Security Proofs for the BLT Signature Scheme

Ahto Buldas, Risto Laanoja, and Ahto Truu

GuardTime AS, Tammsaare tee 60, 11316 Tallinn, Estonia.

Abstract. We present security proofs for the BLT signature scheme in the model, where hash functions are built from ideal components (random oracles, ideal ciphers, etc.). We show that certain strengthening of the Pre-image Awareness (PrA) conditions like boundedness of the extractor, and certain natural properties (balancedness and the so-called output one-wayness) of the hash function are sufficient for existential unforgeability of the BLT signature scheme.

1 Introduction

By the BLT signature scheme, we mean the server-based hash-tree signature scheme proposed recently in [7, 8]. To sign a message m at time t, the signer first combines m and a pseudorandom number ("password") z_t and obtains a time-stamp $S_{t'}$ for a hash $h(z_t, m)$. The signature $(m, z_t, S_{t'}(m, z_t))$ is valid only if $t' = t$, which means that the already-used pseudorandom numbers cannot later be abused to create new signatures $(m', z_t, S_{t''}(m', z_t))$, because $t'' > t$. For such a scheme to be useful and secure in practice, several additional requirements must be met.

First, the time-stamping service used must be trustworthy because signatures can be forged by back-dating the pair (m', z_t) of a new message m' and an already used password z_t. Therefore, the hash-tree keyless time-stamping schemes [15, 1, 10] and keyless signatures [4] are especially suitable for the scheme. The aim of keyless time-stamping is to avoid key-based cryptography and trusted third parties so that time-stamps become irrefutable proofs of time. A collection of C 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 [18] enable to create compact "keyless signatures" of size $O(\log C)$ for each of the C documents. Every such signature consists of all sibling hash values in the path from a document (a leaf of the tree) to the root of the tree. After the root hash value is published, it will be impossible for anyone to *back-date* a new document in terms of creating a hash chain from a new document to the already published hash value. Formal security proofs exist for these schemes about their resistance to back-dating attacks [10, 9] assuming the collision-resistance of the hash function, and somewhat tighter proofs [5, 6] assuming stronger security than the collision-resistance.

Second, there should be a mechanism to verify that a password z_t was indeed intended to use at time t. In [7, 8], the solution to this was to use an iterated one-way function so that all the pseudorandom passwords are computed by recursively iterating a one-way function f on the random seed z_ℓ, so that $z_i = f(z_{i+1})$ and z_0 is the public key. In order to verify that $z = z_t$, the verifier iterates f on z exactly t times and checks if $\underbrace{f(...f}_{t}(z)...) = z_0$. In order to make such a verification more efficient, one may compute a Merkle tree with leaves $z_1, \ldots, z_\ell$ and use its root hash r as the public key. This reduces verification from $O(\ell)$ to $O(\log \ell)$.

In [7, 8], several implementation-related questions are analyzed, but so far there are no formal security proofs for the new signature scheme. All the known proofs [10, 9, 5, 6] only consider the security against back-dating but not against existential forgeries as mostly required from signature

schemes. In this work, we try to figure out how to formalize the security of server-based keyless signatures and present a security proof in an ideal hash function model where the hash function uses an ideal component (such as an ideal cipher, or a fixed-length random oracle). We show that under a slight strengthening of the Pre-image Awareness (PrA) assumption, the so-called Bounded Pre-image Awareness (BPrA) [6]. the BLT signature scheme is existentially unforgeable.

The paper is organized as follows. In Section 2, we present the preliminary concepts about general theory of cryptographic protocols and hash functions, hash tree keyless (data) signatures, and the notion of security against back-dating. In Section 3, we present a formal security model for the BLT signature scheme and a new notion of security against back-dating, which we call *strong unforgeability*, and which, as we show, is sufficient for the security of BLT against existential forgeries. In Section 4, we prove that the BPrA condition (with some natural extensions) is sufficient to prove the strong unfogeability of the so-called rank 1 (not the same as rate 1) constructions H^P in which the ideal component P is used just once. In Section 5, we generalize the proofs for the rank k schemes, and show that their strong unforgeability follows from a weaker assumption, the so-called k-grade strong unforgeability.

2 Preliminaries

2.1 Cryptographic Protocols and Security

A cryptographic protocol usually consists of several functionalities that define the rules of communication between the parties. Formally, we may define a protocol Π as a family of functions of type $\{0,1\}^* \to \{0,1\}^*$ along with a matching between these functions and Turing machines implementing them. However, we can usually compose these functions into one single function – we just use the first few input bits to tell the function which sub-function we want to use. This means that we can still formalize the primitive as one single function, although it may be a little counter-intuitive. A primitive is usually defined in terms of functional requirements that the instances of the primitive must satisfy before it makes sense to talk about their security. These requirements, though, are just syntactic and have nothing to do with security. For example, every permutation is an instance of the one-way permutation primitive, however, it does not have to be a secure instance.

In cryptography we also have to define the *security* of protocols. Note that security may mean different things considering the attacking scenarios we want to avoid in practice. For example, by *secure hash function*, one may mean *one-wayness*, *collision-resistance*, or any other particular security condition.

Definition 1 (Security Condition, Adversaries and Advantage). *A* security condition $\mathcal{C}$ *is defined by any function (*advantage function*)* $\textsc{Adv}^{\mathcal{C}}(\cdot,\cdot)$*, which given as input a cryptographic protocol* Π*, and an oracle Turing machine* $\mathsf{A}^{\mathcal{O}}$ *(an* adversary*) returns a real number* $\textsc{Adv}^{\mathcal{C}}(\Pi, \mathsf{A}^{\mathcal{O}}) \in [0,1]$ *(the* advantage *of* $\mathsf{A}^{\mathcal{O}}$ *when breaking* Π*). The function* $\textsc{Adv}^{\mathcal{C}}(\Pi,\cdot)$ *is extended to probabilistic Turing machines by taking the average over their randomness strings*[1]*.*

We emphasize that our definition says nothing about the efficiency of Π. The function may even be non-computable, as long as the advantage that can be gained by any adversary is negligible. In practice, one needs protocols that are both efficient and secure. Commonly, it is required that

[1] Each fixed randomness string gives a deterministic poly-time Turing machine A for which $\textsc{Adv}^{\mathcal{C}}(\Pi, A)$ is already defined.

we can compute the function Π with a (uniform) poly-time Turing machine for Π to be called *efficient*.

The same applies to the advantage function $\textsc{Adv}^{\mathcal{C}}(\cdot,\cdot)$. In some security conditions, the advantage may not be efficiently computable. However, in most practical cases, the advantage function is efficiently computable. The following important class of security conditions covers all our needs in this work.

