Security Proofs for Hash Tree Time-Stamping Using Hash Functions with Small Output Size

Ahto Buldas[1,2,3] and Risto Laanoja[1,2,⋆]

[1] GuardTime AS, Tammsaare tee 60, 11316 Tallinn, Estonia
[2] Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia
[3] Cybernetica AS, Mealuse 2/1, 12618 Tallinn, Estonia

Abstract. The known security proofs for hash tree time-stamping assume collision-resistance (CR). An asymptotically optimally tight proof has the security loss formula $\frac{t'}{\delta'} \approx 14\sqrt{C}\left(\frac{t}{\delta}\right)^{1.5}$, where $\frac{t'}{\delta'}$ is the time-success ratio of a collision-finder, $\frac{t}{\delta}$ is the ratio of a back-dating adversary and C is the size of the hash tree created in every time unit. Practical schemes use 256-bit hash functions that are just 2^{128}-secure because of the birthday bound. For having a 2^{80}-secure time-stamping scheme, we have $C < 10^3$ that is insufficient for global scale solutions. Due to tightness bounds for CR, practically relevant security proofs must use assumptions stronger than CR. We show that under the random oracle (RO) assumption, the security loss is independent of C. We establish a linear-preserving security reduction under the Pre-Image Awareness (PrA) assumption. We present a new slightly stronger assumption SPrA that leads to much tighter proofs. We also show that bounds on C are necessary—based on any PrA/SPrA function, we construct a PrA/SPrA function that is insecure for unbounded time-stamping.

1 Introduction

Hash tree (keyless) time-stamping was first introduced by Haber and Stornetta [11] in order to eliminate secret-based cryptography and trusted third parties. In their scheme, a collection of N documents is hashed down to a single digest of few dozen bytes that is then published in widely available media such as newspapers. Merkle hash trees [14] enable to create compact certificates (of size $\log N$) for each of the N documents. Such certificates just consist of all sibling hash values in the path from a document (a leaf of the tree) to the root of the tree. The certificate is sufficient to re-compute the root hash value from the document and can be used as a *proof of membership*. Haber and Stornetta drafted a large-scale time-stamping scheme [1] where at every unit of time a large hash tree is created co-operatively by numerous servers all over the world and the root value is published in newspapers.

⋆ This research was supported by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, and by Estonian Research Council's personal research grant PUT-2.

It might seem obvious that the security of a hash-then-publish time-stamping scheme can be reduced to collision-resistance of the hash function. However, the first correct security proof of such a scheme was published as late as in 2004 [6]. It turned out that the potential number N of time-stamps explicitly affects the tightness of security proofs. The proof in [6] shows that if there is an adversary with running time t that backdates a document with probability δ, then there is a collision-findiner with running time $t' \approx 2t$ and success probability $\delta' \approx \frac{\delta^2}{N}$. When measuring security in terms of time-success ratio [12] we have to use $2N \cdot \frac{t}{\delta^2}$-collision resistant hash functions to have a $\frac{t}{\delta}$-secure time-stamping scheme, i.e. the hash function must be roughly $\frac{2N}{\delta}$ times more secure than the time-stamping system constructed from it. As N could be very large in considering global-scale time-stamping, the security requirements for the hash function may grow unreasonably large. Indeed, it is said in [6] that such a security proof is practical only for hash functions with about 400-bit output.

The tightest known security proof [4] for hash tree time-stamping has the security loss formula $\frac{t'}{\delta'} \approx 14\sqrt{N}\frac{t}{\delta^{1.5}}$, which is insufficient because the security level it implies for systems that use 256-bit hash functions is not that one expects today. In order to have a 2^{80}-secure system with the total capacity of N times-tamps, we need $n = \log_2 N + 248$ output bits. If $n = 256$, then $N < 2^8 = 256$, which is clearly too small. Moreover, the proof of [4] is asymptotically optimally tight, if the collision-resistance property is used as the security assumption. So, the only way out is to use stronger (or incomparable) security assumptions for hash functions. In this paper, we first show that if the hash function is assumed to be a random oracle, the security loss does not depend on N. Next, we establish a linear-preserving reduction that assumes the hash functions to be pre-image aware (PrA) constructions from ideal components. We also present a new slightly stronger than PrA security condition (SPrA) under which the security proof is much tighter than all previous ones and is very close to the tightness in the random oracle model. Finally, we show that the bounded capacity N is a necessary assumption to prove the security of hash-tree time-stamping schemes under the PrA/SPrA assumption. Based on arbitrary PrA/SPrA hash function, we construct another PrA/SPrA hash function that is totally insecure for unbounded time-stamping. This negativity result is a generalization of a somewhat weaker oracle-separation based result [6] presented in Asiacrypt 2004 about the proofs that use collision-resistance.

2 Preliminaries

2.1 Security Proofs and Their Tightness

The security of a cryptographic protocol (or a primitive) is measured by the amount of resources (such as running time) needed for an adversary to break the primitive. A protocol is said to be S-secure, if it can be broken by no adversaries with less than S units of resources available. Considering that the running time t and the success probability δ of the known practical attacks against the protocol may vary, Luby [12] proposed the *time-success ratio* $\frac{t}{\delta}$ as a universal security measure. This means that a protocol is S-secure, if the success probability of any t-time adversary does not exceed $\frac{t}{S}$.

In a typical security proof for a protocol $\mathcal{P}$ built from a primitive $\mathcal{Q}$, it is shown that if $\mathcal{Q}$ is S_q-secure, then $\mathcal{P}$ is S_p-secure. Bellare and Rogaway [2,3] first emphasized the importance of studying the *tightness* of security proofs in practical applications. Informally, tightness shows how much security of the primitive is transformed to the protocol. Numerically, we may express tightness as the ratio S_p/S_q. The notion opposite to tightness is *security loss*.

Security proofs are often presented as *reductions*, i.e. we assume that we have an adversary for the protocol $\mathcal{P}$ with running time t and success probability δ and then construct an adversary for the primitive $\mathcal{Q}$ with running time t' and success probability δ'. This means that for having a protocol that is secure against adversaries with running time t and with success probability δ, we have to use a $\frac{t'}{\delta'}$-secure primitive. The ratio $\frac{t'}{\delta'}$ is a function of t and δ, but not always the function of $\frac{t}{\delta}$. For example, if $\frac{t'}{\delta'} = c \cdot \frac{t}{\delta^2}$, then there is no dependence of type $\frac{t'}{\delta'} = F\left(\frac{t}{\delta}\right)$. However, due to $t \geq 1$, we have an inequality $\frac{t'}{\delta'} \leq c \cdot \left(\frac{t}{\delta}\right)^2$.

2.2 Security Properties of Hash Functions

In this paper, we study the security properties of hash functions H^P that use some kind of ideal functionality P (random permutations, random functions, ideal ciphers, etc.) as an oracle. For example, in case of the Merkle-Damgård hash functions, the compression function and the output transform are often assumed to be ideal objects. In this section, we describe some of the properties of hash functions, starting from the strongest ones.

