We conclude that the $\mathcal{PFC}$ property for MDHF is not necessarily secure against LEA.

Paper outline. Section 2 introduces the notations and definitions used in this paper. The related work about the variants of $\mathcal{PRO}$-MDHF and the common structural property derived are provided in Section 3. Section 4 explores the $\mathcal{PFC}$ non-necessity for a $\mathcal{PRO}$-MDHF. Finally, in Section 5, we show there is no relationship between the resistance against LEA and the $\mathcal{PFC}$ property for an MDHF.

2 Preliminaries

In this section, we introduce notations and properties which will be used throughout this paper.

Notations. Let $\mathcal{A}$ be the distinguisher (adversary) and $\langle M\rangle$ represents the length of a message M in bits. MDHF denotes the Merkle-Damgård hash function, $\mathcal{RO}$ represents the random oracle and $\mathcal{S}$ the simulator. Let $\mathcal{CF}$ be the compression function, $\texttt{Pad}$ the padding algorithm, $\mathcal{PFC}$-MD the prefix free computing Merkle-Damgård and $\mathcal{PFE}$-MD the prefix free encoding Merkle-Damgård. Assigning a random value from $\{0,1\}^n$ to z is denoted by $z \xleftarrow{\$} \{0,1\}^n$.

Random oracle ($\mathcal{RO}$). The fixed-input-length random oracle (FIL-RO) is defined by a function f such that f: $\{0,1\}^m \mapsto \{0,1\}^n$ chooses the value $f(x)$ randomly from $\{0,1\}^n$ for each $x \in \{0,1\}^m$. More precisely, assume that a subset of the input-output relations of f is known: $f(x_1) = y_1, f(x_2) = y_2, \ldots, f(x_t) = y_t$. For simplicity, we denote $f(X) = Y$ where $X = \{x_1, x_2, \ldots, x_t\}$ and $Y = \{y_1, y_2, \ldots, y_t\}$. For any $x \in \{0,1\}^m\backslash X$, $\Pr[f(x) = y|f(X) = Y] = \frac{1}{2^n}$ for any $y \in \{0,1\}^n$. In this paper, a random oracle $\mathcal{RO}$ is referred to as the one with arbitrary-length inputs, whose domain is usually denoted as $\{0,1\}^*$. Let $\mathcal{F}$ be the set of all functions f with the same domain and range. The compression function $\mathcal{CF}$ can be equally regarded as a function f uniformly and randomly selected from $\mathcal{F}$.

Merkle-Damgård hash functions (MDHF). MD hash function uses four basic parts which are defined as follows:

(1) An initial value $\mathrm{IV} \in \{0,1\}^n$.

(2) An underlying compression function $\mathcal{CF}$: $\{0,1\}^{n+b} \mapsto \{0,1\}^n$ such that $\mathcal{CF}(h_{i-1}, m_i) = h_i$.

(3) A classical iterated function $\mathcal{CF}^+_{\mathrm{IV}}$: $(\{0,1\}^{n+b})^+ \mapsto \{0,1\}^n$ defined as $\mathcal{CF}^+_{\mathrm{IV}}(m_1, \ldots, m_\ell) = \mathcal{CF}(\cdots\mathcal{CF}(\mathcal{CF}(\mathrm{IV}, m_1), m_2), \cdots, m_\ell)$

(4) An injective function called padding function $\texttt{Pad} : \{0,1\}^* \mapsto (\{0,1\}^b)^+$ such that for a message $M \in \{0,1\}^*$, $\texttt{Pad}(M) = (m_1, \ldots, m_\ell)$ where ℓ is the number of blocks in the message M. One of the roles of ℓ is to make the length of the message compatible with the domain of $\mathcal{CF}^+_{\mathrm{IV}}$.

So, the classical iterated MDHF is defined for all $M \in \{0,1\}^*$ as $\mathrm{MD}^{\mathcal{CF}}_{\mathrm{IV,Pad}}(M) = \mathcal{CF}^+_{\mathrm{IV}}(\mathrm{Pad}(M))$.

Indifferentiability. According to [10,11], the indifferentiability is defined as follows.

Definition 1. An algorithm $\mathcal{H}$ with access to an ideal primitive $\mathcal{CF}$ is said to be (t_A, t_S, q, ϵ) indifferentiable from an ideal primitive $\mathcal{RO}$ if there is a simulator $\mathcal{S}$ such that for any distinguisher $\mathcal{A}$ it holds that

$$\mathrm{Adv}(\mathcal{A}) = |\Pr[\mathcal{A}^{\mathcal{H},\mathcal{CF}} = 1] - \Pr[\mathcal{A}^{\mathcal{RO},\mathcal{S}} = 1]| \leqslant \epsilon,$$

where $\mathcal{A}$ runs at most t_A time. $\mathcal{S}$ has oracle access to $\mathcal{RO}$ and runs at most t_S time, and makes at most q queries. In this paper, $\mathcal{CF}$ is defined as FIL-RO, $\mathcal{H}$ is the hash function. $\mathcal{RO}$ has the same domain and range with $\mathcal{H}$. The role of the simulator $\mathcal{S}$ is to simulate $\mathcal{CF}$. $\mathcal{A}$ interacts with $(\mathcal{O}_1, \mathcal{O}_2)$, where $\mathcal{O}_1$ can be $\mathcal{H}$ or $\mathcal{RO}$ and $\mathcal{O}_2$ can be $\mathcal{CF}$ or $\mathcal{S}$. The goal of the distinguisher is to distinguish after queries to $(\mathcal{O}_1, \mathcal{O}_2)$ which scenario it is.

View. Denote the pair of oracles that $\mathcal{A}$ interacts with as $(\mathcal{O}_1, \mathcal{O}_2)$, which is either $(\mathcal{H}, \mathcal{CF})$ or $(\mathcal{RO}, \mathcal{S})$. Denote the i-th query of $\mathcal{A}$ as q_i, and q_i is either an arbitrarily long message M queried to $\mathcal{O}_1$ or a fixed-length pair (x, y) queried to $\mathcal{O}_2$, where x denotes the hash chaining value and y denotes the message block. Denote the response of the oracles on q_i as z_i. Denote the i-th query-response relation as r_i, which is either $(Y_i = \mathtt{Pad}(M_i),\ z_i)$ or $(x_i,\ y_i,\ z_i)$. For the simplicity of the description, we use (IV, Y_i, z_i) to denote $(Y_i,\ z_i)$ and thus unify r_i as triple relations (a, b, c). From the first i query-response relations r_1, $r_2, \ldots, r_i$, $\mathcal{A}$ derives more relations using the following rules:

- From (a, b, c) and (a', b', c') with $c = a'$, $(a, b\|b', c')$ is derived;
- From $(a, b_1\|b_2, c)$ and (a', b', c') with $(a, b_1) = (a', b')$, (c', b_2, c) is derived.

Denote all the relations that $\mathcal{A}$ knows, namely the knowledge of $\mathcal{A}$, after the first i query-responses as R_i. Note that R_i consists of both previous query-responses $r_1, r_2, \ldots, r_i$ and the derived relations. Denote IV and all the hash chaining values in R_i as $H_i := \{a : (a, b, c) \in R_i\} \cup \{c : (a, b, c) \in R_i\} \cup \{\mathrm{IV}\}$. Following the folklore, we also separate the queries into trivial queries and non-trivial queries. For a query q_i made to $\mathcal{O}_1$, where q_i is an arbitrary long message, it is called trivial if there is a relation (a, b, c) in R_{i-1} such that (a, b) is equal to (IV, $\mathtt{Pad}(q_i)$), and non-trivial otherwise. For a query q_i made to $\mathcal{O}_2$, where q_i is a fixed-length pair (x, y), it is called trivial if there is a relation (a, b, c) in R_{i-1} such that (a, b) is equal to (x, y), and non-trivial otherwise. A trivial query is either repeating a previous query or verifying a relation derived from previous query-responses. Since all the oracles $\mathcal{RO}, \mathcal{S}, \mathcal{H}$ and $\mathcal{CF}$ always produce the same response on the same query, repeating a previous query cannot improve the advantage of $\mathcal{A}$ at all. Thus we assume that $\mathcal{A}$ does not repeat previous queries without the loss of generality. So in the rest of the paper, we assume that trivial queries are used to verify derived relations. If $(\mathcal{O}_1, \mathcal{O}_2)$ is $(\mathcal{H}, \mathcal{CF})$, the query-response from a trivial query q_i is always consistent with the previous derived relations in R_{i-1}. On the other case, $(\mathcal{O}_1, \mathcal{O}_2) = (\mathcal{RO}, \mathcal{S})$, the query-response from a trivial query may contradict with the previous derived relations in R_{i-1} if $\mathcal{S}$ fails to mimic.

3 Revisiting previous results

3.1 $\mathcal{PRO}$-MDHF

Coron et al. [11] proved that the original MDHF is not $\mathcal{PRO}$. They showed that it is vulnerable against the LEA. Nevertheless, they provide several patches to ensure the indifferentiability of MDHF. In addition, several $\mathcal{PRO}$-Merkle-Damgård variants have been proposed in [11, 15, 21, 22].