Definition 2 (t-time Falsifiable Security Condition). *By a* t-time falsifiable security condition *$\mathcal{C}$, we mean a security condition for which the advantage function is in the form*

$$\textsc{Adv}^{\mathcal{C}}(\Pi, A) = \mathsf{Pr}\left[1 \leftarrow \mathcal{E}^{\Pi,A}\right] \ ,$$

where $\mathcal{E}$ (the so-called environment*) is a t-time probabilistic oracle Turing machine, where oracle calls are assumed to be unit-cost.*

The security of cryptographic protocol Π is measured by the amount of resources needed for an adversary to break the primitive. A scheme 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 scheme may vary, Luby [16] proposed the *time-success ratio* $\frac{t}{\delta}$ as a measure for attacking resources. The following definition is based on the approach of [16].

Definition 3 (S-Secure Protocol). *A cryptographic protocol Π is said to be* S-secure *in terms of a security condition $\mathcal{C}$, if every t-time adversary A has advantage $\textsc{Adv}^{\mathcal{C}}(\Pi, A) \leq \frac{t}{S}$. In terms of oracle-adversaries, Π is said to be S-secure relative to $\mathcal{O}$, if every t-time oracle adversary has advantage $\textsc{Adv}^{\mathcal{C}}(\Pi, A^{\mathcal{O}}) \leq \frac{t}{S}$.*

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 the *tightness* S_p/S_q of security proofs in practical applications. Informally, tightness shows how much security of the primitive is transformed to the protocol. Security proofs are mostly *reductions*: an adversary for $\mathcal{P}$ with running time t and success probability δ is transformed to an adversary for $\mathcal{Q}$ with running time t' and success probability δ'. This means that for having $\frac{t}{\delta}$-secure $\mathcal{P}$, we have to use a $\frac{t'}{\delta'}$-secure $\mathcal{Q}$.

2.2 Random Oracles

By a *random oracle* Ω, we mean a function that is chosen randomly from the set of all functions of type $\{0,1\}^m \rightarrow \{0,1\}^n$. 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 random oracle. The random oracle heuristic was first introduced by Bellare and Rogaway [2]. Although it was proved later by Canetti et al [11] that the random oracle heuristic fails in certain theoretical cases, proofs in the random oracle model are still considered valuable security arguments, especially if no better security proofs are known.

2.3 Pseudorandom Functions

Pseudorandom functions are functions that are (computationally) indistinguishable from true random oracles, though the family of functions they are chosen from is much smaller than the set of all functions from which the random oracle is chosen.

Definition 4 (PRF). *A function $h\colon \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^n$ is an S-secure pseudorandom function family (PRF) if for every t-time distinguisher D:*

$$\textsc{Adv}^{\mathrm{PRF}}(h, D) = \left| \mathsf{Pr}\left[1 \leftarrow D^{h(z,\cdot)}\right] - \mathsf{Pr}\left[1 \leftarrow D^{\Omega}\right] \right| \leq \frac{t}{S} \ ,$$

*where $z \leftarrow \{0,1\}^k$ and Ω is a random oracle chosen from the set of all functions $\{0,1\}^m \to \{0,1\}^n$. We say that h is an S-*secure PRF relative to an oracle P, *if h and D are allowed to use P-calls, i.e. we use h^P, $D^{h(z,\cdot),P}$, and $D^{\Omega,P}$ in the definition.*

If h is pseudorandom and Π is a cryptographic protocol in which $h(z,\cdot)$ (with random z) is used, then we can replace such calls of h with calls to a true random oracle Ω without changing the security of the protocol, if the changed (randomized) version of the protocol is secure then so is the original protocol. The following theorem formalizes this same idea. In order to replace some of the h-calls with random oracle calls, we first represent Π^{h^P} as a special case of a more general protocol $\Sigma^{f,g}$ (f and g represent two different instances of the h-calls made by Π) such that $\Pi^{h^P} \equiv \Sigma^{h^P(z,\cdot),h^P}$, i.e. the first type of h-calls of Π (equivalently, the f-calls of Σ) are all in the form $x \mapsto h(z,x)$, where z is picked randomly and uniformly and is the same for all these calls. We may now replace these calls with calls to a true random oracle, to get the so-called *randomized protocol* $\Pi^{h^P}_{\mathrm{rnd}} \equiv \Sigma^{\Omega,h^P}$.

Theorem 1. *Let Π^{h^P} be any protocol that can be represented as a special case of a more general protocol $\Sigma^{f,g}$ such that $\Pi^{h^P} \equiv \Sigma^{h^P(z,\cdot),h^P}$. If $h^P(z,\cdot)$ is an S-secure PRF relative to P and the randomized protocol $\Pi^{h^P}_{\mathrm{rnd}} \equiv \Sigma^{\Omega,h^P}$ is S-secure relative to P in terms of a t'-time falsifiable security condition $\mathcal{C}$, then the original protocol Π^{h^P} is $\frac{S}{2t'}$-secure relative to P in terms of $\mathcal{C}$.*

Proof. Let A^P be a t-time adversary that breaks Π^{h^P} with advantage $\delta = \textsc{Adv}^{\mathcal{C}}(\Pi^{h^P}, A^P) = \mathsf{Pr}\left[1 \leftarrow \mathcal{E}^{\Pi^{h^P},A^P}\right]$. Let $D^{\Phi,P} \equiv \mathcal{E}^{\Sigma^{\Phi,h^P},A^P}$. The running time of D does not exceed tt'. Note that $\mathsf{Pr}\left[1 \leftarrow D^{h^P(z,\cdot),P}\right] = \mathsf{Pr}\left[1 \leftarrow \mathcal{E}^{\Sigma^{h^P(z,\cdot),h^P},A^P}\right] = \textsc{Adv}^{\mathcal{C}}(\Sigma^{h^P(z,\cdot),h^P}, A^P) = \textsc{Adv}^{\mathcal{C}}(\Pi^{h^P}, A^P) = \delta$, and $\mathsf{Pr}\left[1 \leftarrow D^{\Omega,P}\right] = \mathsf{Pr}\left[1 \leftarrow \mathcal{E}^{\Sigma^{\Omega,h^P},A^P}\right] = \mathsf{Pr}\left[1 \leftarrow \mathcal{E}^{\Pi^{h^P}_{\mathrm{rnd}},A^P}\right] = \textsc{Adv}^{\mathcal{C}}(\Pi^{h^P}_{\mathrm{rnd}}, A^P) \leq \frac{t}{S}$. Therefore,

$$\delta \leq \textsc{Adv}^{\mathcal{C}}(\Pi^{h^P}_{\mathrm{rnd}}, A^P) + \left| \mathsf{Pr}\left[1 \leftarrow D^{h(z,\cdot),P}\right] - \mathsf{Pr}\left[1 \leftarrow D^{\Omega,P}\right] \right| \leq \frac{t}{S} + \frac{tt'}{S} \leq 2t'\frac{t}{S} \ ,$$

which means that Π^{h^P} is $\frac{S}{2t'}$-secure relative to P. □

2.4 Pseudorandom Oracles

There is a stronger notion of pseudorandom function that applies to functions H^P that use ideal components accessible via an oracle P. For example, one may assume that the compression function of a hash function is made of an ideal cipher or a random oracle. In such a model, it would be fair

to assume that adversaries also have access to the P oracle. Moreover, sometimes H^P is used as a building block in cryptographic protocols the security proofs of which (for simplicity) assume that H^P is replaced with a true random oracle Ω. For such a replacement (the so-called *random oracle heuristics*) to be correct, it is sufficient that H^P is a *pseudo-random oracle*:

Definition 5 (PRO). *We say that H^P is an (S, t')-secure* pseudo-random oracle (PRO) *if there is a t'-time simulator $\mathcal{S}$, such that for every t-time distinguisher D:*

$$\textsc{Adv}^{\text{PRO}}(H^P, D) = \left| \mathsf{Pr}\left[1 \leftarrow D^{H^P,P}\right] - \mathsf{Pr}\left[1 \leftarrow D^{\Omega,\mathcal{S}^\Omega}\right] \right| \leq \frac{t}{S} \ .$$

This notion is first studied by Maurer et al [17] and was adapted to hash functions by Coron et al [12]. The most important practical implication of the pseudo-random oracle property of H^P is that any application that uses H^P as a hash function is almost as secure as if a random oracle 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 [12] that the commonly used Merkle-Damgård style hash functions do not satisfy the PRO property.

