Indifferentiable Security Analysis of choppfMD, chopMD, a chopMDP, chopWPH, chopNI, chopEMD, chopCS, chopESh, a pfCM-chopMD Hash Domain Extensions

Donghoon Chang[1], Jaechul Sung[2], Seokhie Hong[1], and Sangjin Lee[1]

[1] Center for Information Security Technologies(CIST), Korea University, Korea
pointchang@gmail.com
{hsh,sangjin}@cist.korea.ac.kr
[2] Department of Mathematics, University of Seoul, Korea
jcsung@uos.ac.kr

Abstract. First, we provide indifferentiable security analyses of choppfMD, chopMD, a chopMDP (where the permutation P is to be xored with any non-zero constant.), chopWPH (the chopped version of Wide-Pipe Hash proposed in [16]), chopEMD, chopNI, chopCS, chopESh hash domain extensions. Even though there are security analysis of them in the case of no-bit chopping (i.e., $s = 0$), there is no unified way to give security proofs. Next, we design a prefix-free-Counter-Masking-MD (pfCM-MD) and a prefix-free-Counter-Masking-chopMD (pfCM-chopMD) hash domain extensions and provide indifferentiable security proofs of them. Our two new domain extensions also guarantee the full n-bit security bound against the Kelsey-Schneier second preimage attack and a new second preimage attack proposed by Elena *et al.*, where n is the output bit-size of domain extensions. All our proofs in this paper follow the technique introduced in [3]. These proofs are simple and easy to follow.

1 Introduction

Till now, known indifferentiable security analysis of hash domain extensions are not easy and the proofs are very long. The following question is natural : how can we easily and simply prove the indifferentiable security of any given hash domain extension? Recently, Bertoni *et al.* [3] showed the possibility of simple indifferentiable security. In this paper, we revisit their proof technique, and through this work, we give the indifferentiable security of eight constructions and their truncated versions. *We hope that submitters of candidate of SHA-3 can prove the indifferentiable security of their hash functions as we prove nine constructions in this paper.*

Remark. In the case of SHA-2 family, SHA-224 is defined by truncating the least significant 32 bits of the final hash output. Likewise, SHA-384 is defined by truncating the least significant 128 bits of the final hash output. The truncation is attractive method to get a hash family for supporting variable output sizes. Among nine constructions, in the case of WPH, there is no indifferentiable security proof. Even though there security proofs for chopMD construction [9], the proof is a little bit complicated. And in the case of choppfMD, there is only a theorem statement without any security proof [19].

2 Some Notations and Results

In the keyless setting, we consider the compression function $f : \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$. We write $||m||_b = t$ if $m \in \{\{0,1\}^b\}^t$, where t is the b-bit block size. Similarly, we write $||c||_n = t$ if $c \in \{\{0,1\}^n\}^t$, where t is the n-bit block size. In the dedicated-key setting, we consider the compression function $f : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$, where $\{0,1\}^k$ is a key space. When a key K is fixed, we write f with K by $f_K(\cdot,\cdot)$ or $f(K,\cdot,\cdot)$.

MD. The traditional Merkle-Damgård extension (MD) [20, 11] works as follow: for a message $M = m_1||\cdots||m_t$, $\mathrm{MD}^f(M) = f(\cdots f(f(IV, m_1), m_2)\cdots, m_t)$, where f is a compression function and IV is the initial value.

Padding. Except chopEMD, chopCS, chopESh, we say any injective and length-consistent function $g : \{0,1\}^* \rightarrow (\{0,1\}^b)^+$ as a padding rule. In cases of chopEMD, chopCS, chopESh, we say any injective and length-consistent function $g : \{0,1\}^* \rightarrow (\{0,1\}^b)^+ \times \{0,1\}^{b-n}$ as a padding rule. We say g is a prefix-free padding if for any $M \neq M'$ $g(M)$ is not a prefix of $g(M')$.

chop. For $0 \leq s \leq n$ we define $\mathrm{chop}_s(x) = x_L$ where $x = x_L \parallel x_R$ and $|x_R| = s$.

last. For $0 \leq s \leq n$ we define $\mathrm{last}_s(x) = x_R$ where $x = x_L \parallel x_R$ and $|x_R| = s$.

pfMD. prefix-free MD (shortly, pfMD) is defined as follows : $\mathrm{pfMD}^f_g(M) = \mathrm{MD}^f(g(M))$ where g is a prefix-free padding.

chopMD. For $0 \leq s \leq n$ we define $\mathrm{chopMD}^f_g(M) = \mathrm{chop}_s(\mathrm{MD}^f(g(M)))$, where g is any padding rule.

choppfMD. chop-prefix-free MD (shortly, choppfMD) is defined as follows : $\mathrm{choppfMD}^f_g(M) = \mathrm{chop}_s(\mathrm{MD}^f(g(M)))$ where g is a prefix-free padding. Note that choppfMD with $s = 0$ is pfMD. In other words, pfMD is a special case of choppfMD. HAIFA [7] is an example of choppfMD.

MDP. MD with a permutation (shortly, MDP) [12] is defined as follows : $\mathrm{MDP}^f_g(M) = f(P(\mathrm{MD}^f(\mathrm{chop}_b(g(M)))), \mathrm{last}_b(g(M)))$ where P is a permutation, g is any padding rule and $\mathrm{last}_s(x) = x_R$ where $x = x_L \parallel x_R$ and $|x_R| = s$. And P and P^{-1} is efficiently computable. In this paper, we only consider to be xored with a non-zero constant P.

chopMDP. chopMDP is defined as follows : $\text{chopMDP}_g^f(M) = \text{chop}_s\,(f(P(\text{MD}^f(\,\text{chop}_b(g(M)))), \text{last}_b(g(M))))$ where P is a permutation, g is any padding rule and $\text{last}_s(x) = x_R$ where $x = x_L \parallel x_R$ and $|x_R| = s$. And P and P^{-1} is efficiently computable. Note that chopMDP with $s = 0$ is MDP. In other words, MDP is a special case of chopMDP. In this paper, we only consider to be xored with a non-zero constant P.

WPH. Wide-Pipe Hash (shortly, WPH) is proposed by Lucks [16]. Wide-Pipe Hash use two independent functions f_1 and f_2, where $f_1 : \{0,1\}^w \times \{0,1\}^b \to \{0,1\}^w$ and $f_2 : \{0,1\}^w \to \{0,1\}^n$ and $w \geq 2n$. Given any padding rule g, WPH works as follows : for a message M, $\text{WPH}_g^{f_1,f_2}(M)$=$f_2(\text{MD}^{f_1}(g(M)))$, where the initial value of IV is w-bit.

chopWPH. The chopped Wide-Pipe Hash (shortly, chopWPH) works as follows : for a message M, $\text{chopWPH}_g^{f_1,f_2}(M)$=$\text{chop}_s(f_2(\text{MD}^{f_1}(g(M))))$. In this paper, we provide an indifferentiable security bound for any n and w. That is, there is no restriction that $w \geq 2n$. Note that chopWPH with $s = 0$ is WPH. In other words, WPH is a special case of chopWPH.

EMD. EMD [5] is defined as follows : $\text{EMD}^f(M) = f(IV_2, \text{MD}^f(Q)||M_t)$, where IV_2 is a fixed value different from IV, $Q||M_t = g(M) = M||10^r||\mathsf{bin}_{64}(|M|)$, where $|M_t| = b - n$, $\mathsf{bin}_i(x)$ means the i-bit binary representation of x, r is the smallest non-negative integer such that $|g(x)| - (b - n)$ is a multiple of b.

chopEMD. chopEMD is defined as follows : $\text{chopEMD}^f(M) = \text{chop}_s(\text{EMD}^f(M))$. Note that chopEMD with $s = 0$ is EMD. Since we focus on the indifferentiable security of chopEMD, *we assume that g is not such specific padding rule but any padding rule.*

NI. Nested Iteration (shortly, NI) [1] is defined as follows : $\text{NI}_g^f(K_1, K_2, M) = f(K_2, \text{MD}^{f_{K_1}}(\text{chop}_b(g(M))), \text{last}_b(g(M)))$, where g is any padding rule.

chopNI. Chopped Nested Iteration (shortly, chopNI) is defined as follows : $\text{chopNI}_g^f(K_1, K_2, M) = \text{chop}_s(\text{NI}_g^f(K_1, K_2, M))$, where g is any padding rule. Note that chopNI with $s = 0$ is NI.

CS. Chain Shift (shortly, CS) [18] is defined as follows : $\text{CS}_g^f(K, M)$=$f(K$,IV_2, $\text{MD}^{f_K}(\text{chop}_{b-n}(g(M)))$, $\text{last}_{b-n}(g(M)))$, where g is any padding rule.

chopCS. Chopped Chain Shift (shortly, chopCS) is defined as follows : $\text{chopCS}_g^f(K, M) = \text{chop}_s(\text{CS}_g^f(K, M))$, where g is any padding rule. Note that chopCS with $s = 0$ is CS.

ESh. Enveloped Shoup (shortly, ESh) [6] is defined as follows : $\mathrm{ESh}^f_g(K,(K_0, \cdots, K_r),M) = f_K(IV_2 \oplus K_{\nu(1)}, \mathrm{Shoup}^f_{K,\overline{K}}(M') \oplus K_{\nu(t)}, M_t)$, where $\nu_2(i) = j$ if $2^j | i$ and $2^{j+1} \not| i$, for $1 \leq i \leq t-1$ $|M_i| = b$, $|M_t| = b-n$, $\mathrm{Shoup}^f_{K,\overline{K}}(M') = f_K(\cdots f_K(f_K(IV_1 \oplus K_{\nu(1)}, m_1) \oplus K_{\nu(2)}, m_2) \oplus K_{\nu(3)}, \cdots \oplus K_{\nu(t-1)}, m_{t-1})$, $M' = m_1||\cdots||m_{t-1}$, $\overline{K} = K_1||\cdots||K_{\nu(t-1)}$, g is any padding rule such that $g(M) = m_1||\cdots||m_t$.

chopESh. Chopped Enveloped Shoup (shortly, chopESh) is defined as follows : $\mathrm{chopESh}^f_g(K,M) = \mathrm{chop}_s(\mathrm{ESh}^f_g(K,M))$, where g is any padding rule. Note that chopESh with $s = 0$ is ESh.

pfCM-MD. CM-MD (MD with a counter-masking) works similar to MD as follow : for given a message M such that $g(M) = m_1||\cdots||m_t$, $\mathrm{CM\text{-}MD}^f_g(M) = f(\cdots f(f(IV \oplus c_0, m_1) \oplus c_1, m_2) \oplus c_3, \cdots, m_t)$, where g is any padding rule. When for any two $c = c_0||\cdots||c_t$ and $c' = c'_0||\cdots||c'_{t'}$ c is not a prefix of c', we say its counter-masking is prefix-free. So, pfCM-MD means prefix-free-Counter-Masking-MD. In this paper, we consider a special case that for any $c = c_0||\cdots||c_t$, $c_i = i$ for $0 \leq i \leq t-1$ and $c_t = P$, where P a fixed value bigger than other counter c_j's. For example, when the maximum bit-size of an input message is $2^{64} - 1$, any value $P \geq 2^{64}$ is ok. When the maximum bit-size of an input message is $2^{128} - 1$, any value $P \geq 2^{128}$ is ok.

