An Investigation of the Enhanced Target Collision Resistance Property for Hash Functions *

Mohammad Reza Reyhanitabar, Willy Susilo, and Yi Mu

Centre for Computer and Information Security Research,
School of Computer Science and Software Engineering
University of Wollongong, Australia
{rezar, wsusilo, ymu}@uow.edu.au

Abstract. We revisit the enhanced target collision resistance (eTCR) property as a newly emerged notion of security for dedicated-key hash functions, which has been put forth by Halevi and Krawczyk at CRYPTO'06, in conjunction with the Randomized Hashing mode to archive this property. Our contribution is twofold. Firstly, we provide a full picture of the relationships between eTCR and each of the seven security properties for a dedicated-key hash function, considered by Rogaway and Shrimpton at FSE'04; namely, collision resistance (CR), the three variants of second-preimage resistance (Sec, aSec, eSec) and the three variants of preimage resistance (Pre, aPre, ePre). The results show that, for an *arbitrary* dedicated-key hash function, eTCR is *not* implied by any of these seven properties, and it can only imply three of the properties; namely, eSec (TCR), Sec, Pre. In the second part of the paper, we analyze eTCR preservation capabilities of several domain extension transforms (a.k.a. modes of operation) for hash functions, including (Plain, Strengthened, and Prefix-free) Merkle-Damgård, Randomized Hashing (variant in the dedicated-key hash function setting), Shoup, Enveloped Shoup, XOR Linear Hash (XLH), and Linear Hash (LH) methods. From this analysis it turns out that, with the exception of a *nested variant* of LH construction, *none* of the investigated transforms can preserve eTCR property.

Key words: Hash Functions, Security Notions, eTCR, Relationships, Domain Extension

1 Introduction

Cryptographic hash function are functions that can map variable length strings to fixed length strings while providing some required security properties. They are used in a vast variety of cryptographic applications and are indispensable part of digital signatures and message authentication codes (e.g. HMAC). Originally designed to make digital signatures more efficient, application of hash functions in schemes following hash-and-sign paradigm, like DSA, requires them to provide collision resistance (CR) property. Hash functions are also asked to provide several different security properties depending on the specific security requirements of the higher-level protocols utilizing them. Although CR is one of the most important and well-known security properties for a hash function, they are often asked to provide many other security properties that, depending on the requirements of the higher-level applications, may range from merely being a one-way function (i.e. preimage resistance property) to acting as a truly random function (i.e. a random oracle). Hence, unlike many other cryptographic primitives which are only aimed to fulfill a specific security notion, hash functions as workhorses of cryptography are usually assumed to provide a wide application dependent spectrum of security properties.

Halevi and Krawczyk at CRYPTO'06 [12] introduced eTCR property as a new "*enhanced*" variant of the well-known target collision resistance (TCR) property for a *dedicated-key* hash function. (We note that "TCR" [5] is an alternative name for the notion of UOWHF [17], and it is also called "eSec" according to [24].) Halevi and Krawczyk also introduced the Randomized Hashing mode (announced by NIST as SP 800-106 [19]) to realize an eTCR hash function to be used in digital signatures. This is motivated by the fact that, the CR property is known to be a very demanding property from theoretical viewpoint [26, 5, 20],

* A preliminary version of this paper appeared in [21].

and a practically broken or endangered property for many of the in-use hash functions like MD5 and SHA-1 [30, 29, 9]. In response to the recent cryptanalytic results against the standard hash functions, NIST has created a design competition for the next generation hash function standard which will be called SHA-3 [18]. It is "*hoped*" that SHA-3 standard will resist against all *known* attacks, especially the powerful statistical methods like differential cryptanalysis which have been used to attack MD5, SHA-1 and many other hash functions [30, 29, 28]. Meanwhile, the Randomized Hashing mode aims at providing a "safety net" by relaxing the current complete reliance on CR property without having to change the internals of an already implemented hash function like SHA-1. In a nutshell, Randomized Hashing construction, as shown in Fig. 1, converts a *keyless* hash function H (e.g. SHA-1) to a *dedicated-key* hash function $\tilde{H}$ defined as $\tilde{H}_K(M) = H(K||(M_1 \oplus K)|| \cdots ||(M_L \oplus K))$, where H is a Merkle-Damgård iterated hash function based on a compression function h. ($M = M_1|| \cdots ||M_L$ denotes the padded input message.)

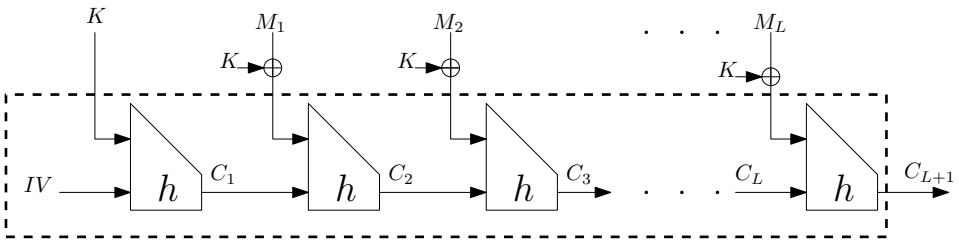


Fig. 1. Randomized Hashing construction

In [12], the eTCR security of this *dedicated-key hash function* $\tilde{H}$ is based on some new assumptions, called c-SPR and e-SPR, on the underlying *keyless compression function* h. These new assumptions on h are weaker security assumptions compared to the the CR assumption, and hence a keyless compression function h may remain secure in c-SPR or e-SPR sense, despite being broken in CR sense. We note that, as a result of future cryptanalytical results, the eTCR security of this specific Randomized Hashing construction $\tilde{H}$, may be threatened when implemented by a specific keyless hash functions (e.g. SHA-1), but the notion of eTCR and the problem of designing new eTCR-secure hash functions will still remain interesting independently from this specific construct. For instance, as noticed in [12], employing an eTCR hash function in a hash-and-sign digital signature scheme removes the need to sign the key K used for the hashing; it is only required to sign $H_K(M)$ and the key K can be sent in public to the verifier as part of the signed message [12]. This can be considered as an improvement (from efficiency viewpoint) compared to using a TCR (UOWHF) hash function where one has to sign $H_K(M)||K$ [5].

In pursuit of a clearer understanding of the notion of eTCR, in this paper we investigate and answer to the following two essential questions: (1) what are the formal relationships between eTCR and the previously known security notions (specially CR) for a *dedicated-key* hash function?, and (2) how can one convert an eTCR-secure compression function to a full-fledged eTCR-secure hash function, *i.e.* how to construct an eTCR-preserving domain extension transform?

Working out the formal relationships (implications and/or separations) between a new notion of security and other well-studied security notions is an essential step to clarify the relative position of the new property among the previously known ones. In regard to the security notions for hash functions, there are a few works in this line of research, e.g. [16, 27, 24]. As one of the most comprehensive works in this line of research, Rogaway and Shrimpton in [23] provided the relationships among the seven variants of the basic security notions for dedicated-key hash functions; namely, the CR (denoted by 'Coll' in [23]), Sec, aSec, eSec (TCR), Pre, aPre, and ePre properties.

The possibility of designing a property-preserving "domain extension transform"(a.k.a. "mode of operation" or "mode of iteration"), is another important issue to be considered with regard to a new security

property. The problem is that whether given a fixed-input-lenght (FIL) hash function (a.k.a. a compression function) which has a security property xxx, one can construct a full-fledged hash function, *i.e.* a variable-input-length (VIL) or arbitrary-input-length (AIL) hash function, that achieves the "*same*" security property xxx. In the case of the CR property, the seminal works of Merkle [15] and Damgård [8] showed that Merkle-Damgård (MD) iteration with strengthening padding is a CR-preserving domain extender. Analysis and design of (multi-)property-preserving domain extenders for hash function has been recently attracted new attention in several works considering different security properties, such as [5, 3, 2, 1].

Our Contributions. As our first contribution, we provide a full picture of the relationships, by working out all implications and separations, between the eTCR property and each of the seven variants of the basic security properties; namely, the CR, Sec, aSec, eSec, Pre, aPre and ePre properties. The summary of the results is depicted in Fig. 2. Interestingly and somewhat surprisingly, the results show that, for an *arbitrary* dedicated-key hash function, eTCR is *not* implied by any of the seven properties; in particular, we note that even the strong CR property does not imply eTCR in general. We stress that all these properties are formally defined for a dedicated-key hash function, and hence the CR notion here is different from the strong collision resistance *assumption* for a *keyless* hash function which cannot be formally defined. (We will briefly review the latter "foundations-of-hashing dilemma" regarding the definition of the notion of collision resistance for a keyless hash function.)

As our second contribution, we consider the problem of eTCR-preserving domain extension. We investigate eight domain extension transforms for this purpose; namely Plain MD [15, 8], Strengthened MD [15, 8], Prefix-free MD [7, 14], Randomized Hashing [12] (considered in dedicated-key hash setting), Shoup [25], Enveloped Shoup [2], XOR Linear Hash (XLH) [5], and a variant of Linear Hash (LH) [5] methods. Interestingly, we show that the only eTCR preserving method among these methods is a *nested variant* of LH (defined based on a variant proposed in [5]) which has the drawback of having linear key expansion factor. From this analysis, design of a new and more *efficient* eTCR preserving domain extender can be considered as an interesting open problem for future research. The overview of the constructions and the properties they preserve are shown in Table 1.

Scheme	CR	TCR	eTCR
Plain MD	× [15, 8]	× [5]	<u>×</u>
Strengthened MD	✓[15, 8]	× [5]	<u>×</u>
Prefix-free MD	× [2]	× [2]	<u>×</u>
Randomized Hashing	✓[1]	× [1]	<u>×</u>
Shoup	✓[25]	✓[25]	<u>×</u>
Enveloped Shoup	✓[2]	✓[2]	<u>×</u>
XOR Linear Hash (XLH)	✓[1]	✓[5]	<u>×</u>
Nested Linear Hash (LH)	✓[5]	✓[5]	<u>✓</u>

Table 1. Overview of constructions and the properties they preserve. The symbol "✓" means that the property is provably preserved by the construction; "×" means that it is not preserved. Underlined entries related to eTCR property are shown in this paper.

Organization of the Paper. In Section 2 we define the notations and conventions that we use through the rest of the paper, and review the syntax of the dedicated-key hash function setting. In Section 3 we review definitions of the security notions for a dedicated-key hash function, and the notions of implications and separations. Section 4 contains the main body of our first contribution, where we work out all relationships between eTCR and each of the seven security properties. In Section 5, as our second contribution, we investigate eTCR preservation capability of eight domain extension transforms. We conclude the paper and make some open questions for future research in Section 6.

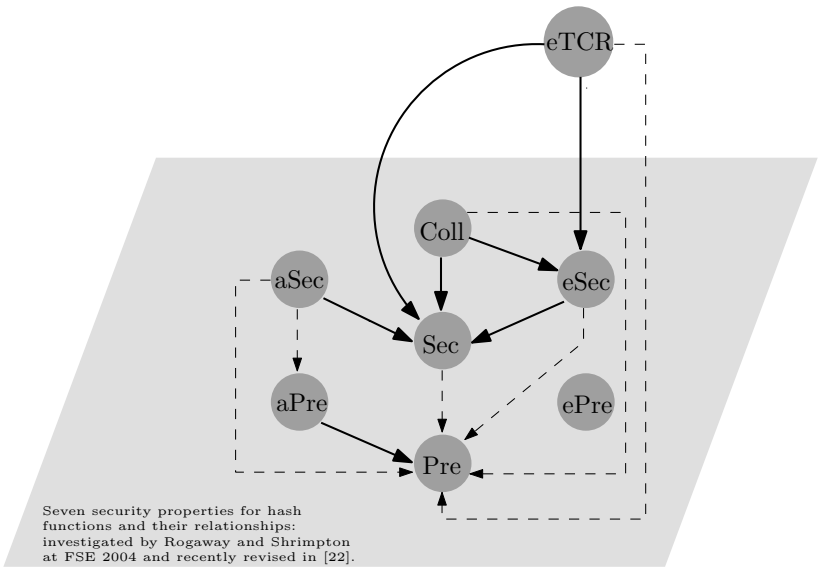


Fig. 2. Relationships between eTCR and the seven security notions for hash functions: a directed path shows an implication (dashed lines represent "provisional implications" in which the strength of the reduction used to show the implication depends on the amount of compression by the hash function), and lack of any directed path shows that there is a separation. New implications and separations between eTCR and each of the seven properties are shown by formal proofs and counterexamples in Sec. 4 of this paper. (Note that Coll and CR are different aliases for the collision resistance property (CR=Coll) and also eSec=TCR=UOWHF.)

2 Preliminaries

2.1 Notations and Conventions

If X is a finite set, by $x \xleftarrow{\$} X$ it is meant that x is chosen from X uniformly at random. For a binary string $M = M_1||M_2||\cdots||M_m$, let $M_{1\ldots n}$ denote the first n bits of M (*i.e.* $M_1||\cdots||M_n$) and $|M|$ denote its length in bits (where $n \leq m = |M|$). Let $x||y$ denote the string obtained from concatenating string y to string x. Let 1^m and 0^m, respectively, denote a string of m consecutive 1 and 0 bits, and 1^m0^n denote the concatenation of 0^n to 1^m. The set of all binary strings of length n bits (for some positive integer n) is denoted as $\{0,1\}^n$, the set of all binary strings whose lengths are variable but upper-bounded by N is denoted by $\{0,1\}^{\leq N}$ and the set of all finite binary strings is denoted by $\{0,1\}^*$. If S is a finite set we denote size of S by $|S|$. The symbol $\wedge$ denotes logical 'AND' operation, and the symbol $\vee$ denotes logical 'OR' operation.

Let $val(.)$ be the function that accepts any binary string M, considers it as an unsigned binary number with the rightmost bit (*i.e.* $M_{|M|}$) representing the least significant bit, and returns its decimal value as a non-negative integer. Let $\langle . \rangle_b$ denotes the operation that, accepts a non-negative integer z such that $\lceil log_2(z) \rceil \leq b$, and returns the binary representation of z in a b-bit long string $Z = Z_1||\cdots||Z_b$ in which Z_b is the least significant bit. (Note that $val(\langle z \rangle_b) = z$ as the domain of the function $\langle . \rangle_b$ is restricted to non-negative integers z s.t. $\lceil log_2(z) \rceil \leq b$, but we have $\langle val(Z) \rangle_b = Z$ only if *if* $|Z| \leq b$.)

By time complexity of an algorithm A, we mean its worst case running time, relative to some fixed model of computation (e.g. TM or RAM model), plus the size of the description of the algorithm using some fixed and reasonable encoding method. By time complexity of a function f, we mean the time complexity of the most efficient algorithm that can compute f.