Random- and Pseudorandom Oracles. By a *fixed length (FIL-) random oracle* $\mathcal{R}$, we mean a function that is chosen randomly from the set of all functions of type $\{0,1\}^m \to \{0,1\}^n$. There are $(2^n)^{2^m} = 2^{n2^m}$ possible choices of $\mathcal{R}$. By a *variable length (VIL-) random oracle* $\mathcal{V}$, we mean a function that is chosen randomly from the set Ω of all functions of type $\{0,1\}^* \to \{0,1\}^n$. The probability distribution of $\mathcal{V}$ is defined so that for any fixed input length m, the restriction of $\mathcal{V}_m$ to $\{0,1\}^m$ is distributed like a FIL-random oracle.

By the *random oracle heuristic* we mean a security argument when an application of a hash function (e.g. a time-stamping scheme, a signature scheme) is proved to be secure in the so-called *random oracle model*, where the hash function is replaced with a VIL-random oracle. The random oracle heuristic was first introduced by Bellare and Rogaway [2]. Although it was proved later by Canetti et al [7] that the random oracle heuristic fails in certain theoretical cases, proofs in the random oracle model are still considered as valuable security arguments, especially if no better security proofs are known.

Definition 1 (PRO). *We say that H^P is a* pseudo-random oracle (PRO) *if there is an efficient simulator $S^{\mathcal{V}}$, such that for every efficient distinguisher D, the following difference is negligible:*

$$\left| \Pr\left[1 \leftarrow D^{H^P,P}\right] - \Pr\left[1 \leftarrow D^{\mathcal{V},S^{\mathcal{V}}}\right] \right| \quad .$$

This notion is first studied by Maurer et al [13] and was adapted to hash functions by Coron et al [9]. The most important practical implication of the pseudo-random oracle property of H^P is that any application (e.g. a time-stamping scheme) that uses H^P as a hash function is almost as secure as if a variable length random oracle $\mathcal{V}$ is used instead of H^P. This means that the *random oracle heuristics* applies in case of the particular application, i.e. we can prove the security of the application in the random oracle model and then replace the oracle by a more realistic (but still ideal!) model H^P of the hash function. Note that the PRO-property is a very strong assumption and often we would like to know if some lighter assumptions would also be sufficient for the security of the application. For example, it was shown [9] that the commonly used Merkle-Damgård style hash functions do not satisfy the PRO property.

Pre-image Awareness. Informally, pre-image awareness of a (hash) function H means, that if we first commit an output y and later come up with an input x, such that $y = H(x)$, then it is safe to conclude that we knew x before committing y. This notion was first formalized by Dodis et al. [10] for hash functions H^P that are built using an ideal primitive P in a black box way. For H^P being pre-image aware, there has to be an efficient deterministic algorithm $\mathcal{E}$ (the so-called *extractor*) which when given y and the list α of all previously made P-calls (by the adversary), outputs x, such that $H^P(x) = y$, or $\perp$ if $\mathcal{E}$ was unable to find such an x. The adversary tries to find x and y so that $\mathcal{E}(\alpha, y) \neq x$ and $y = H^P(x)$.

To define pre-image awareness of H^P in a precise way, we set up an experiment **Exp** (see Figure 1), specified as a game which an attacker B is trying to win. B is constrained to oracle access to P, via a wrapper oracle P, which records all P-calls made by B as an advise string α. Likely, the extractor $\mathcal{E}$ is also accessible through another wrapper oracle Ex, which uses global arrays $\mathtt{Q}$ (initially $\perp$ everywhere) and $\mathtt{V}$ (initially blank). $\mathtt{Q}$ is used to record all input parameters to $\mathcal{E}$; $\mathtt{V}$ is used to store all successfully extracted values corresponding to $\mathcal{E}$'s inputs. The adversary B tries to output a value x such that $H^P(x) = y$, $\mathtt{Q}[y] = 1$ and $\mathtt{V}[y] \neq x$, i.e. $\mathcal{E}$ tried to invert y, but was either unsuccessful ($\mathtt{V}[y] = \perp$) or found a different pre-image $x' \neq x$ (a *collision* for H^P). As P- and Ex-calls are unit cost, the running time of B does not depend on the running time of $\mathcal{E}$.

Definition 2 (Pre-image Awareness). *A function H^P is S-secure* pre-image aware *(PrA) if there is an efficient extractor $\mathcal{E}$, so that for every t-time B:*

$$\mathbf{Adv}^{\mathrm{pra}}_{H,P,\mathcal{E}}(B) = \Pr\left[1 \leftarrow \mathbf{Exp}^{\mathrm{pra}}_{H,P,\mathcal{E},B}\right] \leq \frac{t}{S} \quad . \tag{1}$$

It is easy to see that pre-image awareness of H^P implies collision-resistance of H^P. Hence, as there is the so-called birthday bound for the collision-resistance which says that no function with n-bit output can be more than $2^{n/2}$-secure, we conclude that no function with n-bit output can be more than $2^{n/2}$-secure pre-image aware. The Pre-image awareness property alone is not sufficient for proving the PRO property. However, together with some additional assumptions this has been proved to be possible [10,8].

```
$\mathbf{Exp}^{\mathrm{pra}}_{H,P,\mathcal{E},B}$:
$x \leftarrow B^{\mathsf{P},\mathsf{Ex}}$
$y \leftarrow H^P(x)$
If $\mathtt{Q}[y] = 1$ and $\mathtt{V}[y] \neq x$ return 1,
else return 0
```

```
oracle $\mathsf{P}(m)$:
$c \leftarrow P(m)$
$\alpha \leftarrow \alpha || (m, c)$
return $c$
```

```
oracle $\mathsf{Ex}(y)$:
$\mathtt{Q}[y] \leftarrow 1$
$\mathtt{V}[y] \leftarrow \mathcal{E}(y, \alpha)$
return $\mathtt{V}[y]$
```

Fig. 1. Preimage awareness experiment and the wrapper oracles

Collision Resistance. Informally, the collision resistance of a hash function H^P means that it is infeasible for adversaries to find two different inputs x and x' that have the same hash value, i.e. $H^P(x) = H^P(x')$. This definition makes sense only if the ideal primitive P contains some randomness, because for fixed functions, there always exist collisions that can be "wired" into the adversary.

Definition 3 (Collision Resistance). *A function H^P is S-secure* collision resistant *(CR) if for every adversary B with running time t:*

$$\mathbf{Adv}^{\mathrm{cr}}_{H,P}(B) = \mathsf{Pr}\left[x, x' \leftarrow B^P : x \neq x',\ H^P(x) = H^P(x')\right] \leq \frac{t}{S} \ . \tag{2}$$

Note that due to the so-called Birthday attack, functions with n-bit output can only be $2^{\frac{n}{2}}$-secure collision resistant.

2.3 Hash-Tree Time-Stamping Schemes and Their Security

Hash trees were introduced by Merkle [14]. Let $h\colon \{0,1\}^{2n} \to \{0,1\}^n$ be a hash function. By a *hash-tree* we mean a tree-shaped data structure that consists of nodes each of which contains an n-bit hash value. Each node is either a *leaf* which means it has no child nodes, or an *internal node* that has two child nodes (the left- and the right child) whereas the hash value y of an internal node is computed as a hash $y = h(y_0, y_1)$, where y_0 and y_1 are the hash values of the left- and the right child, respectively. There is one *root node* that is not a child of any other node. If $\mathcal{T}$ is a hash tree with m leaves with hash values $x_1, \ldots, x_m$ and r is the hash value of the root node, then we use the notation $r = \mathcal{T}(x_1, \ldots, x_m)$.