Theorem 2. *If a cryptographic protocol Π^Ω (where Ω is a random oracle) is S-secure in a t'-time falsifiable sense in the random oracle model, and H^P is (S, t'')-secure PRO, then Π^{H^P} is $\frac{S}{t'+t''}$-secure relative to P in the same sense.*

Proof. Let A^P be a t-time adversary that breaks Π^{H^P} with probability $\delta = \textsc{Adv}^{\mathcal{C}}(\Pi, A^P) = \mathsf{Pr}\left[1 \leftarrow \mathcal{E}^{\Pi^{H^P},A^P}\right]$. Let B^Ω be an adversary that behaves like A, except that it simulates its P-queries using the t''-time simulator S^Ω. The running time of B does not exceed tt''. As Π^Ω is S-secure, the probability that B^Ω breaks Π^Ω is $\delta' = \textsc{Adv}^{\mathcal{C}}(\Pi^\Omega, B^\Omega) = \mathsf{Pr}\left[1 \leftarrow \mathcal{E}^{\Pi^\Omega,B^\Omega}\right] \leq \frac{tt''}{S}$. Let $D^{\Phi,\pi}$ be a distinguisher that simulates the attack of A^π against Π^Φ and outputs 1 iff A is successful. Hence, $\mathsf{Pr}\left[1 \leftarrow D^{\Phi,\pi}\right] = \textsc{Adv}^{\mathcal{C}}(\Pi^\Phi, A^\pi) = \mathsf{Pr}\left[1 \leftarrow \mathcal{E}^{\Pi^\Phi,A^\pi}\right]$. The running time of D does not exceed tt', which means that

$$\begin{aligned}\delta &= \mathsf{Pr}\left[1 \leftarrow \mathcal{E}^{\Pi^{H^P},A^P}\right] = \mathsf{Pr}\left[1 \leftarrow D^{H^P,P}\right] \leq \mathsf{Pr}\left[1 \leftarrow D^{\Omega,S^\Omega}\right] + \frac{tt'}{S} = \mathsf{Pr}\left[1 \leftarrow \mathcal{E}^{\Pi^\Omega,A^{S^\Omega}}\right] + \frac{tt'}{S} \\ &= \mathsf{Pr}\left[1 \leftarrow \mathcal{E}^{\Pi^\Omega,B^\Omega}\right] + \frac{tt'}{S} \leq \frac{tt''}{S} + \frac{tt'}{S} = (t'+t'')\frac{t}{S} \ ,\end{aligned}$$

which means that Π^{H^P} is $\frac{S}{t'+t''}$-secure. □

2.5 Pre-Image Awareness

Pre-Image Awareness (PrA) of a (hash) function H means, that if we first commit to 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. [14] for hash functions

H^P that are built using an ideal primitive P as a black box. For H^P being PrA, there has to be an efficient deterministic extractor $\mathcal{E}$ which when given y and the list α of all previously made P-calls, outputs an input 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 $x \neq \mathcal{E}(\alpha, y)$ and $y = H^P(x)$. A weaker form of PrA (so-called WPrA) allows $\mathcal{E}$ output a set $\mathcal{L}$ of inputs x, and the adversary tries to find x, such that the query $\mathcal{L} \leftarrow \mathcal{E}(\alpha, y)$ was made, $y = H^P(x)$, but $x \notin \mathcal{L}$. Obviously, WPrA becomes PrA if the number of elements in $\mathcal{L}$ is limited to one, i.e. $|\mathcal{L}| \leq 1$. To define pre-image awareness of H^P in a precise way,

$\mathbf{Exp}^{\mathrm{pra}}_{H,P,\mathcal{E},B}$:
$x \leftarrow B^{\mathsf{P},\mathsf{Ex}}$
$y \leftarrow H^P(x)$
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 $\mathsf{Ex}(y)$:
if $\mathtt{Q}[y] \neq 1$, then:
  $\mathtt{Q}[y] \leftarrow 1$
  $\mathtt{V}[y] \leftarrow \mathcal{E}(y, \alpha)$
return $\mathtt{V}[y]$

Fig. 1. Preimage awareness experiment with the oracles P and Ex.

we set up an experiment **Exp** (see Fig. 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 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 unsuccessful. As P- and Ex-calls are unit cost, the running time of B does not depend on the running time of $\mathcal{E}$. Note that PrA implies collision-resistance, but WPrA does not.

Definition 6 (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:*

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

2.6 Bounded Pre-Image Awareness

In [6], a stronger than PrA security condition was defined that is called *Bounded Pre-Image Awareness* (BPrA) which is a stronger version of the PrA condition. The BPrA condition assumes the existence of a PrA-extractor $\mathcal{E}$ that is bounded in the sense that for efficiently computable query strings α, the number of outputs y for which $\mathcal{E}(y, \alpha)$ succeeds does not exceed the number of queries in α.

Definition 7. *A function $H^P\colon \{0,1\}^{2n} \to \{0,1\}^n$ is S-secure* Bounded Pre-Image Aware *(BPrA) if it is S-secure PrA, and for any t-time adversary $\alpha \leftarrow Q^P$:*

$$\mathrm{Adv}(\mathcal{E}, Q^P) = \mathsf{Pr}\left[\alpha \leftarrow Q^P\colon \ |\{y\colon \mathcal{E}(y, \alpha) \neq \perp\}| > |\alpha|\right] \leq \frac{t^2}{2^n} \ ,$$

where $\mathcal{E}$ is the extractor from the PrA condition.

Note that for BPrA, the extractor must always return $\perp$ if the query string is empty, i.e. $\mathcal{E}((), z) = \perp$ for every $z \in \{0,1\}^n$, because otherwise Q may break $\mathcal{E}$ with doing nothing and hence producing an empty query list.

2.7 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, as the collisions or fixed functions can always be "wired" into the adversary.

Definition 8 (Collision Resistance). *A function H^P is S-secure* collision resistant *(CR) if for every adversary B with running time t:*

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

Due to the so-called Birthday bound, functions with n-bit output can only be up to $2^{\frac{n}{2}}$-secure collision resistant.