If A is a probabilistic algorithm then by $y = A(x_1, \cdots, x_n; R)$ it is meant that y is the output of A on inputs $x_1, \cdots, x_n$ when it is provided with the random tape (or coins) R. It is assumed that R is of some length $r(|x|)$ where $r(.)$ is some known function called the length of the random tape of A and $|x|$ is the (total) length of the input(s). Whenever A needs to toss a coin, it simply reads the next bit on R. By

$y \xleftarrow{\$} A(x_1, \cdots, x_n)$ it is meant that $R \xleftarrow{\$} \{0,1\}^{r(|x|)}$ and $y = A(x_1, \cdots, x_n; R)$. If A has running time $t(|x|)$ then clearly it cannot read more than $t(|x|)$ bits from its random tape; hence, $r(|x|) \leq t(|x|)$. To show that an algorithm A is run without any input (*i.e.* when the input is an empty string) we use either the notation $y \xleftarrow{\$} A()$ or $y \xleftarrow{\$} A(\varnothing)$.

2.2 Two Settings for Hash Functions

For a formal treatment of hash functions and their security notions, one should clarify whether a keyless hash function or a dedicated-key hash function is to be considered. In the traditional keyless hash function setting, a hash function refers to a single-argument function $H : \mathcal{M} \to \{0,1\}^n$ (e.g. SHA-1 : $\{0,1\}^{<2^{64}} \to \{0,1\}^{160}$) that maps a variable-length input string to a fixed-length output string. In the dedicated-key setting, a hash function refers to a two-argument function $H : \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ which is also called a family of hash functions considering the key as an indexed (or salt) to select a instance function from the family.

The difference between two settings is worth emphasizing; for instance, some security properties like TCR and eTCR are defined and only make sense for a dedicated-key hash function [23, 12]. In this paper we consider the dedicated-key hash function setting, as the setting in which the eTCR property is defined.

Syntax. A dedicated-key hash function is a function $H : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$, where $\mathcal{C} = \{0,1\}^n$ for some positive integer n, the key space $\mathcal{K}$ is some nonempty *finite* set and the message space $\mathcal{M} \subseteq \{0,1\}^*$; such that $\{0,1\}^m \subseteq \mathcal{M}$ for at least a positive integer m, where $m > n$ if one insists that a hash function must compress.

For any $M \in \mathcal{M}$ and $K \in \mathcal{K}$, we use the notations $H_K(M)$ and $H(K, M)$ interchangeably, to denote the computed n-bit long hash value. We use $T_{H,\,\delta}$ to denote the time complexity of the most efficient algorithm that can compute $H(K, M)$, for any $M \in \{0,1\}^\delta \subseteq \mathcal{M}$ and $K \in \mathcal{M}$, plus the time complexity of the most efficient algorithm that can sample from the (finite) set $\mathcal{K}$.

Depending on the structure of $\mathcal{M}$, we can have: an FIL hash function (usually called a "compression function"), where $\mathcal{M} = \{0,1\}^m$; a VIL hash function, where $\mathcal{M} = \{0,1\}^{<\lambda}$ for some (huge) value λ (e.g. $\lambda = 2^{64}$ as in SHA-1), or an AIL hash function, where $\mathcal{M} = \{0,1\}^*$. We note that almost all of the iterated hash functions in practice are actually VIL and not AIL, but this difference becomes unimportant for practical uses where the message sizes will be always much less than, say 2^{64}.

3 Definitions

3.1 Security Notions

We briefly recall the following conventions from the concert-security framework, which are simplified for our purpose here. An adversary A is modeled as an algorithm that can be a probabilistic one (*i.e.* may use some randomness). For multi-stage adversarial computations, an adversary A may be viewed as consisting of several sub-algorithms, as $A = (A1, A2, \cdots, A_n)$, which are linked using a state variable to pass any information through the stages. Let $\mathrm{Adv}^{\mathsf{xxx}}_H(A)$ denote a probabilistic measure of A's success, *i.e.* its advantage, in attacking the xxx property of H, Let the resource parameterized function $\mathrm{Adv}^{\mathsf{xxx}}_H(t, \ell)$ denote the maximal value of the adversarial advantage; *i.e.* $\mathrm{Adv}^{\mathsf{xxx}}_H(t, \ell) = max_A \{\mathrm{Adv}^{\mathsf{xxx}}_H(A)\}$, over all adversaries A attacking the xxx property of H, with time complexity at most t and using messages of length at most ℓ bits. We say that H is (t, ℓ, ϵ)-xxx secure if $\mathrm{Adv}^{\mathsf{xxx}}_H(t, \ell) < \epsilon$. The resource parameter ℓ may be omitted from the notations if it is irrelevant in the context.

The advantage measures defining the security notions are shown in Fig. 3. Note that all the notions, except Coll (or CR) and ePre, are parameterized by a parameter δ where $\{0,1\}^\delta \subseteq \mathcal{M}$.

The Role of parameter δ. We notice that this parametrization (by δ) is firstly aimed to handle the technical issue that the efficient sampling from a set according to the uniform distribution requires the set

$$
\begin{aligned}
\mathrm{Adv}_H^{CR}(A) &= \Pr\left[K \xleftarrow{\$} \mathcal{K}; (M, M') \xleftarrow{\$} A(K) : \quad M \neq M' \ \wedge \ H_K(M) = H_K(M')\right] \\
\mathrm{Adv}_H^{Sec[\delta]}(A) &= \Pr\left[\begin{array}{l} K \xleftarrow{\$} \mathcal{K}; M \xleftarrow{\$} \{0,1\}^{\delta}; \\ M' \xleftarrow{\$} A(K, M) \qquad\qquad : \quad M \neq M' \ \wedge \ H_K(M) = H_K(M') \end{array}\right] \\
\mathrm{Adv}_H^{aSec[\delta]}(A) &= \Pr\left[\begin{array}{l} (K, State) \xleftarrow{\$} A_1(); \\ M \xleftarrow{\$} \{0,1\}^{\delta}; \\ M' \xleftarrow{\$} A_2(M, State) \quad : \quad M \neq M' \ \wedge \ H_K(M) = H_K(M') \end{array}\right] \\
\mathrm{Adv}_H^{eSec[\delta]}(A) &= \Pr\left[\begin{array}{l} (M, State) \xleftarrow{\$} A_1(); \\ K \xleftarrow{\$} \mathcal{K}; \\ M' \xleftarrow{\$} A_2(K, State) \quad : \quad M \neq M' \ \wedge \ H_K(M) = H_K(M') \end{array}\right] \\
\mathrm{Adv}_H^{Pre[\delta]}(A) &= \Pr\left[\begin{array}{l} K \xleftarrow{\$} \mathcal{K}; M \xleftarrow{\$} \{0,1\}^{\delta}; Y \leftarrow H_K(M); \\ M' \xleftarrow{\$} A(K, Y) \qquad\qquad : \quad H_K(M') = Y \end{array}\right] \\
\mathrm{Adv}_H^{aPre[\delta]}(A) &= \Pr\left[\begin{array}{l} (K, State) \xleftarrow{\$} A_1(); \\ M \xleftarrow{\$} \{0,1\}^{\delta}; Y \leftarrow H_K(M); \\ M' \xleftarrow{\$} A_2(Y, State) \quad : \quad H_K(M') = Y \end{array}\right] \\
\mathrm{Adv}_H^{ePre}(A) &= \Pr\left[(Y, State) \xleftarrow{\$} A_1(); K \xleftarrow{\$} \mathcal{K}; M' \xleftarrow{\$} A_2(K, State) : \quad H_K(M') = Y\right] \\
\mathrm{Adv}_H^{eTCR[\delta]}(A) &= \Pr\left[\begin{array}{l} (M, State) \xleftarrow{\$} A_1(); \\ K \xleftarrow{\$} \mathcal{K}; \\ K', M' \xleftarrow{\$} A_2(K, State) : \ (K, M) \neq (K', M') \ \wedge \ H_K(M) = H_{K'}(M') \end{array}\right]
\end{aligned}
$$

Fig. 3. Definitions of the security notions for a dedicated-key hash function $H : \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^n$. In the case of eSec (=TCR) and eTCR notions the parameter δ is assumed to be the length of the first (*i.e.* the target) message M output by A_1 in the first stage of the attack.

to be finite. Therefore, if an AIL hash function is to be considered, *i.e.* $\mathcal{M} = \{0,1\}^*$, then the message space is infinite and cannot be sampled uniformly at random. Secondly, it is motivated by the following observation. The ideal security level for (variants of) the second-preimage resistance and preimage resistance properties of a hash function with n-bit output is 2^n, due to a simple generic (random search) attack. Clearly, if the length of the target message is known to be shorter than the hash size n, then one will be able to search the input message space in less than 2^n steps. On the other hand, for practical iterated hash functions if the length of the target message is too long; e.g. 2^l blocks for some large l, then there are generic long message second-preimage attacks, put forth by Kelsey and Schneier [13], with complexity of about $l2^{n/2+1} + 2^{n-l+1}$ which becomes much less than the ideal 2^n level if the target message is too long, e.g. $l = n/2$. Therefore, explicitly parameterizing the security notions by the length of the target messages, i.e. δ, can help clarify these dependencies of the advantage functions on the target message length.

CR for a Keyless Hash Function. It is well-known that collision resistance as a security property cannot be formally defined for a keyless hash function $H : \mathcal{M} \rightarrow \{0,1\}^n$. *Informally*, one would say that it is "*infeasible*" to find two distinct messages M and M' such that $H(M) = H(M')$. But it is easy to see that if the hash function is compressing then there are many colliding pairs and hence, trivially there *exists* an *efficient* program that can always output a colliding pair M and M', namely a simple one with M and M' included in its code. That is, *infeasibility* cannot be formalized by an statement like "there exists no efficient adversary with non-negligible advantage" as clearly there are many such adversaries as mentioned before. The point is that *no human being knows* such a program [22], but the latter concept cannot be formalized mathematically. Therefore, in the context of keyless hash functions, CR can only be treated as a strong *assumption* to be used in a constructive security reduction following human-ignorance framework of [22]. We will call such a CR assumption about a keyless hash function as **keyless-CR assumption** to distinguish it from the formally definable CR (=Coll) notion for a dedicated-key hash function. We note

that the recent collision finding attacks show that the keyless-CR assumption is completely invalid for MD5 [30] and theoretically endangered assumption for SHA-1 [29].

3.2 Notions of Implication and Separation

We will use the following notions of implication and separation among the security properties in this paper.

Definition 1 (Implication). Let xxx and yyy be two security notions defined for an *arbitrary* hash function $H : \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^n$, and fix δ such that $\{0,1\}^\delta \subseteq \mathcal{M}$. We say that xxx implies yyy if $\mathrm{Adv}_H^{\mathsf{yyy}}(t') \leq c\mathrm{Adv}_H^{\mathsf{xxx}}(t) + \mu$; where $t' = t - c'T_{H,\delta}$, c and c' are some non-negative constants, and μ is a function of the hash function parameters (*i.e.* input, output, and/or key sizes). The exact strength of the implication will depend on μ as well as the constants c and c', and we may have:

- *Security-Preserving Implications.* If $\mu = 0$ then we have a security-preserving implication which is denoted by xxx $\rightarrow$ yyy.
- *Provisional Implications.* If $\mu \neq 0$ then we have a provisional implication, denoted by xxx $\dashrightarrow$ yyy, whose strength is provisioned on μ and hence may vanish if μ becomes "large". For example, for $\mu = 2^{n-\delta}$, the strength of such a provisional implication will depend on the amount of compression done by the hash function.

Definition 2 (Separation). Let xxx and yyy be two security notions defined for an *arbitrary* hash function $H : \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^n$, and fix δ such that $\{0,1\}^\delta \subseteq \mathcal{M}$. We use xxx $\nrightarrow$ yyy to show that the notion xxx does not imply the notion yyy, and this is proved by providing counterexamples. Namely, assuming that there exists a dedicated-key hash function $H : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$, which is $(t, \epsilon) -$ xxx secure, we construct (as a counterexample) a dedicated-key hash function $G : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$ which is also $(t', \epsilon') -$ xxx secure, but *completely insecure* in yyy sense; *i.e.* $\mathrm{Adv}_G^{\mathsf{yyy}}(c'') = 1$, where c'' is a small constant. The concrete relations between adversarial advantages (*i.e.* $\epsilon = \mathrm{Adv}_H^{\mathsf{xxx}}(t)$ and $\epsilon' = \mathrm{Adv}_G^{\mathsf{xxx}}(t')$) will be given in one of the following two forms:

1. $\mathrm{Adv}_G^{\mathsf{xxx}}(t') \leq c\mathrm{Adv}_H^{\mathsf{xxx}}(t) + \mu(n,k,\delta)$
2. $\mathrm{Adv}_G^{\mathsf{xxx}}(t') \leq c\mathrm{Adv}_H^{\mathsf{xxx}}(t) + c'\sqrt{\mathrm{Adv}_H^{\mathsf{xxx}}(t)} + \mu(n,k,\delta)$

, where $t' = t - c''T_{H,\delta}$; c, c' and c'' are some non-negative constants, and $\mu(n,k,\delta)$ depends on the hash function parameters n, k and δ.

Remark 1. The first-type bound above for establishing a separation, is the (strong) conventional form that used in the context of comparing the seven security properties in [24]. But, while investigating the relationships between eTCR and the seven properties, in some cases it appears a non-trivial task for us (if possible at all) to provide counterexamples yielding to such a strong bound, and hence we demonstrate some of the separations by counterexamples supporting the second-type bound (*i.e.* with a quadratic security degradation).

4 Relationships among the Security Notions

4.1 eTCR *vs.* CR

In this section, we investigate the relationship between the eTCR and CR properties. We provide mutual separations between the two properties in the conventional sense. The separations between eTCR and CR are of special theoretical and practical interest, considering the fact that collision resistance property has been known, for a long time, as one of the most challenged and demanding properties for a hash function,

both theoretically [26, 5, 20] and practically [30, 29]. Our separation results show that, "*in general*", the new eTCR property neither implies nor is implied by the CR property when both notions are considered for an "*arbitrary*" dedicated-key hash function $H : \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^n$. We note that, although such separation results do not rule out the possibility of designing a "*specific*" dedicated-key hash function in which eTCR might be easier to achieve compared to CR (or vice versa), *they emphasize the point that any such a specific construction should explicitly and clearly shows that this is indeed the case.*

We emphasize that these mutual separation (incomparability) results are demonstrated from dedicated-key hash function setting viewpoint, and not from the existential viewpoint in the complexity-theoretic sense. Indeed in the complexity-theoretic sense, existence of an eTCR function is implied by that of a one-way function [31], but there is a strong evidence that this is not true for the case of a collision resistance function [26]. That is, eTCR assumption is *existentially no stronger* than CR in the complexity-theoretic sense.

Theorem 1. *eTCR and CR are two incomparable security properties in the dedicated-key hash function setting; in the sense that, for an arbitrary hash function $H : \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^n$ neither of the properties implies the other.*

The proof is obtained by combining the separations in Lemma 1 and Lemma 2.