pfCM-chopMD. The chop Merkle-Damgård extension (chopMD) works as follow: for a message M such that $g(M) = m_1||\cdots||m_t$, $\mathrm{chopMD}^f = \mathrm{chop}_s(f(\cdots f(f(IV, m_1), m_2) \cdots, m_t))$. CM-chopMD (chopMD with a counter-masking) works similar to chopMD as follow : for given a message M such that $g(M) = m_1||\cdots||m_t$, $\mathrm{CM\text{-}chopMD}^f_g(M) = \mathrm{chop}_s(f(\cdots f(f(IV \oplus c_0, m_1) \oplus c_1, m_2) \oplus c_3, \cdots, m_t))$. When for any two $c = c_0||\cdots||c_t$ and $c' = c'_0||\cdots||c'_{t'}$ c is not a prefix of c', we say its counter-masking is prefix-free. So, pfCM-chopMD means prefix-free-Counter-Masking-chopMD. In this paper, we consider a special case that for any $c = c_0||\cdots||c_t$, $c_i = i$ for $0 \leq i \leq t-1$ and $c_t = P$, where P a fixed value bigger than other counter c_j's. Note that pfCM-chopMD with $s = 0$ is the same as pfCM-MD. That is, pfCM-MD is a special case of pfCM-chopMD. So, in this paper, we focus on providing an indifferentiable security proof of pfCM-chopMD with any s.

Inequality. The following inequality will be used to prove Theorem 2-10.

Ineq 1. For any $0 \leq a_i \leq 1$, $\prod_{i=1}^{q}(1 - a_i) \geq 1 - \sum_{i=1}^{q} a_i$. One can prove it by induction on q.

Random Oracle Model : f is said to be a *random oracle* from X to Y if for each $x \in X$ the value of $f(x)$ is chosen randomly from Y [4]. More precisely, $\Pr[f(x) = y \mid f(x_1) = y_1, f(x_2) = y_2, \ldots f(x_q) = y_q] = \frac{1}{T}$, where $x \notin \{x_1, \ldots, x_q\}$, $y, y_1, \cdots, y_q \in Y$ and $|Y| = T$. In the case that $X = \{0,1\}^d$

for a fixed value d, we say f is a FIL (Fixed Input Length) random oracle. In the case that $X = \{0,1\}^*$, we say f is a VIL (Variable Input Length) random oracle. A VIL random oracle is usually denoted by R.

The cost of Queries. The security bound of a scheme is usually described using the number q of queries and the maximum length l of each queries. On the other hand, in [3], the notion *cost* is used to describe the security bound of sponge construction. The notion *cost* denotes the total block length of q queries. The notion *cost* is significant because the unit of time complexity corresponds to the time of an underlying function call and the total time complexity depends on how many the underlying function is called. The notion *cost* exactly reflects how many the underlying function is called. So, we can consider two cases. The first case is that the number of queries is bounded q. The second case is that the cost of queries is bounded by q. Without loss of generality, for describing notions and some results in this section, we assume that the number of queries is bounded q.

View. A is a probabilistic algorithm with access to a tuple of oracles $O = (O_1, O_2, \cdots, O_t)$. r is a random coin string of A. A can make a query adaptively as follows. Let x_i be a i-th query and y_i be a response of a oracle for the i-th query x_i.

$$A(v_{i-1}) = x_i,$$

where $v_{i-1} = ((x_1, y_1), \cdots, (x_{i-1}, y_{i-1})$ and $A(\lambda) = x_0$. When the number of queries is q, v_q is said to be a *possible final view* of A, which is a tuple of query-response pairs. We may use the symbol v instead of v_q. The final view v is determined by the random coins of A and those of the tuple of oracles. The role of a random coin of A helps A to randomly choose one among possible choices during A's execution. Without loss of generality, we can assume that the number of coin tosses of A is fixed, because we only consider polynomial time algorithms. More precisely, since A is a polynomial time algorithm, the number of coin tosses is bounded by some t_q, where q is the number of queries of A. In the case that the number of coin tosses in less than t_q, we can construct A' by adding dummy coin tosses to A. Such A' identically behaves as A. Therefore,

$$\Pr[A(X) = Y] = \frac{|\{r | A'_r(X) = Y\}|}{2^{t_q}},$$

where X is an input of A, X can be v_{i-1}, Y is an output of A for X, Y is y_i or 0 or 1. From now, we assume that the number of coin tosses is fixed as t_q.

We define $\alpha_A(v) = \Pr[A(v_{i-1}) = x_i, \forall i, 1 \leq i \leq q]$, where $A(\mathsf{null}) = x_1$, and $v = ((x_1, y_1), \cdots, (x_q, y_q))$. We can also say that a view v is possible if $\alpha_A(v) \neq 0$. The set of possible final views of A is denoted by V_A. And for each random coin r of A, we similarly define

$$\alpha_{A_r}(v) = \Pr[A_r(v_{i-1}) = x_i, \forall i, 1 \leq i \leq q].$$

So, for any $v \in V_A$, $\alpha_{A_r}(v)$=0 or 1, because A_r is a deterministic algorithm for each r. Finally, A outputs 0 or 1 from the final view v, which consists of q query-response pairs, '0' denotes any output which is not '1'. More precisely, given a possible view v, The output value of $A(v)$ depends on the random coin tosses of A. We write $V_A^1 = \{v|A(v) = 1\}$ and $V_A^0 = \{v|A(v) = 0\}$. Since A_r with a fixed r is a deterministic algorithm, we can also define the set of possible views of A_r, which is denoted by V_{A_r}. So, for any view $v \in V_{A_r}$, $A_r(v)$ outputs 1 or 0 with probability 1, that is, a function $f_{A_r} : V_{A_r} \rightarrow \{0, 1\}$ is defined. And $V_{A_r}^1 = \{v|A_r(v) = 1\}$ and $V_{A_r}^0 = \{v|A_r(v) = 0\}$ are defined in the same way.

Computational Distance. Let $F = (F_1, F_2, \cdots, F_t)$ and $G = G(G_1, G_2, \cdots, G_t)$ be tuples of probabilistic oracle algorithms. We define the computational distance of a probabilistic attacker A at distinguishing F from G as

$$\mathbf{Adv}_A(F, G) = |\Pr[A^F = 1] - \Pr[A^G = 1]|.$$

Statistical Distance. Let $F = (F_1, F_2, \cdots, F_t)$ and $G = G(G_1, G_2, \cdots, G_t)$ be tuples of probabilistic oracle algorithms. We define the statistical distance of a deterministic attacker A at distinguishing F from G as

$$\mathbf{Stat}_A(F, G) = \frac{1}{2} \sum_{v \in V_A} |\Pr[F = v] - \Pr[G = v]|.$$

And we let the maximum statistical distance of F and G against any deterministic algorithm A be $\mathbf{Stat}(F, G)$, where the number of queries of A is bounded by q.

Computational Distance vs. Statistical Distance

Lemma 1. *Let $F = (F_1, F_2, \cdots, F_t)$ and $G = G(G_1, G_2, \cdots, G_t)$ be tuples of probabilistic oracle algorithms. For any probabilistic algorithm A who can make at most q queries*

$$\mathbf{Adv}_A(F, G) \leq \mathbf{Stat}(F, G).$$

Proof. We provide our proof on above lemma. Without loss of generality, we assume that A makes q queries. Since for A_r with any fixed r $\Pr[A_r^O = 1]+\Pr[A_r^O = 0] = \sum_{v \in V_{A_r}^1} Pr[F = v] + \sum_{v \in V_{A_r}^0} Pr[F = v] = 1$, where $\Pr[O = v]$ denotes $\Pr[O(x_i) = y_i, 1 \leq i \leq q, v = ((x_1, y_1), \cdots, (x_q, y_q))]$, the following inequality 2 holds.

Ineq 2. $|\sum_{v \in V_{A_r}^1} (\Pr[F = v] - \Pr[G = v])| \leq \frac{1}{2} \sum_{v \in V_{A_r}} |\Pr[F = v] - \Pr[G = v]|$.

And, $\Pr[A^O = 1]$
$= \sum_{v \in V_A^1} \alpha_A(v) \Pr_A[O(x_i) = y_i, \forall i, v = ((x_1, y_1), \cdots, (x_q, y_q))]$
$= \sum_{r \in \{0,1\}^{t_q}} \frac{1}{2^{t_q}} \sum_{v \in V_A^1} \alpha_{A_r}(v) \Pr[O(x_i) = y_i, 1 \leq i \leq q, v = ((x_1, y_1), \cdots, (x_q, y_q))]$

$= \sum_{r\in\{0,1\}^{t_q}} \frac{1}{2^{t_q}} \sum_{v\in V^1_{A_r}} \Pr[O(x_i) = y_i, 1 \le i \le q, v = ((x_1, y_1), \cdots, (x_q, y_q))].$

Therefore,

$\mathbf{Adv}_A(F, G) = |\Pr[A^F = 1] - \Pr[A^G = 1]|$
$= |(\sum_{v\in V^1_A} \alpha_A(v)\Pr[F = v]) - (\sum_{v\in V^1_A} \alpha_A(v)\Pr[G = v])|$
$= |(\sum_{r\in\{0,1\}^{t_q}} \frac{1}{2^{t_q}} \sum_{v\in V^1_{A_r}} \Pr[F = v]) - (\sum_{r\in\{0,1\}^{t_q}} \frac{1}{2^{t_q}} \sum_{v\in V^1_{A_r}} \Pr[G = v])|$
$= \sum_{r\in\{0,1\}^{t_q}} \frac{1}{2^{t_q}} \sum_{v\in V^1_{A_r}} |\Pr[F = v] - \Pr[G = v]|$
$\le \sum_{r\in\{0,1\}^{t_q}} \frac{1}{2^{t_q}} \cdot \frac{1}{2} \sum_{v\in V_{A_r}} (|pr[F = v] - \Pr[G = v]|$ by Ineq 1
$\le \frac{1}{2}\mathsf{Max}_{r\in\{0,1\}^{t_q}}(\sum_{v\in V_{A_r}} |\Pr[F = v] - \Pr[G = v]|)$
$= \frac{1}{2} \sum_{v\in V_{A_{r'}}} |\Pr[F = v] - \Pr[G = v]|$ (maximized by $r = r'$)
$= \mathbf{Stat}_{A_{r'}}(F, G)$ ($A_{r'}$ is a deterministic algorithm.)
$\le \mathsf{Max}_B(\mathbf{Stat}_B(F, G))$. (for any deterministic algorithm B.)
$= \mathbf{Stat}(F, G)$. ■

Lemma 2. *Let $F = (F_1, F_2, \cdots, F_t)$ and $G = G(G_1, G_2, \cdots, G_t)$ be tuples of probabilistic oracle algorithms. Let Bad be an event. For any deterministic algorithm A who can make at most q queries*

$$\mathbf{Stat}_A(F, G) \le \frac{1}{2} \sum_{v\in V_A \wedge Bad} |\Pr[F = v] - \Pr[G = v]| + \frac{1}{2} \sum_{v\in V_A \wedge \overline{Bad}} \Pr[F = v] + \frac{1}{2} \sum_{v\in V_A \wedge \overline{Bad}} \Pr[G = v]$$

Proof. Since this lemma is clear, we skip the proof. ■

Indifferentiability

We give a brief introduction of the indifferentiable security notion.