2.8 Hash-Tree Schemes and their Security Against Back-Dating

Hash trees were introduced by Merkle [18]. Let $h: \{0,1\}^{2n} \to \{0,1\}^n$ be a hash function. By a *hash-tree* we mean a binary tree-shaped data structure that consists of nodes labeled with n-bit hash values. Each node is either a *leaf* which means it has no children, or an *internal node* with two child nodes (the left and the right child). 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 children. There is one *root node* that is not a child of any node. If $\mathcal{T}$ is a hash tree with m leaves labeled with hash values $x_1, \ldots, x_m$ and r is the root hash value, then we use the notation $r = \mathcal{T}(x_1, \ldots, x_m)$.

Encoding the Leaves of a Hash Tree. Nodes of a hash tree can be named in a natural way with finite bit-strings. The root node is named by the empty string $\lfloor\rfloor$. If a node is named by ℓ, then its left and right child nodes are named by $\ell 0$ and $\ell 1$, respectively. The name ℓ of a node is also 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 children depending on the corresponding bit in ℓ.

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 the shape $\{00, 01, 10, 11\}$. If the root hash value is denoted by r (instead of $r_{\lfloor\rfloor}$) 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 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 a *prefix-free code*.

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 r, it is sufficient to present all the sibling hashes of the nodes on the unique path from r_ℓ to the root r. For example, in the left tree of Fig. 2, to prove that r_{01} belongs to the tree, one has to present the hashes r_{00} and r_1 that enable us to compute $r_0 = h(\underline{r_{00}}, r_{01})$ and $r = h(r_0, \underline{r_1})$. Formally, hash chains can be defined as follows [5]:

Fig. 2. Trees with shape $\{00, 01, 10, 11\}$ (left) and $\{000, 001, 01, 1\}$ (right).

Definition 9 (Hash-Chain). *If $h\colon \{0,1\}^{2n} \to \{0,1\}^n$ be a hash function. An* h-hash-chain *(or simply, a* hash chain*) from x to r (where $x, r \in \{0,1\}^n$) is a (possibly empty) sequence $c = (c_1, c_2, \ldots, c_m)$ of triples $c_i = (x_i, x'_i; y_i)$ (where $y_i = h(x_i, x'_i)$), such that $y_i \in \{x_{i+1}, x'_{i+1}\}$ for every $i = 1, \ldots, m-1$. We write $x \overset{c}{\rightsquigarrow} r$ to denote that c is a hash chain from x to r. Note that $x \overset{()}{\rightsquigarrow} x$ for every $x \in \{0,1\}^n$.*

Definition 10 (Subchains, Reduced Chains). *By a* subchain *of a hash chain $x \overset{c}{\rightsquigarrow} r$ is an arbitrary sub-sequence of c' of c that itself is a hash chain; c' is said to be* proper *if $x \overset{c'}{\rightsquigarrow} r$. A hash chain c is* reduced*, if all its triples $(x_i, x'_i; y_i)$ have unique output values y_i, i.e. if all the values y_i are different. Note that shortest proper subchains of c are always reduced and can be efficiently found by using a simple backwards wave-propagation technique.*

Hash-Tree Keyless Signature Schemes. The signing (time-stamping) procedure runs as follows. During every time unit t (e.g. one second) the 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 *signature* for x_i) that proves the participation 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 hash chain c_i (the signature) 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 $x_m = r$.

Security Against Back-Dating. 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 $x \overset{c}{\rightsquigarrow} r$. We use the formal security condition from [5] that involves a two-stage back-dating adversary $A = (A_1, A_2)$. The first stage A_1 creates a public repository $\mathcal{R}$ of hash values that may be created in an arbitrary way, not necessary by using hash trees. The second stage A_2 of A presents a high-entropy x and a hash chain $x \overset{c}{\rightsquigarrow} r$ with $r \in \mathcal{R}$. The high entropy of x is crucial because otherwise x could have been pre-computed or guessed by A_1 before r is published and hence x could be in fact older than r and thereby not really back-dated by A_2. Therefore, only *unpredictable* adversaries (that produce high-entropy x) are considered, i.e. x must be hard to guess for A_2 even if the contents of $\mathcal{R}$ and all the internal computations of A_1 are known. There might be several ways to define unpredictability, but we use the *strong unpredictability* assumption from [5]:

Definition 11 (Strong Unpredictability). *We say that a back-dating adversary $A = (A_1, A_2)$ is* strongly unpredictable *if the conditional min-entropy $H_\infty[x \mid \mathcal{R}, a]$ of x (back-dated by A_2) is*

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 $\frac{1}{2^{n-1}}$.

Definition 12 (Security against Back-Dating). *A hash function h is S-secure against back-dating if for every t-time strongly unpredictable (A_1, A_2):*

$$\textsc{Adv}^{\text{BD}}(h, A) = \mathsf{Pr}\left[(\mathcal{R}, a) \leftarrow A_1, (x, c) \leftarrow A_2(\mathcal{R}, a) \colon x \overset{c}{\rightsquigarrow} \mathcal{R}\right] \leq \frac{t}{S} \ , \tag{3}$$

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

2.9 Merkle-Damgård Hash Functions

Merkle-Damgård (or iterated) hash functions use a compression function $F(m, v)$ to iteratively compute a hash of an arbitrary size message m divided into equal blocks $m_1, \ldots, m_\ell$ of suitable size. The hash $h = H(m)$ is computed as follows: (1) $h \leftarrow IV$; (2) for $i \in \{1, \ldots, \ell\}$ do: $h \leftarrow F(m_i, h)$; (3) and output $H(m) = h$. Here, IV is a public and standard initial value. It has been proved [14] that if F is PrA, then so is H.

2.10 Blockcipher-Based Compression Functions

Many hash functions are constructed from secure block-ciphers. The most common approach for creating a $2n \to n$ hash function is to use a block-cipher with n-bit block and n-bit key and make a compression function that makes one call to the block-cipher. Such constructions were first analyzed by Preneel et al. [19] and are called *rate-1 schemes*. Higher rate schemes compress several n-bit blocks at one iteration. The most general approach is that of Stam [21], where the compression function is defined by the following three steps:

1. Prepare key and plaintext: $(k, x) \leftarrow C_{\text{pre}}(m, v)$;
2. Use the blockcipher: $y \leftarrow E_k(x)$;
3. Output the digest: $w \leftarrow C_{\text{post}}(m, v, y)$.

There are two types of rank-1 compression functions.

Definition 13. *A block-cipher-based rate-1 compression function F^E is called Type-I iff: (1) C_{pre} is bijective; (2) $C_{\text{post}}(m, v, \cdot)$ is bijective for all m, v; and (3) $C_{\text{aux}}(\cdot) = C_{\text{post}}(C_{\text{pre}}^{-1}(k, \cdot), y)$ is bijective for all k, y.*

Definition 14. *A block-cipher-based rate-1 compression function F^E is called Type-II iff: (1) C_{pre} is bijective; (2) $C_{\text{post}}(m, v, \cdot)$ is bijective for all m, v; and $C_{\text{pre}}^{-1}(k, \cdot)$ (restricted to its second output v) is bijective for all k.*

Type-I functions are collision-resistant. Type-II functions become collision-resistant when iterated as Merkle-Damgård hash functions. It has been shown [14] that rate-1 Type-I compression functions are PrA and that if the rate-1 Type-II compression function is iterated, then it becomes PrA.

