1-1-2009

Analysis of property-preservation capabilities of the ROX and ESh hash domain extenders

Reza Reyhanitabar
University of Wollongong, rezar@uow.edu.au

Willy Susilo
University of Wollongong, wsusilo@uow.edu.au

Yi Mu
University of Wollongong, ymu@uow.edu.au

Analysis of property-preservation capabilities of the ROX and ESh hash domain extenders

Abstract

Two of the most recent and powerful multi-property preserving (MPP) hash domain extension transforms are the Ramdom-Oracle-XOR (ROX) transform and the Enveloped Shoup (ESh) transform. The former was proposed by Andreeva et al. at ASIACRYPT 2007 and the latter was proposed by Bellare and Ristenpart at ICALP 2007. In the existing literature, ten notions of security for hash functions have been considered in analysis of MPP capabilities of domain extension transforms, namely CR, Sec, aSec, eSec (TCR), Pre, aPre, ePre, MAC, PRF, PRO. Andreeva et al. showed that ROX is able to preserve seven properties; namely collision resistance (CR), three flavors of second preimage resistance (Sec, aSec, eSec) and three variants of preimage resistance (Pre, aPre, ePre). Bellare and Ristenpart showed that ESh is capable of preserving five important security notions; namely CR, message authentication code (MAC), pseudorandom function (PRF), pseudorandom oracle (PRO), and target collision resistance (TCR). Nonetheless, there is no further study on these two MPP hash domain extension transforms with regard to the other properties. The aim of this paper is to fill this gap. Firstly, we show that ROX does not preserve two other widely-used and important security notions, namely MAC and PRO. We also show a positive result about ROX, namely that it also preserves PRF. Secondly, we show that ESh does not preserve other four properties, namely Sec, aSec, Pre, and aPre. On the positive side we show that ESh can preserve ePre property. Our results in this paper provide a full picture of the MPP capabilities of both ROX and ESh transforms by completing the property-preservation analysis of these transforms in regard to all ten security notions of interest, namely CR, Sec, aSec, eSec (TCR), Pre, aPre, ePre, MAC, PRF, PRO.

Keywords

era2015

Disciplines

Physical Sciences and Mathematics

Publication Details

Reyhanitabar, M., Susilo, W. & Mu, Y. (2009). Analysis of property-preservation capabilities of the ROX and ESh hash domain extenders. C. Boyd & J. GonzÃ¡lez Nieto In Information Security and Privacy, 14th Australasian Conference, ACISP 2009, July 2009, Brisbane, Australia. Lecture Notes in Computer Science, 5594 153-170.

Analysis of Property-Preservation Capabilities of the ROX and ESh Hash Domain Extenders

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
{mrr790, wsusilo, ymu}@uow.edu.au

Abstract. Two of the most recent and powerful multi-property-preserving (MPP) hash domain extension transforms are the Ramdom-Oracle-XOR (ROX) transform and the Enveloped Shoup (ESh) transform. The former was proposed by Andreeva et al. at ASIACRYPT 2007 and the latter was proposed by Bellare and Ristenpart at ICALP 2007. In the existing literature, ten notions of security for hash functions have been considered in analysis of MPP capabilities of domain extension transforms, namely CR, Sec, aSec, eSec (TCR), Pre, aPre, ePre, MAC, PRF, PRO. Andreeva et al. showed that ROX is able to preserve seven properties; namely collision resistance (CR), three flavors of second preimage resistance (Sec, aSec, eSec) and three variants of preimage resistance (Pre, aPre, ePre). Bellare and Ristenpart showed that ESh is capable of preserving five important security notions; namely CR, message authentication code (MAC), pseudorandom function (PRF), pseudorandom oracle (PRO), and target collision resistance (TCR). Nonetheless, there is *no* further study on these two MPP hash domain extension transforms with regard to the other properties. The aim of this paper is to fill this gap. Firstly, we show that ROX *does not preserve* two other widely-used and important security notions, namely MAC and PRO. We also show a positive result about ROX, namely that it also preserves PRF. Secondly, we show that ESh *does not preserve* other four properties, namely Sec, aSec, Pre, and aPre. On the positive side we show that ESh can preserve ePre property. Our results in this paper provide a full picture of the MPP capabilities of both ROX and ESh transforms by completing the property-preservation analysis of these transforms in regard to all ten security notions of interest, namely CR, Sec, aSec, eSec (TCR), Pre, aPre, ePre, MAC, PRF, PRO.

Key words: Hash Functions, Domain Extension, MPP, ROX, ESh

1 Introduction

A cryptographic hash function is a function that can map variable length strings to fixed length strings. Hash functions have been used in a vast variety of applications, e.g. digital signature, MAC, PRF, and must provide different security properties depending on the security requirements of the applications. The most well-known property for a hash function is collision resistance (CR). Nevertheless, hash functions are often asked to provide many other security properties ranging from merely being a one-way function (i.e. preimage resistance property) to acting as a truly random function (i.e. a random oracle).

In a formal study of cryptographic hash functions two different but related settings can be considered. The first setting is the traditional unkeyed hash function setting where a hash function refers to a single function $H : \mathcal{M} \rightarrow \{0,1\}^n$ (e.g. SHA-1) that maps variable length messages to a fixed length output hash value. In the second setting, a hash function is considered as a family of functions $H : \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^n$, also called a "dedicated-key hash function" [3], indexed by a key space $\mathcal{K}$. The exact role of the hash function key is application-dependent; it can be a public parameter, e.g. when the hash function is used in a digital signature, or a secret key like in MAC and PRF applications. In this paper, we consider hash functions and their security notions in the dedicated-key hash function setting.

Almost all cryptographic hash functions are designed based on the following two-step approach: first a compression function is designed which is only capable of hashing fixed-input-length (FIL) messages and then a domain extension transform is applied to get a full-fledged hash function which can hash variable-input-length (VIL) or arbitrary-input-length (AIL) messages, depending on the transform. Assume that we

have a (dedicated-key) 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 (dedicated-key) hash function $H : \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^n$, where the message space $\mathcal{M}$ can be either $\{0,1\}^*$ (in which case H is an AIL hash function) or $\{0,1\}^{<2^\lambda}$, for some huge positive integer λ, e.g. $\lambda = 64$ (in which case H is a VIL hash function). For instance, the strengthened-MD domain extension transform [12, 8, 3] yields to a VIL hash function while the Prefix-free domain extension transform [7, 3] yields to an AIL hash function. In practice the difference between being VIL or AIL hash function will not be of a concern as for typical value of $\lambda = 64$ almost all messages will have length less than 2^{64} bits, i.e. will belong to $\{0,1\}^{<2^{64}}$.

From security viewpoint, the crux sought from a domain extension transform is its property preserving capability; that is, if the underlying compression function h possesses some security property P, then the obtained full-fledged hash function H should also *provably* possess the property P. The most well-known domain extension transform is the strengthened Merkle-Damgård (MD) construction which was shown by Merkle [12] and Damgård [8] to be a CR preserving transform. Bellare and Rogaway in [6] showed that strengthened MD, despite preserving CR property, is unable to preserve UOWHF property (put forth by Naor and Yung [15]) which is a weaker than CR property. They renamed UOWHF as target collision resistance (TCR) and provided four domain extension transforms for preserving the TCR property. Shoup in [17] provided a transform (improving XLH transform of Bellare-Rogaway in [6]), which is shown to be UOWHF and CR preserving. Mironov [14] showed that Shoup's transform is optimal from key expansion viewpoint among masking based serial transforms for TCR preservation. Coron et al. [7] introduced the notion of random oracle preservation and provided prefix-free MD transform which is capable of preserving (pseudo-)random oracle, which means that if the compression function is modeled as a random oracle then the AIL hash function obtained by applying prefix-free MD transform will also be indifferentiable from a random oracle [7, 11]. A new line of research recently has been initiated by Bellare and Ristenpart in [4], and followed in several other works, e.g. [3, 1], with the aim of designing multi-property-preserving (MPP) domain extension transforms. An MPP transform is capable of preserving multiple security properties simultaneously while extending the domain of a compression function.

Two of the most recent and powerful MPP transforms are the Enveloped Shoup (ESh) transform designed by Bellare and Ristenpart in [3] and the Random-Oracle XOR (ROX) transform by Andreeva et al. in [1]. Both ESh and ROX are variants of Shoup (Sh) transform proposed in [17].

ESh is a standard model transform and was shown to be the best-performing, in terms of property preserving capability, among the nine transforms studies in [3]. It is shown in [3] that ESH preserves five security notions; namely CR, TCR, MAC, PRF, and PRO.

ROX was shown to be the only transform among the twelve transforms investigated in [1] which is able to preserve seven security notions; namely CR, Sec, aSec, eSec, Pre, aPre, and ePre as put forth by Rogaway and Shrimpton in [16]. But unlike to other transforms, ROX "..., quite controversially, uses a random oracle in the iteration." [1], although Andreeva et al. in [1] provide arguments justifying the merits of such a *limited application* of auxiliary FIL random oracles in their construction from practical viewpoint.

Our Contribution. We complete property-preservation analysis of the ROX and ESh transforms by providing new negative and positive results in regard to their MPP capabilities. Our results complete the property preservation analysis of ROX and ESh in regard to all ten security notions of interest, namely CR, Sec, aSec, eSec (TCR), Pre, aPre, ePre, MAC, PRF, PRO. For the ROX transform, we show that it *does not preserve* MAC and PRO. This settles the open question of [1] about MAC and PRO preservation capability of the ROX, in a negative way. On the positive side we notice that the ROX is also a PRF preserving transform. Regarding the ESh transform, we show that ESh *does not preserve* Sec, aSec, Pre, and aPre properties. As a positive result about ESh we show that it also preserves ePre property.