Lemma 1 (CR $\nrightarrow$ eTCR). *Assume that there exists a dedicated-key hash function $H : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$ (where $m \geq n$) which is $(t, \epsilon) - CR$. Select (and* fix*) an arbitrary message $M^* \in \{0,1\}^m$ and an arbitrary key $K^* \in \{0,1\}^k$ (e.g. $M^* = 1^m$ and $K^* = 1^k$). The dedicated-key hash function $G : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$ shown in this lemma is $(t', \epsilon') - CR$, where $t' = t - cT_{H,m}$ and $\epsilon' = \epsilon + 2^{-k}$, but it is completely insecure in eTCR sense. $T_{H,m}$ denotes the time for one computation of H, and c is a small constant.*

$$G_K(M) = \begin{cases} M^*_{1\cdots n} & \text{if } M = M^* \ \vee \ K = K^* \quad (1) \\ H_K(M^*) & \text{if } M \neq M^* \ \wedge \ K \neq K^* \ \wedge \ H_K(M) = M^*_{1\cdots n} \quad (2) \\ H_K(M) & \text{otherwise} \quad (3) \end{cases}$$

The proof is valid for any arbitrary selection of parameters $M^* \in \{0,1\}^m$ and $K^* \in \{0,1\}^k$, and hence, this construction actually shows 2^{m+k} such counterexample functions, which are CR but not eTCR.

Proof. Let's first demonstrate that G as a dedicated-key hash function is completely insecure in eTCR sense. This can be shown by the following simple adversary $A = (A_1, A_2)$ playing eTCR game against G. In the first stage of eTCR attack, A_1 outputs the target message as $M = M^*$. In the second stage of the attack, A_2, after receiving the first randomly selected key K (where $K \xleftarrow{\$} \{0,1\}^k$), outputs a different message $M' \neq M^*$ and selects the second key as $K' = K^*$. It can be seen easily that the adversary $A = (A_1, A_2)$ always wins the eTCR game, as $M' \neq M^*$ implies that $(M^*, K) \neq (M', K^*)$ and by the construction of G we have $G_K(M^*) = G_{K^*}(M') = M^*_{1\cdots n}$.

To complete the proof, we need to show that G inherits the CR property of H. Let A be an adversary that can win CR game against G with probability ϵ' using time complexity t'. We construct an adversary B against CR property of H with success probability of at least $\epsilon = \epsilon' - 2^{-k}$, and time $t = t' + cT_{H,m}$ as stated in the lemma.

The construction of B is as follows:

Algorithm $B(K)$
10: $(M, M') \xleftarrow{\$} A(K)$;
20: if $[M = M^* \wedge H_K(M') = M^*_{1\cdots n}]$ then **return** (M, M');
30: if $[M' = M^* \wedge H_K(M) = M^*_{1\cdots n}]$ then **return** (M, M');
40: if $[M \neq M^* \wedge H_K(M) = M^*_{1\cdots n} \wedge M' \neq M^* \wedge H_K(M') \neq M^*_{1\cdots n}]$ then
return (M^*, M');
50: if $[M' \neq M^* \wedge H_K(M') = M^*_{1\cdots n} \wedge M \neq M^* \wedge H_K(M) \neq M^*_{1\cdots n}]$ then
return (M, M^*);
60: **return** (M, M');

We claim that if $K \neq K^*$ then B will return a valid collision for H whenever A returns a valid collision (M, M') for G. Referring to the definition of G, if A returns a valid collision (M, M') under G_K, we can analyze all possible cases that this can happen and show that in each case the algorithm B also returns a collision for H_K. Let **(i)-(j) Collision** mean that the colliding messages M and M' output by A for G_K, respectively, satisfy conditions in line (i) and line (j) in definition of the function G. Then we have the following cases (assuming that $K \neq K^*$):

1. **(1)-(1) Collision**, **(1)-(3) Collision** and **(3)-(1) Collision** are not possible. A **(1)-(1) Collision** implies that $M = M'$ which is not possible as it is assumed that (M, M') is a valid collision for G_K. Now, note that the condition in line (3) of the definition of G (implicitly denoted as "*otherwise*") can be explicitly shown as: [if $M \neq M^* \wedge K \neq K^* \wedge H_K(M) \neq M^*_{1\cdots n}$]; that is, the hash value computed in line (3) of G, is always different from $M^*_{1\cdots n}$ and therefore **(1)-(3) Collision** and **(3)-(1) Collision** are impossible.
2. **(1)-(2) Collision**: When A outputs a valid **(1)-(2) Collision** for G (*i.e.* $M' \neq M \wedge G_K(M') = G_K(M)$), referring to the definition of G and remembering the assumption that $K \neq K^*$, it can be seen that $M = M^*$ and $H_K(M') = M^*_{1\cdots n}$ because this is a **(1)-(2) Collision** and from $G_K(M') = G_K(M)$ we have $H_K(M^*) = M^*_{1\cdots n}$. In this case, the adversary B returns (M, M') in line 20 of its code as a collision for H_K and wins because $H_K(M) = H_K(M^*) = M^*_{1\cdots n} = H_K(M')$.
3. **(2)-(1) Collision**: The proof of this case is symmetric to the case of **(1)-(2) Collision**; in this case, B returns (M, M') in line 30 of its code as collision for H_K.
4. **(2)-(3) Collision**: We show that in this case, B returns (M^*, M') as a collision for H_K in line 40 of its code and wins. Whenever A outputs a valid **(2)-(3) Collision** for G then (by referring to the definition of G, remembering the assumption $K \neq K^*$ and considering the condition in line (3) of G explicitly) it can be seen that $M \neq M^*$, $H_K(M) = M^*_{1\cdots n}$, $M' \neq M^*$ and $H_K(M') \neq M^*_{1\cdots n}$. Hence, as (M, M') output by A is a valid collision for G, *i.e.* $G_K(M') = G_K(M)$, we have that $H_K(M') = H_K(M^*)$ and therefor (M^*, M') returned by B in line 40, is a valid collision for H_K.
5. **(3)-(2) Collision**: The proof of this case is symmetric to the case of **(2)-(3) Collision**; in this case, B returns (M, M^*) in line 50 of its code as a collision for H_K.
6. **(2)-(2) Collision** and **(3)-(3) Collision**: It can be seen that in these two cases, the adversary B returns (M, M') as a collision for H_K in line 60 of its code. Referring to the definition of G, whenever A outputs a valid collision (M, M') for G_K as either a **(2)-(2) Collision** or **(3)-(3) Collision** (that is, $M \neq M' \wedge G_K(M) = G_K(M')$ and both M and M' belong to the same sub-domain of G) then (M, M') will also be a valid collision for H_K. Note that $G_K(M) = G_K(M')$ implies that in the **(2)-(2) Collision** case we have $H_K(M) = H_K(M') = H_K(M^*)$ and in the **(3)-(3) Collision** case we have $H_K(M) = H_K(M')$.

The above case analysis shows that if $K \neq K^*$ then B will be successful in finding a valid collision for H_K whenever A can find a valid collision for G_K. If $K = K^*$ then the returned pair of messages by B will not

necessarily be a valid collision for H. Therefore, we have $\epsilon = \Pr[B \text{ succeeds}] = \Pr[A \text{ succeeds} \wedge K \neq K^*] \geq \Pr[A \text{ succeeds}] - \Pr[K = K^*] = \epsilon' - 2^{-k}$. □

Lemma 2 (eTCR $\nrightarrow$ CR). *Assume that there exists a dedicated-key hash function $H : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$, where $m > k \geq n$, which is $(t, \epsilon) - eTCR$. The dedicated-key hash function $G : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$ shown in this lemma is $(t', \epsilon') - eTCR$, where $t' = t - c$, $\epsilon' = \epsilon + 2^{-k+1}$, but it is completely insecure in CR sense. (c is a small constant.)*

$$G_K(M) = \begin{cases} H_K(0^{m-k}||K) & \text{if } M = 1^{m-k}||K \\ H_K(M) & \text{otherwise} \end{cases}$$

Note that the structural assumption about $H : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$, namely that we have $m > k \geq n$ is quite reasonable even for practical scenarios. For instance, in Randomized Hashing which should provide a dedicated-key hash function with eTCR property, the key length k is fixed and equal to the block length of the underlying keyless hash function (e.g using SHA-1 we have $k = 512$, $n = 160$) while message length m can be very large (just less than 2^{64}).

Proof. We firstly demonstrate that G is completely insecure in CR sense, by the following simple adversary A. On receiving the key K, the adversary A outputs two different messages as $M = 1^{m-k}||K$ and $M' = 0^{m-k}||K$ and wins the CR game as we have $G_K(1^{m-k}||K) = H_K(0^{m-k}||K) = G_K(0^{m-k}||K)$.

It remains to show that that G is an eTCR-secure hash function. Let $A = (A_1, A_2)$ be an adversary which wins the eTCR game against G with probability ϵ' and using time complexity t'. We construct an adversary $B = (B_1, B_2)$ which uses A as a subroutine and wins eTCR game against H with success probability at least $\epsilon = \epsilon' - 2^{-k+1} (\approx \epsilon'$, for large $k)$ and spending time complexity $t = t' + c$ where small constant c can be determined from the description of algorithm B.

To make the proof easier to follow, we use a boolean variable "*bad*" as a flag whose initial value is assumed to be '*false*'. This flag is set by B when an undesirable event happens that could make B unsuccessful even if A was successful. We note that, the conditional statement checking the occurrence of the bad event (in line 30) and setting the flag *bad* to *true* is dummy and can be omitted from the code of B. Algorithm B is as follows:

Algorithm $B_1()$	**Algorithm** $B_2(K, M, State)$
10: $(M, State) \xleftarrow{\$} A_1()$;	30: if $\left[M = 1^{m-k}\|\|K \vee M = 0^{m-k}\|\|K\right]$ then $bad \leftarrow true$;
20: **return** $(M, State)$;	40: $(K', M') \xleftarrow{\$} A_2(K, M, State)$;
	50: if $M' = 1^{m-k}\|\|K'$ then **return** $(K', 0^{m-k}\|\|K')$;
	60: **return** (K', M');

As it can be seen from B's description, in the first stage of eTCR attack B_1 just merely runs A_1 and returns whatever it returns as the first message(M) and any possible state information to be passed on to the second stage algorithm. In the second stage of the attack, let **Bad** be the event that $[M = 1^{m-k}||K \vee M = 0^{m-k}||K]$; that is, the flag *bad* is set to *true*. Let $\overline{\textbf{Bad}}$ denote the complement event for **Bad**, *i.e.* $[M \neq 1^{m-k}||K \wedge M \neq 0^{m-k}||K]$.

Using the following simple case analysis, we can show that if **Bad** does not happen then B will succeed in eTCR attack against H whenever A succeeds in eTCR attack against G:

1. **Case 1:** $M' = 1^{m-k}||K'$**.** In this case, we have $(K, M) \neq (K', 1^{m-k}||K')$ and $G_K(M) = G_{K'}(1^{m-k}||K')$ (because we assume A succeeds in eTCR attack against G), and this in turn implies that $(K, M) \neq (K', 0^{m-k}||K')$ and $H_K(M) = H_{K'}(0^{m-k}||K')$ (according to the description of G and the assumption that **Bad** does not happen). Hence, in this case B becomes successful by returning $(K', 0^{m-k}||K')$ in

line 50 of its code. (It might seem non-trivial why *in this case* $(K, M) \neq (K', 1^{m-k}||K')$ implies that $(K, M) \neq (K', 0^{m-k}||K')$. To verify this, note that if $K \neq K'$ this becomes obvious, and if $K = K'$ then from the assumption that **Bad** has not happened we know that $M \neq 0^{m-k}||K$.)

2. **Case 2:** $M' \neq 1^{m-k}||K$**.** In this case, B succeeds by just returning (K', M') in (line 60 of its code in) the second stage, *i.e.* the same message and key pair as A returns in its second stage. This is easy to verify as in this case from the description of G we have $G_K(M) = H_K(M)$ and $G_{K'}(M') = H_{K'}(M')$, and so B wins against H if A wins against G.

Now note that $\Pr[\mathbf{Bad}] = \Pr[M = 1^{m-k}||K] + \Pr[M = 0^{m-k}||K] = 2^{-k} + 2^{-k} = 2^{-k+1}$, as K is selected uniformly at random just after the message M is fixed in the eTCR game. Hence, we have $\epsilon = \Pr[B \text{ succeeds}] = \Pr[A \text{ succeeds} \wedge \overline{\mathbf{Bad}}] \geq \Pr[A \text{ succeeds}] - \Pr[\mathbf{Bad}] = \epsilon' - 2^{-k+1}$. □

The Case for the Randomized Hashing. Randomized Hashing method as shown in Fig. 1 is a simple method to obtain a dedicated-key hash function $\tilde{H} : \mathcal{K} \times M \rightarrow \{0,1\}^n$ from an iterated (keyless) hash function H as $\tilde{H}(K, M) \triangleq H\big(K||(M_1 \oplus K)|| \cdots ||(M_L \oplus K)\big)$, where $\mathcal{K} = \{0,1\}^b$ and H itself is constructed by iterating a keyless compression function $h : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ and using a *fixed* initial chaining value IV. The analysis in [12] reduces the security of $\tilde{H}$ in eTCR sense to some assumptions, called c-SPR and e-SPR, on the keyless compression function h which are weaker than the keyless-CR assumption on h.

Here, we are interested in a somewhat different question, namely whether (formally definable) CR for this specific design of dedicated-key hash function $\tilde{H}$ implies that it is eTCR or not. Interestingly, we can gather a strong evidence that CR for $\tilde{H}$ implies that it is also eTCR, by the following argument. First, from the construction of $\tilde{H}$ it can be seen that CR for $\tilde{H}$ implies keyless-CR for a hash function H^* which is identical to the H except that its initial chaining value is a random and *known* value $IV^* = h(IV||K)$ instead of the prefixed IV (Note that K is selected at random and is provided to the adversary at the start of CR game). This is easily proved, as any adversary that can find collisions for H^* (*i.e.* breaks it in keyless-CR sense) can be used to construct an adversary that can break $\tilde{H}$ in CR sense. Second, from recent cryptanalysis methods which use differential attacks to find collisions [30, 29], we have a strong evidence that finding collisions for H^* under known IV^* would not be harder than finding collisions for H under IV, for a practical hash function like MD5 or SHA-1. That is, we argue that if H^* is keyless-CR then H is also keyless-CR. Finally, we note that keyless-CR assumption on H in turn implies that $\tilde{H}$ is eTCR as follows. Consider a successful eTCR attack against $\tilde{H}$ where on finishing the attack we will have $(K, M) \neq (K', M')$ and $\tilde{H}(K, M) = \tilde{H}(K', M')$; where, $M = M_1|| \cdots ||M_L$ and $M' = M'_1|| \cdots ||M'_{L'}$. Referring to the construction of $\tilde{H}$ this is translated to $H\big(K||(M_1 \oplus K)|| \cdots ||(M_L \oplus K)\big) = H\big(K'||(M'_1 \oplus K')|| \cdots ||(M'_{L'} \oplus K')\big)$ and from $(K, M) \neq (K', M')$ we have that $K||(M_1 \oplus K)|| \cdots ||(M_L \oplus K) \neq K'||(M'_1 \oplus K')|| \cdots ||(M'_{L'} \oplus K')$. Hence, we have found a collision for H and this contradicts the assumption that H is keyless-CR. Therefore, for the case of the specific dedicated-key hash function $\tilde{H}$ obtained via Randomized Hashing mode, it can be argued that CR implies eTCR.