3 Security Model for the BLT Signature Scheme

3.1 Parties

The following parties (Fig. 3) are involved in the BLT server-based signature scheme:

- **Server** authenticates clients and creates keyless signatures $S_t(y)$ for signers' requests y. For serving several clients, and multiple signature requests by one client, the server has to process requests in batches, create a hash tree for each batch, and send the root hash r_t (together with t) to the Repository. For every request y, a hash chain $y \overset{c}{\rightsquigarrow} r_t$ is extracted from the hash tree and is sent as a keyless signature $S_t(y)$ back to client.
- **Repository** $\mathcal{R}$ is a trusted database that receives: (i) write requests r_t which are stored in append-only manner as an array; and (ii) read requests t which are answered with $\mathcal{R}[t]$. The repository also has the public verification hash z_0 of every signer.
- **Signer** $\mathcal{S}$ who signs documents. For that, client first hashes the document to produce a hash value x and then, combining the hash with a suitable one-time password z_t, hashing the combination and sending the hash $y = h(z_t, x)$ (together with t) as a request to the server. After having received the keyless signature $S_t(y) = y \overset{c}{\rightsquigarrow} r_t$, the signer makes a read request t to the repository and obtains r_t, and verifies $S_t(y)$ against r_t (i.e. computes the output hash of c and compares it to r_t). Only after a successful verification, the signer can send the full signature (t, z_t, c) to the verifier.
- **Verifier** is a party who has to verify signatures created by signers. For verifying a signature (t, z_t, c) on m, the verifier makes a read-request to the repository to obtain r_t. After that, the verifier computes $h(z_t, x)$ and compares the result with y, computes the output of the hash chain c and compares the result to r_t, and it also has to verify that z_t was intended for using at t. For example, in the scheme of [7, 8], the verifier iterates $\underbrace{f(...f}_{t}(z_t)...)$ and compares the result to z_0 obtained from the repository. In this paper, we use a somewhat different scheme.

3.2 Adversary

We assume that both the verifier and the server may co-operate against the signer, trying to forge signatures and hence, the verifier and the server are assumed to be under complete control of the adversary (dashed box in Fig. 3). The adversary $A^{\mathcal{S},\mathcal{R}}$ is assumed to make oracle calls x to signer $\mathcal{S}$ and repository $\mathcal{R}$ (Fig. 4). The objective of $A^{\mathcal{S},\mathcal{R}}$ is to produce a correct signature (t, z_t, c) for an x that was never used to call $\mathcal{S}$ during the attack, i.e. the attack scenario describes a standard *existential forgery* setting.

The signer and the repository are defined in such a way that the adversary has to write r_t to $\mathcal{R}$ before it is possible to access z_t, which intuitively means that z_t cannot be abused by $A^{\mathcal{S},\mathcal{R}}$, unless $A^{\mathcal{S},\mathcal{R}}$ is able to back-date z_t relative to r_t, i.e. to first store r_t and then (having got z_t) find a hash chain $y' \rightsquigarrow r_t$ from $y' = h(z_t, x')$ (where x' is the new hash to be signed) to an already stored root hash r_t.

However, formal security proof is not trivial, even in the model with the repository $\mathcal{R}$ as an ideal object, mostly because of two reasons:

- Before storing r_t, the adversary has partial knowledge about z_t, namely the one-way image $f(z_t) = z_{t-1}$ (and also the just-received request $y = h(z_t, x)$). This has not been foreseen in the

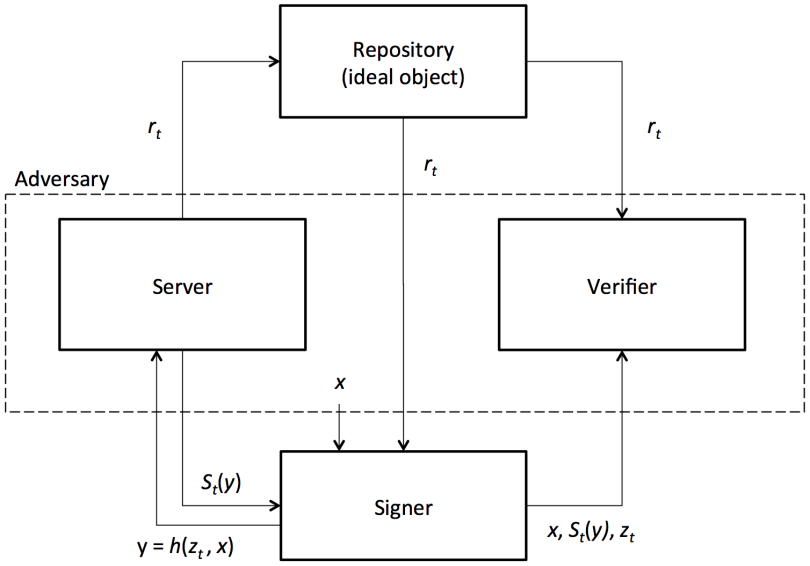

Fig. 3. Parties involved in the BLT signature scheme.

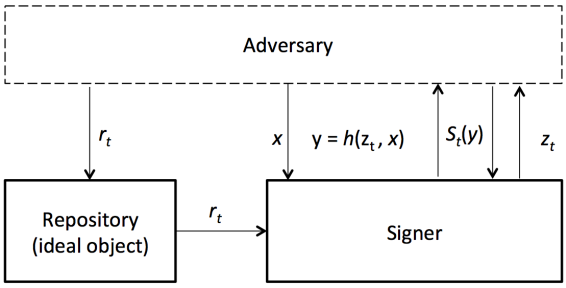

Fig. 4. Adversarial model of the BLT signature scheme.

standard security conditions that consider back-dating [10, 9, 5], where the back-dated data was generated randomly after the first stage $(r, a) \leftarrow A_1$ of the back-dating adversary has finished.
– There is another way for the adversary to try to forge a signature with z_t, namely using the partial information that $y = h(z_t, x)$ may provide clues for creating a hash $y' = h(z_t, x')$ for $x' \neq x$ knowing only y. The security condition about h that prevents doing so resembles very much the so-called *non-malleability* of h, which is considered a very strong security condition.

"Common sense analysis" suggests that if the hash function is non-malleable and secure against back-dating, then it is also secure for the BLT setting. Assume that the adversary A_1 has $y = h(z_t, x)$ and it commits to a hash value r, i.e. $(r, a) \leftarrow A_1$, where a is an advice string. Say that A_2 is able to come up with a forgery, i.e. $(x', c) \leftarrow A_2(a, z_t)$, where $x' \neq x$, and c is a hash chain $y' \overset{c}{\rightsquigarrow} r$ from $y' = h(z_t, x')$ to the published (by A_1) hash r. Informally, we may argue that either:

– (A_1, A_2) found a collision for h, i.e. $h(z_t, x) = y = y' = h(z_t, x')$;
– A_1 was able to compute y' from y before publishing r; or

– A_2 was able to back-date a new hash value y' relative to r.