The overview of the results are shown in Table 1. A "Yes" means that the property is provably preserved by the transform. A "No" means that the property is not preserved and this is shown either by showing a counterexample compression function or by some attacks benefiting from the structural weakness of the transform in regard to the specific security property. Unreferenced entries are the results shown in this paper. We leave the question of a ten-property-preserving transform *without any random oracle* as an interesting open question.

	ESh	ROX
CR (Coll)	Yes [3]	Yes [1]
Sec	**No**	Yes [1]
aSec	**No**	Yes [1]
eSec (TCR or UOWHF)	Yes [3]	Yes [1]
Pre	**No**	Yes [1]
aPre	**No**	Yes [1]
ePre	**Yes**	Yes [1]
MAC	Yes [3]	**No**
PRF	Yes [3]	**Yes**
PRO	Yes [3]	**No**

Table 1. Overview of the MPP capabilities of the ESh and ROX hash domain extension transforms in regard to ten security notions. Unreferenced entries are the results shown in this paper.

2 Preliminaries

2.1 Notations

If A is a probabilistic algorithm with access to some oracle $f(.)$ then by $y \xleftarrow{\$} A^{f(.)}(x_1, \cdots, x_n)$ it is meant that y is the output random variable which is defined by running A, given inputs $x_1, \cdots, x_n$ and having oracle access to $f(.)$. To show that an algorithm A is run without any input, we use the notation $y \xleftarrow{\$} A()$. By time complexity of an algorithm we mean the running time, relative to some fixed model of computation plus the size of the description of the algorithm using some fixed encoding method. If X is a finite set, by $x \xleftarrow{\$} X$ it is meant that x is chosen from X uniformly at random. By $X \leftarrow Y$ it is meant that the value Y is simply assigned to the variable X. 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. By (x, y) we mean an injective encoding of two strings x and y, from which one can efficiently recover x and y. For a binary string M, let $|M|$ denote its length in bits and $|M|_b \triangleq \lceil |M|/b \rceil$ denote its length in b-bit blocks. Let $M[i]$ denote the i-th bit of M, and $M_{i...j}$ denote the bits from i-th to j-th positions, i.e. $M_{i...j} = M[i] \cdots M[j]$. If S is a finite set we denote size of S by $|S|$. For a positive integer m, let $\langle m \rangle_\lambda$ denote its representation as a binary string of length exactly λ bits. 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 binary strings of arbitrary length is denoted by $\{0,1\}^*$. The set of all functions $f : Dom \rightarrow Rng$ (from a domain Dom to a range Rng) is denoted by $\mathrm{Func}(Dom, Rng)$.

2.2 Definition of Security Notions

In this section, we recall definition of ten security notions for hash functions; namely, the seven notions (Coll, Sec, aSec, eSec, Pre, aPre and ePre) formalized in [16] as well as PRF, MAC, PRO. All definitions are for a

dedicated-key hash 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 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. For any $M \in \mathcal{M}$ and $K \in \mathcal{K}$, we use the notations $H_K(M)$ and $H(K,M)$ interchangeably. The advantage measures for an adversary A attacking H are defined in Fig. 1 for the ten security notions.

$$
\begin{aligned}
\mathrm{Adv}_H^{Coll}(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}{ll} K \xleftarrow{\$} \mathcal{K}; M \xleftarrow{\$} \{0,1\}^\delta; & \\ M' \xleftarrow{\$} A(K,M) & : \quad M \neq M' \ \wedge \ H_K(M) = H_K(M') \end{array}\right] \\
\mathrm{Adv}_H^{aSec[\delta]}(A) &= \Pr\left[\begin{array}{ll} (K, State) \xleftarrow{\$} A(); & \\ M \xleftarrow{\$} \{0,1\}^\delta; & \\ M' \xleftarrow{\$} A(M, State) & : \quad M \neq M' \ \wedge \ H_K(M) = H_K(M') \end{array}\right] \\
\mathrm{Adv}_H^{eSec}(A) &= \Pr\left[\begin{array}{ll} (M, State) \xleftarrow{\$} A(); & \\ K \xleftarrow{\$} \mathcal{K}; & \\ M' \xleftarrow{\$} A(K, State) & : \quad M \neq M' \ \wedge \ H_K(M) = H_K(M') \end{array}\right] \\
\mathrm{Adv}_H^{Pre[\delta]}(A) &= \Pr\left[\begin{array}{ll} K \xleftarrow{\$} \mathcal{K}; M \xleftarrow{\$} \{0,1\}^\delta; Y \leftarrow H_K(M); & \\ M' \xleftarrow{\$} A(K,Y) & : \quad H_K(M') = Y \end{array}\right] \\
\mathrm{Adv}_H^{aPre[\delta]}(A) &= \Pr\left[\begin{array}{ll} (K, State) \xleftarrow{\$} A(); & \\ M \xleftarrow{\$} \{0,1\}^\delta; Y \leftarrow H_K(M); & \\ M' \xleftarrow{\$} A(Y, State) & : \quad H_K(M') = Y \end{array}\right] \\
\mathrm{Adv}_H^{ePre}(A) &= \Pr\left[(Y, State) \xleftarrow{\$} A(); K \xleftarrow{\$} \mathcal{K}; M' \xleftarrow{\$} A(K, State) : \quad H_K(M') = Y\right] \\
\mathrm{Adv}_H^{MAC}(A) &= \Pr\left[K \xleftarrow{\$} \mathcal{K}; (M, tag) \xleftarrow{\$} A^{H_K(.)}() : \quad H_K(M) = tag \ \wedge \ \text{M not queried}\right] \\
\mathrm{Adv}_H^{PRF}(A) &= \left|\Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{H_K(.)}() \Rightarrow 1\right] - \Pr\left[\rho \xleftarrow{\$} \mathrm{Func}(\mathcal{M}, \mathcal{C}) : A^{\rho(.)}() \Rightarrow 1\right]\right| \\
\mathrm{Adv}_H^{PRO}(A) &= \left|\Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{H_K^h(.),\, h_K(.)}(K) \Rightarrow 1\right] - \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{F}(.),\, \mathcal{S}^{\mathcal{F}}(K,.)}(K) \Rightarrow 1\right]\right|
\end{aligned}
$$

Fig. 1. Definitions of ten security notions for a hash function family H [16, 3].

We say that H is (t, l, ϵ)-xxx, for xxx $\in$ {Coll, Sec[δ], aSec[δ], eSec, Pre[δ], aPre[δ], ePre}, if the advantage of any adversary A with time complexity at most t and using messages of length at most l, is less than ϵ, in attacking H in xxx sense. Note that four of the notions (namely, Sec[δ], aSec[δ], Pre[δ] and aPre[δ]) are parameterized by δ where $\{0,1\}^\delta \subseteq \mathcal{M}$. If H is a compression function (i.e. an FIL hash function), then parameter δ and the resource parameter l for the adversary will be the same as the fixed input length of the compression function and hence omitted from the notations. It is shown in [16] that the strength of provisional implications between different notions depends on the relative size of δ and the hash size n. For more related details we refer to [16].

For xxx $\in \{MAC, PRF\}$, we say that H is (t, q, l, ϵ)-xxx if the advantage of any adversary A having time complexity at most t and making at most q queries with maximum query length of l bits is at most ϵ.

PRO Notion. The definition of pseudorandom oracle preservation for a hash function was first considered by Coron et al. in [7] using the indifferentiability framework of Maurer et al. in [11], and further studied in the following works, e.g. in [4, 3]. The definition for the dedicated-key setting that we consider in this paper, as shown in Fig. 1, is due to Bellare and Ristenpart [3].

PRO is defined formally as follows. Adversary A is given '*oracles access*' to either the VIL hash function $H_K^h(.)$ and FIL random oracle $h_K(.)$, or a VIL random oracle $\mathcal{F}(.)$ and a simulator $\mathcal{S}^{\mathcal{F}}(K,.)$. A must differentiate between these two worlds and the simulator's goal is to mimic the FIL random oracle $h_K(.)$ in a way that convinces adversary A that $H_K^h(.)$ is $\mathcal{F}(.)$ (i.e. the two worlds become indifferentiable from A's

view). The PRO advantage of an adversary A against H is defined as the difference between the probability that A outputs a one when given *oracle* access to $H^h_K(.)$ and $h_K(.)$ and the probability that it outputs a one when given oracle access to $\mathcal{F}(.)$ and the simulator $\mathcal{S}^{\mathcal{F}}(K,.)$. We say that H is $(t_A, t_{\mathcal{S}}, q_1, q_2, l, \epsilon)-$PRO, if for any adversary A having time complexity at most t and making at most q_1 queries from its first (left) oracle and q_2 queries from its second (right) oracle with maximal query length of l bits, there exits a simulator $\mathcal{S}$ with time complexity $t_{\mathcal{S}}$ such that makes $\mathrm{Adv}^{PRO}_H(A) < \epsilon$.

The Special Case of ROX Construction. In the case of a hash function obtained using ROX construction, the VIL hash function H utilizes two FIL random oracles RO_1 and RO_2 in its construction as well as a compression function h. In this case the definitions should be straightforwardly adapted to consider the existence of these two auxiliary FIL random oracles, namely adversary A will be also given oracle access to RO_1 and RO_2 and the number of queries from these oracles should also be considered as additional resource parameters for the adversary A. One also must use the generalized PRO notion based on the indifferentiability framework of [11] to involve these additional random oracles. We provide the required generalized definition in section 3.1 of this paper, following [11, 4]. Briefly saying, the simulator will have to simulate three random oracles for the adversary, namely the compression function itself (which for PRO notion is modeled as an FIL random oracle), as well as the two auxiliary random oracles used by the ROX in addition to h. More details are given in section 3.1 of this paper.