Prefix-free encoding Merkle-Damgård hash function ($\mathcal{PFE}$-MDHF). Coron et al. [11] proposed to apply the prefix free encoding ($\mathcal{PFE}$) property on MDHF in order to prove its indifferentiability from $\mathcal{RO}$ when the compression function is FIL-RO.

The prefix-free encoding ($\mathcal{PFE}$) property is applied specifically to the padding function $\mathtt{Pad}$ which is said $\mathcal{PFE}$ if for any $M_1 \neq M_2$, $\mathtt{Pad}(M_1)$ is not prefix of $\mathtt{Pad}(M_2)$. They showed that $\mathcal{PFE}$ property for an iterated MDHF is sufficient to claim the indifferentiability of this hash function mode when assuming that the compression function is FIL-RO.

Chop Merkle-Damgård hash function. It chops several bits of the output and uses the other bits as the hash digest. It is widely used in practice. For example, SHA-348 is obtained by dropping some bits from SHA-512. It has been shown that such dropping prevents the LEA.

HMAC. It is widely used for performing message authentication. It consists to hash the message with $\mathrm{MD}^{\mathcal{CF}}$ to get an output h' and then using this later to make an extra call to the compression function

($\mathcal{CF}$ used in MD$^{\mathcal{CF}}$) with the initial value (IV) as its input.

Remark 1. Coron et al. [11] justify the use of the first message (0^k) by the concern to ensure a non-interdependence between the use of the same IV on h' and the first message block.

NMAC. It is the basis of HMAC. Instead of using only one $\mathcal{CF}$ like HMAC, it applies another function G to the output of the original MD to produce a hash digest.

SFPF. It is based on three different FIL-$\mathcal{RO}$ f_1, f_2 and f_3 in order to hash a message M [15]. The first and last message blocks are processed by f_1 and f_3 respectively and all the intermediate message blocks are processed by f_2. f_3 ensures that MDHF is prefix free whereas f_1 provides the suffix freeness. The authors showed that prefix free property for an iterated MDHF is necessary to claim the indifferentiability of this hash function mode when assuming that the compression function is FIL-RO.

Merkle-Damgård with a permutation (MDP). It is a variant of MDHF which inserts a permutation before the last compression function [21].

Sequential hashing with minimum padding. It extends the MDP by using two distinct permutations π_1 and π_2 for domain separation [22]. A typical candidate for these is bitwise XOR with different nonzero constants.

3.2 Prefix free computing ($\mathcal{PFC}$) property

We notice that all the aforementioned $\mathcal{PRO}$ Merkle-Damgård have a common point. They modify only the operation on the last block in order to counter the vulnerability to the LEA. This means that these variants of MD share a common property that does not allow to compute $\mathcal{H}(M')$ from $\mathcal{H}(M)$ even if $M' = M||m$ and m is known. We call this common property prefix free computing ($\mathcal{PFC}$). Roughly speaking, an MDHF $\mathcal{H}^{\mathcal{CF}}$ is considered as $\mathcal{PFC}$ if, for any messages chosen M and M' by any computationally unbounded $\mathcal{A}$ with oracle access to $\mathcal{H}^{\mathcal{CF}}$ and $\mathcal{CF}$, the probability to compute $\mathcal{H}^{\mathcal{CF}}(M||M')$, given $\mathcal{H}^{\mathcal{CF}}(M)$ and any M', is negligible. We adopt an encoding Input/output tuples for $\mathcal{CF}$ and MDHF $\mathcal{H}$ in order to provide a definition of $\mathcal{PFC}$-MDHF.

Tuple description. We propose a novel representation for $\mathcal{CF}$ and MDHF $\mathcal{H}$. Indeed, this representation adopts an input/output encoding tuple for $\mathcal{CF}$ which can be extended to $\mathcal{H}$ by a sequential representation. This (sequential) tuple representation will be the basis of the definitions and proofs that will describe below.

Assume that the tuple $(\mathcal{CF}, x||y, z)$ represents $\mathcal{CF}(x, y) = z$ where the input (x, y) of $\mathcal{CF}$ are the chaining variable and the message block respectively and the output z is the updated chaining variable. Let $(\mathcal{CF}, \text{IV}||m_1, h_1)$, $(\mathcal{CF}, h_1||m_2, h_2), \ldots,$ $(\mathcal{CF}, h_{n-1}||m_n, h)$ be the sequential tuples representation of $\mathcal{H}^{\mathcal{CF}}(M)$ for message $M = m_1 m_2 \cdots m_n$ with $\mathcal{H}^{\mathcal{CF}}(M) = (\mathcal{CF} \cdots (\mathcal{CF}(\mathcal{CF}(\text{IV}, m_1), m_2), \ldots, m_n)$.

Prefix of tuples. A sequential tuples representation for proceeding multiple blocks $(\mathcal{CF}, h'_0||m'_0, h'_1)$, $\cdots, (\mathcal{CF}, h'_i||m'_i, h'_{i+1})$ is said a prefix of another sequential tuples representation $(\mathcal{CF}, h_0||m_0, h_1)$, $\ldots,$ $(\mathcal{CF}, h_i||m_i, h_{i+1}), \ldots, (\mathcal{CF}, h_n||m_n, h_{n+1})$, if for each tuple $(\mathcal{CF}, h'_k||m'_k, h'_{k+1})$ and $(\mathcal{CF}, h_k||m_k, h_{k+1})$ in each representation with $(0 \leqslant k \leqslant i)$: $(h'_k = h_k)$, $(m'_k = m_k)$ and $(h'_{k+1} = h_{k+1})$.

Definition 2. MDHF $\mathcal{H}^{\mathcal{CF}}$ with oracle access to a compression function $\mathcal{CF}$ is said $\mathcal{PFC}$ if there are no two distinct messages M and M' such that the sequential tuples representation of $\mathcal{H}(M)$ is a prefix of $\mathcal{H}(M')$.

Remark 2. Obviously $\mathcal{PFE}$-MDHF is surely a specific case of $\mathcal{PFC}$-MDHF because the block sequence of a message after padding is not a prefix of the block sequences of any other message after padding. Moreover, we notice that $\mathcal{PFC}$ definition also covers other $\mathcal{PRO}$-MDHF, including Chop-MD, NMAC and HMAC [11]. Here we use HMAC as an example and refer to Appendix B for more examples such as NMAC, Chop-MD, MDP, SFPF. For a message padded and divided into blocks $m_0, \ldots, m_n$, the sequential tuples representation of $\mathcal{H}^{\mathcal{CF}}(\text{IV}, M) = (\mathcal{CF}, h_0||m_0, h_1)$, $(\mathcal{CF}, h_1||m_1, h_2), \ldots, (\mathcal{CF}, h_i||m_i, h_{i+1})$, $\ldots,$ $(\mathcal{CF}, h_n||m_n, h_{n+1}), (\mathcal{CF}, h_0||h_{n+1}, h)$ can be prefix of another sequential tuples representation of $\mathcal{H}^{\mathcal{CF}}(\text{IV}, M||m) = (\mathcal{CF}, h_0||m_0, h_1)$, $(\mathcal{CF}, h_1||m_1, h_2), \ldots, (\mathcal{CF}, h_i||m_i, h_{i+1}), \ldots, (\mathcal{CF}, h_n||m_n, h_{n+1}), (\mathcal{CF},$ $h_{n+1}||m, h_{n+2}), (\mathcal{CF}, h_0||h_{n+2}, h')$ if and only if $h_n = h_0$ and $m_n = h_{n+1}$ and $h_{n+1} = h$ which is possible with a negligible probability.

In what follows, we will analyze the necessity of $\mathcal{PFC}$ property for a $\mathcal{PRO}$-MDHF.

4 Unnecessary $\mathcal{PFC}$ property for a $\mathcal{PRO}$ MDHF

This section studies the necessity of $\mathcal{PFC}$ property for an MDHF to be $\mathcal{PRO}$. More precisely, we present a concrete example of an MDHF $\mathcal{H}^{\mathcal{CF}}$ that is not $\mathcal{PFC}$ but is indifferentiable from $\mathcal{RO}$. The intuition behind this study is came up by trying to set up a no-$\mathcal{PFC}$ MDHF $\mathcal{H}^{\mathcal{CF}}$ function using redundant padding. This construction is inspired by the design proposed by Liskov [23] in which each block is processed twice. However, by doing so, we got a function that is not $\mathcal{PRO}$. That is why we argue that we should limit the intervention on the padding function as much as possible.

4.1 Counter-example

4.1.1 *Specification of the counterexample $\mathcal{H}$*

Our example mainly modifies the padding algorithm, comparing with the original MDHF. The specification of the counterexample is as follows. Assume that the maximum length of the input message is 2^p bits.

Padding algorithm $\texttt{Pad}(\cdot)$**.** The padding algorithm is composed of two padding algorithms: $\texttt{Pad}(M) = \texttt{Pad}_1(M)\|\texttt{Pad}_2(M)$, where $\|$ denotes the concatenation operator. To illustrate it, let M be a message, $\langle M\rangle$ the bit length of M, and $\mathrm{len}(M)_b$ is the m-bit binary encoding of $\langle M\rangle$, where m is the message block length. v is the smallest integer such that $\langle M\rangle + 1 + v$ is a multiple of m.