4.2 Other Relationships

In the previous subsection, we investigated the relationship between eTCR and CR. Now, we continue to complete all the remaining new relationships between eTCR and each of the other six properties; namely, Sec, aSec, eSec (TCR), Pre, aPre, ePre.

Theorem 2 (Implications). *For any dedicated-key hash function $H : \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^n$ and for any fixed value of δ such that $\{0,1\}^\delta \subseteq \mathcal{M}$, we have:*

1. *eTCR $\rightarrow$ eSec:* $Adv_H^{eSec[\delta]}(t) \leq Adv_H^{eTCR[\delta]}(t)$
2. *eTCR $\rightarrow$ Sec:* $Adv_H^{Sec[\delta]}(t) \leq Adv_H^{eTCR[\delta]}(t)$

3. *eTCR ⇢ Pre:* $Adv_H^{Pre[\delta]}(t') \leq 2Adv_H^{eTCR[\delta]}(t) + 2^{n-\delta}$

Proof. The security preserving implications (cases 1 and 2 above) are quite straightforward to show by simply looking at the definitions of these properties in Fig. 3. The provisional implication 'eTCR ⇢ Pre is also easily deduced combining 'eTCR → Sec' with the known fact from [24] that 'Sec ⇢ Pre; namely, $\mathrm{Adv}_H^{Pre[\delta]}(t') \leq 2\mathrm{Adv}_H^{Sec[\delta]}(t) + 2^{n-\delta}$, where $t' = t - cT_{H,\delta}$, for a constant c. □

Interestingly, except the *three* (simple) implications stated in Theorem 2, all the remaining *eleven* relationships are of the separation type. We have already proved the two separations between eTCR and CR as they seem to be the most interesting ones from practical viewpoint. In Theorem 3, we complete the picture of the relationships by providing the remaining *nine* separations.

Theorem 3. *Let eTCR, Sec, aSec, eSec, Pre, aPre and ePre be the security notions as defined in Fig. 3 for some fixed value of the parameter δ. The following separations hold:*

1. *eTCR ↛ aSec*
2. *eTCR ↛ aPre*
3. *eTCR ↛ ePre*
4. *Sec ↛ eTCR*
5. *aSec ↛ eTCR*
6. *eSec ↛ eTCR*
7. *Pre ↛ eTCR*
8. *aPre ↛ eTCR*
9. *ePre ↛ eTCR*

We note that the first three separations above are new and cannot be deduced from any previously known results. In the following, we provide complete proofs of these three separations in Lemma 3 and Lemma 4. The remaining separations, *i.e.* cases 4-9 above, can be easily deduced combining the fact that eTCR → TCR (=eSec) and the known separations shown by Rogaway and Shrimpton in [24] (for example, from 'Sec ↛ TCR' [24] and 'eTCR → TCR' we get that Sec ↛ eTCR, and so on).

Lemma 3 (eTCR ↛ aSec and eTCR ↛ aPre). *Assume that there exists a dedicated-key hash function $H : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$ which is $(t, \epsilon) - eTCR$. Select (and* fix*) an arbitrary key $K^* \in \{0,1\}^k$ and an arbitrary hash value $C^* \in \{0,1\}^n$ (e.g. $K^* = 0^k$ and $C^* = 0^n$). The dedicated-key hash function $G1 : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$ shown in this lemma is $(t', \epsilon') - eTCR$, where $t' = t - c$ (where c is a constant) and $\epsilon' \leq \epsilon + \sqrt{\epsilon} + 2^{-k+1}$, but it is completely insecure in both aSec and aPre senses.*

$$G1_K(M) = \begin{cases} C^* & \text{if } K = K^* \quad (1) \\ \\ H_K(M) & \text{otherwise} \quad (2) \end{cases}$$

Proof. Let's first demonstrate that $G1$ is completely insecure in both aSec and aPre senses.

- $\mathrm{Adv}_{G1}^{aSec}(c') = 1$: Consider the following simple adversary $A = (A_1, A_2)$ playing aSec game against $G1$. A_1 chooses the key as $K = K^*$, and A_2 after receiving the first randomly selected message M, outputs any different message $M' \neq M$. It can be easily seen that this adversary, spending a small constant c', always wins the aSec game because $M' \neq M$, and by the construction of $G1$ we have $G1_{K^*}(M') = G1_{K^*}(M) = C^*$.
- $\mathrm{Adv}_{G1}^{aPre}(c') = 1$: Consider the following simple adversary $A = (A_1, A_2)$ playing aPre game against $G1$. A_1 chooses the key as $K = K^*$, and A_2 after receiving the hash value $Y = G1_{K^*}(M) = C^*$, outputs any arbitrary message $M' \in \{0,1\}^m$. Adversary $A = (A_1, A_2)$ always wins the aPre game because, according to the construction of $G1$, we have $G1_{K^*}(M') = C^*$ for any $M^* \in \{0,1\}^m$.

To complete the proof, we show that $G1$ inherits the eTCR property of H by demonstrating that $\epsilon' \leq \epsilon + \sqrt{\epsilon} + 2^{-k+1}$.

Let $A = (A_1, A_2)$ be any adversary that can win eTCR game against $G1$ with success probability ϵ' and having time complexity at most t'. Consider the following adversary $B = (B_1, B_2)$ against eTCR property of H which uses A as a subroutine (and forwards whatever it outputs):

Algorithm $B_1()$	**Algorithm** $B_2(K, M, State)$
10: $(M, State) \xleftarrow{\$} A_1()$;	30: if $K = K^* \vee H_K(M) = C^*$ then $bad \leftarrow true$;
20: **return** $(M, State)$;	40: $(K', M') \xleftarrow{\$} A_2(K, M, State)$;
	50: **return** (K', M');

To make the proof easier to follow, we used a boolean variable "*bad*" as a flag whose initial value is assumed to be '*false*'. This flag is set by B when an *undesirable* event happens that could make B unsuccessful even if A was successful.(This is shown by line 30, which is dummy otherwise, and can be omitted from the code of B without affecting its operation.)

Let **Bad** be the event that in the eTCR game $K = K^* \vee H_K(M) = C^*$; *i.e.* in line 30 the flag *bad* is set to *true*. We show that if **Bad** does not happen then B will succeed in eTCR attack against H whenever A succeeds in eTCR attack against $G1$. Note that A succeeds in eTCR attack against G whenever $(K, M) \neq (K', M')$ and $G_K(M) = G_{K'}(M')$. Assuming that the event **Bad** does not happen; that is, $K \neq K^* \wedge H_K(M) \neq C^*$, and referring to the construction of $G1$, it can be seen that in this case from $G1_K(M) = G1_{K'}(M')$ we get that $H_K(M) = H_{K'}(M')$; that is, B also succeeds in eTCR attack against H. Hence, we have: $\epsilon \geq \Pr[B \text{ succeeds}] = \Pr[A \text{ succeeds} \wedge \overline{\mathbf{Bad}}] \geq \Pr[A \text{ succeeds}] - \Pr[\mathbf{Bad}] = \epsilon' - \Pr[\mathbf{Bad}]$. Rearranging the terms we have:

$$\epsilon' \leq \epsilon + \Pr[\mathbf{Bad}] \tag{1}$$

Now we need to upperbound $\Pr[\mathbf{Bad}] = \Pr[K = K^* \vee H_K(M) = C^*]$. Using the union bound we have:

$$\Pr[\mathbf{Bad}] \leq \Pr[K = K^*] + \Pr[H_K(M) = C^*] = 2^{-k} + \Pr[H_K(M) = C^*] \tag{2}$$

It remains to upperbound $p = \Pr[H_K(M) = C^*]$. We claim that:

Claim. $p = \Pr[H_K(M) = C^*] \leq 2^{-k} + \sqrt{\epsilon}$.

Before continuing to prove this claim, note that the inequalities (1), (2) and the above claim gives the target upper-bound as $\epsilon' \leq \epsilon + \sqrt{\epsilon} + 2^{-k+1}$. Clearly, (ignoring the time for the dummy operation in line 30 of B) the time complexity of B is that of A plus a small constant time c, *i.e.* $t = t' + c$.

Proof of the Claim: The first and main step is to express our problem in a format which can be considered as an special case of the Reset Lemma of [4], and then we can apply the probabilistic analysis of the Reset Lemma.

Referring to the description of B, it can be seen that p equals to the probability that the following experiment returns 1; where, the probability is taken over the randomness used by A_1 and the random selection of the first key K:

Experiment I
$(M, State) \xleftarrow{\$} A_1()$;
$K \xleftarrow{\$} \{0,1\}^k$
If $H_K(M) = C^*$ then **return 1** else **return 0**;

Let $R \in \{0,1\}^r$ denote the random tape used by the (randomized) algorithm A_1. Let $\text{Verify}(M,K,C)$ be a predicate which is defined as follows:

$$\text{Verify}(M,K,C) = \begin{cases} 1 & \text{if } H_K(M) = C \\ 0 & \text{otherwise} \end{cases}$$

Now, we can rewrite Experiment I as below, where $\varnothing$ denotes an 'empty string':

Experiment I
$R \xleftarrow{\$} \{0,1\}^r$; $(M, State) = A_1(\varnothing; R)$;
$K \xleftarrow{\$} \{0,1\}^k$; $d = \text{Verify}(M,K,C^*)$;
Return d

Let q be the probability that the following (reset) experiment returns 1:

Experiment II (*Reset Experiment*)
$R \xleftarrow{\$} \{0,1\}^r$; $(M, State) = A_1(\varnothing; R)$;
$K_1 \xleftarrow{\$} \{0,1\}^k$; $d_1 = \text{Verify}(M,K_1,C^*)$;
$K_2 \xleftarrow{\$} \{0,1\}^k$; $d_2 = \text{Verify}(M,K_2,C^*)$;
If $(d_1 = 1 \ \wedge \ d_2 = 1 \ \wedge \ K_1 \neq K_2)$ then **return 1** else **return 0**

Proposition 1. $p \leq \sqrt{q} + 2^{-k}$.

The proof of this proposition can be deduced as a special case of that of the Reset Lemma in [4]. We provide the proof here for completeness. For any $R \in \{0,1\}^r$, let M_R denote the target message output by A using the random tape R; that is, $(M_R, State_R) = A(\varnothing; R)$. Define two functions $X : \{0,1\}^r \to [0,1]$ and $Y : \{0,1\}^r \to [0,1]$ as follows:

$$X(R) \triangleq \Pr[\text{Verify}(M_R, K, C^*) = 1] \tag{3}$$

where the probability is taken over random selection of K from the key space $\{0,1\}^k$, and

$$Y(R) \triangleq \Pr[\text{Verify}(M_R, K_1, C^*) = 1 \ \wedge \ \text{Verify}(M_R, K_2, C^*) = 1 \ \wedge \ K_1 \neq K_2] \tag{4}$$

where the probability is taken over random and *independent* selection of K_1 and K_2 from the key space $\{0,1\}^k$. By a simple argument, noting that K_1 and K_2 are chosen independently and using the fact that $\Pr(E \ \wedge \ \overline{F}) \geq \Pr(E) - \Pr(F)$ for any two events E and F, we have:

$$Y(R) = \Pr[\text{Verify}(M_R, K_1, C^*) = 1] \,.\, \Pr[\text{Verify}(M_R, K_2, C^*) = 1 \ \wedge \ K_1 \neq K_2] \geq X(R)[X(R) - 2^{-k}] \tag{5}$$

We can view functions X and Y as random variables over sample space $\{0,1\}^r$ of random tape (coins) used by probabilistic algorithm A. Now, note that the probabilities that Experiment I and Experiment II return 1 are, respectively, the expected values of the random variables X and Y with respect to R, i.e. $p = \mathbf{E}[X]$ and $q = \mathbf{E}[Y]$. Using the inequality (5) and letting $c = 2^{-k}$ we have:

$$q = \mathbf{E}[Y] \geq \mathbf{E}[X(X-c)] = \mathbf{E}[X^2] - c\mathbf{E}[X] \geq \mathbf{E}[X]^2 - c\mathbf{E}[X] = p^2 - cp$$

Using the above relation we have:

$$(p - \frac{c}{2})^2 = p^2 - cp + \frac{c^2}{4} \leq q + \frac{c^2}{4}$$

and using the fact that $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ for $a, b \geq 0$ we have:

$$p - \frac{c}{2} \leq \sqrt{q} + \frac{c}{2}$$

Remembering that $c = 2^{-k}$, we get the final result as $p \leq \sqrt{q} + 2^{-k}$.

To complete the proof of the Claim, we show that $q \leq \epsilon$. We construct an adversary $C = (C_1, C_2)$ against eTCR property of H; such that $\mathrm{Adv}_H^{eTCR}(C) = q$ as follows: C_1 runs A_1 (by providing its random tape R as shown in Experiment II) and forwards M as its output in the first stage of the eTCR attack against H. C_2, on receiving the random key $K1$, simply chooses another random key $K2$ and returns $(K2, M)$ to finish the eTCR attack. Clearly the advantage of C in the eTCR game will be the same as the probability that Experiment II returns 1. Note that Experiment II returns 1 if $\mathrm{Verify}(M, K_1, C^*) = 1 \wedge \mathrm{Verify}(M, K_2, C^*) = 1 \wedge K_1 \neq K_2$, and from the definition of the predicate $\mathrm{Verify}(.,.,.)$ this implies that $H(K_1, M) = H(K_2, M) = C^* \wedge K_1 \neq K_2$. Hence, whenever Experiment II returns 1 we have $(K_1, M) \neq (K_2, M)$ and $H(K_1, M) = H(K_2, M)$, *i.e.* A succeeds in the eTCR attack game against H. □

Lemma 4 (eTCR $\nrightarrow$ ePre). *Assume that there exists a dedicated-key hash function $H : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^n$, where $m \geq k$, which is $(t, \epsilon) - eTCR$. Select (and* fix*) an arbitrary hash value $C^* \in \{0,1\}^n$ (e.g. $C^* = 0^n$). The dedicated-key hash function $G2 : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^n$ shown in this lemma is $(t', \epsilon') - eTCR$, where $t' = t - c$ (where c is a constant) and $\epsilon' \leq \epsilon + \sqrt{\epsilon} + 2^{-k+1}$, but it is completely insecure in ePre sense.*

$$G2_K(M) = \begin{cases} C^* & \text{if } val(K) = val(M) \\ \\ H_K(M) & \text{otherwise} \end{cases}$$

Proof. First we show that $G2$ is completely insecure in ePre sense, by using the following simple adversary $A = (A_1, A_2)$. A_1 puts the target hash value as $Y = C^*$. A_2, after receiving the random key K, outputs a message $M \in \{0,1\}^m$ whose decimal value equals to that of K, i.e. $M = \langle val(K) \rangle_m$. Clearly, the adversary $A = (A_1, A_2)$ always wins the ePre game against $G2$, as we have $G2_K(M) = C^* = Y$ for such a message M and K which satisfy the condition $val(K) = val(M)$. To complete the proof, we show that $G2$ inherits the eTCR property of H by demonstrating that $\epsilon' \leq \epsilon + \sqrt{\epsilon} + 2^{-k+1}$.