2.3 Hash Domain Extension

Assume that we have a compression function $h : \{0,1\}^k \times \{0,1\}^{n+b} \to \{0,1\}^n$ 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 $H : \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$, where the message space $\mathcal{M}$ can be either $\{0,1\}^*$ or $\{0,1\}^{<2^\lambda}$, for some positive integer λ (e.g. $\lambda = 64$). 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.

A *domain extension transform* comprises two functions: an injective *'padding function'* and an *'iteration function'*. First, the padding function $Pad : \mathcal{M} \to D_I$ 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 $f : \mathcal{K} \times D_I \to \{0,1\}^n$ 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. In the case of ROX transform both the padding algorithm (rox-pad) and the iteration algorithm need small-input random oracles as well.

The padding functions used in the Sh, ESh and ROX domain extension transforms are 'Strengthening', 'Strengthened Chain Shift' and 'rox-pad' defined as follows, where 2^λ is the maximum message length in bits (typically $\lambda = 64$) :

- **Strengthening**: $\mathrm{pad}_s : \{0,1\}^{<2^\lambda} \to \bigcup_{L\geq 1}\{0,1\}^{Lb}$, where $\mathrm{pad}_s(M) = M||10^p||\langle|M|\rangle_\lambda$ and p is the minimum number of 0's required to make the length of $\mathrm{pad}_s(M)$ a multiple of block length.
- **Strengthened Chain Shift**: $\mathrm{padCS}_s : \{0,1\}^{<2^\lambda} \to \bigcup_{L\geq 1}\{0,1\}^{Lb+b-n}$, where $\mathrm{padCS}_s(M) = M||10^r||\langle|M|\rangle_\lambda||0^p$, and parameters p and r are defined in two ways depending on the block length b. If $b \geq n+\lambda$ 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.
- **rox-pad**: $\text{rox-pad}^{RO_2} : \{0,1\}^{<2^\lambda} \to \bigcup_{L\geq 1}\{0,1\}^{L.b}$, where $RO_2 : \{0,1\}^k \times \{0,1\}^\lambda \times \{0,1\}^{\lceil log\ b\rceil} \to \{0,1\}^{2n}$ is an auxiliary FIL random oracle, and

$$\text{rox-pad}^{RO_2}(M) = M||RO_2(M_{1\ldots k}, \langle|M|\rangle_\lambda, \langle 1\rangle)||RO_2(M_{1\ldots k}, \langle|M|\rangle_\lambda, \langle 2\rangle)||\cdots,$$

  where the last block of padded message must contain at least $2n$ bits generated by RO_2 which implies adding a new final block just for padding if necessary.

The *iteration functions* for Sh, ESh and ROX transforms are shown in Fig. 2, where IV, IV_1, and IV_2 are some known initial values and $IV_1 \neq IV_2$. $RO_1 : \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^{\lceil log\ \lambda \rceil} \rightarrow \{0,1\}^n$ is a random oracle used by the ROX iteration function to generate required key masks.

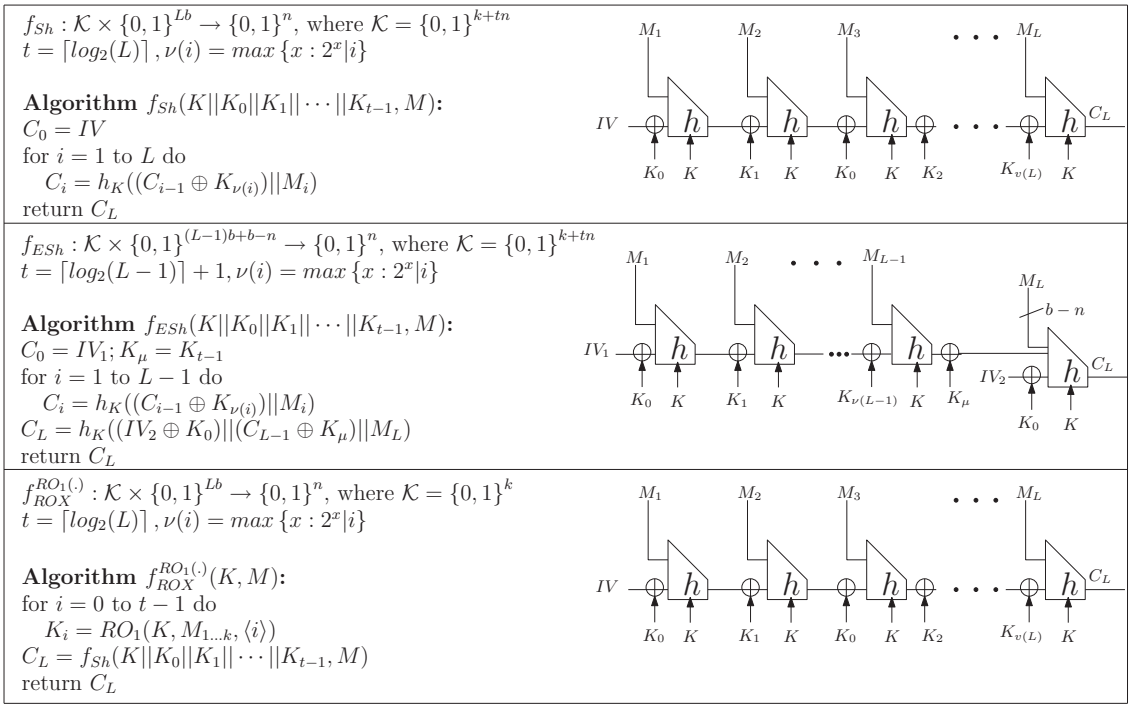


Fig. 2. Iteration functions of Shoup (Sh), Enveloped Shoup (ESh), and ROX transforms

The variable-input-length (VIL) hash function $H : \mathcal{K} \times \{0,1\}^{<2^\lambda} \rightarrow \{0,1\}^n$, for $H \in \{Sh, ESh, ROX\}$, obtained by applying Sh, ESh, or ROX domain extension transforms on a fixed-input-length (FIL) hash function $h : \{0,1\}^k \times \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is defined, respectively, as follows:

$$Sh(\mathbf{K}, M) = f_{Sh}(\mathbf{K}, \text{pad}_s(M)),\ \text{where}\ \mathbf{K} = K||K_0||\cdots||K_{t-1} \in \{0,1\}^{k+tn}$$

$$ESh(\mathbf{K}, M) = f_{ESh}(\mathbf{K}, \text{padCS}_s(M)),\ \text{where}\ \mathbf{K} = K||K_0||\cdots||K_{t-1} \in \{0,1\}^{k+tn}$$

$$ROX^{RO_1,RO_2}(K, M) = f_{ROX}^{RO_1(.)}(K, \text{rox-pad}^{RO_2(.)}(M)),\ \text{where}\ K \in \{0,1\}^k$$

3 Property Preservation Analysis of the Transforms

In this section we analyze property preserving capability of the ROX and ESh transforms in terms of the ten security notions defined in section 2.2, namely CR (Coll), Sec, aSec, eSec(TCR), Pre, aPre, ePre, MAC, PRF, PRO.

ROX was already shown in [1] to be able to preserve seven properties, namely: CR (Coll), Sec, aSec, eSec, Pre, aPre, and ePre. We complete a property-preservation analysis of ROX in regard to the other three important security notions (i.e. MAC, PRF, PRO) and gather both negative and positive results. On the negative side we show that ROX cannot preserve MAC and PRO notions. As a positive result we note that ROX can also preserve PRF. Next we investigate ESh transform. ESh was already shown in [3] to be

able to preserve five properties, namely: CR (Coll), MAC, PRF, PRO, and TCR (eSec). We complete the property preservation analysis of ESh with respect to the remaining five properties among the ten notions by showing, as negative results, that ESh does not preserve four properties, namely Sec, aSec, Pre, aPre, and as a positive result we show that it can also preserve ePre.

3.1 Analysis of the ROX Transform

In this section we provide two negative results about PRO and MAC preservation and one positive result about PRF preserving capability of the ROX domain extension transform. Among these results, the negative result showing that ROX does not preserve PRO seems more interesting regarding the fact that ROX, unlike all other hash domain extension transforms, uses random oracles in its construction.

Indifferentiability Analysis of the ROX Construction. Our aim is to show that ROX transform does not preserve PRO, i.e., the VIL hash function obtained using ROX transform is differentiable from a true VIL random oracle. We first provide the required generalization of PRO notion for the case of ROX construction based on the indifferentiability framework of [11]. Then we provide our negative result in Theorem 1.