Encoding the Leaves of a Hash Tree. Each node in a hash tree can be *naturally named* by using finite bit-strings in the following way. The root node is named by the empty string $\sqcup$. If a node is named by a bit-string ℓ, then its left- and right child nodes are named by $\ell 0$ and $\ell 1$, respectively. The name ℓ of a node resembles an *address* of the node, considering that one starts the searching process from the root node, and then step by step, chooses one of the child nodes depending on the corresponding bit in ℓ, i.e. 0 means "left" and 1 means "right".

Shape of a Hash Tree. Hash tree has a particular *shape* by which we mean the set of all names of the leaf-nodes. For example a balanced complete tree with four nodes (Fig. 2, left) has shape $\{00, 01, 10, 11\}$. If the root hash value is denoted by r (instead of $r_{[]}$) and r_ℓ denotes the hash value of a node with name ℓ, then in this example, the relations between the nodes are the following: $r = h(r_0, r_1)$, $r_0 = h(r_{00}, r_{01})$, and $r_1 = h(r_{10}, r_{11})$. The shape $\{000, 001, 01, 1\}$ represents a totally unbalanced tree with four leaves (Fig. 2, right), with the hash values being in the following relations: $r = h(r_0, r_1)$, $r_0 = h(r_{00}, r_{01})$, and $r_{00} = h(r_{000}, r_{001})$. Note also that the shape is always a *prefix-free code.*

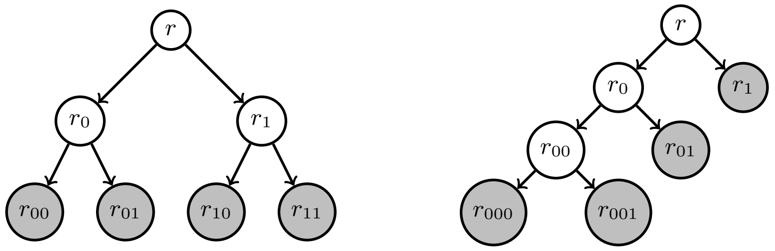

Fig. 2. A balanced tree (left) and an unbalanced tree (right)

Hash Chains. In order to prove that a hash value r_ℓ (where $\ell_1\ell_2 \ldots \ell_m$ is the binary representation of ℓ) participated in the computation of the root hash value r, it is sufficient to present all the sibling hash values of the nodes on the unique path from r_ℓ to the root r. For example, in the balanced tree with four leaves (Fig. 2, left), to prove that r_{01} belongs to the tree, we have to show the sibling hash values r_{00} and r_1, which enable a verifier to compute $r_0 = h(\underline{r_{00}}, r_{01})$ and $r = h(r_0, \underline{r_1})$. In general, we define a hash chain as follows:

Definition 4 (Hash-chain). *A* hash-link *from x to r (where $x, r \in \{0,1\}^n$) is a pair (s, b), where $s \in \{0,1\}^n$ and $b \in \{0,1\}$, such that either $b = 0$ and $r = h(x\|s)$, or $b = 1$ and $r = h(s\|x)$. A* hash-chain *from x to r is a (possibly empty) list $c = ((s_1, b_1), \ldots, (s_m, b_m))$, such that either $c = ()$ and $x = r$; or there is a sequence $x_0, x_1, \ldots, x_m$ of hash values, such that $x = x_0$, $r = x_m$, and (s_i, b_i) is an hash-link from x_{i-1} to x_i for every $i \in \{1, \ldots, m\}$. We denote by $x \overset{c}{\rightsquigarrow} r$ the proposition that c is a hash chain from x to r. Note that $x \overset{()}{\rightsquigarrow} x$ for every $x \in \{0,1\}^n$. By the* shape *$\ell(c)$ of c we mean the m-bit string $b_1b_2 \ldots b_m$.*

Hash-Tree Time-Stamping Schemes. The time-stamping procedure runs as follows. During every time unit t (e.g. one second) the time-stamping server receives a list $\mathcal{X}_t = (x_1, \ldots, x_m)$ of requests (n-bit hash values) from clients, computes the root hash value $r_{(t)} = \mathcal{T}(x_1, \ldots, x_m)$ of a hash tree $\mathcal{T}$ and publishes $r_{(t)}$ in a public repository $\mathcal{R} = (r_{(1)}, r_{(2)}, \ldots, r_{(t)})$ organized as an append-only list. Each request x_i is then provided with a hash chain c_i (the *time stamp* for x_i) that proves the participance of x_i in the computation of the root hash value

$r_{(t)}$. A request $x \in \mathcal{X}_t$ is said to precede another request $x' \in \mathcal{X}_{t'}$ if $t < t'$. The requests of the same batch are considered simultaneous. In order to verify the time stamp c_i of a request x_i, one computes the output hash value of c_i (the last hash value x_m in the sequence) and checks whether $x_m = r$.

Bounded, Unbounded and Shape-Compact Time-Stamping Schemes. It was first mentioned by Buldas et al [6] that for proving the security of time-stamping schemes, there must be restrictions on the shape $\ell(c)$ of the hash chains that are acceptable in time stamps. In general, we denote the set of allowed shapes by $\mathcal{S}$. If $\mathcal{S}$ is finite then there is always a binary tree $\mathcal{T}(\mathcal{S})$ such that for any $\ell \in \mathcal{S}$ it contains a node with name ℓ. For example, if $\mathcal{S} = \{0, 000, 01, 1, 11, 100\}$, the smallest such binary tree (assuming that every non-leaf node has two children) is depicted in Fig. 3. We say that a time-stamping scheme is *unbounded*,

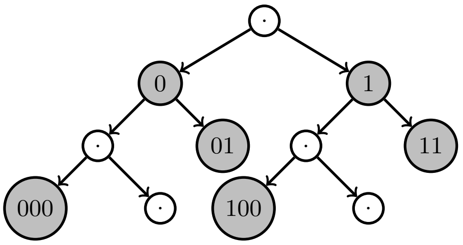

Fig. 3. The smallest binary tree induced by $\mathcal{S} = \{0, 000, 01, 1, 11, 100\}$

if $\mathcal{S} = \{0,1\}^*$. We say that a hash-tree time-stamping scheme is *C-bounded* if $|\mathcal{S}| \leq C$. A C-bounded hash-tree time-stamping scheme is said to be *shape-compact* if the tree $\mathcal{T}(\mathcal{S})$ induced by allowed shapes has no more than $2C$ vertices and $|\ell| \leq 2\log_2 C$ for every $\ell \in \mathcal{S}$, where $|\ell|$ denotes the bit length of ℓ.