The proof for this part is quite similar to that of Lemma 3 and is briefly provided below for completeness.

Let $A = (A_1, A_2)$ be any adversary that can win eTCR game against $G2$ with success probability ϵ' and having time complexity at most t'. Consider the following adversary $B = (B_1, B_2)$ against eTCR property of H which uses A as its subroutine (and simply forwards whatever it outputs):

Algorithm $B_1()$	**Algorithm** $B_2(K, M, State)$
10: $(M, State) \xleftarrow{\$} A_1()$;	30: if $val(K) = val(M) \vee H_K(M) = C^*$ then $bad \leftarrow true$;
20: **return** $(M, State)$;	40: $(M', K') \xleftarrow{\$} A_2(K, M, State)$;
	50: **return** (M', K');

Let **Bad** be the event that in the eTCR game $val(K) = val(M) \vee H_K(M) = C^*$; *i.e.* in line 30, the flag *bad* is set to *true*.

We show that if **Bad** does not happen then B will succeed in eTCR attack against H whenever A succeeds in eTCR attack against $G2$. Note that A succeeds in the eTCR attack against $G2$ whenever $(M, K) \neq (M', K')$ and $G2_K(M) = G2_{K'}(M')$. Assuming that the event **Bad** does not happen; *i.e.* $val(K) \neq val(M) \wedge H_K(M) \neq C^*$, and referring to the construction of $G2$, it can be seen that in this case from $G2_K(M) = G2_{K'}(M')$ we get that $H_K(M) = H_{K'}(M')$; that is, B also succeeds in its eTCR attack against H. Hence, we have: $\epsilon \geq \Pr[B \text{ succeeds}] = \Pr[A \text{ succeeds} \wedge \overline{\mathbf{Bad}}] \geq \Pr[A \text{ succeeds}] - \Pr[\mathbf{Bad}] = \epsilon' - \Pr[\mathbf{Bad}]$. Rearranging the terms we have:

$$\epsilon' \leq \epsilon + \Pr[\mathbf{Bad}] \tag{6}$$

Now we need to upper bound $\Pr[\mathbf{Bad}] = \Pr[val(K) = val(M) \vee H_K(M) = C^*]$. Using the union bound we have:

$$\Pr[\mathbf{Bad}] \leq \Pr[val(K) = val(M)] + \Pr[H_K(M) = C^*] \leq 2^{-k} + \Pr[H_K(M) = C^*] \tag{7}$$

By the same probabilistic argument as detailed in Lemma 3, we get the bound $\Pr[H_K(M) = C^*] \leq 2^{-k} + \sqrt{\epsilon}$, and using the inequalities (6) and (7) we get the targte upper bound as $\epsilon' \leq \epsilon + \sqrt{\epsilon} + 2^{-k+1}$. □

5 Domain Extension and eTCR Property Preservation

In this section we investigate the eTCR preserving capability of eight domain extension transforms, namely Plain MD [15, 8], Strengthened MD [15, 8], Prefix-free MD [7, 14], Randomized Hashing [12], Shoup [25], Enveloped Shoup [2], XOR Linear Hash (XLH)[5], and Linear Hash (LH) [5] methods.

Assume that we have a compression function $h : \{0,1\}^k \times \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ that can only hash messages of fixed length $(n+b)$ bits. A domain extension transform can use this compression function (as a black-box) to construct a hash function $H : \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^n$, where the message space $\mathcal{M}$ can be either $\{0,1\}^*$ or $\{0,1\}^{<2^m}$, for some positive integer m (e.g. $m = 64$). The key space $\mathcal{K}$ is determined by the construction of a domain extender. Clearly $log_2(|\mathcal{K}|) \geq k$, as H involves at least one invocation of h. The difference between $log_2(|\mathcal{K}|)$ (i.e. the key length of H) and k (i.e. the key length of h) is called the 'key expansion' of domain extension transform and is a measure of its efficiency: the less key expansion is, the more efficient the domain extension transform will be.

A domain extension transform comprises of two functions: an injective padding function Pad and an iteration function f_I. First, the padding function $Pad : \mathcal{M} \rightarrow D_I$ is applied to an input message $M \in \mathcal{M}$ to convert it to the padded message $Pad(M)$ in a domain D_I. Then, the iteration function $f_I : \mathcal{K} \times D_I \rightarrow \{0,1\}^n$ uses the compression function h as many times as required, and outputs the final hash value. The full-fledged hash function H is obtained by combining the two functions. It is known that the property preserving capability of a domain extension transform depends on both the padding function and iteration function, for example 'Plain MD' (*i.e.*, plain padding and MD iteration) is not CR preserving domain extender, but 'Strengthened MD' (*i.e.*, strengthening padding and MD iteration) does preserve CR [15, 8, 2]. Hence, precisely speaking, we can have several domain extenders using the same iteration function but with different padding function, e.g. Plain MD, Strengthened MD, Prefix-free MD, which are considered as three different domain extenders that have different capabilities from property preserving viewpoint [2].

The padding functions used in the eight domain extension transforms that we consider in this paper are defined as follows:

- **Plain**: $pad : \{0,1\}^* \rightarrow \bigcup_{L \geq 1} \{0,1\}^{Lb}$, where $pad(M) = M||10^p$ and p is the minimum number of 0's required to make the length of $pad(M)$ a multiple of block length.
- **Strengthening**: $pad_s : \{0,1\}^{<2^m} \rightarrow \bigcup_{L \geq 1} \{0,1\}^{Lb}$, where $pad_s(M) = M||10^p||\langle |M| \rangle_m$ and p is the minimum number of 0's required to make the length of $pad_s(M)$ a multiple of block length.
- **Prefix-free**: $padPF : \{0,1\}^* \rightarrow \bigcup_{L \geq 1} \{0,1\}^{Lb}$, where $padPF$ transforms the input message space $\{0,1\}^*$ to a prefix-free message space, *i.e.* $padPF(M)$ is not a prefix of $padPF(M')$ for any two *distinct* messages M and M'. An example of a Prefix-free padding function, which we consider in this paper, is as follows. Append 10^p to the message where p is the minimum number of 0's required to make the length of the resulted message a multiple of $b-1$ bits. Parse the resulted message into blocks of $b-1$ bits and prepend a '0' to all blocks but the final block where a '1' must be prepended.

– **Strengthened Chain Shift**: $padCS_s : \{0,1\}^{<2^m} \rightarrow \bigcup_{L\geq 1} \{0,1\}^{Lb+b-n}$, where $padCS_s(M) = M||10^r||\langle |M| \rangle_m ||0^p$, and parameters p and r are defined in two ways depending on the block length b. If $b \geq n+m$ then $p = 0$, otherwise $p = b - n$. Then r is the minimum number of 0's required to make the padded message a member of $\{0,1\}^{Lb+b-n}$, for some positive integer L.

The iteration functions for MD, Randomized Hashing, Shoup, Enveloped Shoup, XLH and LH are shown in Fig. 4.

5.1 Merkle-Damgård Does not Preserve eTCR

MD iteration function as shown in Fig. 4 can be used together with Plain (pad), Strengthening(pad_s), or Prefix-free($padPF$) padding function to construct a domain extension transform, which is called Plain MD, Strengthened MD, or Prefix-free MD, respectively. In this section we show that none of these three domain extension transforms can be used as an eTCR preserving domain extender.

Theorem 4 (Negative Result). *Plain MD, Strengthened MD, and Prefix-free MD do not preserve eTCR.*

Proof. We borrow the construction of the following counterexample from [5] where it was used in the context of TCR property. Assume that there is a dedicated-key compression function $g : \{0,1\}^k \times \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ with $b > k$ which is (t, ϵ)-eTCR secure. Set $b = k + b'$ where $b' > 0$ by the assumption that $b > k$. Consider the following dedicated-key compression function $h : \{0,1\}^k \times \{0,1\}^{(n+k)+b'} \rightarrow \{0,1\}^{n+k}$:

$$h(K, X||Y||Z) = h_K(X||Y||Z) = \begin{cases} g_K(X||Y||Z)||K & \text{if } K \neq Y \\ 1^{n+k} & \text{if } K = Y \end{cases}$$

where $K \in \{0,1\}^k, X \in \{0,1\}^n, Y \in \{0,1\}^k, Z \in \{0,1\}^{b'}$ ($n + k$ is chaining variable length and b' is block length for h).

To complete the proof, we first show in Lemma 5 that h_K inherits the eTCR property from g_K. Note that this cannot be directly inferred from the proof in [5] that h_K inherits the weaker notion TCR from g_K. Then, we show a simple attack in each case to show that the hash function obtained via either of Plain, Strengthened, or Prefix-free MD transform by extending domain of h_K is completely insecure in eTCR sense.

Lemma 5. *The dedicated-key compression function h is (t', ϵ')-eTCR secure, where $\epsilon' = \epsilon + 2^{-k+1} \approx \epsilon$ and $t' = t - c$, for a small constant c.*

Proof. Let $A = (A_1, A_2)$ be an adversary which wins the eTCR game against h_K with probability ϵ' and using time complexity t'. We construct an adversary $B = (B_1, B_2)$ which uses A as a subroutine and wins eTCR game against g_K with success probability of at least $\epsilon = \epsilon' - 2^{-k+1} (\approx \epsilon'$, for large $k)$ and spending time complexity $t = t' + c$ where small constant c can be determined from the description of algorithm B. Algorithm B is as follows:

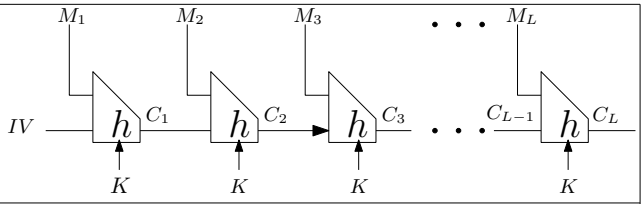
Algorithm $B_1()$
$(M_1 = X_1||Y_1||Z_1, State) \xleftarrow{\$} A_1()$;
return $(M_1, State)$;

Algorithm $B_2(K_1, M_1, State)$
Parse M_1 as $M_1 = X_1||Y_1||Z_1$
if $\left[K_1 = Y_1 \vee K_1 = 1^k\right]$ **return 'Fail'**;
$(M_2 = X_2||Y_2||Z_2, K_2) \xleftarrow{\$} A_2(K_1, M_1, State)$;
return (M_2, K_2);

At the first stage of eTCR attack, B_1 just merely runs A_1 and returns whatever it returns as the first message (*i.e.* $M_1 = X_1||Y_1||Z_1$) and any possible state information to be passed to the second stage algorithm. At the second stage of the attack, let **Bad** be the event that $[K_1 = Y_1 \vee K_1 = 1^k]$. If **Bad** happens

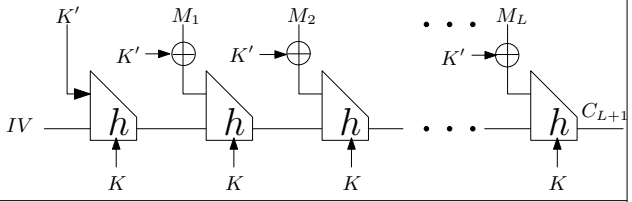
$MD^h_{IV} : \mathcal{K} \times \{0,1\}^{Lb} \rightarrow \{0,1\}^n$, where $\mathcal{K} = \{0,1\}^k$

Algorithm $MD^h_{IV}(K, M)$**:**
$C_0 = IV$
for $i = 1$ to L do
  $C_i = h_K(C_{i-1}||M_i)$
return C_L



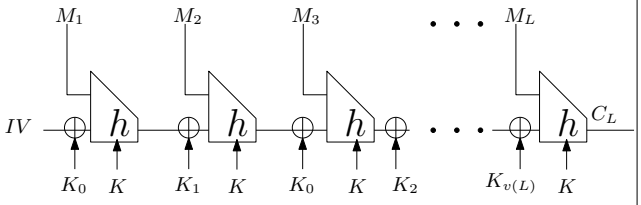
$RH^h_{IV} : \mathcal{K} \times \{0,1\}^{Lb} \rightarrow \{0,1\}^n$, where $\mathcal{K} = \{0,1\}^{k+b}$

Algorithm $RH^h_{IV}(K||K', M)$**:**
$C_0 = IV$
$C_1 = h_K(C_0||K')$
for $i = 2$ to $L + 1$ do
  $C_i = h_K(C_{i-1}||(M_{i-1} \oplus K'))$
return C_{L+1}



$Sh^h_{IV} : \mathcal{K} \times \{0,1\}^{Lb} \rightarrow \{0,1\}^n$, where $\mathcal{K} = \{0,1\}^{k+tn}$
$t = \lceil log_2(L) \rceil, \nu(i) = max\{x : 2^x | i\}$

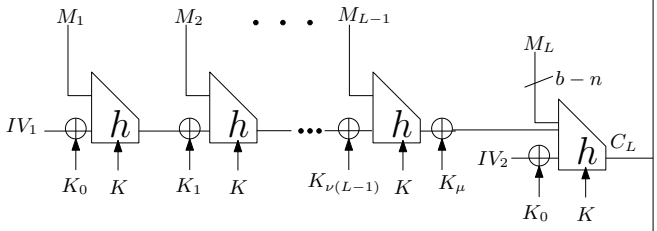
Algorithm $Sh^h_{IV}(K||K_0||K_1||\cdots||K_{t-1}, M)$**:**
$C_0 = IV$
for $i = 1$ to L do
  $C_i = h_K((C_{i-1} \oplus K_{\nu(i)})||M_i)$
return C_L



$ESh^h_{IV_1,IV_2} : \mathcal{K} \times \{0,1\}^{(L-1)b+b-n} \rightarrow \{0,1\}^n$, where $\mathcal{K} = \{0,1\}^{k+tn}$
$t = \lceil log_2(L-1) \rceil + 1, \nu(i) = max\{x : 2^x | i\}$

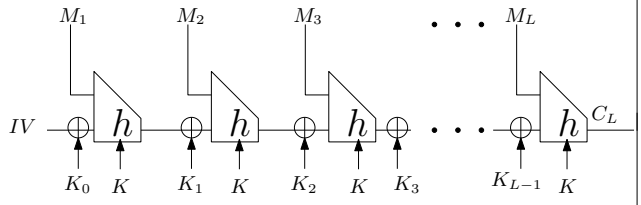
Algorithm $ESh^h_{IV_1,IV_2}(K||K_0||K_1||\cdots||K_{t-1}, M)$**:**
$C_0 = IV_1$; $K_\mu = K_{t-1}$
for $i = 1$ to $L - 1$ do
  $C_i = h_K((C_{i-1} \oplus K_{\nu(i)})||M_i)$

return $h_K((IV_2 \oplus K_0)||(C_{L-1} \oplus K_\mu)||M_L)$



$XLH^h_{IV} : \mathcal{K} \times \{0,1\}^{Lb} \rightarrow \{0,1\}^n$, where $\mathcal{K} = \{0,1\}^{k+Ln}$

Algorithm $XLH^h_{IV}(K||K_0||K_1||\cdots||K_{L-1}, M)$**:**
$C_0 = IV$
for $i = 1$ to L do
  $C_i = h_K((C_{i-1} \oplus K_{i-1})||M_i)$
return C_L



$LH^h_{IV} : \mathcal{K} \times \{0,1\}^{Lb} \rightarrow \{0,1\}^n$, where $\mathcal{K} = \{0,1\}^{Lk}$

Algorithm $LH^h_{IV}(K_1||K_2||\cdots||K_L, M)$**:**
$C_0 = IV$
for $i = 1$ to L do
  $C_i = h_{K_i}(C_{i-1}||M_i)$
return C_L

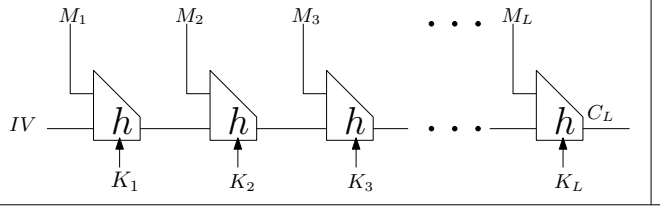


Fig. 4. Iteration functions used in domain extension transforms: Merkle-Damgård (MD), Randomized Hashing (RH), Shoup (Sh), Enveloped Shoup (ESh), XLH and LH. The iteration functions are ordered top-down based on their efficiency in terms of key expansion, MD iteration does not expand the key length of underlying compression function and is the most efficient transform and LH is the least efficient transform.

then algorithm B_2 (and hence B) will fail in eTCR attack; otherwise (*i.e.* if $\overline{\mathbf{Bad}}$ happens) we show that B will be successful in eTCR attack against g whenever A succeeds in eTCR attack against h.