For the purpose of PRO analysis (in the dedicated key hash setting [3]), the compression function $h : \{0,1\}^k \times \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is modeled as a family of FIL random oracles, i.e. $h_K : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is assumed to be an FIL random oracle for any value of the key K where $h_K(.) = h(K,.)$. We note that the ROX transform itself utilizes two additional FIL random oracles, namely RO_1 and RO_2 in generation of the key masks used in the iteration function and padding, respectively. Hence the VIL hash function $ROX^{h_K,RO_1,RO_2} : \{0,1\}^k \times \{0,1\}^{<2^\lambda} \rightarrow \{0,1\}^n$ will have access to three FIL random oracles, namely $h_K(.), RO_1(.)$ and $RO_2(.)$. According to the general indifferentiability framework of [11] which is used to define PRO in [7, 4, 3], adversary A will be given access to four oracles; namely, a VIL oracle $\mathcal{O}_1 : \{0,1\}^{<2^\lambda} \rightarrow \{0,1\}^n$, and three FIL oracles as $\mathcal{O}_2 : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$, $\mathcal{O}_3 : \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^{\lceil log\ \lambda \rceil} \rightarrow \{0,1\}^n$ and $\mathcal{O}_4 : \{0,1\}^k \times \{0,1\}^\lambda \times \{0,1\}^{\lceil log\ b \rceil} \rightarrow \{0,1\}^{2n}$, and must differentiate between the following two worlds:

- **World 1**: A random key $K \xleftarrow{\$} \{0,1\}^k$ is selected. The oracles are set as $\mathcal{O}_1(.) = ROX^{h_K,RO_1,RO_2}(K,.)$, $\mathcal{O}_2(.) = h_K(.)$, $\mathcal{O}_3(.) = RO_1(.)$, $\mathcal{O}_4(.) = RO_2(.)$. A is given K as input and has access to the four oracles.
- **World 2**: A random key $K \xleftarrow{\$} \{0,1\}^k$ is selected. $\mathcal{O}_1(.) = \mathcal{F}(.)$ where $\mathcal{F} : \{0,1\}^{<2^\lambda} \rightarrow \{0,1\}^n$ is a true VIL random oracle. A simulator $\mathcal{S}^{\mathcal{F}}(K) = (\mathcal{S}_1^{\mathcal{F}}(K), \mathcal{S}_2^{\mathcal{F}}(K), \mathcal{S}_3^{\mathcal{F}}(K))$, having access to the oracle $\mathcal{F}(.)$ and receiving K as input, simulates the role of the three FIL random oracles for the adversary. That is, in this world when adversary queries the first oracle (i.e. the VIL oracle) $\mathcal{O}_1$ the response comes from the true VIL random oracle $\mathcal{F}(.)$, but when adversary queries any of the three FIL oracles $\mathcal{O}_2(.), \mathcal{O}_3(.)$ and $\mathcal{O}_4(.)$, the queries are forwarded to the the simulator $\mathcal{S}$. Simulator will respond to these queries trying to mimic the oracles $h_K(.), RO_1(.)$ and $RO_2(.)$, respectively, by its sub-algorithms $\mathcal{S}_1, \mathcal{S}_2$ and $\mathcal{S}_3$ in a way that convinces A that $\mathcal{O}_1(.)$ is $ROX^{h_K,RO_1,RO_2}(K,.)$ although it is now actually $\mathcal{F}(.)$.

Let $H_K(.) = H(K,.) = ROX^{h_K,RO_1,RO_2}(K,.)$. The PRO advantage of the adversary in differentiating H from $\mathcal{F}$ is defined as follows:

$$\begin{aligned}\mathrm{Adv}_H^{PRO}(A) &= \left|\Pr\left[A^{\mathcal{O}_1,\,\mathcal{O}_2,\,\mathcal{O}_3,\,\mathcal{O}_4}(K) \Rightarrow 1 \,|\text{World 1}\right] - \Pr\left[A^{\mathcal{O}_1,\,\mathcal{O}_2,\,\mathcal{O}_3,\,\mathcal{O}_4}(K) \Rightarrow 1 \,|\text{World 2}\right]\right| \\ &= \left|\Pr\left[A^{H_K(.),\,h_K(.),\,RO_1(.),\,RO_2(.)}(K) \Rightarrow 1\right] - \Pr\left[A^{\mathcal{F}(.),\,\mathcal{S}_1^{\mathcal{F}}(K),\,\mathcal{S}_2^{\mathcal{F}}(K),\,\mathcal{S}_3^{\mathcal{F}}(K)}(K) \Rightarrow 1\right]\right|\end{aligned}$$

We say that an adversary A is a $(t_A, t_{\mathcal{S}}, q_1, q_2, q_3, q_4, l, \epsilon)$-differentiating adversary against H if its PRO advantage is at least ϵ against any simulator $\mathcal{S}$ having time complexity at most $t_{\mathcal{S}}$, where the time complexity

of the adversary is at most t_A, the number of queries from i-th oracle is at most q_i (for $1 \leq i \leq 4$) and the length of each query is at most l bits.

Now we are ready to state our negative result which shows the inability of ROX to preserve PRO.

Theorem 1 (Negative Result: PRO). *ROX domain extension transform does not preserve pseudorandom oracle.*

Proof. We show a $(c, t_{\mathcal{S}}, 2, 1, 1, 2, 3b-2n, 1-2^{-n})$-differentiating adversary against H, i.e. A has overwhelming PRO advantage of $1-2^{-n}$ in differentiating the VIL hash function $ROX^{h_K, RO_1, RO_2} : \{0,1\}^{<2^\lambda} \rightarrow \{0,1\}^n$ from a true VIL random oracle $\mathcal{F} : \{0,1\}^{<2^\lambda} \rightarrow \{0,1\}^n$, with respect to any simulator $\mathcal{S}$ with *arbitrary time complexity* $t_{\mathcal{S}}$. A has time complexity $t_A = c$, where c is a small constant, and it asks only: two queries from the first oracle $\mathcal{O}_1$, one query from the second oracle $\mathcal{O}_2$, one query from the third oracle $\mathcal{O}_3$ and two queries from the fourth oracle $\mathcal{O}_4$. The maximum query length is $3b-2n$ bits.

Adversary A acts as follows:

1. $M \xleftarrow{\$} \{0,1\}^{2b-2n}$ and $Y \leftarrow \mathcal{O}_1(M)$;
2. $IP \leftarrow \mathcal{O}_4(M_{1...k}, \langle 2b-2n \rangle_\lambda, \langle 1 \rangle)$;
3. $M' \xleftarrow{\$} \{0,1\}^{b-2n}$ and $Z \leftarrow \mathcal{O}_1(M||IP||M')$;
4. $K_0 \leftarrow \mathcal{O}_3(K, M_{1...k}, \langle 0 \rangle)$;
5. $OP \leftarrow \mathcal{O}_4(M_{1...k}, \langle 3b-2n \rangle_\lambda, \langle 1 \rangle)$;
6. $Z' \leftarrow \mathcal{O}_2\big((Y \oplus K_0)||M'||OP\big)$;
7. If $Z = Z'$ then return 1 (i.e. guess that it is World 1) else return 0 (i.e. guess that it is World 2)

From the construction of the ROX hash function and the description of the World 1, it can be seen that $\Pr\left[A^{H_K(.),\, h_K(.),\, RO_1(.),\, RO_2(.)}(K) \Rightarrow 1\right] = 1$. We claim that $\Pr\left[A^{\mathcal{F}(.),\, \mathcal{S}_1^{\mathcal{F}}(K),\, \mathcal{S}_2^{\mathcal{F}}(K),\, \mathcal{S}_3^{\mathcal{F}}(K)}(K) \Rightarrow 1\right] = 2^{-min\{n, 2b-2n-k\}}$. This can be seen by noting that in World 2, the only queries that the simulator $\mathcal{S}^{\mathcal{F}}(K) = (\mathcal{S}_1^{\mathcal{F}}(K), \mathcal{S}_2^{\mathcal{F}}(K), \mathcal{S}_3^{\mathcal{F}}(K))$ can see and must respond to are the queries from $\mathcal{O}_2, \mathcal{O}_3$, and $\mathcal{O}_4$ oracles. It is worth reminding that, the simulator cannot see query-response sequence between the adversary A and the first oracle $\mathcal{O}_1$ due to the definition of indifferentiability [7, 4, 3], as these queries are answered directly by the true VIL random oracle $\mathcal{F}$ in World 2. Hence referring to the description of the adversary A, it can be seen that the simulator $\mathcal{S}$ just is given the first k bits of the first message M, i.e. $M_{1...k}$, and has *no information* about the remaining $2b-2n-k$ bits of the message M. Hence to make A output a one (i.e. to fool A), simulator $\mathcal{S}$ must either guess these $2b-2n-k$ unknown bits of M (with success probability of $2^{-(2b-2n-k)}$) or guess the correct value of Z (with success probability of 2^{-n}) in order to be able to provide a correct value Z' at step 6 of A's differentiating attack. (Note that the success probability of $\mathcal{S}$ in guessing the correct value of these unknown bits is independent of the time complexity of $\mathcal{S}$, i.e. $t_{\mathcal{S}}$, as $\mathcal{S}$ has no information about these bits.) So, we have $\mathrm{Adv}_H^{PRO}(A) = 1 - 2^{-min\{n, 2b-2n-k\}}$. It remains to verify that $2b-2n-k \geq n$. According to the construction of $rox-pad^{RO_2}$ it must be the case that $b \geq 2n$ and referring to [1], typical values for k and n are suggested as $k = 80$ bits and $n = 160$ bits for an 80-bit security level, i.e. we have $k \approx n/2$. Hence $min\{n, 2b-2n-k\} = n$ and $\mathrm{Adv}_H^{PRO}(A) = 1 - 2^{-n}$ as claimed. □

MAC Preservation Analysis of the ROX. We show that the ROX transform does not preserve MAC (unforgeability). This is done by providing as a counterexample, a compression function h which is a secure MAC but for which the VIL hash function obtained by using ROX transform will be insecure in MAC sense.