The problem is though that in the standard model, it is very hard to define "knowledge", i.e. what does it mean that A_1 knows y' before commiting to r. For this reason, we prove the security in a model with ideal objects and use the Preimage-Awareness (PrA) assumption.

3.3 Verifiable Pseudorandom Passwords

We analyze the necessary and sufficient requirements that the one-time passwords z_i have to meet. It turns out that they should not necessarily be generated in iterative manner: $z_i = f(z_{i+1})$, and $z_\ell = s \leftarrow \{0,1\}^n$. If we use an additional hash tree structure on $z_1, \ldots, z_\ell$ for more efficient verification, then we may use the values $z_i = h(i, s)$ instead, assuming that there is an easily verifiable relation between i and the shape of the hash chain for z_i in the hash tree.

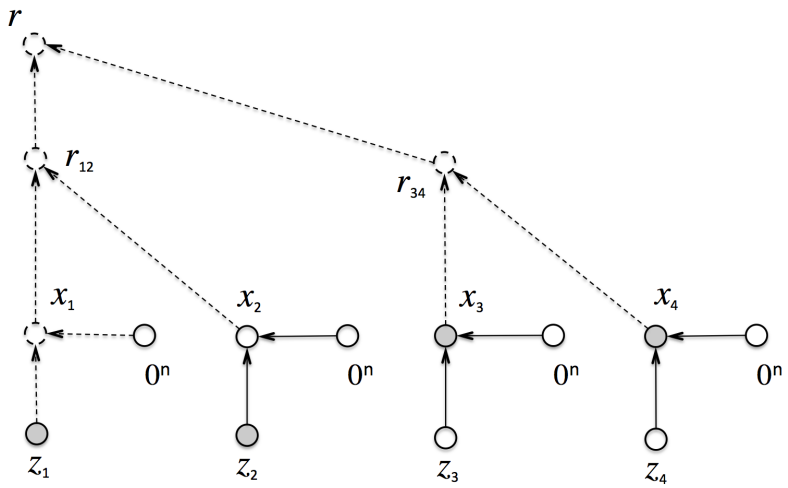

Fig. 5. Hash password scheme. Gray nodes denote the information available to the adversary after using z_2.

The one-time password generation scheme $z_1, \ldots, z_\ell, x_1, \ldots, x_\ell \leftarrow \mathcal{G}(\ell)$ is defined as follows (Fig. 5):

– Pick a random seed $s \leftarrow \{0,1\}^n$.
– For all $i = 1 \ldots \ell$: $z_i \leftarrow h(s, i)$ and $x_i \leftarrow h(z_i, 0^n)$.

The values $x_1, \ldots, x_\ell$ are assumed to be known by adversaries, because the hash chains that prove the authenticity of z_i contain them. Once we assume that all x_i are known to the adversary, the hash chains do not provide any additional information about z_i, because the hash chains can be easily computed from x_i. We also define the randomized scheme $z_1, \ldots, z_\ell, x_1, \ldots, x_\ell \leftarrow \mathcal{G}_{\text{rnd}}(\ell)$ is defined as follows:

– For all $i = 1 \ldots \ell$: $z_i \leftarrow \{0,1\}^n$ and $x_i \leftarrow h(z_i, 0^n)$.

The following theorem is a direct corollary of Thm. 1.

Theorem 3. *If $h(z, \cdot)$ is an S-secure PRF and the random password generation scheme $\mathcal{G}_{\text{rnd}}$ is S-secure in a t'-time falsifiable sense, then the original password scheme $\mathcal{G}$ is $\frac{S}{2t'}$-secure in the same sense.*

3.4 Formal Security Definition

Considering the model above, we give the following formal definition to the security of the BLT-type signature scheme.

Definition 15. *A keyless signature scheme is said to be S-secure, if every t-time adversary $A^{\mathcal{S},\mathcal{R}}$ (that does not make $\mathcal{S}$-queries with x) has success probability*

$$\mathsf{Pr}\left[(z_1,\ldots,z_\ell,x_1,\ldots,x_\ell)\leftarrow\mathcal{G}(\ell),(m,i,z,c)\leftarrow A^{\mathcal{S},\mathcal{R}}(x_1,\ldots,x_\ell)\colon h(m,z)\overset{c}{\rightsquigarrow}\mathcal{R}[i],z=z_i\right]\leq\frac{t}{S}\ .$$

Oracle $\mathcal{S}$ (signing oracle)
Initialize:
$i=0$
Query $\mathcal{S}(x)$:
$i=i+1$
Output $h(z_i,x)$
wait Query $\mathcal{S}(r)$:
If $\mathcal{R}[i]=r$ then output z_i

Oracle $\mathcal{R}$ (repository)
Initialize:
$i=0$
Query $\mathcal{R}(r)$: (write query)
$i=i+1$
$R[i]=r$
Query $\mathcal{R}[j]$: (read query)
If $j\leq i$ then output $R[j]$,
otherwise output $\perp$

Fig. 6. The oracles used in the security condition.

It turns out that the following *strong unforgeability* condition (Def. 16) is sufficient for the BLT scheme to be secure. It may seem that the strong unforgeability condition is somewhat similar to the security against back-dating (Def. 12) but as we discussed in Sec. 3.2, the main difference is that the "back-dated" entity z was generated before A_1 publishes $\mathcal{R}$ and A_1 has a partial knowledge on z in the form of an oracle query $h(z,x)$. On the other hand, the strong unforgeability does not assume that the output of A_2 is unpredictable, which means that by running A_2 twice with the same input (a,z) may produce exactly the same output (m,c) which means that we cannot extract a collision as it was done in the proofs [10, 9].

Definition 16 (Strong Unforgeability). *We say that a hash function $h\colon\{0,1\}^*\to\{0,1\}^n$ is S-secure strongly unforgeable, if for every t-time $A=(A_1,A_2)$ (that queries $h(z,\cdot)$ only once and not with x):*

$$\mathsf{Pr}\left[z\leftarrow\{0,1\}^n,\,(\mathcal{R},a)\leftarrow A_1^{h(z,\cdot)}(h(z,0^n)),\,(x,c)\leftarrow A_2(a,z)\colon\ h(z,x)\overset{c}{\rightsquigarrow}\mathcal{R},x\neq 0^n\right]\leq\frac{t}{S}\ . \quad (4)$$

In this definition, the oracle $h(z,\cdot)$ returns $h(z,v)$ for any $v\in\{0,1\}^n$. Note that we may assume without loss of generality that the hash chain produced by A is always *reduced* (Def. 10).