- $\texttt{Pad}_1(\cdot)$ pads M with a bit '1', then pads v '0's, and pads $\mathrm{len}(M)_b$ to the last m bits, that is
$$\texttt{Pad}_1(M) = M\|1\|0^v\|\mathrm{len}(M)_b.$$
- $\texttt{Pad}_2(\cdot)$ pads M with a bit '0', then pads v '1's, and pads $\mathrm{len}(M)_b$, that is
$$\texttt{Pad}_2(M) = M\|0\|1^v\|\mathrm{len}(M)_b.$$

Overall, the padding of the message M is as follows:
$$\texttt{Pad}(M) = M\|1\|0^v\|\mathrm{len}(M)_b\|M\|0\|1^v\|\mathrm{len}(M)_b.$$

Hash process. After padding, the padded message is split into 2ℓ blocks $m_1||m_2||\cdots||m_\ell||m_{\ell+1}||m_{\ell+2}||\cdots||m_{2\ell}$. Note that $m_i = m_{i+\ell}$ holds for $1 \leqslant i \leqslant \ell-2$. If $m_{\ell-1}$ and $m_{2\ell-1}$ contain padding bits, $m_{\ell-1} \neq m_{2\ell-1}$. For the last block, without loss of generality we always assume that the length of a message block is longer than the representation of $\langle M\rangle$ in bits, and thus $m_\ell \neq m_{2\ell}$ holds for any message M. The message blocks are hashed in a sequential order as described in the iterative MDHF $h_i = \mathcal{CF}(h_{i-1}, m_i), 1 \leqslant i \leqslant 2\ell$, where h_0 is the initial vector (IV). Finally $h_{2\ell}$ is the output as the hash digest.

4.1.2 *The counter-example is not $\mathcal{PFC}$*

To prove that our counter-example is not $\mathcal{PFC}$, it is enough to find two messages M and M' such that the sequential tuples representation of $\mathcal{H}(\mathrm{IV}, M) = (\mathcal{CF}, \mathrm{IV}||m_1, h_1)$, $(\mathcal{CF}, h_1||m_2, h_2), \ldots, (\mathcal{CF}, h_{n-1}||m_n, h)$ is prefix of the sequential tuples representation of $\mathcal{H}(\mathrm{IV}, M') = (\mathcal{CF}, \mathrm{IV}||m_1', h_1')$, $(\mathcal{CF}, h_1'||m_2', h_2), \ldots, (\mathcal{CF}, h_{n'-1}'||m_{n'}', h')$.

First, let choose a message M with a single block m_1 such that $\texttt{Pad}(M) = m_1m_2m_1m_2'$ where $m_2 = 10\cdots0\langle M\rangle$ and $m_2' = 01\cdots1\langle M\rangle$. The sequential tuples representation of $\mathcal{H}(\mathrm{IV}, M)$ is $(\mathcal{CF}, \mathrm{IV}||m_1, h_1)$, $(\mathcal{CF}, h_1||m_2, h_2)$, $(\mathcal{CF}, h_2||m_1, h_3)$, $(\mathcal{CF}, h_3||m_2', h_4)$.

Then, let choose another message $M' = \texttt{Pad}(M)$ and $\texttt{Pad}(M') = m_1m_2m_1m_2'mm_1m_2m_1m_2'm'$. The sequential tuples representation of $\mathcal{H}(\mathrm{IV}, M')$ is $(\mathcal{CF}, \mathrm{IV}||m_1, h_1)$, $(\mathcal{CF}, h_1||m_2, h_2)$, $(\mathcal{CF}, h_2||m_1, h_3)$,

$(\mathcal{CF}, h_3||m'_2, h_4)$, $(\mathcal{CF}, h_4||m, h_5)$, $(\mathcal{CF}, h_5||m_1, h_6)$, $(\mathcal{CF}, h_6||m_2, h_7)$, $(\mathcal{CF}, h_7||m_1, h_8)$, $(\mathcal{CF}, h_8||m'_2, h_9)$, $(\mathcal{CF}, h_9||m', h_{10})$.

It is clear that the sequential tuples representation of $\mathcal{H}(\mathrm{IV}, M)$ is a prefix of $\mathcal{H}(\mathrm{IV}, M')$ and which means that our example is not $\mathcal{PFC}$.

4.2 Indifferentiability analysis

Theorem 1. The above Merkle-Damgård hash function construction $\mathcal{H}^{\mathcal{CF}} : \{0,1\}^* \rightarrow \{0,1\}^n$ based on a FIL-$\mathcal{RO}$ $\mathcal{CF} : \{0,1\}^m \times \{0,1\}^n \rightarrow \{0,1\}^n$ is (t_A, t_s, q, ϵ) indifferentiable from a $\mathcal{RO}$ with the same domain and range for any $t_{\mathcal{A}}$ and $t_{\mathcal{S}} \leqslant \frac{q(q+1)}{2}$ with $\epsilon = O(\frac{(\ell q)^2}{2^n})$, where q denotes the total number of queries made by $\mathcal{A}$ and ℓ is the maximum length of a query made by $\mathcal{A}$.

Proof. The proof of indifferentiability involves two steps. First, describe a simulator that should simulate the $\mathcal{CF}$ in the random oracle model (ROM). Then, show that the probability that $\mathcal{A}$ can distinguish between $(\mathcal{H}^{\mathcal{CF}}, \mathcal{CF})$ and $(\mathcal{RO}, \mathcal{S})$ is negligible.

In what follows we discuss only the simulator $\mathcal{S}$ which allows us to demonstrate the indifferentiability of our example. The full proof of indifferentiability is provided in Appendix A. It is based on a hybrid argument which is widely used in [11, 20, 21]. It is strongly inspired from the indifferentiabilty proof of prefix-free-encoding Merkle-Damgård construction given by Coron et al. [11]. The proof contains an informal description of six games G_i, $1 \leqslant i \leqslant 6$, which allows $\mathcal{A}$ to pass from game G_1 that emulates $(\mathcal{RO}, \mathcal{S})$ to game G_6 that emulates $(\mathcal{H}^{\mathcal{CF}}, \mathcal{CF})$ by small transformations.

The simulator $\mathcal{S}$. Let $\mathcal{S}$ be a simulator which simulates $\mathcal{CF}$. $\mathcal{S}$ should be programmed such that its relation with $\mathcal{RO}$ is consistent as the relation between $(\mathcal{H}^{\mathcal{CF}}, \mathcal{CF})$. $\mathcal{S}$ can call $\mathcal{RO}$ if needed but it does not see the $\mathcal{A}$'s queries to $\mathcal{RO}$ or $\mathcal{H}$.

On a query (x, y) such that $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$, $\mathcal{S}$ responds with a value $z \in \{0,1\}^n$, and stores the query-response relation (x, y, z) in a table T, which is initialized to empty and has three columns (T_x, T_y, T_z). The query (x, y) is stored in T_x and T_y respectively and the correspondent response z is stored in T_z. Moreover, $\mathcal{S}$ updates table T with more relations using the following rule: for a new tuple (x, y, z) if there is a relation (a, b, c) in T with $(c = x)$, $\mathcal{S}$ adds a new relation $(a, b||y, z)$ to T. In the following, we propose $\mathcal{S}$'s algorithm which uses two subroutines (defined below `find_prefix` and `is_padding`) in order to produce response z for a query (x, y). Firstly, we introduce these two algorithms and then we describe $\mathcal{S}$.

`is_padding` subroutine verifies if an input X is a correct padding of a message M or not (according to the definition of padding given above). Recall that $\mathtt{Pad}(M) = \mathtt{Pad}_1(M)||\mathtt{Pad}_2(M)$. The algorithm analyzes the length of the X to check whether it is a correct padding or not. An incorrect length means that the padding is false. Otherwise, X is split into two equal parts which are compared without considering the last two blocks since these blocks may contain the message length and the padding bits. A formal algorithm is presented as Algorithm 1.

`find_prefix`. Following the definition of padding function `Pad`, there is two possibilities to reconstruct a message M from a sequence of message blocks $X = y_1 y_2 \cdots y_{2\ell}$. The first one is from the message blocks X if it is a valid padding. The second one is from the message blocks X if it exists a message M' such that $\mathtt{Pad}(M) = \mathtt{Pad}(M')||X$. The purpose of `find_prefix` algorithm is to return a message M' (if it exists) for a given message X such that $\mathtt{Pad}(M) = \mathtt{Pad}(M')||X$ where M and M' are not given. Indeed, this is possible under constraints that X is a suffix of $\mathtt{Pad}(M)$, the last block of X contains the block length of message M and the block length of message X is bigger than the block length of message M'.

Now, we show how this algorithm works. Denote $\mathtt{Pad}(M')$ as $m'_1||m'_2||\cdots||m'_{2\ell}$. $\mathtt{Pad}(M) = m'_1||\cdots||m'_{2\ell}||m_1||\cdots||m_{2t}$. As $\mathtt{Pad}(M) = \mathtt{Pad}(M')||X$ and the length of $\mathtt{Pad}(M)$ and $\mathtt{Pad}(M')$ are even then the length of X should be even. Without loss of generality, assume that the padding bits are all located between m'_ℓ and $m'_{2\ell}$ for $\mathtt{Pad}(M')$ and between m'_{l+t} and m_{2t} for $\mathtt{Pad}(M)$.