Definition 1. *Indifferentiability. [17] A Turing machine H with oracle access to an ideal primitive f is said to be $(t_D, t_S, q, \varepsilon)$ indifferentiable from an ideal primitive R if there exists a simulator S such that for any distinguisher D it holds that :*

$$|\Pr[D^{H,f} = 1] - \Pr[D^{R,S} = 1] < \varepsilon$$

The simulator has oracle access to R and runs in time at most t_S. The distinguisher runs in time at most t_D and makes at most q queries. Similarly, H^f is said to be (computationally) indifferentiable from R if ε is a negligible function of the security parameter k (for polynomially bounded t_D and t_S).

The following Theorem [17] shows the relation between indifferentiable security notion and the security of a cryptosystem.

Theorem 1. *[17] Let $\mathcal{P}$ be a cryptosystem with oracle access to an ideal primitive R. Let H be an algorithm such that H^f is indifferentiable from R. Then cryptosystem $\mathcal{P}$ is at least as secure in the f model with algorithm H as in the R model.*

Above theorem says that if a domain extension (with a padding rule) based on a FIL random oracle f is indifferentiable from a VIL random oracle R, then a cryptosystem, which is proved in the VIL random oracle model, with negligible loss of security can use the domain extension (with a padding rule) based on a FIL random oracle f instead of R.

3 Construction of the Simulator

In this section, we define simulators as follows. In the next section, simulators S_{choppf}, S_{chop}, $S_{chopMDP}$, $S_{chopWPH}$, $S_{chopEMD}$, S_{chopNI}, S_{chopCS}, $S_{chopESh}$ and S_{pfCM} will be used in order to prove the indifferentiable security of choppfMD, chopMD, chopMDP, chopWPH, chopEMD, chopNI, chopCS, chopESh and pfCM-chopMD, respectively. For defining the simulators, We follow the style of construction of the simulator in [9].

Definition of Simulator S_{choppf}

INITIALIZATION :

1. A partial function $e_1 : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ initialized as empty,
2. a partial function $e_1^* = \mathrm{MD}^{e_1} : (\{0,1\}^b)^* \rightarrow \{0,1\}^n$ initialized as $e_1^*(\lambda) = \mathrm{IV}$.
3. a set $C = \{\mathrm{IV}\}$ and a set $I = \{\lambda\}$.

On query $S^R_{choppf}(x, m)$:

001 `if` $(e_1(x, m) = x')$
    `return` x';

002 `else if` $(\exists\ M'\ and\ M, e_1^*(M') = x, g(M) = M'||m))$
    $y = R(M)$;
    `choose` $w \in_R \{0,1\}^s$;
    `define` $e_1(x, m) = z := y \parallel w$;
    `return` z;

003 `else if` $(\exists\ M', e_1^*(M') = x)$
    `choose` $z \in_R \{0,1\}^n \setminus C \cup I$;
    `define` $e_1(x, m) = z$;
    `define` $C = C \cup \{z\}$;
    `define` $e_1^*(M', m) = z$;
    `return` z;

004 `else`
    $z \in_R \{0,1\}^n$;
    `define` $e_1(x, m) = z$;
    `define` $I = I \cup \{x\}$;
    `return` z;

Definition of Simulator S_{chop}

Initialization :

1. A partial function $e_1 : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ initialized as empty,
2. a partial function $e_1^* = \mathrm{MD}^{e_1} : (\{0,1\}^b)^* \rightarrow \{0,1\}^n$ initialized as $e_1^*(\lambda) = \mathrm{IV}$.
3. a set $C = \{\mathrm{IV}\}$ and a set $I = \{\lambda\}$.

On query $S_{chop}^R(x, m)$:

001 `if` $(e_1(x, m) = x')$
    `return` x';

002 `else if` $(\exists\ M'\ and\ M, e_1^*(M') = x, g(M) = M'||m))$
    $y = R(M)$;
    `choose` $w \in_R \{0,1\}^s \setminus \{w' : y \parallel w' \in C \cup I\}$;
    `define` $e_1(x, m) = z := y \parallel w$;
    `define` $C = C \cup \{z\}$;
    `define` $e_1^*(M', m) = z$;
    `return` z;

003 `else if` $(\exists\ M', e_1^*(M') = x)$
    `choose` $z \in_R \{0,1\}^n \setminus C \cup I$;
    `define` $e_1(x, m) = z$;
    `define` $C = C \cup \{z\}$;
    `define` $e_1^*(M', m) = z$;
    `return` z;

004 `else`
    $z \in_R \{0,1\}^n$;
    `define` $e_1(x, m) = z$;
    `define` $I = I \cup \{x\}$;
    `return` z;

Definition of Simulator $S_{chopMDP}$

Initialization :

1. A partial function $e_1 : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ initialized as empty,
2. a partial function $e_1^* = \mathrm{MD}^{e_1} : (\{0,1\}^b)^* \rightarrow \{0,1\}^n$ initialized as $e_1^*(\lambda) = \mathrm{IV}$.
3. a set $C = \{\mathrm{IV}\}$ and a set $I = \{\lambda\}$.

On query $S_{chopMDP}^R(x, m)$:

001 `if` $(e_1(x, m) = x')$
    `return` x';

002 else if $(\exists\ M'\ and\ M, e_1^*(M') = x \oplus P, g(M) = M'||m))$
  $y = R(M)$;
  choose $w \in_R \{0,1\}^s$;
  define $e_1(x,m) = z := y \parallel w$;
  return z;

003 else if $(\exists\ M', e_1^*(M') = x)$
  choose $z \in_R \{0,1\}^n \setminus C \cup \{a \oplus P : a \in C\} \cup I \cup \{a \oplus P : a \in I\}$;
  define $e_1(x,m) = z$;
  define $C = C \cup \{z\}$;
  define $e_1^*(M', m) = z$;
  return z;

004 else
  $z \in_R \{0,1\}^n$;
  define $e_1(x,m) = z$;
  define $I = I \cup \{x\}$;
  return z;

Definition of Simulator $S_{chopWPH}$

INITIALIZATION :

1. A partial function $e_1 : \{0,1\}^{w+b} \to \{0,1\}^w$ initialized as empty,
2. A partial function $e_2 : \{0,1\}^w \to \{0,1\}^n$ initialized as empty,
3. a partial function $e_1^* = \mathrm{MD}^{e_1} : (\{0,1\}^b)^* \to \{0,1\}^w$ initialized as $e_1^*(\lambda) = \mathrm{IV}$.
4. a set $C = \{\mathrm{IV}\}$ and a set $I = \{\lambda\}$.

On query $S^R_{chopWPH}(x,m)$:

001 if $(e_1(x,m) = x')$
  return x';

002 else if $(\exists\ M', e_1^*(M') = x)$
  choose $z \in_R \{0,1\}^n \setminus C \cup I$;
  define $e_1(x,m) = z$;
  define $C = C \cup \{z\}$;
  define $e_1^*(M', m) = z$;
  return z;

003 else
  $z \in_R \{0,1\}^n$;
  define $e_1(x,m) = z$;
  define $I = I \cup \{x\}$;
  return z;

On query $S^R_{chopWPH}(x)$:

004 `if` $(e_2(x) = x')$
`return` x';

005 `else if` $(\exists\ M'\ \textit{and}\ M, e_1^*(M') = x, g(M) = M'))$
$y = R(M)$;
`choose` $w \in_R \{0,1\}^s$;
`define` $e_2(x) = z := y \parallel w$;
`return` z;

006 `else`
$z \in_R \{0,1\}^n$;
`define` $e_2(x) = z$;
`define` $I = I \cup \{x\}$;
`return` z;

Definition of Simulator $S_{chopEMD}$

INITIALIZATION :

1. A partial function $e_1 : \{0,1\}^{n+b} \to \{0,1\}^n$ initialized as empty,
2. a partial function $e_1^* = \mathrm{MD}^{e_1} : (\{0,1\}^b)^* \to \{0,1\}^n$ initialized as $e_1^*(\lambda) = \mathrm{IV}$.
3. a set $C = \{\mathrm{IV}, \mathrm{IV}_2\}$ and a set $I = \{\lambda\}$.

On query $S^R_{chopEMD}(x, m)$:

001 `if` $(e_1(x, m) = x')$
`return` x';

002 `else if` $(x = IV_2$ and $\exists\ M'\ \textit{and}\ M, e_1^*(M') = \mathrm{chop}_{b-n}(m), g(M) = M'||m))$
$y = R(M)$;
`choose` $w \in_R \{0,1\}^s$;
`define` $e_1(x, m) = z := y \parallel w$;
`return` z;

003 `else if` $(\exists\ M', e_1^*(M') = x)$
`choose` $z \in_R \{0,1\}^n \setminus C \cup I$;
`define` $e_1(x, m) = z$;
`define` $C = C \cup \{z\}$;
`define` $e_1^*(M', m) = z$;
`return` z;

004 `else`
$z \in_R \{0,1\}^n$;
`define` $e_1(x, m) = z$;
`define` $I = I \cup \{x\}$;
`return` z;

Definition of Simulator S_{chopNI}

Initialization :

1. given K_1 and K_2,
2. A partial function $e_1 : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$ initialized as empty,
3. a partial function $e_1^* = \mathrm{MD}^{e_1} : \{K_1\} \times (\{0,1\}^b)^* \rightarrow \{0,1\}^n$ initialized as $e_1^*(\lambda) = \mathrm{IV}$, where e_1 is only defined if its key is K_1.
4. a set $C = \{\mathrm{IV}\}$ and a set $I = \{\lambda\}$.