Security of Time-Stamping. Informally, we want that no efficient adversary can *back-date* any request x, i.e. first publishing a hash value r, and only after that generating a new "fresh" x (not pre-computed by the adversary), and a hash chain c, so that $x \overset{c}{\rightsquigarrow} r$. To formalize the security condition, we use the so-called *entropy-based security* condition [5] inspired by the following attack-scenario with a two-stage adversary $A = (A_1, A_2)$ cooperating with the server. At the first stage A_1 the adversary forces server to create a public repository $\mathcal{R}$ of commitments. Note that there is no guarantee that the hash values in $\mathcal{R}$ are created in the proper way, i.e. by using the hash tree. After that, the second stage A_2 of the adversary presents a high-entropy (unpredictable) x and a hash chain c so that $x \overset{c}{\rightsquigarrow} r$ for an $r \in \mathcal{R}$. The unpredictability of x is crucial because otherwise x could have been pre-computed (guessed) by A before r is published and hence x could be in fact older than r and thereby not back-dated by A.

Hence, for defining the security of time-stamping schemes, the class of possible adversaries is restricted. Only *unpredictable* adversaries that produce unpredictable x are considered, i.e. the output component x is assumed to be unpredictable even if the contents of $\mathcal{R}$ and all the internal computations of the adversary (while computing $\mathcal{R}$) are known. The original security definition from [5] is somewhat inconvenient to use for exact security estimations, because it extensively uses the polynomial security model. In this paper, we slightly weaken the adversary by assuming the so-called *strong unpredictability.* Intuitively, the strong unpredictability means that $x \in \{0,1\}^n$ is almost identically distributed, i.e. its conditional min-entropy $H_\infty[x \mid \mathcal{R}, a]$ must be at least $n-1$ bits, i.e. for every input of A_2 and for any possible value x_0 of x, the probability of $x = x_0$ is upper bounded by $\frac{1}{2^{n-1}}$. For practical justification of such an assumption, note that in practical applications, x is mostly a cryptographic hash of a (much) longer document X that contains a considerable amount of new (fresh) information. Cryptographic hash functions are assumed to be good entropy-extractors, and hence the assumption of strong unpredictability is practically reasonable.

Definition 5 (Security against back-dating). *A time-stamping scheme is S-secure against back-dating if for every t-time strongly unpredictable (A_1, A_2):*

$$\mathsf{Pr}\left[(\mathcal{R}, a) \leftarrow A_1, (x, c) \leftarrow A_2(\mathcal{R}, a)\colon\ x \overset{c}{\leadsto} \mathcal{R},\ \ell(c) \in \mathcal{S}\right] \leq \frac{t}{S}\ , \tag{3}$$

where by $x \overset{c}{\leadsto} \mathcal{R}$ we mean that $x \overset{c}{\leadsto} r$ for some $r \in \mathcal{R}$, and a is an advice string that contains possibly useful information that A_1 stores for A_2.

Existing Security Proofs and Their Tightness. We present the tightness parameters of two known security proofs for hash-tree time-stamping schemes: the first correct proof [6] that was presented in Asiacrypt 2004, and a tighter proof [4] from ACISP 2010 that was also proved to be asymptotically optimally tight. Both proofs assume the collision-resistance property of the hash function. Both proofs apply only to N-bounded time-stamping schemes and their tightness depends on the capacity N of the system. It was proved in [6] by using oracle separation that collision-resistance is insufficient for proving the security of *unbounded* time-stamping schemes. Both proofs are in the form of a reduction: a t-time backdating adversary with success probability δ is converted to a t'-time collision-finding adversary with success probability δ'.

In Tab. 1 we present the parameters of these two security proofs. Closer analysis shows that the parameter N used in these security proofs [6,4] can be expressed by $N = C \cdot |\mathcal{R}|$ in terms of this paper, where $\mathcal{R}$ is the hash repository created by the first stage A_1 of the back-dating adversary. As $|\mathcal{R}|$ is always upper bounded by the running time t of the adversary, we have $N \leq C \cdot t$. The third column of Tab. 1 presents the converted tightness formulae assuming $N \approx Ct$. The fourth column presents a formula for the required output size n of the hash function, assuming that we want the time-stamping scheme to be S-secure and that the hash function is secure near to the birthday barrier, i.e. n-bit hash function is assumed to be $2^{n/2}$-secure. The last column presents the output size in a particular case, where $S = 2^{80}$ (standard requirement for "commercial" security), and $C = 2^{64}$, i.e. acceptable hash chains are assumed to be no longer than 64 steps. Note that this does not mean that we assume a hash-tree with 2^{64} nodes is built in every second! The exponent 64 is only a theoretical bound for the length of a hash chain, which is quite realistic in a system of global scale with a multi-level structure of servers all over the world that compute hash trees.

The required output size 312 is too large, because one would like to use smaller hash functions such as SHA2-256 in time-stamping schemes. As the proof of ACISP 2010 is optimally tight, we have no hope to construct tighter proofs under the collision-resistance assumption. Schemes with smaller hash functions can only be proved secure under assumptions stronger than collision-resistance.

Table 1. Tightness parameters of two security proofs

Proof	Formula	Output Size Formula $n = n(C, S)$	$n(2^{64}, 2^{80})$
Asiacrypt 2004	$\frac{t'}{\delta'} \approx 2C\left(\frac{t}{\delta}\right)^2$	$n = 2\log_2 C + 4\log_2 S + 2$	448
ACISP 2010	$\frac{t'}{\delta'} \approx 14\sqrt{C}\left(\frac{t}{\delta}\right)^{1.5}$	$n = \log_2 C + 3\log_2 S + 8$	312

3 Security under RO and PrA Assumptions

Theorem 1. *If $h : \{0,1\}^{2n} \to \{0,1\}^n$ is a random oracle, then the corresponding (bounded or unbounded) hash-tree time-stamping schemes are $2^{\frac{n-1}{2}}$-secure.*

Proof. Let $A = (A_1, A_2)$ be a strongly unpredictable adversary with running time t, as described in (3). Let t_1, t_2 denote the running times of A_1 and A_2, respectively. Considering that $(\mathcal{R}, a) \leftarrow A_1$, let $R_1 \subseteq \{0,1\}^n$ be the set of all x-s so that the h-calls performed by A_1 induce a hash-chain from x to an $r \in \mathcal{R}$. Note that $\mathcal{R} \subseteq R_1$, as an empty hash-chain is always induced by any set of h-calls. We assume without loss of generality that the advice string a contains R_1. Because of strong unpredictability of A, we have $\mathsf{Pr}\left[x \in R_1\right] \le \frac{|R_1|}{2^{n-1}}$.