- $t = \ell$ case. Note that $m'_{2\ell}$ is used as the last block of $\mathrm{Pad}(M')$ and the last block of $\mathrm{Pad}_1(M)$. Since the bit length of M' and M are different, a contradiction occurs. Hence `find_prefix` can not find M' in this case and return $\perp$.

Algorithm 1 is_padding(X)

```
1: Split the message $X$ to blocks $m_1||m_2||\cdots||m_\ell||m_{\ell+1}||\cdots||m_k$;
2: if ($k$ is odd) then
3:     Return false;
4: else
5:     $\ell = k/2$;
6: end if
7: for $1 \leqslant i \leqslant (\ell - 2)$ do
8:     if ($m_{\ell+i} \neq m_i$) then
9:         Return false;
10:    else
11:        if ($m_{\ell-1} = m_{2\ell-1}$) then
12:            Return true;
13:        else
14:            if ($m_{\ell-1} \neq m_{2\ell-1}$) and ($m_{\ell-1}$, $m_{2\ell-1}$) contains a padding bits then
15:                Return true;
16:            else
17:                Return false;
18:            end if
19:        end if
20:    end if
21: end for
```

• $t<\ell$ case. $\texttt{Pad}_1(M)$ should be the first $\ell+t$ blocks of Pad(M'): $\texttt{Pad}_1(M) = m'_1||\cdots||m'_\ell||m'_1||\cdots||m'_t$. By comparing $\texttt{Pad}_2(M)$ with Pad(M') from blocks $\ell+t+1$ to 2ℓ, we get $m'_{t\times i+1}||m'_{t\times i+2}||\cdots||m'_{t\times(i+1)}$ $= m'_1||\cdots||m'_t$, $0 \leqslant i \leqslant \ell/t$. Let s be the value of ℓ/t and p be $\ell \mod t$. Denote $m'_1||\cdots||m'_t$ as M_1. Denote $m'_1||\cdots||m'_p$ as M_2. Then $\texttt{Pad}(M')$ is $(M_1)^s||M_2$, where $(M_1)^s$ is repeating M_1 s times. And we have $\texttt{Pad}_1(M)$ is $(M_1)^s||M_2||M_1$. Now we focus on the first $(\ell - t)$ blocks of $\texttt{Pad}_2(M)$: $M_1^{(s-1)}||M_2$. It should be equal to the last $\ell - t$ blocks of $\texttt{Pad}(M')$: $(M_1)^{s1}||M'_2$, where M'_2 ia obtained by padding the last block of M' using $\texttt{Pad}_2$ instead of $\texttt{Pad}_1$ and thus $M'_2 \neq M_2$. So a contradiction occurs. Hence `find_prefix` cannot find M' in this case and return $\perp$.

• $t > \ell$ case. We can define $M'' = m_{t-\ell}\cdots m_t$. Check if $(m_1 \cdots m_{t-\ell-1})$ is suffix of M'' and M'' is a valid $\texttt{Pad}_2$ of any message M. If it is the case, then it can derive a message M from $\texttt{Pad}_2(M)$ and find M' from $\texttt{Pad}(M) = \texttt{Pad}(M')||X$. Hence `find_prefix` return M'.

A formal algorithm of find_prefix of an input X is presented as Algorithm 2.

Algorithm 2 find-prefix(X)

```
1: // Take a message $X$ as input and return $\perp$ or $M'$ such that $\texttt{Pad}(M) = \texttt{Pad}(M')||X$;
2: Split the message $X$ to blocks such that $X = m_1m_2\cdots m_im_{i+1}\cdots m_t$; // $m_t$ contains $\langle M\rangle = \ell$;
3: if ($t$ is odd) then
4:     Return $\perp$;
5: end if
6: if ($t \geqslant \ell + 1$) then
7:     Define $M'' = m_{t-\ell}\cdots m_t$;
8:     if ($m_1 \cdots m_{t-\ell-1}$) is suffix of $M''$ then
9:         if ($M''$) is a valid $\texttt{Pad}_2(M)$ then
10:            Derive $M$ from $\texttt{Pad}_2(M)$;
11:            Compute $\texttt{Pad}(M)$;
12:            Derive $M'$ from $\texttt{Pad}(M) = \texttt{Pad}(M')||X$.
13:            Return $M'$;
14:        else
15:            Return $\perp$;
16:        end if
17:    else
18:        Return $\perp$;
19:    end if
20: else
21:    Return $\perp$;
22: end if
```

Describe $\mathcal{S}$. For a query (x_i, y_i), $\mathcal{S}$ searches in the table T for a tuple (x_i, y_i, z_i). If the table T contains the tuple tuple (x_i, y_i, z_i) then $\mathcal{S}$ return z_i. In the other case, $\mathcal{S}$ has to build a new answer to this query. For that, it searches in the table T in order to construct a sequence of tuples $(x_1, y_1, x_2) \cdots (x_{i-1}, y_{i-1}, x_i)$ such that:

- $X = y_1 y_2 \cdots y_{i-1} y_i$ is a valid padding of message M.
- x_1= initial value (IV).

If $\mathcal{S}$ constructs such a sequence of tuples C, it can give a response that is consistent with $\mathcal{RO}$ on $\mathtt{Pad}^{-1}(y_1 y_2 \cdots y_{i-1} y_i) = M$ by calling $\mathcal{RO}$ to get $z_i = \mathcal{RO}(M)$. If it does not find such a sequence and before chooses a random z_i (and return it), it checks if $X = (y_1 y_2 \cdots y_{i-1} y)$ is suffix of a valid padding $\mathtt{Pad}(M)$ where M is to define. Firstly it calls the subroutine `find_prefix` to find an eventual M' such that $\mathtt{Pad}(M) = \mathtt{Pad}(M')||X$. If it finds such M', it computes $x' = \mathcal{RO}(M')$. As our MDHF $\mathcal{H}$ construction is not $\mathcal{PFC}$, $\mathcal{S}$ checks if $x' = x_1$ where (x_1, y_1, z_1) is the first tuple of C. If it is then $\mathcal{S}$ reconstructs a valid $\mathtt{Pad}(M)$ from which it can derive M. $\mathcal{S}$ queries M to $\mathcal{RO}$ and return $z_i = \mathcal{RO}(M)$. In all of these cases, $\mathcal{S}$ stores (x_i, y_i, z_i) into the table T. For more details see Algorithm 3.

The running time $T_\mathcal{S}$ of $\mathcal{S}$'s algorithm (Algorithm 3) is $T_\mathcal{S} \leqslant \frac{q(q+1)}{2}$ where q $= 2\ell q_{\mathcal{RO}} + q_\mathcal{S}$ such that $q_{\mathcal{RO}}$ is the number of queries to $\mathcal{RO}$ and $q_\mathcal{S}$ is the number of queries to $\mathcal{CF}$. This bound of the running time can be explained on the one hand by the fact that the number of entries in table T of $\mathcal{S}$ is $\leqslant q$ and on the other hand by the fact that $\mathcal{S}$ responds to each query (x, y) by checking whether there are any relations with previous queries in order to be consistent with $\mathcal{RO}$.

Corollary 1. An Merkle-Damgård hash function that is indifferentiable from a $\mathcal{RO}$ is not necessarily $\mathcal{PFC}$.

Proof. This corollary flows directly from the definition of our counter-example which is not $\mathcal{PFC}$ whereas it is proven $\mathcal{PRO}$ (see Appendix A).

5 $\mathcal{PFC}$ property and LEA

This section studies the relationship between $\mathcal{PFC}$ and being secure against LEA of MDHF. We noticed that all constructions [11, 20, 21] proposed in the literature have been proved secure against LEA and are $\mathcal{PFC}$. So, one may wonder whether $\mathcal{PFC}$ is necessary for an MDHF to be secure against LEA. However, in the following, we show that the $\mathcal{PFC}$ property is not required to ensure that an MDHF is secured against LEA by giving, on the one hand, a non $\mathcal{PFC}$ MDHF that resist to LEA.

On the other hand, we provide an example of a $\mathcal{PFC}$ MDHF that is not secure against LEA. Hence, these two examples allow concluding that there is no implication relationship between $\mathcal{PFC}$ and being secure against LEA.

5.1 A non-$\mathcal{PFC}$ MDHF that is resistant to LEA

From the MDHF construction proposed in Section 4 that is not $\mathcal{PFC}$ although proved $\mathcal{PRO}$, and knowing that indifferentiability has been specially crafted to capture attacks in general and LEA in particular, we can claim that our non-$\mathcal{PFC}$ MDHF construction is secure against LEA. Therefore, we can conclude that a LEA resistant MDHF is not necessary to be $\mathcal{PFC}$.

5.2 A $\mathcal{PFC}$ MDHF not resistant to LEA

5.2.1 *Specification of the example $\mathcal{H}'$*

Our example mainly introduces a permutation π after the last iteration of the MDHF while keeping the specificity of a plain MDHF unchanged.