On query $S^R_{chopNI}(K, x, m)$:

001 `if` $(e_1(K, x, m) = x')$
 `return` x';

002 `else if` $(K = K_2$ and $\exists\ M'$ *and* $M, e_1^*(M') = x, g(M) = M'||m))$
 $y = R(M)$;
 `choose` $w \in_R \{0,1\}^s$;
 `define` $e_1(K, x, m) = z := y \parallel w$;
 `return` z;

003 `else if` $(\exists\ M', e_1^*(M') = x)$
 `choose` $z \in_R \{0,1\}^n \setminus C \cup I$;
 `define` $e_1(K, x, m) = z$;
 `define` $C = C \cup \{z\}$;
 `if` $K = K_1$, `define` $e_1^*(M', m) = z$;
 `return` z;

004 `else`
 $z \in_R \{0,1\}^n$;
 `define` $e_1(K, x, m) = z$;
 `define` $I = I \cup \{x\}$;
 `return` z;

Definition of Simulator S_{chopCS}

Initialization :

1. given K,
2. A partial function $e_1 : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$ initialized as empty,
3. a partial function $e_1^* = \mathrm{MD}^{e_1} : \{K\} \times (\{0,1\}^b)^* \rightarrow \{0,1\}^n$ initialized as $e_1^*(\lambda) = \mathrm{IV}$, where e_1 is only defined if its key is K.
4. a set $C = \{\mathrm{IV}, \mathrm{IV}_2\}$ and a set $I = \{\lambda\}$.

On query $S^R_{chopCS}(K', x, m)$:

001 `if` $(e_1(K', x, m) = x')$
  `return` x';

002 `else if` ($K' = K$ and $x = IV_2$ and $\exists\, M'$ *and* $M, e_1^*(M') = \text{chop}_{n-s}(m), g(M) = M' || m)$)
  $y = R(M)$;
  `choose` $w \in_R \{0,1\}^s$;
  `define` $e_1(K', x, m) = z := y \,\|\, w$;
  `return` z;

003 `else if` $(\exists\, M', e_1^*(M') = x)$
  `choose` $z \in_R \{0,1\}^n \setminus C \cup I$;
  `define` $e_1(K', x, m) = z$;
  `define` $C = C \cup \{z\}$;
  `if` $K' = K$, `define` $e_1^*(M', m) = z$;
  `return` z;

004 `else`
  $z \in_R \{0,1\}^n$;
  `define` $e_1(K', x, m) = z$;
  `define` $I = I \cup \{x\}$;
  `return` z;

Definition of Simulator $S_{chopESh}$

Initialization :

1. given K and $\overline{K}$,
2. A partial function $e_1 : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$ initialized as empty,
3. a partial function $e_1^* = \text{Shoup}^{e_1}_{K,\overline{K}} : (\{0,1\}^b)^* \rightarrow \{0,1\}^n$ initialized as $e_1^*(\lambda) = \text{IV} \oplus \text{K}_0$, where e_1 is only defined if its key is K.
4. a set $I = \{\text{IV} \oplus \text{K}_0, \text{IV}_2 \oplus \text{K}_0\}$ and a set $C = \{\lambda\}$.

On query $S^R_{chopESh}(K', x, m)$:

001 `if` $(e_1(K', x, m) = x')$
  `return` x';

002 `else if` ($K' = K$ and $x = IV_2 \oplus K_0$ and $\exists\, M'$ and M, $e_1^*(M') = K_{\mu(i)} \oplus \text{chop}_{n-s}(m)$, $g(M) = M'||m$, $||M'||_b = i - 1$)
  $y = R(M)$;
  `choose` $w \in_R \{0,1\}^s$;
  `define` $e_1(K', x, m) = z := y \,\|\, w$;
  `return` z;

003 `else if` $(\exists\ M',\ e_1^*(M') = K_{\mu(i)} \oplus x,\ ||M'||_b = i-1)$
  `choose` $z \in_R \{0,1\}^n \setminus \{c \oplus K_{\mu(i+1)} : c \in I\} \cup \{a : (i_a, a) \in C\}$
  $\cup \{a \oplus K_{\mu(i_a)} \oplus K_{\mu(i+1)} : (i_a, a) \in C\}$;
  `define` $e_1(K', x, m) = z$;
  `define` $C = C \cup \{(i+1, z)\}$;
  `if` $K' = K$, `define` $e_1^*(M', m) = z$;
  `return` z;

004 `else`
  $z \in_R \{0,1\}^n$;
  `define` $e_1(K', x, m) = z$;
  `define` $I = I \cup \{x\}$;
  `return` z;

Definition of Simulator S_{pfCM}

INITIALIZATION :

1. A partial function $e_1 : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ initialized as empty,
2. a partial function $e_1^* = \text{CM-MD}^{e_1} : (\{0,1\}^b)^* \rightarrow \{0,1\}^n$ initialized as $e_1^*(\lambda) =$ IV.
3. a set $I = \{\text{IV}\}$ and a set $U = \{\lambda\}$.

On query $S_{pfCM}^R(x, m)$:

001 `if` $(e_1(x, m) = x')$
  `return` x';

002 `else if` $(\exists\ M'\ and\ M, e_1^*(M') = x \oplus P, ||M'||_b = i, g(M) = M'||m))$
  $y = R(M)$;
  `choose` $w \in_R \{0,1\}^s$;
  `define` $e_1(x, m) = z := y \parallel w$;
  `return` z;

003 `else if` $(\exists\ M', e_1^*(M') = x \oplus i, ||M'||_b = i)$
  `choose` $z \in_R \{0,1\}^n \setminus \{c \oplus i : c \in I\} \cup \{c \oplus P : c \in I\} \cup \{a : (i_a, a) \in U\}$
  $\cup \{a \oplus P \oplus i : (i_a, a) \in U\} \cup \{a \oplus i_a \oplus i : (i_a, a) \in U\}$
  $\cup \{a \oplus i_a \oplus P : (i_a, a) \in U\}$;
  `define` $e_1(x, m) = z$;
  `define` $U = U \cup \{(i, z)\}$;
  `define` $e_1^*(M', m) = z$;
  `return` z;

004 `else`
  $z \in_R \{0,1\}^n$;
  `define` $e_1(x, m) = z$;
  `define` $I = I \cup \{x\}$;
  `return` z;

Some Important Observations on the Simulator S_{choppf}

The bound of the number of queries. In line 003, the number q of queries of S should be bounded by $q < 2^n$ in order to choose z. If $q \geq 2^n$, the simulator may not work. So, we assume that $q < 2^n$.

The bound of the number of possible input message. Firstly, in 002 and 003, there exists at most one M' such that $e_1^*(M') = x$ by the process of selecting z which is not in the set C in line 003. This first observation corresponds to Lemma 1 in [3]. Secondly, in line 003, by the process of selecting z which is not in the set I, the following holds : if $e_1(x, m)$ is already defined under the assumption that $e_1^*(M') \neq x$ for all previously defined M', no M can be newly defined such that $e_1^*(M) = x$. This second observation corresponds to the second part of proof of Lemma 2 in [3].

Some Important Observations on the Simulator S_{chop}

The bound of the number of queries. In line 002, the number q of queries of S should be bounded by $q < 2^s$ in order to choose z. If $q \geq 2^s$, the simulator may not work. So, we assume that $q < 2^s$.

The bound of the number of possible input message. Firstly, in 002 and 003, there exists at most one M' such that $e_1^*(M') = x$ by the process of selecting z which is not in the set C. This first observation corresponds to Lemma 1 in [3]. Secondly, in line 003, by the process of selecting z which is not in the set I, the following holds : if $e_1(x, m)$ is already defined under the assumption that $e_1^*(M') \neq x$ for all previously defined M', no M can be newly defined such that $e_1^*(M) = x$. This second observation corresponds to the second part of proof of Lemma 2 in [3].

Some Important Observations on the Simulator $S_{chopMDP}$

The bound of the number of queries. In line 003, the number q of queries of S should be bounded by $q < 2^n/2$ in order to choose z. If $q \geq 2^n/2$, the simulator may not work. So, we assume that $q < 2^n/2$.

The bound of the number of possible input message. Firstly, in 002 and 003, there exists at most one M' such that $e_1^*(M') = x$ by the process of selecting z which is not in the set C. This first observation corresponds to Lemma 1 in [3]. Secondly, in line 003, by the process of selecting z which is not in the set I, the following holds : if $e_1(x, m)$ is already defined under the assumption that $e_1^*(M') \neq x$ and $e_1^*(M') \neq x \oplus P$ for all previously defined M', no M can be newly defined such that $e_1^*(M) = x$ or $e_1^*(M) = x \oplus P$. This second observation corresponds to the second part of proof of Lemma 2 in [3].

Some Important Observations on the Simulator $S_{chopWPH}$

The bound of the number of queries. In line 002, the number q of queries of S should be bounded by $q < 2^n$ in order to choose z. If $q \geq 2^n$, the simulator may not work. So, we assume that $q < 2^n$.

The bound of the number of possible input message. Firstly, in 002 and 005, there exists at most one M' such that $e_1^*(M') = x$ by the process of selecting z which is not in the set C. This first observation corresponds to Lemma 1 in [3]. Secondly, in line 002, by the process of selecting z which is not in the set I, the following holds : if $e_1(x, m)$ is already defined under the assumption that $e_1^*(M') \neq x$ for all previously defined M', no M can be newly defined such that $e_1^*(M) = x$. This second observation corresponds to the second part of proof of Lemma 2 in [3].

Some Important Observations on the Simulator $S_{chopEMD}$

The bound of the number of queries. In line 003, the number q of queries of S should be bounded by $q < 2^n - 1$ in order to choose z. If $q \geq 2^n - 1$, the simulator may not work. So, we assume that $q < 2^n - 1$.

The bound of the number of possible input message. Firstly, in 002 and 003, there exists at most one M' such that $e_1^*(M') = x$ by the process of selecting z which is not in the set C in line 003. This first observation corresponds to Lemma 1 in [3]. Secondly, in line 003, by the process of selecting z which is not in the set I, the following holds : if $e_1(x, m)$ is already defined under the assumption that $e_1^*(M') \neq x$ for all previously defined M', no M can be newly defined such that $e_1^*(M) = x$. This second observation corresponds to the second part of proof of Lemma 2 in [3].

Some Important Observations on the Simulator S_{chopNI}

The bound of the number of queries. In line 003, the number q of queries of S should be bounded by $q < 2^n$ in order to choose z. If $q \geq 2^n$, the simulator may not work. So, we assume that $q < 2^n$.

The bound of the number of possible input message. Firstly, in 002 and 003, there exists at most one M' such that $e_1^*(M') = x$ by the process of selecting z which is not in the set C in line 003. This first observation corresponds to Lemma 1 in [3]. Secondly, in line 003, by the process of selecting z which is not in the set I, the following holds : if $e_1(K_1, x, m)$ is already defined under the assumption that $e_1^*(M') \neq x$ for all previously defined M', no M can be newly defined such that $e_1^*(M) = x$. This second observation corresponds to the second part of proof of Lemma 2 in [3].