Assume that the event $\overline{\mathbf{Bad}}$ happens; that is, $[K_1 \neq Y_1 \wedge K_1 \neq 1^k]$. We claim that in this case if A succeeds then B also succeeds. Referring to the construction of (counterexample) compression function h in this lemma, it can be seen that if A succeeds, *i.e.*, whenever $(M_1, K_1) \neq (M_2, K_2) \wedge h_{K_1}(M_1) = h_{K_2}(M_2)$, it must be the case that $g_{K_1}(M_1)||K_1 = g_{K_2}(M_2)||K_2$ which implies that $g_{K_1}(M_1) = g_{K_2}(M_2)$ (and also $K_1 = K_2$). That is, (M_1, K_1) and (M_2, K_2) are also valid a colliding pair for the eTCR attack against g. (Remember that $M_1 = X_1||Y_1||Z_1$ and $M_2 = X_2||Y_2||Z_2$.)

Now note that $\Pr[\mathbf{Bad}] \leq \Pr[K_1 = Y_1] + \Pr[K_1 = 1^k] = 2^{-k} + 2^{-k} = 2^{-k+1}$, as K_1 is selected uniformly at random just after the message M_1 is fixed in the eTCR game. Therefore, we have $\epsilon = \Pr[B \text{ succeeds}] = \Pr[A \text{ succeeds} \wedge \overline{\mathbf{Bad}}] \geq \Pr[A \text{ succeeds}] - \Pr[\mathbf{Bad}] \geq \epsilon' - 2^{-k+1}$.

To complete the proof of Theorem 4, we need to show that MD transforms cannot preserve eTCR while extending the domain of this specific compression function h_K. For this part, the same attacks that used in [5, 2] against TCR property also work for our purpose here as clearly breaking TCR implies breaking its strengthened variant eTCR. The eTCR attacks are as follows:

The Case of Plain MD and Strengthened MD:
Let's denote Plain MD and Strengthened MD domain extension transforms applied on the counterexample compression function h and using an initial value IV, respectively, by $\mathbf{pMD}^h_{IV}$ and $\mathbf{sMD}^h_{IV}$. Note that MD^h_{IV} is used to denote the MD *iteration* function (Fig. 4). Then the full-fledged hash function $H : \{0,1\}^k \times \mathcal{M} \rightarrow \{0,1\}^{n+k}$ will be defined as $H(K, M) = \mathbf{pMD}^h_{IV}(K, M) = MD^h_{IV}(K, pad(M))$ and $H(K, M) = \mathbf{sMD}^h_{IV}(K, M) = MD^h_{IV}(K, pad_s(M))$, for Plain and Strengthened MD case, respectively.

The following adversary $A = (A_1, A_2)$ can break H in eTCR sense for both Plain MD and Strengthened MD cases. A_1 outputs $M_1 = 0^{b'}||0^{b'}$ and A_2, on receiving the first key K, outputs a different message as $M_2 = 1^{b'}||0^{b'}$ together with the same key K as the second key. Considering that the initial value $IV = IV_1||IV_2 \in \{0,1\}^{n+k}$ is fixed before adversary starts the attack game and K is chosen at random afterward in the second stage of the game, we have $\Pr[K = IV_2] = 2^{-k}$. If $K \neq IV_2$ which is the case with probability $1 - 2^{-k}$ then adversary becomes successful as we have:

$$MD^h_{IV}(K, 0^{b'}||0^{b'}) = h_K(h_K(IV_1||IV_2||0^{b'})||0^{b'}) = h_K(g_K(IV_1||IV_2||0^{b'})||K||0^{b'}) = 1^{n+k}$$
$$MD^h_{IV}(K, 1^{b'}||0^{b'}) = h_K(h_K(IV_1||IV_2||1^{b'})||0^{b'}) = h_K(g_K(IV_1||IV_2||1^{b'})||K||0^{b'}) = 1^{n+k}$$

$$\mathbf{pMD}: \begin{cases} H(K, 0^{b'}||0^{b'}) = MD^h_{IV}(K, pad(0^{b'}||0^{b'})) = h_K(MD^h_{IV}(K, 0^{b'}||0^{b'})||10^{b'-1}) = h_K(1^{n+k}||10^{b'-1}) \\ H(K, 1^{b'}||0^{b'}) = MD^h_{IV}(K, pad(1^{b'}||0^{b'})) = h_K(MD^h_{IV}(K, 1^{b'}||0^{b'})||10^{b'-1}) = h_K(1^{n+k}||10^{b'-1}) \end{cases}$$

$$\mathbf{sMD}: \begin{cases} MD^h_{IV}(K, pad_s(0^{b'}||0^{b'})) = h_K(MD^h_{IV}(K, 0^{b'}||0^{b'})||10^{b'-m-1}||\langle 2b' \rangle_m) \\ \qquad\qquad = h_K(1^{n+k}||10^{b'-m-1}||\langle 2b' \rangle_m) \\ MD^h_{IV}(K, pad_s(1^{b'}||0^{b'})) = h_K(MD^h_{IV}(K, 1^{b'}||0^{b'})||10^{b'-m-1}||\langle 2b' \rangle_m) \\ \qquad\qquad = h_K(1^{n+k}||10^{b'-m-1}||\langle 2b' \rangle_m) \end{cases}$$

The Case of Prefix-free MD: Denote Prefix-free MD domain extension transform by **preMD**. The full-fledged hash function $H : \{0,1\}^k \times \mathcal{M} \rightarrow \{0,1\}^{n+k}$ will be defined as $H(K, M) = \mathbf{preMD}^h_{IV}(K, M) = MD^h_{IV}(K, padPF(M))$. Note that we have $\mathcal{M} = \{0,1\}^*$ due to the application of $padPF$ function. The following adversary $A = (A_1, A_2)$ which is used for TCR attack against Prefix-free MD in [2], can also break

H in eTCR sense, as clearly any TCR attacker against H is an eTCR attacker as well. Here, we provide the description of the attack for eTCR, for completeness. A_1 outputs $M_1 = 0^{b'-1}||0^{b'-2}$ and A_2 on receiving the first key K outputs a different message as $M_2 = 1^{b'-1}||0^{b'-2}$ together with the same key K as the second key. Considering that the initial value $IV = IV_1||IV_2 \in \{0,1\}^{n+k}$ is fixed before the adversary starts the attack game and K is chosen at random afterward, we have $\Pr[K = IV_2] = 2^{-k}$. If $K \neq IV_2$ which is the case with probability $1 - 2^{-k}$ then the adversary becomes successful as we have:

$$\begin{aligned} H(K, 0^{b'-1}||0^{b'-2}) &= MD^h_{IV}(K, padPF(0^{b'-1}||0^{b'-2})) = MD^h_{IV}(K, 0^{b'}||10^{b'-2}1) \\ &= h_K(h_K(IV_1||IV_2||0^{b'})||10^{b'-2}1) = h_K(g_K(IV_1||IV_2||0^{b'})||K||10^{b'-2}1) = 1^{n+k}. \end{aligned}$$

$$\begin{aligned} H(K, 1^{b'-1}||0^{b'-2}) &= MD^h_{IV}(K, padPF(1^{b'-1}||0^{b'-2})) = MD^h_{IV}(K, 01^{b'-1}||10^{b'-2}1) \\ &= h_K(h_K(IV_1||IV_2||01^{b'-1})||10^{b'-2}1) = h_K(g_K(IV_1||IV_2||01^{b'-1})||K||10^{b'-2}1) = 1^{n+k}. \end{aligned}$$

5.2 Randomized Hashing Does not Preserve eTCR

Our aim in this section is to show that Randomized Hashing (RH) construction, *if considered as a domain extension for a dedicated-key compression function*, does not preserve eTCR property. Note that (this dedicated-key variant of) RH method as shown in Fig. 4 expands the key length of the underlying compression function by only a constant additive factor of b bits, that is $log_2(|\mathcal{K}|) = k + b$ which is independent from input message length. That is, after MD transfrom, RH is the most efficient method from key expansion point of view. The latter characteristic, *i.e.* a small and message-length-independent key expansion could have been considered a stunning advantage from efficiency viewpoint, if RH had been able to preserve eTCR. Nevertheless, unfortunately we shall show that randomized hashing does not preserve eTCR.

Following the specification of the original scheme for Randomized Hashing in [12], we assume that the padding function is the strengthening padding pad_s and so we use the same name for domain extension as its iteration function, *i.e.* RH^h_{IV} (Fig. 4). The full-fledged hash function $H : \{0,1\}^k \times \mathcal{M} \rightarrow \{0,1\}^{n+k}$ will be defined as $H(K||K', M) = RH^h_{IV}(K||K', pad_s(M))$. Note that we have $\mathcal{M} = \{0,1\}^{<2^m}$ due to the application of pad_s function.

Theorem 5 (Negative Result). *The Randomized Hashing transform does not preserve eTCR.*

Proof. We need to show as a counterexample, a dedicated-key compression function h which is eTCR but for which the dedicated-key hash function H obtained via Randomized Hashing method is completely insecure in eTCR sense. The same counterexample used in Theorem 4 can also be used to show that Randomized Hashing transform (in dedicated-key hash function setting) does not preserve eTCR property.

As we have previously shown in Lemma 5 that the constructed h_K inherits the eTCR property of g_K, it just remains to show that RH^h_{IV} cannot extend the domain of h_K while preserving its eTCR property. Consider an adversary $A = (A_1, A_2)$ that plays the eTCR game against the hash function H, obtained via Randomized Hashing, as follows. A_1 outputs a one-block long target message $M_1 = 0^{b'}$ (note that for the counterexample compression function h_K, b' is the block length and $n + k$ is the chaining variable length). A_2 on getting the first key $K||K'$ for H (in the second stage of eTCR attack), outputs the second message as $M_2 = 1^{b'}$ and puts the second key the same as the first key. As $M_2 \neq M_1$, we just need to show that these two messages collide under the same key, *i.e.* $K||K'$. Considering that the initial value $IV = IV_1||IV_2 \in \{0,1\}^{n+k}$ for RH^h_{IV} is (selected and) fixed before the adversary starts the attack game and $K||K'$ is chosen at random latter in the second stage of the game, we have $\Pr[K = IV_2] = 2^{-k}$. If $K \neq IV_2$ (which is the case with probability $1 - 2^{-k}$) then the adversary $A = (A_1, A_2)$ becomes successful as we have:

$$
\begin{aligned}
&RH^h_{IV}(K||K', pad_s(0^{b'})) = RH^h_{IV}(K||K', 0^{b'}||10^{b'-1-m}\langle b'\rangle_m)\\
&= h_K\Big(h_K\big(h_K(IV_1||IV_2||K')||(K'\oplus 0^{b'})\big)||(K'\oplus 10^{b'-1-m}\langle b'\rangle_m)\Big)\\
&= h_K\Big(h_K\big(g_K(IV_1||IV_2||K')||K||(K'\oplus 0^{b'})\big)||(K'\oplus 10^{b'-1-m}\langle b'\rangle_m)\Big)\\
&= h_K(1^{n+k}||(K'\oplus 10^{b'-1-m}\langle b'\rangle_m)).
\end{aligned}
$$

$$
\begin{aligned}
&RH^h_{IV}(K||K', pad_s(1^{b'})) = RH^h_{IV}(K||K', 1^{b'}||10^{b'-1-m}\langle b'\rangle_m)\\
&= h_K\Big(h_K\big(h_K(IV_1||IV_2||K')||(K'\oplus 1^{b'})\big)||(K'\oplus 10^{b'-1-m}\langle b'\rangle_m)\Big)\\
&= h_K\Big(h_K\big(g_K(IV_1||IV_2||K')||K||(K'\oplus 1^{b'})\big)||(K'\oplus 10^{b'-1-m}\langle b'\rangle_m)\Big)\\
&= h_K(1^{n+k}||(K'\oplus 10^{b'-1-m}\langle b'\rangle_m)).
\end{aligned}
$$

□

5.3 Shoup, Enveloped Shoup and XLH Do not Preserve eTCR

In previous subsections, we have shown that neither MD nor RH are eTCR preserving transforms. The next three most efficient candidates from key expansion viewpoint that we consider are Shoup (Sh), Enveloped Shoup (ESh) and XLH transforms. In Sh and ESh transforms the key expansion depends logarithmically on the input message length. For Sh iteration $log_2(|\mathcal{K}|) = k + \lceil log_2(L)\rceil\, n$ and for ESh iteration $log_2(|\mathcal{K}|) = k+(\lceil log_2(L-1)\rceil+1)n$, where L is the length of *the padded message* in blocks which is input to the iteration function. (Note that Fig. 4 just shows the iteration function of the domain extensions and padding functions are not shown Fig. 4).