In case of $x \notin R_1$, in order to be successful, A_2 has to make additional h-calls so that a chain from x to $r \in \mathcal{R}$ is induced. A necessary condition that A_2 has to satisfy is that it has to find $x' = x'_1 \| x'_2$ so that $x'_1 \notin R_1$ or $x'_2 \notin R_1$ (this means that A_1 did not make h-calls with input x'), but $h(x') \in R_1$. The probability of this condition does not exceed $t_2 \frac{|R_1|}{2^n}$, hence, considering that $|\,R_1\,| \le 2t_1$ and $t_1, t_2 \ge 1$, the overall success probability δ of A can be estimated as follows:

$$\delta \le \frac{|\,R_1\,|}{2^{n-1}} + \left(1 - \frac{|\,R_1\,|}{2^{n-1}}\right) t_2 \frac{|\,R_1\,|}{2^n} \le \frac{2t_1}{2^{n-1}} + \frac{t_1 t_2}{2^{n-1}} \le \frac{3t_1 t_2}{2^{n-1}} \le \frac{(t_1+t_2)^2}{2^{n-1}} = \frac{t^2}{2^{n-1}} \ .$$

Hence, as $\delta^2 \le \delta \le \frac{t^2}{2^{n-1}}$, we have $\frac{t}{\delta} \ge 2^{\frac{n-1}{2}}$. □

Corollary 1. *Hash-tree time-stamping schemes (bounded/non-bounded) are S-secure in the RO model if they use hash functions with $2\log_2 S + 1$ output bits.*

Theorem 2. *If $H^P\colon \{0,1\}^{2n} \to \{0,1\}^n$ is a hash-function built from an ideal primitive P that is S-secure PrA, then the corresponding C-bounded shape-compact hash-tree time-stamping schemes are $\frac{S}{2C}$-secure against strongly unpredictable adversaries with running time $t \ll \frac{2^n}{C}$.*

Proof. Due to the PrA assumption there exists an efficient extractor $\mathcal{E}$. Let $A^P = (A_1^P, A_2^P)$ be a strongly unpredictable back-dating adversary with running time $t \ll 2^n/C$ and success probability δ.

We construct a PrA-adversary $B^{\mathsf{P},\mathsf{Ex}}$ that first simulates $(\mathcal{R}, a) \leftarrow A_1^P$ so that all P-calls of are executed through the P-oracle. After that, for every $r \in \mathcal{R}$, the adversary builds a hash-tree by using the Ex-oracle in the following way. It calls $z \leftarrow \mathsf{Ex}(r)$ and if $z = \perp$, then no more extractions are performed. If $z \neq \perp$ (this means $r = H^P(z)$), then B assigns $r_0 = z_{1\ldots n}$ and $r_1 = z_{n+1\ldots 2n}$, i.e. r_0 equals to the first n bits of z and r_1 equals to the last n bits of z. The same procedure is then applied to r_0 and r_1, etc. until the whole hash-tree $\mathcal{T}(\mathcal{S})$ (induced by the set $\mathcal{S}$ of allowed shapes) is filled with hash values. If the extractor fails in some branches of the tree, the extraction procedure is stopped in this branch but is continued in other branches. For example, in case $\mathcal{S} = \{00, 01, 10, 11\}$ the adversary tries to extract $(r_0, r_1) \leftarrow \mathsf{Ex}(r)$, $(r_{00}, r_{01}) \leftarrow \mathsf{Ex}(r_0)$, $(r_{10}, r_{11}) \leftarrow \mathsf{Ex}(r_1)$. Due to the shape-compactness, the Ex-oracle is called no more than $C \cdot |\mathcal{R}| \leq Ct$ times.

Finally, B simulates A_2^P so that all its P calls are executed through the P-oracle. With probability δ we obtain a hash value x and a hash chain c such that $\ell(c) \in \mathcal{S}$ and $x \overset{c}{\rightsquigarrow} r$ for some $r \in \mathcal{R}$. Due to the strong unpredictability of A and $Ct \ll 2^n$, the probability that x coincides with some of the extracted hash values r_ℓ is upper bounded by $\frac{2Ct}{2^{n-1}}$ which is negligible. Hence, with probability almost δ, we have a hash value x that is not in the extracted tree, and still there is a hash chain $c = \{(c_1, b_1), (c_2, b_2), \ldots, (c_m, b_m)\}$ with output hash value r that certainly is in the extracted tree. Let $x_0, x_1, \ldots, x_m$ be the intermediate hash values (outputs of hash links) as described in Def. 4. Let k be the smallest index such that x_{k-1} is not in the extracted tree but x_k is. For such k,

$$\begin{cases} \mathsf{H}^\mathsf{P}(c_k \| x_{k-1}) = x_k \text{ and } \mathsf{Ex}(x_k) \neq (c_k \| x_{k-1}) & \text{if } b_k = 0 \ ; \\ \mathsf{H}^\mathsf{P}(x_{k-1} \| c_k) = x_k \text{ and } \mathsf{Ex}(x_k) \neq (x_{k-1} \| c_k) & \text{if } b_k = 1 \ . \end{cases}$$

The adversary B outputs $(c_k \| x_{k-1})$ if $b_k = 0$ or $(x_{k-1} \| c_k)$ if $b_k = 1$. Hence, an adversary B with approximate running time $t' = Ct + t \leq 2Ct$ wins the PrA experiment with probability $\delta' \approx \delta$. Hence, $\frac{t}{\delta} \geq \frac{t'}{2C\delta'} \geq \frac{S}{2C}$. □

Analysis. Tightness of security proofs under the CR, PrA, and RO assumptions is compared in Tab. 2. In case of PrA, we assume that the hash function is about $2^{n/2}$-secure, which is a limit because PrA implies CR and the birthday barrier applies. We see that even though PrA seems to be much stronger than CR, the required output length is not much smaller. This is because the security loss is linear in C and not in $\sqrt{C}$ as in the case of the CR assumption.

Table 2. Tightness parameters of security proofs under different assumptions

Assumption	Formula	Output Size Formula $n = n(C,S)$	$n(2^{64}, 2^{80})$
CR	$\frac{t'}{\delta'} \approx 14\sqrt{C}\left(\frac{t}{\delta}\right)^{1.5}$	$n = \log_2 C + 3\log_2 S + 8$	312
PrA	$\frac{t'}{\delta'} \approx 2C\frac{t}{\delta}$	$n = 2(\log_2 C + \log_2 S + 1)$	**290**
RO	$\frac{t}{\delta} \geq 2^{\frac{n-1}{2}}$	$n = 2\log_2 S + 1$	161

4 Strong Pre-image Awareness

By studying the proof of Theorem 2, we may see the following way to improve the tightness of the proof. Why should we extract the whole tree $T(\mathcal{S})$? Maybe we just extract the *hash chain* $x \overset{c}{\leadsto} r$ instead. This would mean that we call the Ex-oracle only $2\log_2 C$ times instead of $C\cdot|\mathcal{R}|$ times. However, this is not allowed. First, we do not know x before simulating A_2 and second, after executing A_2, we have changed the string α in the wrapper oracle. If we extract the tree $T(\mathcal{S})$ after executing A_2, then the contents of the tree (the hash values) depend on x and we cannot apply the strong unpredictability argument. The probability that x is in the tree may not be negligible any more. But, what if we still extract the tree after the execution of A_2 but use the same "old" α that was created after the simulation of A_1? The problem is that though we are able to execute the extractor $\mathcal{E}$ directly with an old α, the results will not be saved in the arrays `V` and `Q` and the adversary is unable to win the game for "technical reasons".