Some Important Observations on the Simulator S_{chopCS}

THE BOUND OF THE NUMBER OF QUERIES. In line 003, the number q of queries of S should be bounded by $q < 2^n - 1$ in order to choose z. If $q \geq 2^n - 1$, the simulator may not work. So, we assume that $q < 2^n - 1$.

THE BOUND OF THE NUMBER OF POSSIBLE INPUT MESSAGE. Firstly, in 002 and 003, there exists at most one M' such that $e_1^*(M') = x$ by the process of selecting z which is not in the set C in line 003. This first observation corresponds to Lemma 1 in [3]. Secondly, in line 003, by the process of selecting z which is not in the set I, the following holds : if $e_1(K, x, m)$ is already defined under the assumption that $e_1^*(M') \neq x$ for all previously defined M', no M can be newly defined such that $e_1^*(M) = x$. This second observation corresponds to the second part of proof of Lemma 2 in [3].

Some Important Observations on the Simulator $S_{chopESh}$

THE BOUND OF THE NUMBER OF QUERIES. In line 003, the number q of queries of S should be bounded by $q < (2^n - 1)/3$ in order to choose z. If $q \geq (2^n - 1)/3$, the simulator may not work. So, we assume that $q < (2^n - 1)/3$.

THE BOUND OF THE NUMBER OF POSSIBLE INPUT MESSAGE. Firstly, in 002 and 003, there exists at most one M' such that $e_1^*(M') = x \oplus K_{\mu(i)}$ by the process of selecting z unrelated to the set C in line 003. This first observation corresponds to Lemma 1 in [3]. Secondly, in line 002 and 003, by the process of selecting z which is not in the set I in line 003, the following holds : if $e_1(K, x, m)$ is already defined under the assumption that $e_1^*(M') \neq x \oplus K_{\mu(i)}$ for for all previously defined M', where $||M'||_b = i - 1$, then no M can be newly defined such that $e_1^*(M) = x \oplus K_{\mu(j)}$, where $||M||_b = j - 1$. This second observation corresponds to the second part of proof of Lemma 2 in [3].

Some Important Observations on the Simulator S_{pfCM}

THE BOUND OF THE NUMBER OF QUERIES. In line 003, the number q of queries of S should be bounded by $q < 2^n/6$ in order to choose z. If $q \geq 2^n/6$, the simulator may not work. So, we assume that $q < 2^n/6$.

THE BOUND OF THE NUMBER OF POSSIBLE INPUT MESSAGE. Firstly, in 002 and 003, there exists at most one M' such that $e_1^*(M') = x \oplus i$ or $e_1^*(M') = x \oplus P$ by the process of selecting z unrelated to the set U in line 003. This first observation corresponds to Lemma 1 in [3]. Secondly, in line 002 and 003, by the process of selecting z which is not in the set I in line 003, the following holds : if $e_1(x, m)$ is already defined under the assumption that $e_1^*(M') \neq x \oplus i$ and $e_1^*(M') \neq x \oplus P$ for for all previously defined M', where $||M'||_b = i$, then no M can be newly defined such that $e_1^*(M) = x \oplus j$ or $e_1^*(M) = x \oplus P$, where where $||M||_b = j$. This second observation corresponds to the second part of proof of Lemma 2 in [3].

4 Indifferentiable Security Analysis of choppfMD, chopMD, chopMDP, chopWPH, chopEMD, chopNI, chopCS, chopESh and pfCM-chopMD Hash Domain Extensions

We will describe the indifferentiable security bound of each domain extension using the notion *cost* of queries. We let the cost be q. For example, with the cost q of queries, A can have access to O_2 q times and no access to O_1. By observations of simulators described in previous section, the following Lemma holds, where for choppfMD $T = 2^n$, for chopMD $T = 2^s$, for chopMDP $T = 2^n/2$, for chopWPH $T = 2^n$, for chopEMD $T = 2^n - 1$, for chopNI $T = 2^n$, for chopCS $T = 2^n - 1$, for chopESh $T = (2^n - 1)/3$, and for pfCM-chopMD $T = 2^n/6$.

Lemma 3. *Let $q < T$. When the total cost of queries to O_1 is t less than q or equal, the queries to O_1 can be converted to t queries to O_2, where O_2 gives at least the same amount of information to an attacker A and has no higher cost than O_1.*

Proof. The proof is the same as that of Lemma 3 in [3]. ■

Above Lemma says that to give all queries to O_2 and no query to O_1 is the best strategy to obtain better computational distance. That is, when the cost of queries is bound by q, for any A there is an attacker B such that the following holds :

$$\mathbf{Adv}_A((H^f, f), (R, S)) \leq \mathbf{Adv}_B(f, S),$$

where H^f=choppfMD_g^f or chopMD_g^f or chopMDP_g^f or $\text{chopWPH}_g^{f_1,f_2}$ or H^f=chopEMD_g^f or chopNI_g^f or chopCS_g^f or chopESh_g^f or $\text{pfCM} - \text{chopMD}_g^f$, and S=$S_{choppfMD}$ or S_{chopMD} or $S_{chopMDP}$ or $S_{chopWPH}$ or S=$S_{chopEMD}$ or S_{chopNI} or S_{chopCS} or $S_{chopESh}$ or S_{pfCM}, respectively. Therefore, we focus on computing the upper bound of the computational distance between f and S as shown in the following theorems.

Theorem 2. *Let $q < 2^n$ be the number of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is a FIL random oracle. $S_{choppfMD}$ is the simulator defined in the previous section. Then for any (deterministic or probabilistic) algorithm A*

$$\mathbf{Adv}_A(f, S_{choppfMD}) \leq \frac{q(q+1)}{2^{n+1}}.$$

Proof. See the Appendix.

Theorem 3. *Let $q < \mathsf{min}(2^{n-s-1}, 2^s)$ be the number of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is a FIL random oracle. S_{chopMD} is the simulator defined in the previous section. Then for any (deterministic or probabilistic) algorithm A*

$$\mathbf{Adv}_A(f, S_{chopMD}) \leq \frac{(n-s)q}{2^s} + \frac{q}{2^{n-s}}.$$

Proof. See the Appendix.

Theorem 4. *Let $q < 2^n/2$ be the number of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is a FIL random oracle. $S_{chopMDP}$ is the simulator defined in the previous section. Then for any (deterministic or probabilistic) algorithm A*

$$\mathbf{Adv}_A(f, S_{chopMDP}) \leq \tfrac{q^2}{2^n}.$$

Proof. See the Appendix.

Theorem 5. *Let $q < 2^n$ be the number of queries and $0 \leq s < n$. $f_1 : \{0,1\}^{w+b} \rightarrow \{0,1\}^w$ and $f_2 : \{0,1\}^w \rightarrow \{0,1\}^n$ are independent FIL random oracles. $S_{chopWPH}$ is the simulator defined in the previous section. Then for any (deterministic or probabilistic) algorithm A*

$$\mathbf{Adv}_A((f_1, f_2), S_{chopWPH}) \leq \tfrac{q(q+1)}{2^{w+1}}.$$

Proof. See the Appendix.

Theorem 6. *Let $q < 2^n - 1$ be the number of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is a FIL random oracle. $S_{chopEMD}$ is the simulator defined in the previous section. Then for any (deterministic or probabilistic) algorithm A*

$$\mathbf{Adv}_A(f, S_{chopEMD}) \leq \tfrac{q(q+3)}{2^{n+1}}.$$

Proof. See the Appendix.

Theorem 7. *Let $q < 2^n$ be the number of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is a FIL random oracle. S_{chopNI} is the simulator defined in the previous section. Then for any (deterministic or probabilistic) algorithm A*

$$\mathbf{Adv}_A(f, S_{chopNI}) \leq \tfrac{q(q+1)}{2^{n+1}}.$$

Proof. See the Appendix.

Theorem 8. *Let $q < 2^n - 1$ be the number of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is a FIL random oracle. S_{chopCS} is the simulator defined in the previous section. Then for any (deterministic or probabilistic) algorithm A*

$$\mathbf{Adv}_A(f, S_{chopCS}) \leq \tfrac{q(q+3)}{2^{n+1}}.$$

Proof. See the Appendix.

Theorem 9. *Let $q < (2^n - 1)/3$ be the number of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is a FIL random oracle. $S_{chopESh}$ is the simulator defined in the previous section. Then for any (deterministic or probabilistic) algorithm A*

$$\mathbf{Adv}_A(f, S_{chopESh}) \leq \frac{q(3q+1)}{2^n}.$$

Proof. See the Appendix.

Theorem 10. *Let $q < (2^n - 1)/6$ be the number of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \to \{0,1\}^n$ is a FIL random oracle. S_{pfCM} is the simulator defined in the previous section. Then for any (deterministic or probabilistic) algorithm A*

$$\mathbf{Adv}_A(f, S) \leq \frac{q(3q-2)}{2^n}.$$

Proof. See the Appendix.

From Lemma 2 and Theorem 2-10, we can get indiffrerentiable security bounds of choppfMD, chopMD, chopMDP, chopWPH, chopEMD, chopNI, chopCS, chopESh and pfCM-chopMD as the following corollaries, respectively.

Corollary 1. *Let $q < 2^n$ be the cost of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \to \{0,1\}^n$ is a FIL random oracle. $S_{choppfMD}$ is the simulator defined in the previous section. Then for any attacker A*

$$\mathbf{Adv}_A((\mathrm{choppfMD}_g^f, f), (R, S_{choppfMD})) \leq \frac{q(q+1)}{2^{n+1}}.$$

Corollary 2. *Let $q < \mathsf{min}(2^{n-s-1}, 2^s)$ be the cost of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \to \{0,1\}^n$ is a FIL random oracle. S_{chopMD} is the simulator defined in the previous section. Then for any attacker A*

$$\mathbf{Adv}_A((\mathrm{chopMD}_g^f, f), (R, S_{chopMD})) \leq \frac{(n-s)q}{2^s} + \frac{q}{2^{n-s}}.$$

Corollary 3. *Let $q < 2^n/2$ be the cost of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \to \{0,1\}^n$ is a FIL random oracle. $S_{chopMDP}$ is the simulator defined in the previous section. Then for any attacker A*

$$\mathbf{Adv}_A((\mathrm{chopMDP}_g^f, f), (R, S_{chopMDP})) \leq \frac{q^2}{2^n}.$$

Corollary 4. *Let $q < 2^n$ be the cost of queries and $0 \leq s < n$. $f_1 : \{0,1\}^{w+b} \to \{0,1\}^w$ and $f_2 : \{0,1\}^w \to \{0,1\}^n$ are independent FIL random oracles. $S_{chopWPH}$ is the simulator defined in the previous section. Then for any attacker A*

$$\mathbf{Adv}_A((\mathrm{chopWPH}_g^{f_1,f_2}, f), (R, S_{chopWPH})) \leq \frac{q(q+1)}{2^{w+1}}.$$