Hash process. The message blocks are hashed in a sequential order as described in the iterative Merkle-Damgård hash function $h_i = \mathcal{CF}(h_{i-1}, m_i), 1 \leqslant i \leqslant \ell$, where IV is the initial vector and outputs h as hash digest.

Algorithm 3 Simulator

```
1: Initialize $T = \emptyset$;
2: On a new query $(x, y)$ to $\mathcal{S}$, the response is $z$;
3: if $((x, y) \in T)$ then
4:     return $z$;
5: else if $(x \notin T_x \vee T_z)$ then
6:     $z \xleftarrow{\$} \{0,1\}^n$;                    %random string
7:     if $(z \in T_x \vee T_z)$ then
8:         bad $\leftarrow$ true;
9:         $z \xleftarrow{\$} \{0,1\}^n$ / $T$;
10:        $T \leftarrow T \cup (x, y, z)$;
11:        return $z$;
12:    end if
13: else if $(x \in T_z)$ then
14:    Construct a chain $C_t$ such that $x$ is the end hash value. typically, $C_t : x_1 \xrightarrow{y_1} x_2 \xrightarrow{y_2} x_3 \cdots \xrightarrow{y_i} x$;
15:    if $(x_1 =$ IV$) \wedge$ (is_Padding$(y_1 y_2 \cdots y_i \,||y)$ then
16:        if more than one chain $C_t$ exist then
17:            $\mathcal{S}$ aborts;
18:        else if exactly one chain $C$ exists then
19:            Derive message $M$ such that $\texttt{Pad}(M) = y_1 y_2 \cdots y_i || y$;
20:            $z \leftarrow \mathcal{RO}(M)$;
21:            $T = T \cup (x, y, z)$;
22:            return $z$;
23:        end if
24:    else
25:        %$y_1 y_2 \ldots y_i \,|| y$ is an incomplete padding and $y$ contains the block length of $\texttt{Pad}(M) = \ell$;
26:        if $(M' \leftarrow$ Find_Prefix$(y_1 y_2 \cdots y_i \,|| y))$ then
27:            $x' \leftarrow \mathcal{RO}(M')$;
28:            if $x' = x_1$ then
29:                if more one chaine $C_t$ exist then
30:                    $\mathcal{S}$ aborts;
31:                else if exactly one Chaine exists then
32:                    $\texttt{Pad}(M) = [\texttt{Pad}(M') \,||\, y_1 y_2 \cdots y_i \,||\, y)]$;
33:                    Derive $(M)$;
34:                    $z \leftarrow \mathcal{RO}(M)$;
35:                    $T = T \cup (x, y, z)$;
36:                    return $z$;
37:                end if
38:            end if
39:        else
40:            $z \xleftarrow{\$} \{0,1\}^n$;
41:            if $z \in T_x \vee T_z$ then
42:                bad $\leftarrow$ true;
43:                $z \xleftarrow{\$} \{0,1\}^n$ / $T$;
44:                $T = T \cup (x, y, z)$;
45:                return $z$;
46:            end if
47:        end if
48:    end if
49: end if
```

5.2.2 *The example is $\mathcal{PFC}$*

Because of the last operation of permutation π, it is clear that the example is $\mathcal{PFC}$ since it is impossible to find two messages M and M' such that the sequential tuples representation of $\mathcal{H}(\mathrm{IV}, M)$ is the prefix of the sequential tuples representation of $\mathcal{H}(\mathrm{IV}, M')$.

5.2.3 *The example is not secure against LEA*

As π is a permutation, it is possible to compute π^{-1} for any hash digest h of unknown message M and get $h_{\ell+1}$. This latter can be used to compute the digest $h' = (\mathcal{CF}, h_{\ell+1}||m')$ of a message $(M||m')$. Hence, this example is not secure against LEA.

6 Conclusion

In this paper, we showed that the different variants of Merkle-Damgård that have been proved to be pseudo random oracle $\mathcal{PRO}$ share, in reality, the same property namely prefix-free computing. We took full advantage of the indifferentiability notion introduced by Coron et al. [11] on the hash functions which overcomes the limitations of the existing security evidence that provided only in the random oracle model. We proved that this common property (prefix-free computing) is not a necessary condition for a $\mathcal{PRO}$-secure MDHF. In addition, we explored the relationship between the prefix free computing and the length extension attack resistance and found that there is no correlation between them.

As a part of future work, we think that it would be interesting to identify the sufficient and necessary condition for a Merkle-Damgård variant to be a pseudo random oracle ($\mathcal{PRO}$).

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61602302, 61472250, 61672347), Natural Science Foundation of Shanghai (Grant No. 16ZR1416400), Shanghai Excellent Academic Leader Funds (Grant No. 16XD1401300), and 13th Five-Year National Development Fund of Cryptography (Grant No. MMJJ20170114).

References

1 Damgård I. A design principle for hash functions. In: Proceedings of the 9th Annual International Cryptology Conference, Santa Barbara, 1989. 416–427

2 Merkle R C. One way hash functions and DES. In: Proceedings of the 9th Annual International Cryptology Conference, Santa Barbara, 1989. 428–446

3 Bellare M, Rogaway P. Random oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, 1993. 62–73

4 Bellare M, Rogaway P. Optimal asymmetric encryption. In: Advances in Cryptology — EUROCRYPT'94. Berlin: Springer, 1995. 92–111

5 Andreeva E, Mennink B, Preneel B. Open problems in hash function security. Des Code Cryptogr, 2015, 77: 611–631

6 Naito Y. Indifferentiability of double-block-length hash function without feed-forward operations. In: Proceedings of the 22nd Australasian Conference on Information Security and Privacy, Auckland, 2017. 38–57

7 Bellare M, Boldyreva A, Palacio A. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In: Advances in Cryptology - EUROCRYPT 2004. Berlin: Springer, 2004. 171–188

8 Canetti R, Goldreich O, Halevi S. The random oracle methodology, revisited (preliminary version). In: Proceedings of the 30th Annual ACM Symposium on the Theory of Computing, Dallas, 1998. 209–218

9 Canetti R, Goldreich O, Halevi S. On the random-oracle methodology as applied to length-restricted signature schemes. In: Theory of Cryptography. Berlin: Springer, 2004. 40–57

10 Maurer U M, Renner R, Holenstein C. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Proceedings of the 1st Theory of Cryptography Conference on Theory of Cryptography, Cambridge, 2004. 21–39

11 Coron J, Dodis Y, Malinaud C, et al. Merkle-damgård revisited: how to construct a hash function. In: Proceedings of the 25th Annual International Cryptology Conference, Santa Barbara, 2005. 430–448

12 Lee J. Indifferentiability of the sum of random permutations toward optimal security. IEEE Trans Inform Theor, 2017, 63: 4050–4054

13 Maurer U, Renner R. From indifferentiability to constructive cryptography (and back). In: Proceedings of the 14th International Conference on Theory of Cryptography, Beijing, 2016. 3–24

14 Moody D, Paul S, Smith-Tone D. Improved indifferentiability security bound for the JH mode. Des Code Cryptogr, 2016, 79: 237–259

15 Bagheri N, Gauravaram P, Knudsen L R, et al. The suffix-free-prefix-free hash function construction and its indifferentiability security analysis. Int J Inf Secur, 2012, 11: 419–434

16 Chang D, Lee S, Nandi M, et al. Indifferentiable security analysis of popular hash functions with prefix-free padding. In: Proceedings of the 12th international conference on Theory and Application of Cryptology and Information Security, Shanghai, 2006. 283–298

17 Chang D, Nandi M. Improved indifferentiability security analysis of chopmd hash function. In: Proceedings of the

15th International Workshop on Fast Software Encryption, Lausanne, 2008. 429–443

18 Chang D, Sung J, Hong S, et al. Indifferentiable security analysis of choppfmd, chopmd, a chopmdp, chopwph, chopni, chopemd, chopcs, and chopesh hash domain extensions. IACR Cryptol ePrint Arch, 2008, 2008: 407

19 Gong Z, Lai X, Chen K. A synthetic indifferentiability analysis of some block-cipher-based hash functions. Des Code Cryptogr, 2008, 48: 293–305

20 Bellare M, Ristenpart T. Multi-property-preserving hash domain extension and the EMD transform. In: Proceedings of the 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, 2006. 299–314

21 Hirose S, Park J H, Yun A. A simple variant of the Merkle-Damgård scheme with a permutation. J Cryptol, 2012, 25: 271–309

22 Hirose S. Sequential hashing with minimum padding. Cryptography, 2018, 2: 11

23 Liskov M. Constructing an ideal hash function from weak ideal compression functions. In: Proceedings of the 13th International Workshop on Selected Areas in Cryptography, Montreal, 2006. 358–375

Appendix A Proof of indifferentiabiliy of our counter example

In this part, we give a formal indifferntaiblity proof of the above described MDHF $\mathcal{H}$ based on a hybrid argument.