Theorem 4. *If h is S-secure strongly unforgeable, then the randomized scheme $\mathcal{G}_{\mathrm{rnd}}$ is $\frac{S}{\ell}$-secure.*

Proof. Let $A^{\mathcal{S},\mathcal{R}}$ be a t-time adversary with success δ. We construct an adversary $A'=(A_1,A_2)$ that breaks strong unforgeability:

$A_1^{h(z,\cdot)}(v)$ generates a random $j\leftarrow\{1,\ldots,\ell\}$ and random $z_1,\ldots,z_{j-1},z_{j+1},\ldots,z_\ell\leftarrow\{0,1\}^n$. It then assigns $x_i\leftarrow h(z_i,0^n)$ (for $i\neq j$), and $x_j\leftarrow v$. Then A_1 simulates $A^{\mathcal{S},\mathcal{R}}$ until it makes the j-th

write query to $\mathcal{R}$. The $j-1$ first $\mathcal{S}$-queries can all be simulated using $z_1, \ldots, z_{j-1}$. To simulate the j-th query x', the adversary A_1 uses the oracle call to obtain $h(z, x')$. The j-th write query r is added to $\mathcal{R}$ by A_1 and the output of A_1 is $(\mathcal{R}, a)$, where a is the current state of $A^{\mathcal{S},\mathcal{R}}$. Note that by the definition of the oracles $\mathcal{S}$ and $\mathcal{R}$, before the j-th $\mathcal{R}$-query, the continuation query of the j-th signature query can be answered by $\perp$.

$A_2(a, z)$ restores the state of $A^{\mathcal{S},\mathcal{R}}$ from a and continues to simulate $A^{\mathcal{S},\mathcal{R}}$. Note that now it has $z_j = z$ and is able to simulate the j-th signing call. Finally, if $A^{\mathcal{S},\mathcal{R}}$ finishes and outputs (x, i, z, c) such that $h(z, x) \overset{c}{\rightsquigarrow} \mathcal{R}[i]$ and $z = z_i$, then if $i = j$, it outputs (x, c), otherwise it outputs $\perp$. Note also that if $A^{\mathcal{S},\mathcal{R}}$ never queries $\mathcal{S}$ with x, then A_1 does not query $h(z, \cdot)$ with x.

As the distribution of z_i and x_i are identical to those generated by $\mathcal{G}_{\text{rnd}}$, the value of j does not have any influence over the success of the simulated $A^{\mathcal{S},\mathcal{R}}$. Hence, the event $j = i$ (with probability $\frac{1}{\ell}$) and the success event of A (with probability δ) are independent and the success probability of A' is $\delta' = \frac{\delta}{\ell}$. The running time t' of A' is about t. Hence, $\delta = \ell\delta' \leq \ell \cdot \frac{t'}{S} \approx \ell \cdot \frac{t}{S}$ and hence, the randomized scheme $\mathcal{G}_{\text{rnd}}$ is $\frac{S}{\ell}$-secure. □

This means that, the strong unforgeability property is sufficient for the security of $\mathcal{G}$, but this property seems to be very strong. For example, it implies the following version of non-malleability of h. Hence, it is not realistic to imply such property from the classical properties of hash functions, like one-wayness, second pre-image resistance, or even collision-freeness.

4 Security Proof under PrA

We study the security properties of h that follow the strong unforgeability of $\mathcal{G}_{\text{rnd}}$ and are hence also sufficient for the security of $\mathcal{G}$. It turns out that the following, application specific, weaker form of non-malleability plays a central role in the security proofs.

Definition 17 (Weak Non-Malleability). *We say that a hash function $h\colon \{0,1\}^{2n} \to \{0,1\}^n$ is S-secure* Weakly Non-Malleable (WMN)*, if for every t-time $A = (A_1, A_2)$ (that queries $h(z, \cdot)$ only once and not with x):*

$$\text{Adv}^{\text{WNM}}(h, A) = \mathsf{Pr}\left[z \leftarrow \{0,1\}^n,\ (\mathcal{R}, a) \leftarrow A_1^{h(z,\cdot)}(h(z, 0^n)),\ x \leftarrow A_2(a, z)\colon\ h(z, x) \in \mathcal{R},\ x \neq 0^n\right] \leq \frac{t}{S}\ .$$

The adversary has hash values $h(z, x')$, $h(z, 0^n)$ and has to produce a list $\mathcal{R}$ of hash values such that if later z is made public, the adversary is able to show x, such that $h(z, x) = y \in \mathcal{R}$. This definition easily generalizes to hash functions h^P that use an ideal primitive P, if both A_1 and A_2 have access to the P-oracle. We will now study the known conditions that would imply WMN, and then the conditions that together with WMN imply strong unforgeability. The following notion of *non-guessability* will simplify later proofs.

Definition 18 (Non-Guessability). *We say that a hash function $h\colon \{0,1\}^{2n} \to \{0,1\}^n$ is S-secure non-guessable, if for every t-time oracle-adversary A (that queries the oracle no more than once):*

$$\text{Adv}^{\text{NG}}(h, A) = \mathsf{Pr}\left[z \leftarrow \{0,1\}^n, z' \leftarrow A^{h(z,\cdot)}(h(z, 0^n))\colon z' = z\right] \leq \frac{t}{S}\ .$$

It turns out that non-guessability follows from weak non-malleability.

Theorem 5. *If h is S-secure WNM, then it is S-secure NG.*

Proof. Let A be a t-time guesser with success $\delta = \mathrm{Adv}^{\mathrm{NG}}(h, A)$. Let $A' = (A_1, A_2)$ be the WNM adversary, such that $A_1^{h(z,\cdot)}(y)$ simulates $z' \leftarrow A^{h(z,\cdot)}(y)$ and if x' was the only oracle query that A produces, choses $x \neq x'$, computes $y = h(z', x)$ and outputs $((y), x)$; and the adversary $A_2(z, x)$ just outputs x. If A is successful, then $z' = z$ and hence $h(z, x) = h(z', x) = y$, which means that A' is also successful. □

Definition 19 (Output One-Wayness, OOW). *A function $f\colon D \to R$ is said to be S-secure output one-way, if every t-time adversary A has success:*

$$\mathrm{Adv}^{\mathrm{OOW}}(f, A) = \mathsf{Pr}\left[y \leftarrow R, x \leftarrow A(y)\colon\ y = f(x)\right] \leq \frac{t}{S}\ .$$

Definition 20 (Quasi-Balanced Function). *A function $f\colon D \to D$ is called* quasi-balanced*, if $|\, f^{-1}(d)\,| \leq \sqrt{|\, D\,|}$ for every $d \in D$.*

Theorem 6. *For any $\alpha > 0$ and for any $S = \alpha 2^{n/2}$, if $h\colon \{0,1\}^{2n} \to \{0,1\}^n$ is a hash function such that $h(\cdot, 0^n)\colon \{0,1\}^n \to \{0,1\}^n$ is quasi-balanced and S-secure output one way, and $h(z, \cdot)$ is an S-secure PRF, then h is $\frac{S}{2+\alpha}$-secure non-guessable.*

Proof. Let A be an adversary that guesses h with success δ. We construct an adversary B that inverts $h(\cdot, 0^n)$ and a distinguisher D^ϕ as follows.