Corollary 5. *Let $q < 2^n - 1$ be the cost of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \to \{0,1\}^n$ is a FIL random oracle. $S_{chopEMD}$ is the simulator defined in the previous section. Then for any attacker A*

$$\mathbf{Adv}_A((\mathrm{chopEMD}_g^f, f), (R, S_{chopEMD})) \leq \frac{q(q+3)}{2^{n+1}}.$$

Corollary 6. *Let $q < 2^n$ be the cost of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \to \{0,1\}^n$ is a FIL random oracle. S_{chopNI} is the simulator defined in the previous section. Then for any attacker A*

$$\mathbf{Adv}_A((\text{chopNI}_g^f, f), (R, S_{chopNI})) \le \frac{q(q+1)}{2^n}.$$

Corollary 7. *Let $q < 2^n - 1$ be the cost of queries and $0 \le s < n$. $f : \{0,1\}^{n+b} \to \{0,1\}^n$ is a FIL random oracle. S_{chopCS} is the simulator defined in the previous section. Then for any attacker A*

$$\mathbf{Adv}_A((\text{chopCS}_g^f, f), (R, S_{chopCS})) \le \frac{q(q+3)}{2^{n+1}}.$$

Corollary 8. *Let $q < (2^n - 1)/3$ be the cost of queries and $0 \le s < n$. $f : \{0,1\}^{n+b} \to \{0,1\}^n$ is a FIL random oracle. $S_{chopESh}$ is the simulator defined in the previous section. Then for any attacker A*

$$\mathbf{Adv}_A((\text{chopESh}_g^f, f), (R, S_{chopESh})) \le \frac{q(3q+1)}{2^{n+1}}.$$

Corollary 9. *Let $q < (2^n - 1)/6$ be the cost of queries and $0 \le s < n$. $f : \{0,1\}^{n+b} \to \{0,1\}^n$ is a FIL random oracle. S_{pfCM} is the simulator defined in the previous section. Then for any attacker A*

$$\mathbf{Adv}_A((\text{pfCM} - \text{chopMD}_g^f, f), (R, S_{pfCM})) \le \frac{q(3q-2)}{2^n}.$$

5 Security of our pfCM-MD and pfCM-chopMD against Known Second-Preimage Attacks

In 2005, Kelsey and Schneier [14] introduced a second preimage-finding attack on Merkle-Damgård Strengthening [15]. More precisely, given a message of 2^k block, they showed that it is possible to find a second-preimage with complexity $k \cdot 2^{n/2+1} + 2^{n-k+1}$. They also suggested a countermeasure against their attack, *dithering with a counter*. In the second NIST hash workshop, Rivest [21] suggested a more efficient dithering method for guaranteeing the full security against the Kelsey-Schneier attack, *a dithering method with abelian square-free sequences*. Recently, Elena *et al.* [2] described a new second preimage attack on Rivest's dithering method with a technique of constructing diamond-structure used in the herding attack proposed by Kelsey and Khono [13].

Security against Known Second-Preimage Attacks. Our pfCM-MD and pfCM-chopMD hash domain extensions uses a counter-masking method, where a counter is xored with a part of the input value of the compression function. The counter-masking method makes it impossible to get expandable messages used in Kelsey-Schneier attack. And, in the case of the attack proposed by Elena *et al.*, since there is no repeated factor of any size in the counter-masking method, the full security is also guaranteed against their attack.

6 Conclusion

Till now, most of previous indifferentiable security analysis are difficult to follow and check the validness of security. In this paper, we have provided indifferentiable security anlalyses of nine constructions and their truncated versions with the technique introduced in [3]. Our proof is clear and very easy to follow and simple. We also give how to prove the indifferentiable security of any hash domain extension. By similar methods, other different domain extensions including their truncated versions can be proved. We expect that designers of new hash functions can easily prove the indifferentiable security of their constructions. Even though we only consider the security of single block length domain extensions in the random oracle model, it is also easy to prove the security of constructions based on a block cipher in the ideal cipher model and generalize our results into any block length construction. We remain it as a future work.

References

1. J. H. An and M. Bellare. Constructing VIL-MACs from FIL-MACs: Message authentication under weakened assumptions. In *Crypto'1999*, volume **1666** of *Lecture Notes in Computer Science*, pages 252–269, Springer-Verlag, 1999.
2. E. Andreeva, C. Bouillaguet, P. Fouque, J. J. Hoch, J. Kelsey, A. Shamir, and S. Zimmer. Second Preimage Attacks on Dithered Hash Functions. In *Eurocrypt'2008*, volume **4965** of *Lecture Notes in Computer Science*, pages 270–288, Springer-Verlag, 2008.
3. G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. On the Indifferentiability of the Sponge Construction. In *Eurocrypt'2008*, volume **4965** of *Lecture Notes in Computer Science*, pages 181–197, Springer-Verlag, 2008.
4. M. Bellare and P. Rogaway. Random Oracles Are Practical : A Paradigm for Designing Efficient Protocols. In *1st Conference on Computing and Communications Security*, ACM, pages 62–73, 1993.
5. M. Bellare and T. Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In *Asiacrypt'2006*, volume **4284** of *Lecture Notes in Computer Science*, pages 299–314, Springer-Verlag, 2006.
6. M. Bellare and T. Ristenpart. Hash Functions in the Dedicated-Key Setting : Design Choices and MPP Transforms. In *ICALP'2007*, volume **4596** of *Lecture Notes in Computer Science*, pages 399–410, Springer-Verlag, 2007.
7. E. Biham and O. Dunkelman. A Framework for Iterative Hash Functions - HAIFA. In *The second NIST Hash Workshop*, 2006.
8. D. Chang, S. Lee, M. Nandi and M. Yung. Indifferentiable Security Analysis of Popular Hash Functions with Prefix-Free Padding. In *Asiacrypt'2006*, volume **4284** of *Lecture Notes in Computer Science*, pages 283–298, Springer-Verlag, 2006.
9. D. Chang and M. Nandi. Improved Indifferentiability Security Proof of chopMD Hash Function. In *FSE'2008*, volume **5086** of *Lecture Notes in Computer Science*, pages 429–443, Springer-Verlag, 2008.
10. J. S. Coron, Y. Dodis, C. Malinaud and P. Puniya. Merkle-Damgard Revisited: How to Construct a Hash Function. In *Advances in Cryptology-Crypto'2005*, volume **3621** of *Lecture Notes in Computer Science*, pages 430–448, Springer-Verlag, 2005.
11. I. B. Damgard. A design principle for hash functions. In *Advances in Cryptology-Crypto'1989*, volume **435** of *Lecture Notes in Computer Science*, pages 416–427, Springer-Verlag, 1989.

12. S. Hirose, J. H. Park and A. Yun. A Simple Variant of the Merkle-Damgård Scheme with a Permutation. In *Asiacrypt'2007*, volume **4833** of *Lecture Notes in Computer Science*, pages 113–129, Springer-Verlag, 2007.

13. J. Kelsey and T. Kohno. Herding Hash Functions and the Nostradamus Attack. In *Advances in Cryptology-Eurocrypt'2006*, volume **4004** of *Lecture Notes in Computer Science*, pages 183–200, Springer-Verlag, 2006.

14. J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *Advances in Cryptology-Eurocrypt'2005*, volume **3494** of *Lecture Notes in Computer Science*, pages 474–490, Springer-Verlag, 2005.

15. X. Lai and J. L. Massey. Hash Functions Based on Block Ciphers. In *Advances in Cryptology-Eurocrypt'1992*, volume **658** of *Lecture Notes in Computer Science*, pages 55–70, Springer-Verlag, 1993.

16. Stefan Lucks. A Failure-Friendly Design Principle for Hash Functions. In *Advances in Cryptology-Asiacrypt'20052*, volume **3788** of *Lecture Notes in Computer Science*, pages 474–494, Springer-Verlag, 2005.

17. U. Maurer, R. Renner and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In *TCC'2004*, volume **2951** of *Lecture Notes in Computer Science*, pages 21–39, Springer-Verlag, 2004.

18. U. Maurer and J. Sjödin. Single-key AIL-MACs from any FIL-MAC. In *ICALP'2005*, volume **3580** of *Lecture Notes in Computer Science*, pages 472–484, Springer-Verlag, 2005.

19. Ueli Maurer and Stefano Tessaro. Domain Extension of Public Random Functions: Beyond the Birthday Barrier. In *Advances in Cryptology-Crypto'2007*, volume **4622** of *Lecture Notes in Computer Science*, pages 187–204, Springer-Verlag, 2007.

20. R. C. Merkle. One way hash functions and DES. In *Advances in Cryptology-Crypto'1989*, volume **435** of *Lecture Notes in Computer Science*, pages 428–446, Springer-Verlag, 1990.

21. R. L. Rivest. Abelian square-free dithering for iterated hash functions. In *the second NIST hash workshop*, 2005.

Appendix.

Proof of Theorem 2. Let S be $S_{choppfMD}$. By Lemma 1, we only focus on computing an upper bound of $\mathbf{Stat}(f,S)$. Note that $\mathbf{Stat}(f,S)$ is defined over all deterministic algorithms. So when the oracle is f, the number of possible views is 2^{nq}. And for any deterministic algorithm A, each view occurs with probability $1/2^{nq}$. We let the set of 2^{nq} possible views be V_A. On the other hand, when the oracle is S, the number of possible views is at least $(2^n-1)(2^n-2)\cdots(2^n-q)$. We let the set of least possible views be T_S and the size of T_S be r_q. Since we want to compute an upper bound of $\mathbf{Stat}(f,S)$, we assume that each of T_S views occurs with probability $1/r_q$. Therefore,

$$\mathbf{Stat}_A(f,S) \\ = \tfrac{1}{2}\sum_{v\in V_A} |\Pr[f=v]-\Pr[S=v]|$$

$= \frac{1}{2}\sum_{v\in V_A\setminus T_S}|\Pr[f=v]-\Pr[S=v]| + \frac{1}{2}\sum_{v\in T_S}|\Pr[f=v]-\Pr[S=v]|$
$\leq \frac{1}{2}\sum_{v\in V_A\setminus T_S}|\frac{1}{2^{nq}}-0| + \frac{1}{2}\sum_{v\in T_S}|\frac{1}{2^{nq}}-\frac{1}{r_q}|$
$= \frac{1}{2}\cdot\frac{2^{nq}-r_q}{2^{nq}} + \frac{1}{2}\cdot|\frac{r_q}{2^{nq}}-\frac{r_q}{r_q}|$
$= \frac{1}{2}\cdot(1-\frac{r_q}{2^{nq}}) + \frac{1}{2}\cdot(1-\frac{r_q}{2^{nq}})$
$= 1-\frac{r_q}{2^{nq}}$
$= 1-\prod_{i=1}^{q}(1-\frac{i}{2^n})$
$\leq \sum_{i=1}^{q}(\frac{i}{2^n})$ (by Ineq 1.)
$= \frac{q(q+1)}{2^{n+1}}$. ■