Game 1. We start from the random oracle model. $\mathcal{A}$ has an oracle acces to $\mathcal{RO}$ and $\mathcal{S}$. Define G1 as the event: $\mathcal{A}$ outputs 1 after interacting with $\mathcal{RO}$ and $\mathcal{S}$ which can be represented by

$$\Pr[\mathrm{G1}] = \Pr[\mathcal{A}^{\mathcal{RO},\mathcal{S}}(1^\lambda) = 1].$$

Game 2. Define P as an intermediate program between $\mathcal{A}$ and $\mathcal{RO}$ (without any effect) such that when receiving a random Oracle query from $\mathcal{A}$, it forwards the request to $\mathcal{RO}$ and transmits the $\mathcal{RO}$ output to $\mathcal{A}$ as a response without any alteration. Define G2 as the event: $\mathcal{A}$ outputs 1 after interacting with P and $\mathcal{S}$. It can be represented by $\Pr[\mathrm{G2}]$=$\Pr[\mathcal{A}^{P,\mathcal{S}}(1^\lambda) = 1]$. It is easy to see that the distribution of this game is identical to the previous one and hence $\Pr[\mathrm{G2}]$= $\Pr[\mathrm{G1}]$.

Game 3. We make a small change on $\mathcal{S}$ by introducing conditions that can push the new $\mathcal{S}'$ to collapse. Indeed, these constraints are events representation that $\mathcal{A}$ can exploit to perform $\mathcal{S}$ response in conflict with that of $\mathcal{RO}$. These events are as follows.

Event 1. For a query (x, y), $\mathcal{S}$ responds with z such that $z = z'$ where (x', y', z') is a previous query-response and $(x, y) \neq (x', y')$.

Event 2. For a query (x, y), $\mathcal{S}$ responds with z such that $z = x'$ where (x', y', z') is a previous query-responds and $(x, y) \neq (x', y')$.

In the following, we describe how $\mathcal{A}$ can exploit these events to show an inconsistency in the responses of $\mathcal{S}$ and $\mathcal{RO}$.

If $\mathcal{A}$ find an event 1, he can choose two different messages M and M' such that $\mathtt{Pad}(M)$ and $\mathtt{Pad}(M')$ share a common suffix. So, for a query sequences $\mathtt{Pad}(M)$ and $\mathtt{Pad}(M')$ to $\mathcal{S}$, $\mathcal{A}$ gets the same value z for the end block $(x_{2\ell}, y_{2\ell}, z)$ when for a query $\mathcal{RO}(M)$ and $\mathcal{RO}(M')$ he will get different value z and z'. $\mathcal{S}$ can be consistent with only one of these inputs.

For event 2, $\mathcal{A}$ can force $\mathcal{S}$ to not be consistent with $\mathcal{RO}$ for the same message M. He achieves this goal by querying $\mathcal{S}$ not in the correct sequence $(x_1, y_1), (x_2, y_2), \dots, (x_{2\ell}, y_{2\ell})$. Thus, $\mathcal{A}$ can deduce the value $\mathcal{RO}(M)$ from the responses sequences $\mathcal{S}$ which is different from the response of $\mathcal{RO}(M)$ (it can be same with a probability of 2^{-n}).

Note that because of the suffix freeness of our construction $\mathcal{H}$, $\mathcal{A}$ cannot exploit $S(x, y) = z'$ such that $(x'_{2\ell}, y'_{2\ell}, z')$ is the last tuple of a sequence of queries of a valid $\mathtt{Pad}(Y') = y'_1, \dots, y'_\ell$. Also, as the length of the message is integrated in the padding function, $\mathcal{A}$ cannot exploit $S(x, y) = \mathrm{IV}$ (where IV is the initial value).

Let us compute the probability that $\mathcal{A}$ finds these events. Denote q_p the number of queries made by $\mathcal{A}$ to P (which is equal to $q_{\mathcal{RO}}$ the number made to $\mathcal{RO}$) and $q_{\mathcal{S}'}$ is the number of queries made by $\mathcal{A}$ to $\mathcal{S}'$ (which is equal to $q_{\mathcal{S}}$ the number made to $\mathcal{S}$).

The event 1 can hold if one of this case is realized.

(1) (x_i, y_i) and (x'_j, y'_j) are the last queries in a sequence of $\mathcal{RO}$ query to $\mathcal{S}$. The probability of this event is equal to the probability collision of $\mathcal{RO}$. $\Pr = \frac{q^2_{\mathcal{RO}}}{2^n}$.

(2) (x_i, y_i) and (x'_j, y'_j) are an intermediate queries in a sequence of $\mathcal{RO}$ query to $\mathcal{S}$. The probability of this event is equal to the probability collision of the uniformly random function on $q_{\mathcal{S}}$. $\Pr = \frac{q^2_{\mathcal{S}}}{2^n}$.

(3) (x_i, y_i) is the last query in a sequence of $\mathcal{RO}$ query to $\mathcal{S}$ but (x'_j, y'_j) is an intermediate one. This case does not help $\mathcal{A}$ to improve its advantage to made an inconsistency between $\mathcal{RO}$ and $\mathcal{S}$ since our construction $\mathcal{H}$ is suffixe free. Therefore, the occurrence probability of event 1, Pr[event1], can be bounded by the birthday bound over $(q_{\mathcal{S}} + q_{\mathcal{RO}}) \leqslant \frac{(q_{\mathcal{S}}+q_{\mathcal{RO}})^2}{2^n}$.

For event 2, it can be exploited by $\mathcal{A}$ only if the answer z for a query (x, y) is chosen by $\mathcal{S}$ regardless of $\mathcal{RO}$. In the case of the response of $\mathcal{S}$ is chosen to be consistent with $\mathcal{RO}$ but it collides with a previous input to $\mathcal{S}$, it has no effect on advantage of $\mathcal{A}$. Hence, the occurrence probability of event 2, Pr[event2], can be bounded by $\Pr[\mathrm{event2}] \leqslant \frac{q^2_{\mathcal{S}}}{2^n}$.

Define G3 as the event: $\mathcal{A}$ outputs 1 after interacting with P and $\mathcal{S}'$. It can be represented by $\Pr[\mathrm{G3}]$=$\Pr[\mathcal{A}^{P,\mathcal{S}'}(1^\lambda) = 1]$. Let us analyze the distribution of $\mathcal{S}'$, P function in this game compared to $\mathcal{S}$, P in the previous one.

Before starting, first define some properties of $\mathcal{S}'$.

Claim 1. If $\mathcal{S}'$ does not collapse, then there are no two sequential tuples representation $(\mathcal{S}', x_1||y_1, z_1)$, $(\mathcal{S}', x_2||y_2, z_2), \ldots,$ $(\mathcal{S}', x_i||y_i, z_i)$, $\ldots$, $(\mathcal{S}', x_{2\ell}||y_{2\ell}, z)$ and $(\mathcal{S}', x'_1||y'_1, z'_1)$, $(\mathcal{S}', x'_2||y'_2, z'_2), \ldots,$ $(\mathcal{S}', x'_i||y'_i, z'_i), \ldots, (\mathcal{S}', x'_{2\ell}||y'_{2\ell}, z')$ such that:

- $y_1||\cdots||y_{2\ell}$ and $y'_1||\cdots||y'_{2\ell}$ are a valid padding.
- $x_1 = x'_1 = \mathrm{IV}$.
- For $1 \leqslant j \leqslant 2\ell$, $z_j = x_{j+1}$ and $z'_j = x'_{j+1}$.
- $(x_{2\ell}, y_{2\ell}, z) = (x'_{2\ell}, y'_{2\ell}, z')$.

Proof. Using an induction on $q_{\mathcal{S}'}$, we will show that such sequences of tuples representation cannot exist unless $\mathcal{S}'$ collaps.

For $q_{\mathcal{S}'} = 0$, the claim is true.

Assume that the claim is true when $q_{\mathcal{S}'} = n$ have been made to $\mathcal{S}'$. Suppose after $n+1$ queries to $\mathcal{S}'$ there exists two sequential tuples representation that satisfy claim properties. As $((x_{2\ell}, y_{2\ell}, z) = (x'_{2\ell}, y'_{2\ell}, z'))$, we can deduce that it exists $y_{2\ell-r}\cdots y_{2\ell}$ and $y'_{2\ell-r}\cdots y'_{2\ell}$ such that $\forall s \in \{0, r\} : (x_{2\ell-s}, y_{2\ell-s}, z_{2\ell-s}) = (x'_{2\ell-s}, y'_{2\ell-s}, z'_{2\ell-s})$.

- $(2\ell - r > 1)$ case. If $(x_{2\ell-r} = x'_{2\ell-r})$ then $(z_{2\ell-r-1} = z'_{2\ell-r-1})$. Because of event 1, this push $\mathcal{S}'$ to collapse.
- $(2\ell - r = 1)$ case. If $(x_1, y_1, z_1) = (x'_1, y'_1, z'_1)$ then $y_1\cdots y_{2\ell}$ is same with $y'_1\cdots y'_{2\ell}$ which contradicts the supposition that M and M' are different.

Therefore, there is no two different messages $y_1\cdots y_{2\ell}$ and $y'_1\cdots y'_{2\ell}$ such that $(x_{2\ell}, y_{2\ell}, z) = (x'_{2\ell}, y'_{2\ell}, z')$ if $\mathcal{S}'$ does not collapse.