We assume the same padding functions as proposed in the original schemes, that is, for Shoup [25] and XLH [5] the strengthening padding function (pad_s) is used, and for Enveloped Shoup [2] the padding function is the strengthened chain shift padding ($padCS_s$). So, the full-fledged hash function $H : \{0,1\}^k \times \mathcal{M} \rightarrow \{0,1\}^{n+k}$, obtained via these three domain extension methods, will be defined accordingly as follows:

$$
\begin{aligned}
&\text{Sh: } H(K||K_0||\cdots||K_t, M) = Sh^h_{IV}(K||K_0||\cdots||K_{t-1}, pad_s(M)) && ;\ \text{where } t = \lceil log_2(L)\rceil\\
&\text{ESh: } H(K||K_0||\cdots||K_t, M) = ESh^h_{IV}(K||K_0||\cdots||K_{t-1}, padCS_s(M)) && ;\ \text{where } t = \lceil log_2(L-1)\rceil + 1\\
&\text{XLH: } H(K||K_0||\cdots||K_t, M) = XLH^h_{IV}(K||K_0||\cdots||K_{L-1}, pad_s(M))
\end{aligned}
$$

In the following theorem we show that none of Sh, ESh and XLH transforms can preserve eTCR. That is, we lose the best TCR preserving transform, *i.e.* Sh, as well as the multi-property preserving ESh transform when it comes to eTCR preservation.

Theorem 6 (Negative Results). *Sh, ESh, and XLH transforms do not preserve eTCR.*

Proof. The proof is quite simple but the results are stronger than previous counterexample based proofs, as here the negative results hold for any arbitrary compression function (irrespective of how secure the compression function h is), and not only for some specific counterexamples. That is these XOR masking based domain extension transforms are structurally insecure in eTCR sense. Intuitively, the inability if these domain extenders to preserve eTCR is due to the fact that they use XOR operation to add the key to the internal state (*i.e.* chaining variable), and hence an eTCR adversary will be able to cancel internal differences by taking advantage of its ability to select the value of the second key in the second stage of eTCR attack. For the formal proof, we provide the following simple attacks.

The Case of Shoup:
The following adversary $A = (A_1, A_2)$ can break the hash function H, obtained via Shoup domain extension transfrom (*i.e.* pad_s padding function followed by Sh^h_{IV} iteration method), in the eTCR sense. At the first

stage of the eTCR attack, A_1 outputs a two-block message $M = M_1||M_2$ as the target message which after applying pad_s will become a three-block message $M_1||M_2||(10^{b-1-m}\langle 2b\rangle_m)$ to be input to the three-round Sh^h_{IV} iteration. In the second stage of eTCR game, A_2, after receiving the first key as $K||K_0||K_1||K_0$ from the challenger, chooses the second two-block message as $M' = M'_1||M_2$ which after padding becomes $M'_1||M_2||(10^{b-1-m}\langle 2b\rangle_m)$. A_2 also puts the second key as $K||K_0||K'_1||K_0$, where the value of K'_1 is computed as $K'_1 = K_1 \oplus h_K\big((IV \oplus K_0)||M_1\big) \oplus h_K\big((IV \oplus K_0)||M'_1\big)$. It is easy to see (referring to Fig. 4) that this value for K' cancel the introduced difference in chaining variable which was created due to the different message blocks M_1 and M'_1. So, $(K||K_0||K_1, M)$ and $(K||K_0||K'_1, M')$ constitute a colliding pair for H in eTCR sense. (Note that the key sequence used for iteration function Sh^h_{IV} is $K||K_0||K_1||K_0$ because padded message $pad_s(M)$ has an extra third block containing the length information.)

The Case of Enveloped Shoup:
For the ESh transform the attack strategy is quite similar to Sh case. Adversary $A = (A_1, A_2)$ plays the eTCR game as follows. A_1 outputs two different $(L-1)$-block messages $M = M_1||\cdots||M_{L-1}$ and $M' = M'_1||\cdots||M'_{L-1}$ which after applying $padCS_s$ padding function will become $M_1||\cdots||M_{L-1}||(10^{b-1-m-n}||\langle(L-1)b\rangle_m)$ and $M'_1||\cdots||M'_{L-1}||(10^{b-1-m-n}||\langle(L-1)b\rangle_m)$, respectively. That is, the inputs to ESh iteration function will have the same last block as $M_L = M'_L = 10^{b-1-m-n}\langle|M|\rangle_m$, but their first $(L-1)$ blocks are different (note that in ESh the length of the last block which is used in the final envelop is $b-n$ bits). In the second stage of eTCR attack, A_2, on receiving the first key, puts all blocks of the second key the same as the first given key *except* the last key block K_μ. A_2 simply adjusts the value of this last key block to a new key block $K'_\mu = K_\mu \oplus C_{L-1} \oplus C'_{L-1}$ to cancel the introduced difference in the chaining variables C_{L-1} and C'_{L-1} (related to the computation for M and M', respectively). We stress that this adjustment of the value of K_μ to K'_μ to cancel the difference that appears in final chaining value is possible because "K_μ is only used for the chaining variable fed into the envelope " as stated in [2].

The Case of XLH:
The attack is similar to the case of Shoup. Consider an adversary $A = (A_1, A_2)$ that can break the hash function H, obtained via XLH domain extension transform (*i.e.* pad_s padding function followed by XLH^h_{IV} iteration method), in eTCR sense. A_1 outputs a two-block message $M = M_1||M_2$ as the target message which after applying pad_s will become a three-block message $M_1||M_2||(10^{b-1-m}\langle 2b\rangle_m)$ to be the input to the three-round XLH^h_{IV} iteration. In the second stage of eTCR game, A_2, on receiving the first key as $K||K_0||K_1||K_2$ from the challenger, chooses the second two-block message as $M' = M'_1||M_2$ which after padding becomes $M'_1||M_2||(10^{b-1-m}\langle 2b\rangle_m)$. A_2 then puts the second key as $K||K_0||K'_1||K_2$, where the value of K'_1 is computed as $K'_1 = K_1 \oplus h_K\big((IV \oplus K_0)||M_1\big) \oplus h_K\big((IV \oplus K_0)||M'_1\big)$. It is easy to see (referring to Fig. 4) that this value for K' cancel the introduced difference in chaining variable which was created due to the different message blocks M_1 and M'_1. Hence, $(K||K_0||K_1||K_2, M)$ and $(K||K_0||K'_1||K_2, M')$ constitute a colliding pair for H in eTCR sense. □

Remark. The eTCR adversaries used in the above proofs take advantage of XOR masking based structure of XLH, Sh and ESh transforms to cancel the effect of all accumulated differences in the internal state that may have been introduced by previous different message blocks, by simply adjusting the value of a last free key block. This implies that any class of such XOR masking based transforms that allows this cancellation phenomenon to happen will not be suitable for designing an eTCR preserving domain extender. It can be seen that this is the case for the XTH scheme of [5] as well.

5.4 LH Transform and its Nested Variant

Up to know we have shown that neither of MD, RH, Sh, or XLH transforms can preserve eTCR property. Henceforth, we have lost all efficient methods from key expansion viewpoint and now we have reached to the

same starting point for TCR preserving scenario as in [5], where it was shown that the LH method can be used to preserve TCR *only with respect to equal-length-collision finding adversaries* and its *nested variant* can be used to archive TCR for any variable-length-collision finding adversaries. We should mention that it was pointed out in [5] and latter shown by an explicit counterexample in [1] that LH iteration cannot preserve TCR with respect to variable length collisions.

After the previous series of negative results about inability of several efficient transforms to preserve eTCR, we now consider whether at least (but hopefully not the last) this most non-efficient LH transform or its variants can be used for eTCR preserving domain extension or not. Fortunately, we gather a positive answer for this. The proof for this positive result is a straightforward extension of the methodology used in [5] for the case of TCR, but with some necessary adaptations required for considering eTCR attack scenario where adversary has more power in second stage by getting to choose a different key as well as a different message. Firstly, in Theorem 7 we show that if the compression function h is eTCR secure then the hash function LH^h_{IV} will be secure against a restricted class of eTCR adversaries which only find equal-length colliding pairs. Let's denote this equal-length eTCR notion by eTCR*. Secondly, it is shown in Theorem 8 that a nested variant of LH can be made eTCR secure, *i.e.* against any arbitrary adversary.

Assume that the input messages have length a multiple of block length and the maximum length in blocks is some positive integer N, *i.e.* $|M| \leq Nb$ where b is the length of one block in bits. This restriction of message space to a domain with messages of variable but multiple-block length can be easily removed by using any proper injective padding function like plain padding function *pad*. LH^h_{IV} iteration function can be used to define a hash function as $H(K_1||\cdots||K_N, M) \triangleq LH^h_{IV}(K_1||\cdots||K_m, M)$, where m is the length of M in blocks.

Theorem 7 (Positive Result). *Assume that the compression function $h : \{0,1\}^k \times \{0,1\}^{n+b} \to \{0,1\}^n$ is (t,ϵ)-eTCR. Then the hash function $H : \{0,1\}^{Nk} \times \{0,1\}^{\leq Nb} \to \{0,1\}^n$ obtained using LH^h_{IV} iteration of h, will be (t',ϵ')-eTCR*, where $\epsilon' = N\epsilon$, $t' = t - \Theta(N)\big(T_h + n + b + k\big)$, where T_h is the time for one computation of the compression function h.*

Proof. Assume that $A = (A_1, A_2)$ is an adversary which can break LH^h_{IV} in eTCR* sense (*i.e.* equal-length eTCR sense) with success probability ϵ' and using time complexity t'. We construct an adversary B that uses A to break the compression function h in eTCR sense. First we make the observation that if the adversary A is successful in finding two equal-length colliding messages $M = M_1 \cdots M_m$ and $M' = M'_1 \cdots M'_m$ under the keys $K = K_1||\cdots||K_m$ and $K' = K'_1||\cdots K'_m$, then there must be an $i \in \{1, \cdots, m\}$ which the following two conditions hold:

(1): $LH^h_{IV}(K_1 \cdots K_i, M_1 \cdots M_i) = LH^h_{IV}(K'_1 \cdots K'_i, M'_1 \cdots M'_i)$
(2): $LH^h_{IV}(K_1 \cdots K_{i-1}, M_1 \cdots M_{i-1})||M_i \neq LH^h_{IV}(K'_1 \cdots K'_{i-1}, M'_1 \cdots M'_{i-1})||M'_i$ **OR** $K_i \neq K'_i$

This can be seen by noting that $|M| = |M'|$ and tracing back the computation in LH^h_{IV} iteration which may have made the final collision happen, that is $LH^h_{IV}(K_1 \cdots K_m, M_1 \cdots M_m) = LH^h_{IV}(K'_1 \cdots K'_m, M'_1 \cdots M'_m)$ where $(K, M) \neq (K', M')$ by winning condition for eTCR game.

Using the aforementioned observation we can build an adversary $B = (B_1, B_2)$ which can break eTCR property of h as follows:

Algorithm $B_1()$
$(M, State) \xleftarrow{\$} A_1()$; $m = |M|_b$;
$j \xleftarrow{\$} \{1, \cdots, m\}$;
$K_1, \cdots, K_{j-1} \xleftarrow{\$} \{0,1\}^k$;
$X = LH^h_{IV}(K_1 \cdots K_{j-1}, M_1 \cdots M_{j-1})||M_j$;
$St = (j, M, K_1, \cdots, K_{j-1}, State)$;
return (X, St);

Algorithm $B_2(Key, X, St)$
$(j, M, K_1, \cdots, K_{j-1}, State) \leftarrow St$; $K_j = Key$;
$K_{j+1}, \cdots, K_N \xleftarrow{\$} \{0,1\}^k$;
$(K', M') \xleftarrow{\$} A_2(K_1, \cdots, K_N, M, State)$;
$X' = LH^h_{IV}(K'_1 \cdots K'_{j-1}, M'_1 \cdots M'_{j-1})||M'_j$;
$Key' = K'_j$;
return (Key', X');

At the first stage of the eTCR game, B_1 outputs X as the target message together with the state information St to be passed to B_2 in the second stage of eTCR attack game. B_2 gets the first key for the compression function h denoted by Key which is selected uniformly at random by the challenger according to eTCR game. It outputs (Key', X') as the second key and message to finish eTCR game. It can be seen from the description of B that the distribution on key $K = K_1, \cdots, K_N$ given to A_2 is also uniform as expected in eTCR game against LH^h_{IV}. Now note that if A succeeds, there must be at least one index $i \in \{1, \cdots, m\}$ satisfying the two conditions (aforementioned conditions (1) and (2)) and as index j is selected at random by B_1 and independently from K, the probability that i matches to such an index is at least $\frac{1}{n} \geq \frac{1}{N}$. To complete the proof note that in this case, B also succeeds, that is, we have $(Key, X) \neq (Key', X')$ and $h(Key, X) = h(Key', X')$. This is seen from the way that messages X and X' are computed by algorithms B_1 and B_2, noting that $K_j = Key$ and $K'_j = Key'$ and referring to the two aforementioned conditions. Hence, if A succeeds with probability ϵ' then B also succeeds with probability $\epsilon \geq \frac{\epsilon'}{N}$. The time complexity of B (denote by t) is that of A (denote by t') plus the overhead $\Theta(N).\big(T_h + n + b + k\big)$ by the above reduction, where T_h is the time for one computation of the compression function h. □

The following theorem shows that the composition of a variable input length hash function which is secure only in the equal-length eTCR sense with a compression function which is eTCR secure will yield a variable input length hash function that is secure in eTCR sense.