Proof of Theorem 3. Let S be S_{chopMD}. By Lemma 1, we only focus on computing an upper bound of $\mathbf{Stat}(f,S)$. Note that $\mathbf{Stat}(f,S)$ is defined over all deterministic algorithms. We define an event Bad as follows : for given a view v, there is no r-multicollision in the most significant $n-s$ bits of y_i's (which is the most significant $n-s$ bits of outputs of S or f) of the view v. when the oracle is f, the number of possible views is 2^{nq}. And for any deterministic algorithm A, each view occurs with probability $1/2^{nq}$. We let the set of 2^{nq} possible views be V_A. On the other hand, when the oracle is S and Bad occurs, the number of possible views is at least $2^{(n-s)q}(2^s-r)^q$ by the process of choosing w and y in line 002 of S. We let the set of least possible views be T_S and the size of T_S be r_q. Since we want to compute an upper bound of $\mathbf{Stat}(f,S)$, we assume that each of T_S views occurs with probability $1/r_q$. And for f and S, the most significant $n-s$ bits of their outputs are chosen uniformly at random. So, the probability that the event Bad does not occur is computed as follows : We let $\mu(n-s,r,q)$ be the probability that there is a r-multicollision in the most significant $n-s$ bits of y_i's of the view v. As described in [9], by counting the number of r pairs computable from q responses, we can know that $\mu(n-s,r,q)\leq\frac{\binom{q}{r}}{2^{(n-s)(r-1)}}$. Especially, when $r=n-s$, $\mu(n-s,r,q)\leq\frac{\binom{q}{r}}{2^{(n-s)(r-1)}}<\frac{q^r}{2^{n-s}(r-1)}=(\frac{q}{2^{n-s}})^{n-s-1}\leq\frac{q}{2^{n-s}}$, where $q\leq 2^{n-s-1}$. Therefore,

$\mathbf{Stat}_A(f,S)$
$= \frac{1}{2}\sum_{v\in V_A\wedge Bad}|\Pr[F=v]-\Pr[G=v]|$
$+ \frac{1}{2}\sum_{v\in V_A\wedge\overline{Bad}}\Pr[F=v] + \frac{1}{2}\sum_{v\in V_A\wedge\overline{Bad}}\Pr[G=v]$ (by Lemma 2.)
$\leq \frac{1}{2}\sum_{v\in V_A\wedge Bad}|\Pr[F=v]-\Pr[G=v]| + \frac{1}{2}\frac{q}{2^{n-s}} + \frac{1}{2}\frac{q}{2^{n-s}}$
$= \frac{1}{2}\sum_{v\in(V_A\setminus T_S)\wedge Bad}|\Pr[F=v]-\Pr[G=v]|$
$+ \frac{1}{2}\sum_{v\in T_S\wedge Bad}|\Pr[F=v]-\Pr[G=v]| + \frac{q}{2^{n-s}}$
$\leq \frac{1}{2}\sum_{v\in(V_A\setminus T_S)\wedge Bad}|\frac{1}{2^{nq}}-0| + \frac{1}{2}\sum_{v\in T_S\wedge Bad}|\frac{1}{2^{nq}}-\frac{1}{r_q}| + \frac{q}{2^{n-s}}$
$\leq \frac{1}{2}\cdot(2^{nq}-r_q)\cdot|\frac{1}{2^{nq}}-0| + \frac{1}{2}\cdot r_q\cdot|\frac{1}{2^{nq}}-\frac{1}{r_q}| + \frac{q}{2^{n-s}}$
$= 1-\frac{r_q}{2^{nq}} + \frac{q}{2^{n-s}}$
$= 1-\prod_{i=1}^{q}(1-\frac{r}{2^s}) + \frac{q}{2^{n-s}}$
$\leq \sum_{i=1}^{q}(\frac{r}{2^s}) + \frac{q}{2^{n-s}}$ (by Ineq 1.)
$= \frac{rq}{2^s} + \frac{q}{2^{n-s}} = \frac{(n-s)q}{2^s} + \frac{q}{2^{n-s}}$ ■ (by $r=n-s$.)

Proof of Theorem 4. Let S be $S_{chopMDP}$. By Lemma 1, we only focus on computing an upper bound of $\mathbf{Stat}(f, S)$. Note that $\mathbf{Stat}(f, S)$ is defined over all deterministic algorithms. So when the oracle is f, the number of possible views is 2^{nq}. And for any deterministic algorithm A, each view occurs with probability $1/2^{nq}$. We let the set of 2^{nq} possible views be V_A. On the other hand, when the oracle is S, the number of possible views is at least $(2^n-1)(2^n-3)\cdots(2^n-2q+1)$. We let the set of least possible views be T_S and the size of T_S be r_q. Since we want to compute an upper bound of $\mathbf{Stat}(f, S)$, we assume that each of T_S views occurs with probability $1/r_q$. Therefore,

$$
\begin{aligned}
&\mathbf{Stat}_A(f, S)\\
&\quad = \tfrac{1}{2}\textstyle\sum_{v\in V_A} |\Pr[f = v] - \Pr[S = v]|\\
&\quad = \tfrac{1}{2}\textstyle\sum_{v\in V_A\setminus T_S} |\Pr[f = v] - \Pr[S = v]| + \tfrac{1}{2}\sum_{v\in T_S} |\Pr[f = v] - \Pr[S = v]|\\
&\quad \le \tfrac{1}{2}\textstyle\sum_{v\in V_A\setminus T_S} |\frac{1}{2^{nq}} - 0| + \frac{1}{2}\sum_{v\in T_S} |\frac{1}{2^{nq}} - \frac{1}{r_q}|\\
&\quad = \tfrac{1}{2}\cdot\tfrac{2^{nq}-r_q}{2^{nq}} + \tfrac{1}{2}\cdot|\tfrac{r_q}{2^{nq}} - \tfrac{r_q}{r_q}|\\
&\quad = \tfrac{1}{2}\cdot(1 - \tfrac{r_q}{2^{nq}}) + \tfrac{1}{2}\cdot(1 - \tfrac{r_q}{2^{nq}})\\
&\quad = 1 - \tfrac{r_q}{2^{nq}}\\
&\quad = 1 - \textstyle\prod_{i=1}^{q}(1 - \frac{2i-1}{2^n})\\
&\quad \le \textstyle\sum_{i=1}^{q}(\frac{2i-1}{2^n}) \qquad \text{(by Ineq 1.)}\\
&\quad = \tfrac{q^2}{2^n}. \blacksquare
\end{aligned}
$$

Proof of Theorem 5. Let S be $S_{chopWPH}$. By Lemma 1, we only focus on computing an upper bound of $\mathbf{Stat}(f = (f_1, f_2), S)$. Note that $\mathbf{Stat}((f_1, f_2), S)$ is defined over all deterministic algorithms. So when the oracles are (f_1, f_2), the number of possible views is $2^{wq_1+nq_2}$, where $q = q_1 + q_2$. And for any deterministic algorithm A, each view occurs with probability $1/2^{wq_1+nq_2}$. We let the set of $2^{wq_1+nq_2}$ possible views be V_A. On the other hand, when the oracle is S, the number of possible views is at least $(2^w - 1)(2^w - 2)\cdots(2^w - q_1)(2^n)^{q_2}$. We let the set of least possible views be T_S and the size of T_S be r_q. Since we want to compute an upper bound of $\mathbf{Stat}(f, S)$, we assume that each of T_S views occurs with probability $1/r_q$. Therefore,

$$
\begin{aligned}
&\mathbf{Stat}_A(f, S)\\
&\quad = \tfrac{1}{2}\textstyle\sum_{v\in V_A} |\Pr[f = v] - \Pr[S = v]|\\
&\quad = \tfrac{1}{2}\textstyle\sum_{v\in V_A\setminus T_S} |\Pr[f = v] - \Pr[S = v]| + \tfrac{1}{2}\sum_{v\in T_S} |\Pr[f = v] - \Pr[S = v]|\\
&\quad \le \tfrac{1}{2}\textstyle\sum_{v\in V_A\setminus T_S} |\frac{1}{2^{wq_1+nq_2}} - 0| + \frac{1}{2}\sum_{v\in T_S} |\frac{1}{2^{wq_1+nq_2}} - \frac{1}{r_q}|\\
&\quad = \tfrac{1}{2}\cdot\tfrac{2^{wq_1+nq_2}-r_q}{2^{wq_1+nq_2}} + \tfrac{1}{2}\cdot|\tfrac{r_q}{2^{wq_1+nq_2}} - \tfrac{r_q}{r_q}|\\
&\quad = \tfrac{1}{2}\cdot(1 - \tfrac{r_q}{2^{wq_1+nq_2}}) + \tfrac{1}{2}\cdot(1 - \tfrac{r_q}{2^{wq_1+nq_2}})\\
&\quad = 1 - \tfrac{r_q}{2^{wq_1+nq_2}}\\
&\quad = 1 - \textstyle\prod_{i=1}^{q_1}(1 - \frac{i}{2^w})\\
&\quad \le \textstyle\sum_{i=1}^{q_1}(\frac{i}{2^w}) \qquad \text{(by Ineq 1.)}\\
&\quad = \tfrac{q_1(q_1+1)}{2^{w+1}}
\end{aligned}
$$

$\leq \frac{q(q+1)}{2^{w+1}}$. ■

Proof of Theorem 6. Let S be $S_{chopEMD}$. By Lemma 1, we only focus on computing an upper bound of $\mathbf{Stat}(f, S)$. Note that $\mathbf{Stat}(f, S)$ is defined over all deterministic algorithms. So when the oracle is f, the number of possible views is 2^{nq}. And for any deterministic algorithm A, each view occurs with probability $1/2^{nq}$. We let the set of 2^{nq} possible views be V_A. On the other hand, when the oracle is S, the number of possible views is at least $(2^n - 2)(2^n - 3) \cdots (2^n - q)$. We let the set of least possible views be T_S and the size of T_S be r_q. Since we want to compute an upper bound of $\mathbf{Stat}(f, S)$, we assume that each of T_S views occurs with probability $1/r_q$. Therefore,

$$
\begin{aligned}
&\mathbf{Stat}_A(f, S)\\
&\quad = \tfrac{1}{2} \textstyle\sum_{v \in V_A} |\Pr[f = v] - \Pr[S = v]|\\
&\quad = \tfrac{1}{2} \textstyle\sum_{v \in V_A \setminus T_S} |\Pr[f = v] - \Pr[S = v]| + \tfrac{1}{2} \textstyle\sum_{v \in T_S} |\Pr[f = v] - \Pr[S = v]|\\
&\quad \leq \tfrac{1}{2} \textstyle\sum_{v \in V_A \setminus T_S} |\frac{1}{2^{nq}} - 0| + \tfrac{1}{2} \textstyle\sum_{v \in T_S} |\frac{1}{2^{nq}} - \frac{1}{r_q}|\\
&\quad = \tfrac{1}{2} \cdot \tfrac{2^{nq} - r_q}{2^{nq}} + \tfrac{1}{2} \cdot |\tfrac{r_q}{2^{nq}} - \tfrac{r_q}{r_q}|\\
&\quad = \tfrac{1}{2} \cdot (1 - \tfrac{r_q}{2^{nq}}) + \tfrac{1}{2} \cdot (1 - \tfrac{r_q}{2^{nq}})\\
&\quad = 1 - \tfrac{r_q}{2^{nq}}\\
&\quad = 1 - \textstyle\prod_{i=1}^{q} (1 - \frac{i+1}{2^n})\\
&\quad \leq \textstyle\sum_{i=1}^{q} (\frac{i+1}{2^n}) \qquad \text{(by Ineq 1.)}\\
&\quad = \tfrac{q(q+3)}{2^{n+1}}. \ \blacksquare
\end{aligned}
$$