In the following we show that the only way by which $\mathcal{A}$ can get from $\mathcal{S}$ a random oracle output for an input $\mathtt{Pad}(M) = y_1\cdots y_{2\ell}$ (or $y'_1\cdots y'_{2t}$ such that $\mathtt{Pad}(M) = \mathtt{Pad}(M')||y'_1\ldots y'_{2t})$ is by doing an ordered sequence of queries (x_i, y_i) for $1 \leqslant i \leqslant 2\ell$ (or $1 \leqslant i \leqslant 2t$).

Claim 2. If $\mathcal{S}'$ does not collapse, at a given moment where it has in table T, $(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_{2\ell}, y_{2\ell}, z_{2\ell})$ such that

- $y_1||\cdots||y_{2\ell}$ is a valid padding.
- $x_1 = \mathrm{IV}$.
- For $1 \leqslant j \leqslant 2\ell$, $z_j = x_{j+1}$.

Then $\mathcal{S}'$ has constructed T following an ordered sequence of queries $(x_1, y_1), (x_2, y_2), \cdots, (x_{2\ell}, y_{2\ell})$.

Proof. Using a contradiction proof, we will assume that the sequence $(x_1, y_1, z_1), (x_2, y_2, z_2), \cdots, (x_{2\ell}, y_{2\ell}, z_{2\ell})$ was not done in the right order. Therefore we can deduce that there exists an entry $(x_{i+1}, y_{i+1}, z_{i+1})$ in T before a query (x_i, y_i) for $1 \leqslant i \leqslant 2\ell - 1$ was made. In this case, the response z_i of a query (x_i, y_i) should be chosen regardless of $\mathcal{RO}$ (i.e., randomly) by $\mathcal{S}'$ such as $z_i = x_{i+1}$ (the condition 3 in the claim). Hence, $\mathcal{S}'$ will collapse because of event 2 holds.

Therefore, the only possible interaction between $\mathcal{A}$ and $\mathcal{S}'$ is that mentioned in Claim 2.

Claim 3. If $\mathcal{S}'$ does not collapse, at a given moment where it has in table T, $(x'_1, y'_1, z'_1), (x'_2, y'_2, z'_2), \ldots, (x'_{2t}, y'_{2t}, z'_{2t})$ such that

- For $1 \leqslant j \leqslant 2t$, $z'_j = x'_{j+1}$.
- $Y = (y'_1||y'_2, \ldots, y'_{2t})$ is suffix of $\mathtt{Pad}(M)$ (i.e., $\mathtt{Pad}(M) = \mathtt{Pad}(M')||Y$).
- $x_1 = h$ such that $\mathcal{RO}(M') = h$.

Then $\mathcal{S}'$ has constructed T following an ordered sequence of queries $(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_{2t}, y'_{2t})$.

Proof. The proof is similar with the proof of Claim 2, and thus omitted.

After a review of some properties of $\mathcal{S}'$ let us comeback to view distribution between $(\mathcal{S}', P)$ in Game 3 and $(\mathcal{S}, P)$ in the previous game.

Our goal is to show that

- For any query $(\mathtt{Pad}^{-1}(y_1||\cdots||y_{2\ell})$, P of Game 3 is consistent with P of Game 2.
- For any query (x_i, y_i), $\mathcal{S}'$ of Game 3 is consistent with $\mathcal{S}$ of Game 2.
- The responses of $\mathcal{S}'$ in Game 3 is always consistent with P in the same Game.

For the first point, it is obvious as Games 2 and 3 use the same program P.

For the second point, as $\mathcal{S}'$ is defined on $\mathcal{S}$ with additional constraints of collapses. Hence, we can deduce that $\mathcal{S}$ is consistent with $\mathcal{S}'$ unless the constraints of collapses happen.

We show that the responses of $\mathcal{S}'$ is always consistent with P.

Claim 4. If there is no sequence of entries in table T, $(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_{2\ell}, y_{2\ell}, z_{2\ell})$ such that

- $y_1||\cdots||y_{2\ell}$ is a valid padding.
- $x_1 = \mathrm{IV}$.
- For $1 \leqslant j \leqslant 2\ell$, $z_j = x_{j+1}$.
- The P's response to a $(\mathtt{Pad}^{-1}(y_1||\cdots||y_{2\ell})$ query is different from $(z_{2\ell})$.

Then $\mathcal{S}'$ does not collapse.

Proof. By definition, P is just a relay algorithm between $\mathcal{A}$ and $\mathcal{RO}$. So, P responds for any query $(\mathtt{Pad}^{-1}(y_1||\cdots||y_{2\ell})$ by $\mathcal{RO}(\mathtt{Pad}^{-1}(y_1||\cdots||y_{2\ell})$ from one hand. On other hand, from Claims 2 and 3, we know that if $\mathcal{A}$ want to compute $\mathcal{H}^{\mathcal{S}'}$ on $\mathtt{Pad}(M) = y_1||\cdots||y_{2\ell}$ by using $\mathcal{S}'$, the only way that it has (to not collapse $\mathcal{S}'$) is to queries an ordered sequence $(x_1, y_1), \ldots, (x_{2\ell}, y_{2\ell})$.

From the two-first conditions of the claim, we can say that $\mathcal{S}'$ has in T $(x_1, y_1, z_1), \ldots, (x_{2\ell-1}, y_{2\ell-1}, z_{2\ell-1})$ before the query $(x_{2\ell}, y_{2\ell})$.

By definition, unless $\mathcal{S}'$ collapses, the only possibility response of $\mathcal{S}'$ to a query $(x_{2\ell}, y_{2\ell})$ is $\mathcal{RO}(\mathtt{Pad}^{-1}(y_1||\cdots||y_{2\ell})$ since $(y_1||\cdots||y_{2\ell}$ is valid padding, y_1= IV and for $1 \leqslant j \leqslant 2\ell - 1, z_j = x_{j+1}$.

Hence, unless $\mathcal{S}'$ collapses, it is always consistent with P.

Finally, we can say the view of $\mathcal{A}$ between Games 2 and 3 differs only where $\mathcal{S}'$ collapses. This event probability equals to the one of Event 1 or 2.

$$\begin{aligned}|\Pr[\mathrm{G3}]-\Pr[\mathrm{G2}]| &= \Pr[\mathcal{S}' \text{ collapses}]\\ &= \Pr[\text{event1}]+\ \Pr[\text{event2}]\\ &\leqslant \frac{(q_{\mathcal{S}'}+q_{\mathcal{RO}})^2}{2^n}+\frac{(q_{\mathcal{S}'})^2}{2^n}\\ &\leqslant O\left(\frac{q^2}{2^n}\right).\end{aligned}$$

Game 4. A small change to P occurred such that the new program P' relies on $\mathcal{S}'$ instead of $\mathcal{RO}$ in its responses. Hence, $\mathcal{A}$ has an oracle access to P' which has an oracle access to $\mathcal{S}'$. In order to stay consistent, we give the capability to P' to compute $\texttt{Pad}(Y)=y_1||\cdots||y_{2\ell}$ for any random oracle query (Y) before make successive queries (x_i,y_i) to $\mathcal{S}'$. Therefore, P' is almost the same as our construction $\mathcal{H}$ except of using $\mathcal{S}'$ instead of $\mathcal{CF}$.

Define G4 as the event: $\mathcal{A}$ outputs 1 after interacting with P' and $\mathcal{S}'$. It is represented by Pr[G4]= $\Pr[\mathcal{A}^{P',\mathcal{S}'}(1^\lambda)=1]$.

In the following we show that the view of $\mathcal{A}$ between Games 3 and 4 does not differ because, unless $\mathcal{S}'$ collapses, there is a consistency in responses of

- P and $\mathcal{S}'$ of Game 3.
- $\mathcal{S}'$ of Game 4 and $\mathcal{S}'$ of Game 3.
- P' and $\mathcal{S}'$ of Game 4.
- P' of Game 4 and P of Game 3.

Proof. For the first point, we shown in Claim 4 that P and $\mathcal{S}'$ in Game 3 are consistent unless $\mathcal{S}'$ collapses. For the second point, it is obvious that it is right since $\mathcal{S}'$ in Games 3 and 4 are same. P' and $\mathcal{S}'$ of Game 4 are consistent because by definition P' is based on $\mathcal{S}'$. To show P' of Game 4 is consistent with P of Game 3 we will use a transitivity proof for P, $\mathcal{S}'$ and P'. Recall that, unless $\mathcal{S}'$ collapses, P and $\mathcal{S}'$ are consistent (Claim 4) and $\mathcal{S}'$ is consistent with P' than we can deduce that P and P' are consistent unless $\mathcal{S}'$ collapses.

Therefore, we can conclude that the difference of $\mathcal{A}$'s view between Games 3 and 4 is summed up only when $\mathcal{S}'$ collapses in Games 3 and 4: $|\Pr[G_4]-\Pr[G_3]|\leqslant \Pr[\mathcal{S}'$ collapses in Game 3] + $\Pr[\mathcal{S}'$ collapses in Game 4], where $\Pr[\mathcal{S}'$ collapses in Game 3] $=\frac{(q_{\mathcal{S}'}+q_P)^2}{2^n}+\frac{(q_{\mathcal{S}'})^2}{2^n}$. Let $q=q_{\mathcal{S}'}+q_P$ then $\Pr[\mathcal{S}'$ collapses in Game 3] $\leqslant \frac{q^2}{2^n}$.