This inspires a new stronger notion of PrA in which "old" advise strings α can be used in those queries $\mathsf{Ex}(y)$ where y is created not later than α was formed. So, we define a new Ext-oracle that always uses the "oldest" possible α. For example, if we obtain $x \leftarrow \mathsf{Ext}(y)$ (where $x = x_1x_2$ and $x_1, x_2 \in \{0,1\}^n$) for which the oracle uses α, and later we call $\mathsf{Ext}(x_1)$, the same α is used for extraction, because the oracle remembers that x_1 was created by just "parsing" x and it is thereby as old as x and the use of α is "legal". If the ordinary Ex oracle is replaced with the Ext-oracle, then we call the corresponding security property *strong pre-image awareness (SPrA)*. For the SPrA condition to make sense for functions with variable input length, we use the notion of a *parser*. We also assume a quite natural additional property of the extractor algorithm. We use the notation $\alpha \subseteq \alpha'$ to mean that α is an initial segment of α', i.e. α' is created from α by adding some extra pairs to the top of it.

Definition 6 (Natural extractor). *An extractor $\mathcal{E}$ is* natural*, if $\alpha \subseteq \alpha'$ and $\mathcal{E}(\alpha, x) \neq \mathcal{E}(\alpha', x)$ implies $\mathcal{E}(\alpha, x) = \perp$.*

Definition 7 (n-parser). *By an n-parser Π, we mean any efficiently computable deterministic (not necessarily invertible) function that converts a finite bit-string x to a finite set $\mathcal{X}_x$ of n-bit strings.*

```
$\mathbf{Exp}^{\rm spra}_{H,P,\mathcal{E},\Pi,A}$:
$x \leftarrow A^{\mathsf{P},\mathsf{Ext}_\Pi}$
$y \leftarrow H^P(x)$
If $\mathtt{Q}[y] = 1$ and $\mathtt{V}[y] \neq x$
return 1,
else return 0
```

```
oracle $\mathsf{P}(m)$:
$c \leftarrow P(m)$
$\alpha \leftarrow \alpha || (m, c)$
return $c$
```

```
oracle $\mathsf{Ext}_\Pi(y)$:
if $\mathtt{A}[y] = \perp$ then $\mathtt{A}[y] \leftarrow \alpha$
$\mathtt{Q}[y] \leftarrow 1$
$\mathtt{V}[y] \leftarrow \mathcal{E}(y, \mathtt{A}[y])$
if $\mathtt{V}[y] \neq \perp$ and $\Pi(V[y]) = \{y_1, \ldots, y_m\}$ then for every $i = 1 \ldots m$:
    if $\mathtt{A}[y_i] = \perp$ then $\mathtt{A}[y_i] \leftarrow \mathtt{A}[y]$
return $\mathtt{V}[y]$
```

Fig. 4. Strong pre-image awareness experiment and the wrapper oracles

Definition 8 (Strong Pre-image Awareness). *A function* $H^P\colon \{0,1\}^* \to \{0,1\}^n$ *is* S*-secure* strongly pre-image aware *(SPrA) if there is an efficient natural extractor* $\mathcal{E}$*, so that for every* n*-parser* Π *and for every* t*-time* A*:*

$$\mathbf{Adv}^{\rm spra}_{H,P,\mathcal{E},\Pi}(A) = \Pr\left[1 \leftarrow \mathbf{Exp}^{\rm spra}_{H,P,\mathcal{E},\Pi,A}\right] \leq \frac{t}{S} \ . \tag{4}$$

It can be shown that random oracles are SPrA and the next theorem shows that SPrA is a property the strength of which lies between RO and PrA.

Theorem 3. *SPrA implies PrA.*

Proof. Let H^P be strongly pre-image aware with the corresponding natural extractor $\mathcal{E}$ and let B be an PrA-adversary. We define an SPrA-adversary A so that it simulates B and its P- and Ex oracle calls, ret aining their results in arrays α', $\mathtt{Q}'$ and $\mathtt{V}'$ that A maintains itself. In parallel, A calls the real oracles P and Ext that maintain the arrays $\mathtt{Q}$ and $\mathtt{V}$. For B, the simulated oracles behave exactly like P and Ex (Fig. 5). However, for some of the outputs y it might be

```
$\mathbf{Exp}^{\rm spra}_{H,P,\mathcal{E},\Pi,A}$:
$x \leftarrow A^{\mathsf{P},\mathsf{Ext}_\Pi} \equiv \begin{cases} \text{Initialize } \alpha', \mathtt{V}' \text{ and } \mathtt{Q}' \\ x \leftarrow B^{\mathsf{P}',\mathsf{Ex}'} \end{cases}$
$y \leftarrow H^P(x)$
If $\mathtt{Q}[y] = 1$ and $\mathtt{V}[y] \neq x$ return 1,
else return 0
```

```
simulated $\mathsf{P}'(m)$:
call $c \leftarrow \mathsf{P}(m)$
$\alpha' \leftarrow \alpha' || (m, c)$
return $c$
```

```
simulated $\mathsf{Ex}'(y)$:
$\mathtt{Q}'[y] \leftarrow 1$
$\mathtt{V}'[y] \leftarrow \mathcal{E}(y, \alpha')$
call $v \leftarrow \mathsf{Ext}_\Pi(y)$
return $\mathtt{V}'[y]$
```

Fig. 5. Simulation of oracles by $\mathbf{Exp}^{\rm spra}_{H,P,\mathcal{E},\Pi,A}$

that $\mathcal{E}(\alpha, y) = \mathtt{V}[y] \neq \mathtt{V}'[y] = \mathcal{E}(\alpha', y)$ because the extractions are made using different advise strings. But our construction guarantees that $\alpha \subseteq \alpha'$ and thus

$$\mathtt{V}[y] \neq \mathtt{V}'[y] \quad \Longrightarrow \quad \mathtt{V}[y] = \perp \tag{5}$$

because $\mathcal{E}$ is natural. If B is successful, i.e. finds x with output $y = H^P(x)$, such that $\mathtt{Q}'[y] = 1$ and $\mathtt{V}'[y] \neq x$, then also $\mathtt{Q}[y] = 1$ and $\mathtt{V}[y] \neq x$, because $\mathtt{V}[y] = x$ implies $\mathtt{V}[y] \neq \mathtt{V}'[y]$ which by (5) gives $x = \mathtt{V}[y] = \perp$, a contradiction. Hence, $\mathbf{Adv}^{\rm pra}_{H,P,\mathcal{E}}(B) \leq \mathbf{Adv}^{\rm spra}_{H,P,\mathcal{E},\Pi}(A)$. □

Theorem 4. *If $H^P\colon \{0,1\}^{2n} \to \{0,1\}^n$ is a hash function with ideal primitive P that is S-secure SPrA, then the corresponding C-bounded shape-compact hash-tree time-stamping schemes are $\frac{S}{4\log_2 C}$-secure against strongly unpredictable adversaries with running time $t \ll \frac{2^n}{C}$.*