Proof of Theorem 7. Let S be S_{chopNI}. By Lemma 1, we only focus on computing an upper bound of $\mathbf{Stat}(f, S)$. Note that $\mathbf{Stat}(f, S)$ is defined over all deterministic algorithms. So when the oracle is f, the number of possible views is 2^{nq}. And for any deterministic algorithm A, each view occurs with probability $1/2^{nq}$. We let the set of 2^{nq} possible views be V_A. On the other hand, when the oracle is S, the number of possible views is at least $(2^n - 1)(2^n - 2) \cdots (2^n - q)$. We let the set of least possible views be T_S and the size of T_S be r_q. Since we want to compute an upper bound of $\mathbf{Stat}(f, S)$, we assume that each of T_S views occurs with probability $1/r_q$. Therefore,

$$
\begin{aligned}
&\mathbf{Stat}_A(f, S)\\
&\quad = \tfrac{1}{2} \textstyle\sum_{v \in V_A} |\Pr[f = v] - \Pr[S = v]|\\
&\quad = \tfrac{1}{2} \textstyle\sum_{v \in V_A \setminus T_S} |\Pr[f = v] - \Pr[S = v]| + \tfrac{1}{2} \textstyle\sum_{v \in T_S} |\Pr[f = v] - \Pr[S = v]|\\
&\quad \leq \tfrac{1}{2} \textstyle\sum_{v \in V_A \setminus T_S} |\frac{1}{2^{nq}} - 0| + \tfrac{1}{2} \textstyle\sum_{v \in T_S} |\frac{1}{2^{nq}} - \frac{1}{r_q}|\\
&\quad = \tfrac{1}{2} \cdot \tfrac{2^{nq} - r_q}{2^{nq}} + \tfrac{1}{2} \cdot |\tfrac{r_q}{2^{nq}} - \tfrac{r_q}{r_q}|\\
&\quad = \tfrac{1}{2} \cdot (1 - \tfrac{r_q}{2^{nq}}) + \tfrac{1}{2} \cdot (1 - \tfrac{r_q}{2^{nq}})\\
&\quad = 1 - \tfrac{r_q}{2^{nq}}\\
&\quad = 1 - \textstyle\prod_{i=1}^{q} (1 - \frac{i}{2^n})\\
&\quad \leq \textstyle\sum_{i=1}^{q} (\frac{i}{2^n}) \qquad \text{(by Ineq 1.)}
\end{aligned}
$$

$= \frac{q(q+1)}{2^{n+1}}$. ■

Proof of Theorem 8. Let S be S_{chopCS}. By Lemma 1, we only focus on computing an upper bound of $\mathbf{Stat}(f, S)$. Note that $\mathbf{Stat}(f, S)$ is defined over all deterministic algorithms. So when the oracle is f, the number of possible views is 2^{nq}. And for any deterministic algorithm A, each view occurs with probability $1/2^{nq}$. We let the set of 2^{nq} possible views be V_A. On the other hand, when the oracle is S, the number of possible views is at least $(2^n-2)(2^n-3)\cdots(2^n-q)$. We let the set of least possible views be T_S and the size of T_S be r_q. Since we want to compute an upper bound of $\mathbf{Stat}(f, S)$, we assume that each of T_S views occurs with probability $1/r_q$. Therefore,

$$
\begin{aligned}
&\mathbf{Stat}_A(f, S)\\
&\quad = \tfrac{1}{2}\textstyle\sum_{v\in V_A} |\Pr[f=v]-\Pr[S=v]|\\
&\quad = \tfrac{1}{2}\textstyle\sum_{v\in V_A\setminus T_S} |\Pr[f=v]-\Pr[S=v]| + \tfrac{1}{2}\sum_{v\in T_S} |\Pr[f=v]-\Pr[S=v]|\\
&\quad \le \tfrac{1}{2}\textstyle\sum_{v\in V_A\setminus T_S} |\frac{1}{2^{nq}}-0| + \tfrac{1}{2}\sum_{v\in T_S} |\frac{1}{2^{nq}}-\frac{1}{r_q}|\\
&\quad = \tfrac{1}{2}\cdot\tfrac{2^{nq}-r_q}{2^{nq}} + \tfrac{1}{2}\cdot|\tfrac{r_q}{2^{nq}}-\tfrac{r_q}{r_q}|\\
&\quad = \tfrac{1}{2}\cdot(1-\tfrac{r_q}{2^{nq}}) + \tfrac{1}{2}\cdot(1-\tfrac{r_q}{2^{nq}})\\
&\quad = 1-\tfrac{r_q}{2^{nq}}\\
&\quad = 1-\textstyle\prod_{i=1}^{q}(1-\frac{i+1}{2^n})\\
&\quad \le \textstyle\sum_{i=1}^{q}(\frac{i+1}{2^n}) \qquad \text{(by Ineq 1.)}\\
&\quad = \tfrac{q(q+3)}{2^{n+1}}. \blacksquare
\end{aligned}
$$

Proof of Theorem 9. Let S be $S_{chopESh}$. By Lemma 1, we only focus on computing an upper bound of $\mathbf{Stat}(f, S)$. Note that $\mathbf{Stat}(f, S)$ is defined over all deterministic algorithms. So when the oracle is f, the number of possible views is 2^{nq}. And for any deterministic algorithm A, each view occurs with probability $1/2^{nq}$. We let the set of 2^{nq} possible views be V_A. On the other hand, when the oracle is S, the number of possible views is at least $(2^n-2)(2^n-5)\cdots(2^n-3q+1)$. We let the set of least possible views be T_S and the size of T_S be r_q. Since we want to compute an upper bound of $\mathbf{Stat}(f, S)$, we assume that each of T_S views occurs with probability $1/r_q$. Therefore,

$$
\begin{aligned}
&\mathbf{Stat}_A(f, S)\\
&\quad = \tfrac{1}{2}\textstyle\sum_{v\in V_A} |\Pr[f=v]-\Pr[S=v]|\\
&\quad = \tfrac{1}{2}\textstyle\sum_{v\in V_A\setminus T_S} |\Pr[f=v]-\Pr[S=v]| + \tfrac{1}{2}\sum_{v\in T_S} |\Pr[f=v]-\Pr[S=v]|\\
&\quad \le \tfrac{1}{2}\textstyle\sum_{v\in V_A\setminus T_S} |\frac{1}{2^{nq}}-0| + \tfrac{1}{2}\sum_{v\in T_S} |\frac{1}{2^{nq}}-\frac{1}{r_q}|\\
&\quad = \tfrac{1}{2}\cdot\tfrac{2^{nq}-r_q}{2^{nq}} + \tfrac{1}{2}\cdot|\tfrac{r_q}{2^{nq}}-\tfrac{r_q}{r_q}|\\
&\quad = \tfrac{1}{2}\cdot(1-\tfrac{r_q}{2^{nq}}) + \tfrac{1}{2}\cdot(1-\tfrac{r_q}{2^{nq}})\\
&\quad = 1-\tfrac{r_q}{2^{nq}}\\
&\quad = 1-\textstyle\prod_{i=1}^{q}(1-\frac{3i-1}{2^n})\\
&\quad \le \textstyle\sum_{i=1}^{q}(\frac{3i-1}{2^n}) \qquad \text{(by Ineq 1.)}
\end{aligned}
$$

$= \frac{q(3q+1)}{2^{n+1}}$. ■

Proof of Theorem 10. Let S be S_{pfCM}. By Lemma 1, we only focus on computing an upper bound of $\mathbf{Stat}(f, S)$. Note that $\mathbf{Stat}(f, S)$ is defined over all deterministic algorithms. So when the oracle is f, the number of possible views is 2^{nq}. And for any deterministic algorithm A, each view occurs with probability $1/2^{nq}$. We let the set of 2^{nq} possible views be V_A. On the other hand, when the oracle is S, the number of possible views is at least $(2^n-1)(2^n-7)\cdots(2^n-6q+5)$. We let the set of least possible views be T_S and the size of T_S be r_q. Since we want to compute an upper bound of $\mathbf{Stat}(f, S)$, we assume that each of T_S views occurs with probability $1/r_q$. Therefore,

$$
\begin{aligned}
&\mathbf{Stat}_A(f, S)\\
&\quad = \tfrac{1}{2}\textstyle\sum_{v\in V_A} |\Pr[f = v] - \Pr[S = v]|\\
&\quad = \tfrac{1}{2}\textstyle\sum_{v\in V_A\setminus T_S} |\Pr[f = v] - \Pr[S = v]| + \tfrac{1}{2}\textstyle\sum_{v\in T_S} |\Pr[f = v] - \Pr[S = v]|\\
&\quad \le \tfrac{1}{2}\textstyle\sum_{v\in V_A\setminus T_S} |\tfrac{1}{2^{nq}} - 0| + \tfrac{1}{2}\textstyle\sum_{v\in T_S} |\tfrac{1}{2^{nq}} - \tfrac{1}{r_q}|\\
&\quad = \tfrac{1}{2}\cdot\tfrac{2^{nq}-r_q}{2^{nq}} + \tfrac{1}{2}\cdot|\tfrac{r_q}{2^{nq}} - \tfrac{r_q}{r_q}|\\
&\quad = \tfrac{1}{2}\cdot(1 - \tfrac{r_q}{2^{nq}}) + \tfrac{1}{2}\cdot(1 - \tfrac{r_q}{2^{nq}})\\
&\quad = 1 - \tfrac{r_q}{2^{nq}}\\
&\quad = 1 - \textstyle\prod_{i=1}^{q}(1 - \tfrac{6i-5}{2^n})\\
&\quad \le \textstyle\sum_{i=1}^{q}(\tfrac{6i-5}{2^n}) \qquad \text{(by Ineq 1.)}\\
&\quad = \tfrac{q(3q-2)}{2^n}. \blacksquare
\end{aligned}
$$