Let us compute $\Pr[\mathcal{S}'$ collapses in Game4]. Assume that the maximum length of padding function $\texttt{Pad}$ is 2ℓ then the total number of query that $\mathcal{S}'$ received is $(q_{\mathcal{S}'}+2\ell q_{P'})$.

$$\Pr[\mathcal{S}'\text{collapses in Game 4}]=\frac{(q_{\mathcal{S}'}+2\ell q_{P'})^2}{2^n}+\frac{q_{\mathcal{S}'}^2}{2^n},$$

$$\Pr[\mathcal{S}'\text{collapses in Game 4}]\leqslant\frac{(2\ell q)^2}{2^n}+\frac{q^2}{2^n},$$

$$|\Pr[G_4]-\Pr[G_3]|=O\left[\frac{(\ell q)^2}{2^n}\right].$$

Game 5. This time we make changes to $\mathcal{S}'$ so that the new $\mathcal{S}''$:

- Is independent from $\mathcal{RO}$.
- Does not have to check about the collapse events.

So, for a query (x,y), $\mathcal{S}''$ will look in its table T if there is an entry (x,y,z). If it is, then it returns z. Otherwise it returns a uniformly random z, while updating its table T by (x,y,z).

In the following we show that the view of $\mathcal{A}$ between Games 4 and 5 does not differ unless $\mathcal{S}'$ collapses or $\mathcal{S}''$ returns with z that satisfy one of the collapse events. Indeed, if these two conditions are not satisfied, $\mathcal{A}$ sees $\mathcal{S}'$ and $\mathcal{S}''$ as same because both of them use a random oracle that is uniformly distributed.

As the responses distribution of $\mathcal{S}'$ and $\mathcal{S}''$ are identical, it can be deduced that the probability of occurrence of these two conditions is the same.

Define G_5 as the event: $\mathcal{A}$ outputs 1 after interacting with P' and $\mathcal{S}''$. It is represented by Pr[G5]= $\Pr[\mathcal{A}^{P',\mathcal{S}''}(1\lambda)=1]$. $|\Pr[G_5]-\Pr[G_4]|\leqslant\Pr[\mathcal{S}'$ collapse in Game 4] + $\Pr[\mathcal{S}''$ collapse in Game 5]=$O[\frac{(\ell q)^2}{2^n}]$.

Game 6. We come to the last game that should simulate $\mathcal{CF}$ and $\mathcal{H}$. In this game $\mathcal{S}''$ is replaced by $\mathcal{CF}$. Thus, P' will be based on $\mathcal{CF}$ and become by definition equivalent to $\mathcal{H}^{\mathcal{CF}}$.

It is easy to see that Game 6 is identical to Game 5. Define G$_6$ as the event: $\mathcal{A}$ outputs 1 after interacting with $\mathcal{H}^{\mathcal{CF}}$ and $\mathcal{CF}$. It is represented by $\Pr[G_6]=\Pr[\mathcal{A}^{\mathcal{H}^{\mathcal{CF}},\mathcal{CF}}(1^\lambda)]=1$, as Game 6 is equivalent to Game 5 then $\Pr[G_6]=\Pr[G_5]$. Recall that Game 1 emulates $(\mathcal{RO},\mathcal{S})$ and Game 6 is $(\mathcal{H},\mathcal{CF})$. By transitivity between Games 1 and 6, we can conclude that: Adv($\mathcal{A}$)= $|\Pr[\mathcal{A}^{\mathcal{H},\mathcal{CF}}=1]-\ \Pr[\mathcal{A}^{\mathcal{RO},\mathcal{S}}=1]|$ = O$[\frac{(\ell q)^2}{2^n}]$.

Appendix B Chop-MD, NMAC, SFPF, MDP are $\mathcal{PFC}$

Chop-MD is $\mathcal{PFC}$. For a message $M=m_0,\ldots,m_n$, the sequential tuples representation of Chop-MD hash function $\mathcal{H}^{\mathcal{CF},S}(\mathrm{IV},M)$, using a function S that outputs $(n-s)$-bits for any n-bits input, is $(\mathcal{CF},h_0||m_0,h_1),(\mathcal{CF},h_1||m_1,h_2),\ldots,$ $(\mathcal{CF},h_n||m_n,h_{n+1}),(S,h_{n+1},h_{n+2})$. This sequential tuples representation is prefix of another sequential tuples representation of $\mathcal{H}^{\mathcal{CF},S}(\mathrm{IV},M||m)=(\mathcal{CF},h_0||m_0,h_1),(\mathcal{CF},h_1||m_1,h_2),\ldots,(\mathcal{CF},h_n||m_n,h_{n+1}),(\mathcal{CF},h_{n+1}||m,h'_{n+2}),S(h'_{n+2},h'_{n+3})$

if and only if the adversary $\mathcal{A}$ can recover the h_{n+1} chopped s-bits from h_{n+2} which is possible with probability $(1/2^s)$. So, depending on the length size of chopped-bits (s), Chop-MD is $\mathcal{PFC}$.

NMAC is $\mathcal{PFC}$. NMAC-MD provided by Coron et al. [11] is $\mathcal{PFC}$ because, for a message $M = m_0, \ldots, m_n$, the sequential tuples representation of $\mathcal{H}^{\mathcal{CF}_1,\mathcal{CF}_2}(\mathrm{IV}, M) = (\mathcal{CF}_1, h_0||m_0, h_1), (\mathcal{CF}_1, h_1||m_1, h_2), \ldots, (\mathcal{CF}_1, h_n||m_n, h_{n+1}), (\mathcal{CF}_2, h_0||h_{n+1}, h)$ can be prefix of another sequential tuples representation of $\mathcal{H}^{\mathcal{CF}_1,\mathcal{CF}_2}(\mathrm{IV}, M||m) = (\mathcal{CF}_1, h_0||m_0, h_1), (\mathcal{CF}_1, h_1||m_1, h_2), \ldots, (\mathcal{CF}_1, h_n||m_n, h_{n+1}), (\mathcal{CF}_1, h_{n+1}||m, h_{n+2}), (\mathcal{CF}_2, h_0||h_{n+2}, h)$ if and only if $(\mathcal{CF}_2, h_0||h_{n+1}, h) = (\mathcal{CF}_1, h_n||m_n, h_{n+1})$, that is $h_n = h_0$ and $m_n = h_{n+1}$ and $h_{n+1} = h$. It is obvious that this is possible with a negligible probability.

SFPF is $\mathcal{PFC}$. SFPF provided by Bagheri et al. [15] is $\mathcal{PFC}$ because for a message $M = m_0, \ldots, m_n$, the sequential tuples representation of a SFPF hash function $\mathcal{H}^{\mathcal{CF}_1,\mathcal{CF}_2,\mathcal{CF}_3}(\mathrm{IV}, M)$ is $(\mathcal{CF}_1, h_0||m_0, h_1)$, $(\mathcal{CF}_2, h_1||m_1, h_2)$, $\ldots$, $(\mathcal{CF}_2, h_{n-1}||m_{n-1}, h_n)$, $(\mathcal{CF}_3, h_n||m_n, h_{n+1})$. This later can be prefix of another sequential tuples representation of $\mathcal{H}^{\mathcal{CF}_1,\mathcal{CF}_2,\mathcal{CF}_3}(\mathrm{IV}, M||m) = (\mathcal{CF}_1, h_0||m_0, h_1)$, $(\mathcal{CF}_2, h_1||m_1, h_2)$, $\ldots$, $(\mathcal{CF}_2, h_n||m_n, h'_{n+1})$, $(\mathcal{CF}_3, h'_{n+1}||m, h)$ if and only if $(\mathcal{CF}_3, h_n||m_n, h_{n+1}) = (\mathcal{CF}_2, h_n||m_n, h'_{n+1})$ i.e., $h'_{n+1} = h_{n+1}$ which is possible with a negligible probability $(1/2^n)$ as $\mathcal{CF}_1$ and $\mathcal{CF}_2$ are FIL-$\mathcal{RO}$.

MDP is $\mathcal{PFC}$. MDP proposed by Hirose et al. [21] is $\mathcal{PFC}$ because for a message $M = m_0, \ldots, m_n$, the sequential tuples representation of MDP hash function $\mathcal{H}^{\mathcal{CF},\pi}(\mathrm{IV}, M)$ using a permutation π is $(\mathcal{CF}, h_0||m_0, h_1)$, $(\mathcal{CF}, h_1||m_1, h_2)$, $\ldots$, $(\mathcal{CF}, h_n||m_n, h_{n+1}), (\pi, h_{n+1}, h)$. This later is prefix of another tuples representation of MDP hash function $\mathcal{H}^{\mathcal{CF},\pi}(\mathrm{IV},M||m)$ $= (\mathcal{CF}, h_0||m_0, h_1)$, $(\mathcal{CF}, h_1||m_1, h_2)$, $\ldots$, $(\mathcal{CF}, h_n||m_n, h_{n+1})$, $(\mathcal{CF}, h_{n+1}||m, h_{n+2})$, (π, h_{n+2}, h') if and only if $h = h_{n+1}$. The probability to have a fixed point for an permutation $(\pi(h) = h)$ is negligible $(1/2^n)$.