Assume that there is a compression function $g : \{0,1\}^k \times \{0,1\}^{n+b} \rightarrow \{0,1\}^{n-1}$ which is $(t, q, \epsilon)-$MAC. Consider the following construction from [3] for a compression function $h : \{0,1\}^k \times \{0,1\}^{n+b} \rightarrow \{0,1\}^n$, where $s = \lceil log_2(n) \rceil$:

$$h_K(C||M) = \begin{cases} g_K(C||M)||C[i] & \text{if } M = \langle i \rangle_s ||0^{b-s} \text{ for } i \in \{1, \cdots, n-1\} \\ g_K(C||M)||C[n] & \text{otherwise} \end{cases}$$

It is shown in [3] that if g is $(t, q, \epsilon)-$MAC then h will be $(t - cq, q, \epsilon)-$MAC, where c is a small constant. Note that h leaks the i-th bit of its chaining variable input C to the output if its input block M equals to $\langle i \rangle_s ||0^{b-s}$, for $i \in \{1, \cdots, n-1\}$, and otherwise h leaks the n-th (i.e. the last) bit of C.

We use this counterexample to prove the following negative result.

Theorem 2 (Negative Result: MAC). *ROX domain extension transform does not preserve MAC.*

Proof. Consider the above counterexample function h as an FIL MAC. We show that the VIL function $H(K, .) = ROX^{h,RO_1,RO_2}(K, .)$ obtained by applying the ROX transform on this FIL MAC h will not be a secure MAC. This is done by describing an adversary A which can break $H(K, .)$ in MAC sense, with success probability $\approx \frac{1}{4}$ and whose computational resources are only: one query from the random oracle $RO_2(.)$, $2(n-1)$ queries from the function $H_K(.)$ with maximum length of each query $4b - 2n$ bits, and a constant time complexity proportional to n.

The idea behind the construction of the adversary A is the same "length reduction attacks" used in [3] to show that Sh transform is not MAC preserving. We adapt the attack for the case of ROX transform by considering the special padding function rox-pad^{RO_2} used by the ROX. The algorithm for the adversary A is shown in Fig. 3. It has oracle access to $H_K(.)$ and $RO_2(.)$. Note that in definition of the MAC security notion, the key K is considered secret from the adversary and hence A cannot compute $H_K(.)$ directly.

It selects an all zero string of length $2b - 2n$ bits as $M = 0^{2b-2n}$, for which it tries to return a valid tag T under $H_K(.)$. To this aim, A first queries RO_2 as $IP \leftarrow RO_2(M_{1...k}, \langle 2b - 2n \rangle_\lambda, \langle 1 \rangle)$ to get the 2n-bit response IP (IP stands for 'internal padding'). It then queries oracle $H_K(.)$ on $n-1$ messages, each of length $4b - 2n$ bits, constructed as $QH_i = M||IP||\langle i \rangle_s ||0^{b-s}||0^{b-2n}$, and it receives the response $Y_i = H_K(QH_i)$, for $i \in \{1, \cdots, n-1\}$. Let y_i denote the last bit of Y_i, i.e. $y_i = Y_i[n]$. According to the ROX construction and the structure of the counterexample compression function h, the value of the bit y_i will be computed as $y_i = H_K(M)[i] \oplus K_0[i] \oplus K_2[n]$ with overwhelming probability of at least $1 - 2^{-min\{2n,b-s\}}$. This can be seen from (the Top-Right diagram in) Fig. 3 noting that the final block contains $2n$ bits of random padding string (denoted by OP) as its last bits and hence this final block input to the compression function h is not equal to $\langle i \rangle_s ||0^{b-s}$ with probability at least $1 - 2^{-min\{2n,b-s\}}$, for $i \in \{1, \cdots, n-1\}$, and therefore h will leak the last bit (i.e. n-th bit) of its chaining variable input to the output Y_i. Therefore with probability at least $1 - 2^{-min\{2n,b-s\}}$, the variable y_i (for $1 \leq i \leq n-1$) contains the i-th bit of the tag T for the message M (i.e. $T[i] = H_K(M)[i]$) masked with the unknown key bits $K_0[i]$ and $K_2[n]$. For any typical value of b and n (say $b = 512, n = 160$) we have $1 - 2^{-min\{2n,b-s\}} \approx 1$ and so we assume that this probability is approximately one, to prevent unnecessary complexity in the analysis.

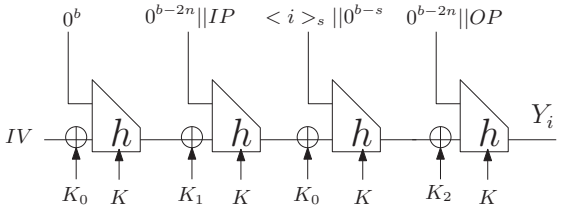
Now A tries to peel off the unknown key bits $K_0[i]$, for $1 \leq i \leq n-1$. It queries $H_K(.)$ on $n-1$ messages, each of length $2b - 2n$ bits, constructed as $qH_i = \langle i \rangle_s ||0^{b-s}||0^{b-2n}$ and receives the response $Z_i = H_K(qH_i)$, for $i \in \{1, \cdots, n-1\}$. Let z_i denote the last bit of the Z_i, i.e. $z_i = Z_i[n]$. According to the description of $H_K(.)$ and the structure of the counterexample function h, the value of bit z_i will be computed as $z_i = H_K(qH_i)[i] = IV[i] \oplus K_0[i] \oplus K_1[n]$ with overwhelming probability of at least $1 - 2^{-2n}$. This can be seen from (the Bottom-Right diagram in) Fig. 3 noting that the final block contains $2n$ bits of random padding string (denoted by OP'_i) as its last bits and hence this final block input to the compression function h is not equal to $\langle i \rangle_s ||0^{b-s}$ with probability at least $1 - 2^{-2n}$, for any $i \in \{1, \cdots, n-1\}$, and hence h will leak the last bit (i.e. n-th bit) of its chaining variable input to the output Z_i. For any typical value of n (say $n = 160$ as in SHA-1) we have $1 - 2^{-2n} \approx 1$ and so we assume that this (overwhelming) probability is approximately one.

Now for each $i \in \{1, \cdots, n-1\}$ adversary builds a variable $t_i = y_i \oplus z_i \oplus IV[i]$ whose value will be $t_i = H_K(M)[i] \oplus K_1[n] \oplus K_2[n]$ (with overwhelming probability of at least $(1 - 2^{-2n})^2 \approx 1$ for typical values of n, say $n = 160$). Note that the value of the remaining unknown masking bit $K_1[n] \oplus K_2[n]$ is independent of the index i, i.e. it is the same for all t_i and therefore adversary can guess this unknown value with probability 1/2. That is, for a random guess $\alpha \xleftarrow{\$} \{0, 1\}$ with probability 1/2 the value $t_i \oplus \alpha$ will be

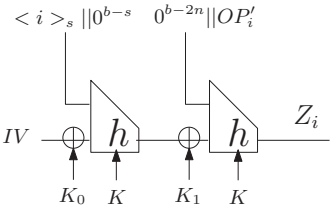
equal to $H_K(M)[i] = T[i]$ (i.e. the correct tag value), for all $i \in \{1, \cdots, n-1\}$. It just remains to compute the last bit of the tag T, i.e. $T[n]$, but A just guesses this one bit and with probability 1/2 this guess will be correct. Hence the adversary A computes a correct MAC tag T for the message M under $H_K(.)$ with probability 1/4 (or more precisely with probability $\frac{1}{4}(1-(n-1)2^{-2n+1})$, which for any typical value of hash size n, say $n = 160$, will be $\approx 1/4$). Note that the message M never is queried from $H_K(.)$ in the attack and so this is a valid forgery attack.

Referring to the algorithm for A it is seen that A is quite efficient as its computational resources are: one query from the random oracle $RO_2(.)$, $2(n-1)$ queries from the function $H_K(.)$ with maximum length of each query $4b-2n$ bits, and a constant time complexity proportional to n which can be easily determined from the description of A.

Algorithm $A^{H_K(.),RO_2(.)}$**:**
$M \leftarrow 0^{2b-2n}$
$IP \leftarrow RO_2(M_{1\ldots k}, \langle 2b-2n \rangle_\lambda, \langle 1 \rangle)$
for $i = 1$ to $n-1$ do
  $Y_i \leftarrow H_K(M||IP||\langle i \rangle_s||0^{b-s}||0^{b-2n})$
  $y_i \leftarrow Y_i[n]$
for $i = 1$ to $n-1$ do
  $Z_i \leftarrow H_K(\langle i \rangle_s||0^{b-s}||0^{b-2n})$
  $z_i \leftarrow Z_i[n]$
$\alpha \xleftarrow{\$} \{0,1\}$
for $i = 1$ to $n-1$ do
  $t_i \leftarrow y_i \oplus z_i \oplus IV[i]$
  $T[i] \leftarrow t_i \oplus \alpha$
$T[n] \xleftarrow{\$} \{0,1\}$
$T \leftarrow T[1] \cdots T[n]$
return (M, T)



$M = 0^{2b-2n}$
$y_i = Y_i[n] = H_K(M)[i] \oplus K_0[i] \oplus K_2[n]$
$OP = RO_2(0^k, \langle 4b-2n \rangle, \langle 1 \rangle)$



$z_i = Z_i[n] = IV[i] \oplus K_0[i] \oplus K_1[n]$
$OP'_i = RO_2(\langle i \rangle_s||0^{k-s}, \langle 2b-2n \rangle, \langle 1 \rangle)$

Fig. 3. (Left) Description of the algorithm for an adversary A against the VIL function $H_K(.) = ROX^{RO_1,RO_2}(K,.)$ based on the (counterexample) FIL MAC function h. **(Right)** The structure of the queries from $H_K(.)$ and computation of the responses according to the ROX construction.