Proof. Let $\mathcal{E}$ be an SPrA-extractor and $A^P = (A_1^P, A_2^P)$ be a strongly unpredictable back-dating adversary with running time $t \ll 2^n/C$ and success δ. We construct an SPrA-adversary $B^{\mathsf{P},\mathsf{Ext}}$ which first simulates $(\mathcal{R}, a) \leftarrow A_1$ and then calls $\mathsf{Ext}(r)$ for every $r \in \mathcal{R}$. This step fixes the advice string α, thus the following tree extraction will be independent of the computations of A_2. After that, B simulates $(x, c) \leftarrow A_2^P(\mathcal{R}, a)$ so that all P-calls of H are executed through the P-oracle. Note that as $Ct \ll 2^n$, the probability that x is in the extraction tree $\mathcal{T}(\mathcal{S})$ is negligible. Finally, B uses the Ext-oracle to extract the hash values along the hash chain c (there are at most $2\log_2 C$ of them). The proof is exactly like in the case of the PrA assumption, except we have a much smaller tree (i.e. a chain instead of a tree). Hence, an adversary B with approximate running time

$$t' \approx t + |\mathcal{R}| + 2\log_2 C \le 2t + 2\log_2 C \le 4t\log_2 C$$

wins the SPrA game with probability $\delta' \approx \delta$. Hence, $\frac{t}{\delta} \ge \frac{t'}{4\log_2 C\cdot\delta'} \ge \frac{S}{4\log_2 C}$. □

Tab. 3 summarizes the results. As we see, a seemingly minor and natural strengthening of PrA leads to much tighter security proofs for time-stamping schemes.

Table 3. Tightness parameters of security proofs under different assumptions

Assumption	Formula	Output Size Formula $n = n(C, S)$	$n(2^{64}, 2^{80})$
CR	$\frac{t'}{\delta'} \approx 14\sqrt{C}\left(\frac{t}{\delta}\right)^{1.5}$	$n = \log_2 N + 3\log_2 S + 8$	344
PrA	$\frac{t'}{\delta'} \approx 2C\frac{t}{\delta}$	$n = 2(\log_2 C + \log_2 S + 1)$	290
SPrA	$\frac{t'}{\delta'} \approx 4\log_2 C\frac{t}{\delta}$	$n = 2(\log_2\log_2 C + \log_2 S + 2)$	**176**
RO	$\frac{t}{\delta} \ge 2^{\frac{n-1}{2}}$	$n = 2\log_2 S + 1$	161

5 Necessity of Boundedness

The security proof in the pure random oracle model does not depend on the capacity C of the time-stamping scheme. In all other cases, the assumption about the boundedness is necessary and the tightness of the reduction depends on C. We show in this section that even under assumptions so strong as PrA, the boundedness is a necessary assumption. Namely, under the assumption that PrA hash functions exist, we construct another PrA hash function that is totally insecure for unbounded time-stamping schemes. For every $n \ge 2$ let $\mathcal{I}_n$ be a subset of $\{0,1\}^{2n}$ defined as follows:

$$\mathcal{I}_n = \{(\mathtt{0}x\mathtt{0}, \mathtt{0}x\mathtt{1})\colon x \in \{0,1\}^{n-2},\, x \ne \mathtt{0}^{n-2}\} \ .$$

Define $\iota_n : \mathcal{I}_n \to \{0,1\}^{n-1}$ so that $\iota_n(\mathtt{0}x\mathtt{0}, \mathtt{0}x\mathtt{1}) = \mathtt{0}x$ for any $x \in \{0,1\}^{n-2} \backslash \{\mathtt{0}^{n-2}\}$. Clearly, ι_n is injective and easily invertible. The computations with ι_n form a tree (with nodes of the form $x = 0x'$, where $x' \in \{0,1\}^{n-1} \backslash \{0^{n-1}\}$). The main property of this tree is that for every $x = 0x'$, where $x' \neq 0^{n-1}$ there is an easily computable hash chain from x to the root node $r = 0^{n-1}1$. For every $F^P : \{0,1\}^* \to \{0,1\}^{n-1}$ and for every extractor $\mathcal{E}: \{0,1\}^{n-1} \times \{0,1\}^* \to \{0,1\}^*$ define $F_0^P : \{0,1\}^* \to \{0,1\}^n$ and $\mathcal{E}_0 : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^*$ so that:

$$F_0^P(x) = \begin{cases} \mathtt{0} \| \iota_n(x) & \text{if } x \in \mathcal{I}_n \\ \mathtt{1} \| F^P(x) & \text{if } x \notin \mathcal{I}_n \end{cases} \quad \mathcal{E}_0(y, \alpha) = \begin{cases} \iota_n^{-1}(y') & \text{if } 0^n \neq y = 0 \| y' \in \{0,1\}^n \\ \mathcal{E}(y', \alpha) & \text{if } y = 1 \| y' \\ \perp & \text{otherwise.} \end{cases}$$

Theorem 5. *F_0^P is insecure for unbounded time-stamping, i.e. there is an efficient strongly unpredictable adversary (A_1, A_2) with success probability $\approx \frac{1}{2} - \frac{1}{2^n}$.*

Proof. The first stage A_1 outputs the root value $r = 0^{n-1}1$ of the computational tree for ι_n. The second stage choses randomly and uniformly $x \leftarrow \{0,1\}^n$, which with probability $\approx \frac{1}{2} - \frac{1}{2^n}$ belongs to the computational tree and hence there is an easily computable hash chain $x \overset{c}{\rightsquigarrow} r$. □

Theorem 6. *If F^P is PrA then so is F_0^P.*

Proof. Let $\mathcal{E}$ be the extractor for F^P and let $\mathcal{E}_0$ be the corresponding extractor for F_0^P. Let B_0 be a PrA adversary for F_0^P. Let Ex_0 be the oracle that corresponds to $\mathcal{E}_0$. We modify the adversary B_0 to an adversary B that uses the Ex-oracle (that corresponds to $\mathcal{E}$) and simulates B's Ex_0-calls with input y as follows:

- if $y = 0 \| y' \in \{0,1\}^n \backslash \{0^n\}$, the Ex_0-call is answered with $\iota_n^{-1}(y')$;
- if $y = 1 \| y'$, it makes an oracle call $x \leftarrow \mathsf{Ex}(y')$ and answers the call with x;
- otherwise, the call is answered with $\perp$.

If $1 \leftarrow \mathbf{Exp}^{\mathrm{pra}}_{F_0,P,\mathcal{E}_0,B_0}$, then during the execution $x \leftarrow B_0^{\mathsf{P},\mathsf{Ex}_0}$ an Ex_0-call was made with an input $y = F_0^P(x)$ such that $\mathcal{E}_0(F_0^P(x), \alpha) \neq x$. Note that $y \neq 0 \| y'$, because otherwise (due to the definition of F_0^P) we would have $y' \in \mathcal{I}_n$, $y' = \iota_n(x)$, and $\mathcal{E}_0(F_0^P(x), \alpha) = \mathcal{E}_0(0 \| y', \alpha) = \iota^{-1}(y') = \iota_n^{-1}(\iota_n(x)) = x$.