Theorem 8 (From eTCR* to eTCR). *Assume that $H_1 : \{0,1\}^{k_1} \times \mathcal{M} \rightarrow \{0,1\}^n$ is (t_1, ϵ_1)-eTCR* hash function and $h : \{0,1\}^{k_2} \times \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is (t_2, ϵ_2)-eTCR compression function, where $b \geq \lceil log_2(|M|) \rceil$, for any $M \in \mathcal{M}$. Then the composition function $H : \{0,1\}^{k_1+k_2} \times \mathcal{M} \rightarrow \{0,1\}^n$, defined as $H(K1||K2, M) = h(K2, H_1(K1, M)||\langle |M| \rangle_b)$, will be (t, ϵ)-eTCR; where $\epsilon = \epsilon_1 + 2\epsilon_2$, and $t = min\{t_1 - k_2, t_2 - k_1 - 2T_{H_1} - 2b\}$.*

Proof. Let $A = (A_1, A_2)$ be a (t, ϵ)-breaking adversary against H, *i.e.* having time complexity t and $\mathrm{Adv}^{eTCR}_H(A) = \epsilon$. The experiment defining the eTCR attack by $A = (A_1, A_2)$ against H is as follows:

$$(M, State) \xleftarrow{\$} A_1(); K1 \xleftarrow{\$} \{0,1\}^{k_1}; K2 \xleftarrow{\$} \{0,1\}^{k_2}; (M', K1'||K2') \xleftarrow{\$} A_2(K1||K2, M, State) \qquad (8)$$

$\mathrm{Adv}^{eTCR}_H(A)$ is defined as the probability that, after running the above experiment in (8), the following success event happens: $H(K1||K2, M) = H(K1'||K2', M') \ \wedge \ (K1||K2, M) \neq (K1'||K2', M')$. Let $x = H_1(K1, M)$ and $x' = H_1(K1', M')$. Let E1, E2, E3 be three events as follows:

- E1: A is successful **AND** $|M| = |M'|$ **AND** $x = x'$ **AND** $K2 = K2'$
- E2: A is successful **AND** $|M| = |M'|$ **AND** $x = x'$ **AND** $K2 \neq K2'$
- E3: A is successful **AND** ($|M| \neq |M'|$ **OR** $x \neq x'$)

Clearly E1, E2, and E3 are three disjoint events, and their union is the event that A succeeds in the eTCR attack against H. Let $p_1 = \Pr[\text{E1}]$, $p_2 = \Pr[\text{E2}]$, $p_3 = \Pr[\text{E3}]$, where probabilities are under the experiment defined in equation (8). That is, we have $\mathrm{Adv}^{eTCR}_H(A) = p_1 + p_2 + p_3$. Therefore, we need to bound p_1, p_2, and p_3. To achieve this goal, using A as a subroutine, we show three adversaries $B = (B_1, B_2)$, $C = (C_1, C_2)$, and $D = (D_1, D_2)$: B can break H_1 in equal-length eTCR sense (whenever E1 happens) and has $\mathrm{Adv}^{eTCR^*}_{H_1}(B) = p_1$, C can break h in eTCR sense (whenever E2) happens and has $\mathrm{Adv}^{eTCR}_h(C) = p_2$, and D can break h in eTCR sense (whenever E3 happens) and has $\mathrm{Adv}^{eTCR}_h(D) = p_3$. From our assumption in the statement of the Theorem 8 hat H_1 is (t_1, ϵ_1)-eTCR* and h is (t_2, ϵ_2)-eTCR, it must be the case that $\mathrm{Adv}^{eTCR}_h(B) = p_1 \leq \epsilon_1$, $\mathrm{Adv}^{eTCR}_h(C) = p_2 \leq \epsilon_2$, $\mathrm{Adv}^{eTCR}_h(D) = p_3 \leq \epsilon_2$, and hence, we have $\mathrm{Adv}^{eTCR}_H(A) = p_1 + p_2 + p_3 \leq \epsilon_1 + 2\epsilon_2$ as stated in the Theorem.

Now, we just need to show the algorithms for $B = (B_1, B_2)$, $C = (C_1, C_2)$ and $D = (D_1, D_2)$. The algorithms are as follows:

Algorithm $B_1()$
$(M, State) \xleftarrow{\$} A_1()$
return $(M, State)$

Algorithm $B_2(K1, M, State)$
$K2 \xleftarrow{\$} \{0,1\}^{k_2}$
$(K1'||K2', M') \xleftarrow{\$} A_2(K1||K2, M, State)$
return $(K1', M')$

Algorithm $C_1()$
$(M, State) \xleftarrow{\$} A_1()$
$K1 \xleftarrow{\$} \{0,1\}^{k_1}$
$x = H_1(K1, M)$
$y = x||\langle|M|\rangle_b$
return $(y, (M, State, K1))$

Algorithm $C_2(K2, y, (M, State, K1))$
$(K1'||K2', M') \xleftarrow{\$} A_2(K1||K2, M, State)$
return $(K2', y)$

Algorithm $D_1()$
$(M, State) \xleftarrow{\$} A_1()$
$K1 \xleftarrow{\$} \{0,1\}^{k_1}$
$x = H_1(K1, M)$
$y = x||\langle|M|\rangle_b$
return $(y, (M, State, K1))$

Algorithm $D_2(K2, y, (M, State, K1))$
$(K1'||K2', M') \xleftarrow{\$} A_2(K1||K2, M, State)$
$x' = H_1(K1', M')$
$y' = x'||\langle|M'|\rangle_b$
return $(K2', y')$

The analysis is straightforward. Consider the eTCR attack experiment in Equation (8) and definition of the events E1, E2, E3. We claim that whenever E1 happens, the adversary $B = (B_1, B_2)$ becomes successful in attacking H_1. Note that when E1 happens $|M| = |M'|$ and hence B is an equal length eTCR attacker against H_1. To prove this claim, consider the definition of E1. Note that when A becomes successful in eTCR attack against $H = h \circ H_1$, we have $(K1||K2, M) \neq (K1'||K2', M')$ and $h(K2, H_1(K1, M)||\langle|M|\rangle_b) = h(K2', H_1(K1', M')||\langle|M'|\rangle_b)$. By definition of E1 we know that $x = H_1(K1, M) = H_1(K1', M') = x'$ and $K2 = K2'$, so the collision found by A must be an internal collision, *i.e.* a collision for H_1 and so adversary $B = (B_1, B_2)$ which attacks H_1 will be successful. That is, we have $\mathrm{Adv}_{H_1}^{eTCR^*}(B) = \Pr[\mathrm{E1}] = p_1$. The time complexity of B is $t_B = t + k_2$ and this is at most t_1 due to the assumption that H_1 is (t_1, ϵ_1)-eTCR*, that is, $t \leq t_1 - k_2$.

The analysis of success probability for the adversaries C and D which attack the eTCR property of the outer function h in $H = h \circ H_1$ can be provided similarly, just by noting the definitions for E2 and E3 events and the description of these adversaries.

Note that when E2 happens, we have $h(K2, x||\langle|M|\rangle_b) = h(K2', x||\langle|M|\rangle_b)$ (because A is successful) and $K2 \neq K2'$, hence adversary C becomes successful in eTCR attack against h as it outputs $y = x||\langle|M|\rangle_b$ in the first stage and $(K2', y)$ in the second stage. Hence $(K2, y) \neq (K2', y)$ and $h(K2, y) = h(K2', y)$ as required for winning eTCR game against h. Therefore, we have $\mathrm{Adv}_h^{eTCR}(C) = \Pr[\mathrm{E2}] = p_2$. The time complexity of C is $t_C = t + k_1 + T_{H_1} + b$ and this is at most t_2 due to the assumption that h is (t_2, ϵ_2)-eTCR, that is, $t \leq t_2 - k_1 - T_{H_1} - b$.

When E3 happens, we have $h(K2, x||\langle|M|\rangle_b) = h(K2', x'||\langle|M'|\rangle_b)$ (because A is successful) and either $|M| \neq |M|'$ or $x \neq x'$. Hence, adversary D becomes successful in eTCR attack against h as it outputs $y = x||\langle|M|\rangle_b$ in the first stage and $(K2', y' = x'||\langle|M'|\rangle_b)$ in the second stage. Hence $(K2, y) \neq (K2', y')$ (because $y \neq y'$) and $h(K2, y) = h(K2', y')$ as required for winning eTCR game against h. Therefore, we have $\mathrm{Adv}_h^{eTCR}(D) = \Pr[\mathrm{E3}] = p_3$. Therefore, we have $\mathrm{Adv}_h^{eTCR}(C) = \Pr[\mathrm{E2}] = p_2$. The time complexity of D is $t_D = t + k_1 + 2T_{H_1} + 2b$ and this is at most t_2 due to the assumption that h is (t_2, ϵ_2)-eTCR, that is, $t \leq t_2 - k_1 - 2T_{H_1} - 2b$.

Note that the bound t in the statement of the Theorem, i.e. $t = min\{t_1 - k_2, t_2 - k_1 - 2T_{H_1} - 2b\}$, satisfies all the three bounds for t as required. □

Nested Linear Hash: Let H_1 be the equal-length eTCR hash function obtained via LH transform as stated in Theorem 7. From Theorem 8 we can obtain a variant of LH which is eTCR secure. This variant which we call it Nested LH is obtained by the composition of H_1 with an eTCR compression function h, that is, LH nested by this final application of the compression function in the way stated in Theorem 8 (*i.e.* final block is just $\langle |M| \rangle_b$). Theorem 8 and Theorem 7 show that this Nested LH will be eTCR if the compression function is eTCR. Alternatively, this Nested LH construction can be seen as obtained using a *variant* of strengthening padding followed by LH iteration on the compression function h. This variant of strengthening padding, which might be called full-final-block strengthening, acts as follows. On input a message M, append the message by 10^r to make its length a multiple of block length and then append another full block which only contains the representation of length of M in an exactly b-bit string, i.e. $\langle |M| \rangle_b$.

6 Conclusion

The invention of the Enhanced Target Collision Resistance (eTCR) property by Halevi and Krawczyk [12] has been proven to be very useful to enrich the notions of hash functions, in particular with its application to construct the Randomized Hashing mode which has been announced by NIST as Draft SP 800-106. Nonetheless, the relationships between eTCR with the existing properties of hash functions need to be further studied. In this paper, we compared the eTCR property with all of the seven security properties for a hash function, formalized by Rogaway and Shrimpton in FSE 2004, and provided a full picture of relationships between eTCR and each of the properties, namely CR, Sec, aSec, eSec, Pre, aPre and ePre, where all these properties are considered formally for a dedicated-key hash function. Furthermore, when considering the problem of eTCR property preserving domain extension, we found that the only eTCR preserving method is a nested variant of LH which has a drawback of having high key expansion factor. Therefore, it is interesting to design a new eTCR preserving domain extension in *standard model*, which is *efficient*. We left this as an open problem in this paper.

References

[1] E. Andreeva, G. Neven, B. Preneel, T. Shrimpton: Seven-Property-Preserving Iterated Hashing: ROX. In: K. Kurosawa (ed.): ASIACRYPT 2007. LNCS, vol. 4833, pp. 130–146. Springer (2007)
[2] M. Bellare, T. Ristenpart: Hash Functions in the Dedicated-Key Setting: Design Choices and MPP Transforms. In L. Arge, C. Cachin, T. Jurdzinski, A. Tarlecki (eds.): ICALP 07. LNCS, vol. 4596, pp. 399–410. Springer (2007)
[3] M. Bellare, T. Ristenpart: Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In X. Lai, K. Chen (eds.): ASIACRYPT 2006. LNCS, vol. 4284, pp. 299–314. Springer (2006)
[4] M. Bellare, A. Palacio: GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In M. Yung (ed.): CRYPTO 2002. LNCS, vol. 2442, pp. 162–177. Springer (2002)
[5] M. Bellare, P. Rogaway: Collision-Resistant Hashing: Towards Making UOWHFs Practical. In B.S. Kaliski Jr. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 470–484. Springer (1997)
[6] S. Contini, Y.L. Yin: Forgery and Partial Key-Recovery Attacks on HMAC and NMAC using Hash Collisions. In: X. Lai, K. Chen (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 37–53. Springer (2006)
[7] J.S. Coron, Y. Dodis, C. Malinaud, P. Puniya: Merkle-Damgård Revisited: How to Construct a Hash Function. In V. Shoup (ed.): CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer (2005)
[8] I. Damgård: A Design Principle for Hash Functions. In G. Brassard (ed.): CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer (1990)
[9] C. De Cannière, C. Rechberger: Finding SHA-1 Characteristics: General Results and Applications. In: X. Lai, K. Chen (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer (2006)
[10] B. den Boer, A. Bosselaers: Collisions for the Compressin Function of MD5. In: T. Helleseth (ed.): EUROCRYPT 1993. LNCS, vol. 765, pp. 293–304. Springer (1993)

[11] Y. Dodis, P. Puniya: Getting the Best Out of Existing Hash Functions; or What if We Are Stuck with SHA? In S.M. Bellovin, R. Gennaro, A.D. Keromytis, M. Yung (eds.): ACNS 2008. LNCS, vol. 5037, pp. 156–173. Springer (2008)
[12] S. Halevi, H. Krawczyk: Strengthening Digital Signatures Via Randomized Hashing. In C. Dwork (ed.): CRYPTO 2006. LNCS, vol. 4117, pp. 41–59. Springer (2006)
[13] John Kelsey and Bruce Schneier: Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In *Advances in Cryptology–EUROCRYPT '05* (2005), Vol. 3494 of *LNCS*, Springer-Verlag, 474–490.
[14] U. Maurer, J. Sjödin: Single-Key AIL-MACs from Any FIL-MAC. In L. Caires, G.F. Italiano, L. Monteiro, C. Palamidessi, M. Yung (eds.): ICALP 05. LNCS, vol. 3580, pp. 472–484. Springer (2005)
[15] R.C. Merkle: One Way Hash Functions and DES. In G. Brassard (ed.): CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer (1990)
[16] I. Mironov: Hash Functions: From Merkle-Damgård to Shoup. In B. Pfitzmann (ed.): EUROCRYPT 2001. LNCS, vol. 2045, pp. 166–181. Springer (2001)
[17] M. Naor, M. Yung: Universal One-Way Hash Functions and Their Cryptographic Applications. In: STOC 1989, pp. 33–43. ACM (1989)
[18] National Institute of Standards and Technology. Cryptographic Hash Algorithm Competition. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.
[19] National Institute of Standards and Technology. Draft NIST SP 800-106: Randomized Hashing for Digital Signatures (August 2008). http://csrc.nist.gov/publications/PubsDrafts.html#SP-800-106.
[20] B. Preneel: The State of Cryptographic Hash Functions. In I. Damgård (ed.): Lectures on Data Security. LNCS, vol. 1561, pp. 158–182. Springer (1999)
[21] M.R. Reyhanitabar, W. Susilo, Y. Mu: Enhanced Target Collision Resistant Hash Functions Revisited. In O. Dunkelman (ed.): FSE 2009. LNCS, vol. 5665, pp. 327–344. Springer (2009)
[22] P. Rogaway: Formalizing Human Ignorance: Collision-Resistant Hashing without the Keys. In P.Q. Nguyen (ed.): VIETCRYPT 2006. LNCS, vol. 4341, pp. 211–228. Springer (2006)
[23] P. Rogaway, T. Shrimpton: Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. Cryptology ePrint Archive: Report 2004/035 (Latest revised version: 9 Aug 2009).
[24] P. Rogaway, T. Shrimpton: Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In B.K. Roy, W. Meier (eds.): FSE 2004. LNCS, vol. 3017, pp. 371–388. Springer (2004)
[25] V. Shoup: A Composition Theorem for Universal One-Way Hash Functions. In: B. Preneel (ed.): EUROCRYPT 2000. LNCS, vol. 1807, pp. 445–452. Springer (2000)
[26] D.R. Simon: Finding Collisions on a One-Way Street: Can Secure Hash Functions be Based on General Assumptions? In: K. Nyberg (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 334–345. Springer (1998)
[27] D.R. Stinson: Some Observation on the Theory of Cryptographic Hash Functions. Journal of *Design, Codes and Cryptography*, Vol. 38, 259–277, 2006.
[28] X. Wang, X. Lai, D. Feng, H. Chen, X. Yu: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In R. Cramer (ed.): EUROCRYPT 2005. LNCS, vol. 3494, pp. 1–18. Springer (2005)
[29] X. Wang, Y.L. Yin, H. Yu: Finding Collisions in the Full SHA-1. In V. Shoup (ed.): CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer (2005)
[30] X. Wang, H. Yu: How to Break MD5 and Other Hash Functions. In R. Cramer (ed.): EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer (2005)
[31] K. Yasuda: How to Fill Up Merkle-Damgård Hash Functions. In: Pieprzyk, J. (ed.): ASIACRYPT 2008. LNCS, vol. 5350, pp. 272–289. Springer (2008).