□

Theorem 3 (Positive Result: PRF). *ROX domain extension transform preserves PRF.*

Proof. This result is just a straightforward corollary of a theorem in [3] (Theorem 5, page 407) showing that *in the dedicated-key hash function setting* (where the compression function is a keyed hash function) Merkle-Damgård transform and all of its variants including Shoup are PRF preserving transforms. As shown in [3, 5] even if the key masks $(K_0, K_1, \cdots)$ in Shoup construction are made public and only the key (K) for the compression function is kept secret then Shoup transform will still be PRF preserving. Clearly a special case will be putting the value of all key masks to zero in which case Shoup iteration will be the same as Merkle-Damgård iteration which is a PRF preserving transform *in the dedicated-key hash setting* [5]. We note that the iteration function of the ROX transform is exactly the same as Shoup where the key masks are generated using a random oracle (Refer to Fig. 2). □

3.2 Analysis of the ESh Transform

The following theorems show our results about the ESh. We show both negative results and a positive result about property preservation capability of ESh.

Theorem 4 (Negative Results: Sec, aSec, Pre, and aPre). *ESh domain extension transform does not preserve any of the Sec, aSec, Pre, and aPre security properties.*

Proof. The proof is done by showing, as a counterexample, a compression function which is secure in xxx sense for xxx $\in$ {Sec, aSec, Pre, aPre} but for which the full-fledged hash function obtained using ESh domain extension transform is completely insecure in xxx sense for xxx $\in$ {Sec[δ], aSec[δ], Pre[δ], aPre[δ]} and for *any value of the parameter* $\delta < 2^\lambda$ (remember that 2^λ is the maximum input message length in bits).

Referring to the description of the strengthened chain shift padding function (padCS$_s$) used in ESh transform, we consider the following two cases depending on the sizes of the parameters b, n and λ (note that for ESh, $b \geq n$ and typical value of $\lambda = 64$):

- Case 1: if $b \geq n + \lambda$ then $\text{padCS}_s(M) = M||10^r||\ \langle|M|\rangle_\lambda$
- Case 2: if $b < n + \lambda$ then $\text{padCS}_s(M) = M||10^r||\ \langle|M|\rangle_\lambda\ ||0^{b-n}$

Assume that there is a compression function $g : \{0,1\}^k \times \{0,1\}^{n+b} \rightarrow \{0,1\}^{n-1}$ which is (t, ϵ)-xxx, where xxx is any of the four properties in {Sec, aSec, Pre, aPre}. For Case 1 and Case 2, respectively, consider the following two compression functions $h_1 : \{0,1\}^k \times \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ and $h_2 : \{0,1\}^k \times \{0,1\}^{n+b} \rightarrow \{0,1\}^n$:

$$h_1(K, M) = \begin{cases} 0^n & \text{if } M_{n+b-\lambda+1\ldots n+b} = \langle\delta\rangle_\lambda \\ g(K, M)||1 & \textit{otherwise} \end{cases}$$

$$h_2(K, M) = \begin{cases} 0^n & \text{if } M_{2n+1\ldots n+b} = 0^{b-n} \\ g(K, M)||1 & \textit{otherwise} \end{cases}$$

Construction of the counterexamples h_1 and h_2 are inspired from the CE$_3$ counterexample in [1] where we make some modifications in the conditions defining these two functions as it is necessary to consider the effect of padCS$_s$ padding in ESh transform. To complete the proof of the theorem we prove and combine the following two lemmas. The first lemma shows that the compression functions h_1 and h_2 inherit security properties xxx $\in$ {Sec, aSec, Pre, aPre} from the compression function g and the second lemma shows that ESh transform cannot preserve these four properties while extending the domain of h_1 or h_2 compression functions. Note that only one of these compression functions are used depending on which of the two conditions specified in Case 1 and Case 2 above are the case.

Lemma 1. *If g is (t, ϵ)-xxx then h_1 is $(t, \epsilon + 2^{-\lambda})$-xxx and h_2 is $(t, \epsilon + 2^{-(b-n)})$-xxx, for any of the notions* xxx $\in$ {*Sec, aSec, Pre, aPre*}.

Proof. The proof is provided in Appendix 1.

Lemma 2. *For any* xxx $\in$ {*Sec[δ], aSec[δ], Pre[δ], aPre[δ]*}*, and for any value of $\delta < 2^\lambda$, there is a simple adversary which can break the domain extended hash function $ESh(\mathbf{K}, M) = f_{ESh}(\mathbf{K}, padCS_s(M))$ using h_1 or h_2 as the compression function.*

Proof. Considering the description of the padCS$_s$ and counterexample compression functions h_1 and h_2, we have $ESh(\mathbf{K}, M) = 0$ for any $M \in \{0,1\}^\delta$. Hence, in Pre[δ] and aPre[δ] attacks adversary A just needs to output any arbitrary $M' \in \{0,1\}^\delta$ and wins with probability one. Similarly, in Sec[δ] and aSec[δ] attacks A only needs to output any $M' \in \{0,1\}^\delta$ which is different from the challenge message $M \in \{0,1\}^\delta$ and wins with probability one. That is, the VIL hash function $ESh : \mathcal{K} \times \{0,1\}^{<2^\lambda} \rightarrow \{0,1\}^n$, defined as $ESh(\mathbf{K}, M) = f_{ESh}(\mathbf{K}, \text{padCS}_s(M))$ using h_1 or h_2 is completely insecure in xxx sense for xxx $\in$ {Sec[δ], aSec[δ], Pre[δ], aPre[δ]} and for any value of the parameter δ, where $\delta < 2^\lambda$.

□

Theorem 5 (Positive Result: ePre). *If the compression function* $h : \{0,1\}^k \times \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ *is* $(t,\epsilon)-ePre$ *then the full-fledged hash function* $ESh : \mathcal{K} \times \{0,1\}^{<2^\lambda} \rightarrow \{0,1\}^n$ *defined as* $ESh(\mathbf{K}, M) = f_{ESh}(\mathbf{K}, padCS_s(M))$ *will be* $(t',\epsilon)-ePre$*, where* $t' = t - c$*, for a small constant* c*.*

Proof. Assume that there is an adversary A against ePre property of the hash function ESh with time complexity t' and advantage ϵ'. We construct an adversary B which can break the compression function h in ePre sense with the same advantage (i.e. $\epsilon = \epsilon'$) and whose time complexity is that of A plus a small constant time (i.e. $t = t' + c$). Adversary B runs A and on receiving the value of Y from A outputs the same value Y as its own target hash value in the first phase of ePre game. B receives K (the random key for h), generates $K_0||\cdots||K_{t-1} \xleftarrow{\$} \{0,1\}^{t.n}$, where $t = \lceil log_2(2^\lambda/b) \rceil + 1$ and sends the key $K||K_0||\cdots||K_{t-1}$ to adversary A as the key for the full-fledged hash function ESh. Note that because B by this phase just knows the Y and does not know the length of the input message to the hash function ESh, it generates a key string of maximum required length for the XOR masks, i.e. $t.n$ bits, for $t = \lceil log_2(2^\lambda/b) \rceil + 1$ where $2^\lambda/b$ is the maximum possible input length in blocks and n is the hash size. Now on receiving the message M' from A (which is to be a preimage for Y under ESh, i.e. $Y = ESh(K||K_0||\cdots||K_{t-1},\ M')$), adversary B simply outputs the value $(IV_2 \oplus K_0)||(C_{L-1} \oplus K_\mu)||M'_L$ as a preimage for Y (refer to Fig. 2) which is the input to the final application of the compression function h in the construction of ESh hash function. Clearly B wins whenever A wins. The time complexity of B is that of A plus the time required to generate t random n-bit keys, where $t = O(\lambda)$ (typically $\lambda = 64$), and the time to compute the hash function ESh on a message of length $|M'|$. □

4 Can We Preserve All Properties?

In the previous section we showed that the ROX transform, which is a random oracle variant of Shoup, does not preserve MAC and PRO notions and also we showed that the Enveloped Shoup (ESh) transform does not preserve Sec, aSec, Pre, aPre notions. An immediate question arises from this analysis is that whether we can preserve all the ten properties simultaneously by a new domain extension transform.

4.1 Using FIL Random Oracles

If we are allowed to use some FIL Random Oracles in our construction in the same way that ROX does (i.e. uses FIL random oracles just for padding and generation of masking keys), then our analysis in the previous section hints us towards a candidate for such a ten-property-preserving transform by just mixing components from both ESh and ROX. We notice that, as shown in Fig. 2, ROX utilizes Shoup's iteration as its underlying iteration function and uses FIL random oracles for generation of masking keys and padding function. Hence the natural candidate for a ten property preserving transforms seems to be a random oracle variant of ESh with some necessary adaptation in a similar way that ROX is obtained from Sh. We call such a transform as Random-Oracle Enveloped Shoup (RO-ESh). The construction of RO-ESh can be seen as a mixture of ESh and ROX elements and like the ROX construction, RO-ESh also needs two "small-input" FIL random oracles. The FIL random oracles in RO-ESH are used only, logarithmic number of times in message length, for message padding and generation of masking keys from a single key, but the compression function h itself is not modeled as a random oracle.