Therefore, $y = F_0^P(x) = 1 \| y'$, which means that $y' = F^P(x)$ and $\mathcal{E}_0(y, \alpha) = \mathcal{E}(y', \alpha)$. This also means that B makes an Ex-call with parameter y', such that

$$\mathcal{E}(F^P(x), \alpha) = \mathcal{E}(y', \alpha) = \mathcal{E}_0(y, \alpha) = \mathcal{E}_0(F_0^P(x), \alpha) \neq x \enspace ,$$

and we have $1 \leftarrow \mathbf{Exp}^{\mathrm{pra}}_{F,P,\mathcal{E},B}$. Hence, $\mathbf{Adv}^{\mathrm{pra}}_{F_0,P,\mathcal{E}_0}(B_0) \leq \mathbf{Adv}^{\mathrm{pra}}_{F,P,\mathcal{E}}(B)$. □

Theorem 7. *If F^P is SPrA then so is F_0^P.*

Proof. Let $\mathcal{E}$ be the extractor for F^P and let $\mathcal{E}_0$ be the corresponding extractor for F_0^P. Let Π_0 be an arbitrary n-parser and B_0 be an SPrA adversary for F_0^P. Let Ext_0 be the oracle that corresponds to $\mathcal{E}_0$ and Π_0. We define an $(n-1)$-parser Π in the following way: if $\Pi_0(x) = \{y_1, \ldots, y_m\}$ and $y_1 = b_1 \| y_1' \ldots y_m = b_m \| y_m'$

(where $b_1, \ldots, b_m \in \{0,1\}$), then $\Pi(x) = \{y'_1, \ldots, y'_m\}$. Let Ext be the oracle that corresponds to $\mathcal{E}$ and Π. We modify the adversary B_0 to an adversary B that uses the Ext-oracle (that corresponds to $\mathcal{E}$) and simulates B's Ext_0-calls with input y as follows:

- if $y = 0\|y' \in \{0,1\}^n \backslash \{0^n\}$, then the Ext_0-call is answered with $\iota_n^{-1}(y')$;
- if $y = 1\|y'$, it makes an oracle call $x \leftarrow \mathsf{Ext}(y')$ and answers the call with x;
- otherwise, the call is answered with $\perp$.

If $1 \leftarrow \mathbf{Exp}^{\mathrm{spra}}_{F_0,P,\mathcal{E}_0,\Pi_0,B_0}$, then during the execution $x \leftarrow B_0^{\mathsf{P},\mathsf{Ext}_0}$ an Ext_0-call was made with an input $y = F_0^P(x)$ such that $\mathcal{E}_0(F_0^P(x), \alpha) \neq x$. Note that $y \neq 0\|y'$, because otherwise (due to the definition of F_0^P) we would have $y' \in \mathcal{I}_n$, $y' = \iota_n(x)$, and $\mathcal{E}_0(F_0^P(x), \alpha) = \mathcal{E}_0(0\|y', \alpha) = \iota^{-1}(y') = \iota_n^{-1}(\iota_n(x)) = x$.

Therefore, $y = F_0^P(x) = 1\|y'$, which means that $y' = F^P(x)$ and $\mathcal{E}_0(y, \alpha) = \mathcal{E}(y', \alpha)$. This also means that B makes an Ext-call with parameter y', such that

$$\mathcal{E}(F^P(x), \alpha) = \mathcal{E}(y', \alpha) = \mathcal{E}_0(y, \alpha) = \mathcal{E}_0(F_0^P(x), \alpha) \neq x \ .$$

Note that due to the definition of the parser Π, the Ext-oracle uses the same α for extracting y' then the EXT_0-oracle would use for extracting $y = 1\|y'$. Thereby $1 \leftarrow \mathbf{Exp}^{\mathrm{spra}}_{F,P,\mathcal{E},\Pi,B}$, and hence, $\mathbf{Adv}^{\mathrm{pra}}_{F_0,P,\mathcal{E}_0,\Pi_0}(B_0) \leq \mathbf{Adv}^{\mathrm{pra}}_{F,P,\mathcal{E},\Pi}(B)$. $\square$

Generalization of an Oracle Separation Result from Asiacrypt 2004. The construction of F_0^P from F^P allows to generalize and simplify the result in [6] which says that there are no black-box reductions that prove the security of unbounded time-stamping scheme based on the collision-freeness of the underlying hash function. Using the construction $F \mapsto F_0$ enables to extend this result form black-box reductions to arbitrary proofs. This is due to the next theorem:

Theorem 8. *If F^P is CR then so is F_0^P (even if F^P does not use P at all).*

The proof relies on the fact that the function F_0^P is injective on $\mathcal{I}_n$ and hence finding a collision for F_0^P is equivalent to finding collisions for F_0.

Corollary: If there exist collision-free hash functions, then there also exist collision free hash functions that are insecure for unbounded hash-then-publish time-stamping schemes. Note that this corollary is a much stronger statement than an oracle separation because it rules out any proving attempts, not only the black-box ones. Therefore, the implication "F is CR $\Rightarrow$ F is secure for unbounded time-stamping" is true only if there exist no collision-free hash functions.

6 Open Questions and Further Research

It would be interesting to know whether the SPrA assumption can be used in the indifferentiability framework in a way analogous to the PrA assumption, i.e. is it weak enough for being preserved by the Merkle-Damgård transform, and can the conventional compression functions (like Davies-Meyer) to be proven SPrA.

References

1. Bayer, D., Haber, S., Stornetta, W.-S.: Improving the efficiency and reliability of digital timestamping. In: Sequences II: Methods in Communication, Security, and Computer Sci., pp. 329–334. Springer, Heidelberg (1993)
2. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: The 1st ACM Conference on Computer and Communications Security: CCS 1993, pp. 62–73. ACM (1993)
3. Bellare, M., Rogaway, P.: The exact security of digital signatures - How to sign with RSA and Rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)
4. Buldas, A., Niitsoo, M.: Optimally tight security proofs for hash-then-publish time-stamping. In: Steinfeld, R., Hawkes, P. (eds.) ACISP 2010. LNCS, vol. 6168, pp. 318–335. Springer, Heidelberg (2010)
5. Buldas, A., Laur, S.: Do broken hash functions affect the security of time-stamping schemes? In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 50–65. Springer, Heidelberg (2006)
6. Buldas, A., Saarepera, M.: On provably secure time-stamping schemes. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 500–514. Springer, Heidelberg (2004)
7. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. JACM 51(4), 557–594 (2004)
8. Chang, D., Yung, M.: Adaptive preimage resistance analysis revisited: requirements, subtleties and implications. In: IACR Cryptology ePrint Archive 209 (2012)
9. 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, Heidelberg (2005)
10. Dodis, Y., Ristenpart, T., Shrimpton, T.: Salvaging Merkle-Damgård for practical applications. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 371–388. Springer, Heidelberg (2009)
11. Haber, S., Stornetta, W.-S.: How to time-stamp a digital document. Journal of Cryptology 3(2), 99–111 (1991)
12. Luby, M.: Pseudorandomness and Cryptographic Applications. Princeton University Press, Princeton (1996)
13. Maurer, U., 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, Heidelberg (2004)
14. Merkle, R.C.: Protocols for public-key cryptosystems. In: Proceedings of the 1980 IEEE Symposium on Security and Privacy, pp. 122–134 (1980)