The padding function of the RO-ESh domain extension transforms is defined as follows, where 2^λ is the maximum message length in bits (typically $\lambda = 64$) :

$$\text{RO-ESh-pad} : \{0,1\}^{<2^\lambda} \rightarrow \bigcup_{L \geq 1} \{0,1\}^{L.b+b-n}$$
$$\text{RO-ESh-pad}^{RO_2}(M) = M||10^r||RO_2(M_{1\ldots k}, \langle |M| \rangle_\lambda)$$

, where r is the minimum number of 0's required to make the padded message a member of $\{0,1\}^{L.b+b-n}$, for some positive integer L, and $RO_2 : \{0,1\}^k \times \{0,1\}^\lambda \to \{0,1\}^{b-n}$ is an FIL random oracle.

The iteration function for RO-ESh transform is shown in Fig. 4, where IV_1 and IV_2 are known initial values and $IV_1 \neq IV_2$. $RO_1 : \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^{\lceil log\ \lambda \rceil} \to \{0,1\}^n$ is a random oracle used by the RO-ESh iteration function to generate required key masks.

$f^{RO_1(.)}_{RO-ESh} : \mathcal{K} \times \{0,1\}^{(L-1).b+b-n} \to \{0,1\}^n$, where $\mathcal{K} = \{0,1\}^k$
$t = \lceil log_2(L-1) \rceil + 1, \nu(i) = max\{x : 2^x | i\}$

Algorithm $f^{RO_1(.)}_{RO-ESh}(K, M)$**:**
for $i = 0$ to $t-1$ do
  $K_i = RO_1(K, M_{1\ldots k}, \langle i \rangle)$
$C_L = f_{ESh}(K||K_0||K_1||\cdots||K_{t-1}, M)$
return C_L

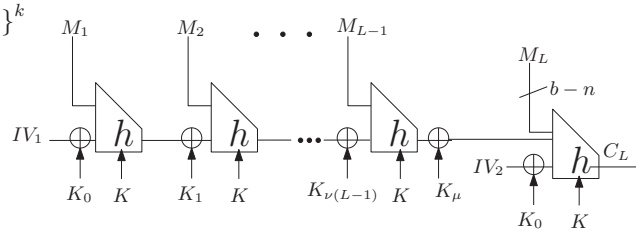


Fig. 4. Iteration function of the RO-ESh transform. The structure of the iteration is the same as ESh and only the key masks are generated using an FIL random oracle RO_1. f_{ESh} is the iteration function of ESh as shown by the diagram on the left and described in Fig. 2.

The VIL hash function $H : \{0,1\}^k \times \{0,1\}^{<2^\lambda} \to \{0,1\}^n$, obtained by applying the RO-ESh domain extension transform on an FIL hash function $h : \{0,1\}^k \times \{0,1\}^{n+b} \to \{0,1\}^n$ is defined as follows:

$$H(K,\ M) = \text{RO-ESh}^{RO_1,\ RO_2}(K, M) = f^{RO_1}_{RO-ESh}(K,\ \text{RO-ESh-pad}^{RO_2}(M))$$

The proof of the following Theorem is obtained by a straightforward (but lengthy) adaptation of the previous results proved in [1] and [3], for the security of ROX and ESh transforms, respectively.

Theorem 6. *If the compression function $h : \{0,1\}^k \times \{0,1\}^{n+b} \to \{0,1\}^n$ has security property $P \in \{CR, Sec, aSec, TCR, Pre, aPre, ePre, MAC, PRF, PRO\}$, then the hash function H obtained by RO-ESh transform will also possess property P, with the following concrete-security reductions and relations between the resources:*

1. *if h is (t,ϵ)-CR then H will be (t', l, ϵ')-CR, where $\epsilon' = \epsilon + \frac{q^2_{RO_2}}{2^{b-n}}$ and $t' = t - 2\tau(l)T_h$*
2. *if h is (t,ϵ)-TCR then H will be (t', l, ϵ')-TCR, where $\epsilon' = \tau(l)\epsilon + \frac{q_{RO_1}}{2^k} + \frac{q^2_{RO_2}}{2^{b-n}}$ and $t' = t - 2\tau(l)T_h$*
3. *if h is (t,ϵ)-aSec then H will be (t', l, ϵ')-aSec[m], where $\epsilon' = \tau(l)\epsilon + \frac{q_{RO_1}}{2^k} + \frac{q^2_{RO_2}}{2^{b-n}}$ and $t' = t - 2\tau(l)T_h$*
4. *if h is (t,ϵ)-Sec then H will be (t', l, ϵ')-Sec[m], where $\epsilon' = \tau(l)\epsilon + \frac{q^2_{RO_2}}{2^{b-n}}$ and $t' = t - 2\tau(l)T_h$*
5. *if h is (t,ϵ)-aPre then H will be (t', l, ϵ')-aPre[m], where $\epsilon' = \epsilon + \frac{q_{RO_1}}{2^k}$ and $t' = t - 2\tau(l)T_h$*
6. *if h is (t,ϵ)-Pre then H will be (t', l, ϵ')-Pre[m], where $\epsilon' = \epsilon$ and $t' = t - \tau(l)T_h$*
7. *if h is (t,ϵ)-ePre then H will be (t', l, ϵ')-ePre, where $\epsilon' = \epsilon$ and $t' = t - \tau(l)T_h$*
8. *if h is (t, q, ϵ)-MAC then H will be (t', q', l, ϵ')-MAC, where $\epsilon' = (q^2/2 + 3q/2 + 1)\epsilon$, $t' = t - c(q'+1)\tau(l)$ and $q' = (q - \tau(l) + 1)/\tau(l)$*

9. *if h is (t, q, ϵ)-PRF then H will be (t', q', l, ϵ')-PRF, where $\epsilon' = \epsilon + q^2\tau(l)^2/2^n$, $t' = t - cq\tau(l)$ and $q' = q/\tau(l)$*

10. *if $h_K = RF_{n+b,\,n}$ be a random oracle for any arbitrary K, then H will be $(t_A, t_{\mathcal{S}}, q_1, q_2, q_3, q_4, l, \epsilon)$-PRO, where running time of adversary t_A is arbitrary, time complexity for simulator $t_S = O(q_2^2)$, q_i is the number of queries by adversary from i-th oracle (for $1 \leq i \leq 4$) and*

$$\epsilon' = \frac{\tau(l)^2 q_1^2 + \tau(l) q_1 q_2 + q_2^2}{2^n} + \frac{\tau(l) q_1 + q_2 + q_3^2}{2^n}$$

In above relations: $\tau(l) = \lceil (l+1)/b \rceil + 1 = L$ is the number of blocks after applying RO-ESh-pad function on an l-bit message M, T_h is the time to compute compression function h, q_{RO_1} and q_{RO_2} are, respectively, the number of queries from $RO_1(.)$ and RO_2 oracles.

□

4.2 Without Any Random Oracle

As it was shown in analysis of ESh, as a *standard model* transform, the four properties, namely; Pre, aPre, Sec, and aSec are not preserved by ESh. *It appears to be a crux to preserve these four properties (simultaneously) in the standard model.*

Sec. Andreeva and Preneel in SAC 2008 [2] proposed a keyed transform to extend the domain of a keyless compression function. The proposed transform yields to a dedicated-key VIL hash function which is CR and Sec secure (where CR and Sec notions are defined for a dedicated-key hash function) based on the assumptions that underlying keyless compression function is, respectively, CR or Sec (here CR and Sec are defined for a keyless compression function). The setting is called "keyless compression function - keyed iteration"' in [2]. We note that the nature of Sec notion for the keyless compression function and that of keyed hash function is different. Unfortunately the proposed scheme cannot be shown to be Pre secure in standard model and only a random oracle argument is provided in [2] for its security in Pre sense .

Pre. To the best of our knowledge, the only transform in the standard model that is pointed out in [1] to be Pre preserving (in the dedicated-key hash setting) is the XOR Tree scheme, but it does not preserve aPre and aSec as shown in [1].

aSec and aPre. There is currently no transform in the literature that can *preserve* aSec and aPre in the standard model. Such an aSec and aPre preserving transform in dedicated-key hash setting will also yield to a construction in keyless hash setting capable of preserving the Second Preimage Resistance and Preimage Resistance (a.k.a One-wayness) of the underlying keyless compression function.

Remark. It is worth reminding that we call a hash domain extension transform (either in dedicated-key or keyless settings) capable of preserving a security property P (defined either for a dedicated-key or keyless hash function) if the constructed VIL hash function provably possesses the property P assuming that the underlying compression function satisfies the same property P. This is different from a scenario where one proves that the VIL hash function has property P if its underlying compression function satisfies a different property P$'$, e.g. [18], or a collection of different assumptions, e.g. [10, 9].

5 Conclusion

In this paper, we analyzed two recently proposed MPP hash domain extension transforms, namely the Random-Oracle-XOR (ROX) transform and the Enveloped Shoup (ESh) transform. We showed that ROX *does not preserve* MAC and PRO notions, but it preserves PRF. We also showed that ESh *does not preserve* Sec, aSec, Pre, and aPre, but it preserves ePre. Our results complete the MPP analysis of both ROX and ESh transforms in regard to all ten security notions of interest, namely CR, Sec, aSec, eSec (TCR), Pre, aPre, ePre, MAC, PRF, PRO, and provide the full picture of their MPP capabilities.

References

[1] Andreeva, E., Neven, G., Preneel, B., Shrimpton, T.: Seven-Property-Preserving Iterated Hashing: ROX. In: Kaoru Kurosawa (ed.): ASIACRYPT 2007. LNCS, vol. 4833, pp. 130–146. Springer (2007)

[2] Andreeva, E., Preneel, B.: A Three-Property-Secure Hash Function. In: Avanzi, R., Keliher, L., Sica, F. (eds.): SAC 2008. Workshop Records, pp. 208–224. (2008)

[3] Bellare, M., Ristenpart, T.: Hash Functions in the Dedicated-Key Setting: Design Choices and MPP Transforms. In Arge, L., Cachin, C., Jurdzinski, T., Tarlecki, A. (eds.): ICALP 07. LNCS, vol. 4596, pp. 399–410. Springer (2007)

[4] Bellare, M., Ristenpart, T.: Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In Lai, X., Chen, K. (eds.): ASIACRYPT 2006. LNCS, vol. 4284, pp. 299–314. Springer (2006)

[5] Bellare, M., Ristenpart, T.: Hash Functions in the Dedicated-Key Setting: Design Choices and MPP Transforms. Cryptology ePrint Archive, Report 2007/271, 2007. http://eprint.iacr.org/.

[6] Bellare, M., Rogaway, P.: Collision-Resistant Hashing: Towards Making UOWHFs Practical. In Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 470–484. Springer (1997)

[7] Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård Revisited: How to Construct a Hash Function. In Shoup, V. (ed.): CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer (2005)

[8] Damgård, I.: A Design Principle for Hash Functions. In Brassard, G. (ed.): CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer (1990)

[9] Dodis, Y., Puniya, P.: Getting the Best Out of Existing Hash Functions; or What if We Are Stuck with SHA?. In: Bellovin, S. M., Gennaro, R., Keromytis, A. D., Yung, M. (eds.): ACNS 2008. LNCS, vol. 5037, pp. 156–173. Springer (2008)

[10] Halevi, S., Krawczyk, H.: Strengthening Digital Signatures Via Randomized Hashing. In: Dwork, C. (ed.): CRYPTO 2006. LNCS, vol. 4117, pp. 41–59. Springer (2006)

[11] Maurer, U. M., Renner, R., Holenstein, C.: Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Naor, M. (ed.): TCC 2004. LNCS, vol. 2951, pp. 21–39. Springer (2004).

[12] Merkle, R.C.: One Way Hash Functions and DES. In Brassard, G. (ed.): CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer (1990)

[13] Mironov, I.: Collision-Resistant No More: Hash-and-Sign Paradigm Revisited. In Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.): PKC 2006. LNCS, vol. 3958, pp. 140–156. Springer (2006)

[14] Mironov, I.: Hash Functions: From Merkle-Damgård to Shoup. In Pfitzmann, B. (ed.): EUROCRYPT 2001. LNCS, vol. 2045, pp. 166–181. Springer (2001)

[15] Naor, M., Yung, M.: Universal One-Way Hash Functions and Their Cryptographic Applications. In: STOC 1989, pp. 33–43. ACM (1989)

[16] Rogaway, P., Shrimpton, T.: Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Roy, B.K., Meier, W. (eds.): FSE 2004. LNCS, vol. 3017, pp. 371–388. Springer (2004)

[17] Shoup, V.: A Composition Theorem for Universal One-Way Hash Functions. In: Preneel, B. (ed.): EUROCRYPT 2000. LNCS, vol. 1807, pp. 445–452. Springer (2000)

[18] Yasuda, K.: How to Fill Up Merkle-Damgård Hash Functions. In: Pieprzyk, J. (ed.): ASIACRYPT 2008. LNCS, vol. 5350, pp. 272–289. Springer (2008)

A Proof of Lemma 1

A.1 The case of h_1

We want to show that if g is (t, ϵ)-xxx then h_1 is $(t, \epsilon + 2^{-\lambda})$-xxx, where xxx $\in$ {Sec, aSec, Pre, aPre}. Let A be an adversary attacking h_1 in xxx sense with time complexity t' and advantage ϵ', we construct an

adversary B that can attack g in the xxx sense with the same time complexity as A, i.e. $t = t'$ and with advantage $\epsilon = \epsilon' - 2^{-\lambda}$. Briefly saying, adversary B plays in xxx game by using A as a subroutine. B simply forwards challenges to A and after receiving responses from A simply outputs them as its own responses. As we show for all cases below, B wins the xxx game whenever A wins ***and*** a specific event, called **Bad**, does not happen. **Bad** is defined as the event that $M_{n+b-\lambda+1\ldots n+b} = \langle\delta\rangle_\lambda$ in which case h_1 always outputs '0' irrespective of the output value of g (refer to the definition of the counterexample h_1). Clearly for a random message $M \in \{0,1\}^{n+b}$ we have $\Pr[\mathbf{Bad}] = 2^{-\lambda}$. For definitions of Sec, aSec, Pre and aPre games refer to Fig. 1. The details of reductions are as follows:

The case of xxx=Sec: Adversary B on receiving the key K and the first message M (where $K \xleftarrow{\$} \{0,1\}^k$ and $M \xleftarrow{\$} \{0,1\}^{n+b}$) checks whether **Bad** has happened, i.e. $M_{n+b-\lambda+1\ldots n+b} = \langle\delta\rangle_\lambda$ or not. If **Bad** happens then B returns 'Fail' and **aborts** here, otherwise it forwards K and M to A, gets the second preimage M' from A for h_1, and outputs M' as its own second preimage for g. We note that if **Bad** does not happen, then according to the construction of h_1 adversary B wins (i.e. M' is a second preimage for M under g) whenever A wins (i.e. if M' is a second preimage for M under h_1). Hence, we have Pr[B wins]=Pr[A wins and $\overline{\mathbf{Bad}}$] $\geq$ Pr[A wins]-Pr[**Bad**] $= \epsilon' - 2^{-\lambda}$. The time complexity of B is the same as A, as it just runs A by forwarding the messages.

The Case of xxx=aSec: Adversary B runs A, receives the key K from it and outputs K as its own key in the first stage of aSec game. On receiving the first message M (where $M \xleftarrow{\$} \{0,1\}^{n+b}$), B checks whether **Bad** has happened, i.e. $M_{n+b-\lambda+1\ldots n+b} = \langle\delta\rangle_\lambda$ or not. If **Bad** happens then B returns 'Fail' and **aborts** here, otherwise it sends M to A, receives the second preimage M' from A for h_1, and outputs M' as its own second preimage for g. It is easy to see from the construction of h_1 that if **Bad** does not happen then adversary B wins (i.e. M' is a second preimage for M under g) whenever A wins (i.e. if M' is a second preimage for M under h_1). Hence, we have Pr[B wins]=Pr[A wins and $\overline{\mathbf{Bad}}$] $\geq$ Pr[A wins]-Pr[**Bad**] $= \epsilon' - 2^{-\lambda}$. The time complexity of B is the same as A, as it just runs A and forwarding the messages.

The Case of xxx=Pre: Adversary B on receiving the key K and the hash value Y (where $K \xleftarrow{\$} \{0,1\}^k$, $M \xleftarrow{\$} \{0,1\}^{n+b}$ and $Y = g(K, M)$), runs A by sending the same key K together with $Y' = Y||1$ (as the hash value under h_1) to A. On receiving the preimage M' (for Y' under h_1) from A, adversary B returns M' as its own preimage (for Y under g). It is easy to see from the construction of h_1 that if **Bad** does not happen (i.e. $M_{n+b-\lambda+1\ldots n+b} \neq \langle\delta\rangle_\lambda$) then B wins (i.e. M' is a preimage for Y under g) whenever A wins (i.e. if M' is a preimage for $Y' = Y||1$ under h_1). Hence, we have Pr[B wins]=Pr[A wins and $\overline{\mathbf{Bad}}$] $\geq$ Pr[A wins]-Pr[**Bad**] $= \epsilon' - 2^{-\lambda}$. The time complexity of B is the same as A, as it just runs A by forwarding the messages.

The Case of xxx=aPre: Adversary B runs A, receives the key K from it and outputs K as its own key in the first stage of aPre game. When B receives the (challenge) hash value Y (where $M \xleftarrow{\$} \{0,1\}^{n+b}$ and $Y = g(K, M)$) it runs A by sending to it $Y' = Y||1$ (as the challenge hash value under h_1). On receiving the preimage M' (for Y' under h_1) from A, adversary B returns M' as its own preimage (for Y under g). It can be seen from the construction of h_1 that if **Bad** does not happen (i.e. $M_{n+b-\lambda+1\ldots n+b} \neq \langle\delta\rangle_\lambda$) then B wins (i.e. M' is a preimage for Y under g) whenever A wins (i.e. if M' is a preimage for $Y' = Y||1$ under h_1). Hence, we have Pr[B wins]=Pr[A wins and $\overline{\mathbf{Bad}}$] $\geq$ Pr[A wins]-Pr[**Bad**] $= \epsilon' - 2^{-\lambda}$. The time complexity of B is the same as A, as it just runs A by forwarding the messages.

A.2 The case of h_2

The proof for this case (that is, to show that if g is (t, ϵ)-xxx then h_2 is $(t, \epsilon + 2^{-(b-n)})$-xxx, where xxx $\in$ {Sec, aSec, Pre, aPre}) is almost the same as the previously proved case for h_1 and the only difference is that now in this case the event **Bad** is defined as the event that $M_{2n+1\ldots n+b} = 0^{b-n}$ in which case h_2 always

outputs '0' irrespective of the output value of g (refer to the definition of the counterexample h_2). Clearly for a random message $M \in \{0,1\}^{n+b}$ we have $\Pr[\mathbf{Bad}] = 2^{-(b-n)}$.