The inversion adversary B having as input $y \in \{0,1\}^n$ simulates the run of $A^{h(z,\cdot)}(y)$, hoping that A will not make any oracle-calls, because B is not able to simulate them. If A succeeds to find its output z' without oracle calls, then B outputs z', otherwise B gives up and outputs $\perp$.

The distinguisher D^ϕ simulates A on input $y = \phi(0^n)$ and answers to A-s oracle calls with the ϕ-oracle. It hopes that A makes at least one oracle call $y' \leftarrow \phi(x)$. If A outputs z', then D checks if $y = h(z', 0^n)$ and $y' = h(z', x)$, and outputs 1 if both equalities hold, otherwise D outputs 0.

Let δ_0 be that probability that A succeeds to invert $h(\cdot, 0^n)$ without making any oracle calls. Then, the success of B is $\delta_0 = \mathrm{Adv}^{\mathrm{OOW}}(h(\cdot, 0^n), B) \leq \frac{t}{S}$.

Let δ_1 be the probability that A succeeds while making at least one oracle call. It is easy to see that $\mathsf{Pr}\left[1 \leftarrow D^{h(z,\cdot)}\right] = \delta_1$. In order to estimate $\mathsf{Pr}\left[1 \leftarrow D^{\Omega}\right]$ (where Ω is a true random oracle), note first that due to the quasi-balancedness of $h(\cdot, 0^n)$, the are no more than $2^{\frac{n}{2}}$ possible z'-s for which $h(z', 0^n) = y$. Hence, no matter how we choose x, there are no more than $2^{\frac{n}{2}}$ possible y'-s such that $y' = h(z', 0^n)$ for a z' such that $h(z', 0^n) = y$. As the output $y' = \Omega(x)$ is uniformly distributed random variable, the probability that such a z' exists does not exceed $\frac{2^{n/2}}{2^n} = 2^{-n/2}$. Hence, as $h(z, \cdot)$ is an S-secure PRF, we have $\delta_1 \leq 2^{-n/2} + \frac{t}{S}$. This implies

$$\delta = \delta_0 + \delta_1 \leq 2\frac{t}{S} + 2^{-n/2} \leq 2\frac{t}{S} + \frac{\alpha}{S} \leq (2+\alpha)\frac{t}{S}\ ,$$

which means that h is $\frac{S}{2+\alpha}$-secure non-guessable. □

Theorem 7. *For any $\alpha > 0$ and $S = \alpha 2^{n/2}$, if a rank 1 hash function $h^P\colon \{0,1\}^{2n} \to \{0,1\}^n$ (queries P only once) is S-secure BPrA and S-secure non-guessable, then it is $\frac{S}{2+\alpha}$-secure weakly non-malleable against adversaries with running time $t \leq 2^{n/2}$.*

Proof. Suppose there is an adversary (A_1, A_2) with running time $t \leq 2^{\frac{n}{2}}$ that breaks weak non-malleability with success δ, and assume also that h^P is BPrA. We construct three adversaries: a guesser A, a BPrA-adversary $B^{\mathsf{P},\mathsf{Ex}}$, and a query-list adversary Q^P.

The guesser $A^{h^P(z,\cdot),P}(y)$ simulates $(\mathcal{R}, a) \leftarrow A_1^{h^P(z,\cdot),P}$, where $\mathcal{R} = (y_1, \ldots, y_m)$. Let $x' \mapsto y' = h^P(z, x')$ be the single call that A_1 makes and let α_0 be the query string that consists of all P-queries made by A_1. Note that α_0 may not contain the P-query of $h^P(z, x')$ and $y = h^P(z, 0^n)$.

1. If $\perp \neq \mathcal{E}(\alpha_0, y) = (z', *) \neq \perp$, then return z';
2. If $\perp \neq \mathcal{E}(\alpha_0, y') = (z', *) \neq \perp$, then return z';
3. Find the first i such that $\mathcal{E}(\alpha, y_i) = (z', *) \neq \perp$ such that $h^P(z', 0^n) = y$ and $h^P(z', x') = y'$, and return z';
4. If there is no such i, then return $\perp$.

The PrA adversary B proceeds as follows:

1. Pick $z \leftarrow \{0,1\}^n$ and simulate $(\mathcal{R}, a) \leftarrow A_1^{h^P(z,\cdot)}(y)$, so that the P-calls are made through the P-oracle, where $y = h^P(z, 0^n)$ and $y' = h^P(z, x')$ is the oracle call of A_1.
2. Compute $X \leftarrow \mathrm{Ex}(y)$ and $X' \leftarrow \mathrm{Ex}(y')$,
3. If $X \neq (z, 0^n)$, then output $(z, 0^n)$ and stop.
4. If $X' \neq (z, x')$, then output (z, x') and stop.
5. Call m Ex-calls $X_1 \leftarrow \mathrm{Ex}(y_1), \ldots, X_m \leftarrow \mathrm{Ex}(y_m)$, where $\mathcal{R} = (y_1, \ldots, y_m)$.
6. Find the fist i for which $X_i = (z', *)$, $h^P(z', 0^n) = y$, and $h^P(z', x') = y'$, output $(z', 0^n)$ and stop.
7. If there is no such i, simulate $x \leftarrow A_2^P(a, z)$ and return (z, x).

The query-list adversary Q^P picks $z \leftarrow \{0,1\}^n$, simulates $(y_1, \ldots, y_m, a) \leftarrow A_1^{h^P(z,\cdot)}(y)$ and outputs the list α_1 of all the P-queries made by A_1, $h^P(z, 0^n)$ and $h^P(z, x')$, where x' is the only $h^(z, \cdot)$-query made by A_1.

Let δ_0 be the probability that (A_1, A_2) succeeds, while either:

(a) $(z, *) \leftarrow \mathcal{E}(\alpha_0, y)$ or
(b) $\perp \leftarrow \mathcal{E}(\alpha_0, y)$ and $\perp \neq (z, *) \leftarrow \mathcal{E}(\alpha_0, y')$ (i.e. A guesses the right z by applying $\mathcal{E}$ to y or y'), or
(c) There is i, for which $\perp \neq (z', *) = \mathcal{E}(\alpha_0, y_i)$ and $h^P(z', 0^n) = y$ and $h^P(z', x') = y'$, and $z = z'$ for the first such i. (i.e. the first "consistent" z' that A finds from y_i-s is the right z).

By definition of A, we have $\mathrm{Adv}^{NG}(h, A) \geq \delta_0$. Let δ_1 be the probability that (A_1, A_2) succeeds, while all of the following conditions hold:

(i) $\mathcal{E}(\alpha_0, y) \neq (z, *)$;
(ii) $\mathcal{E}(\alpha_0, y') \neq (z, *)$;
(iii) One of the following conditions hold:
  (a) For all i such that $\perp \neq \mathcal{E}(\alpha_0, y_i) = (z', *)$, either $h^P(z', 0^n) \neq y$, or $h^P(z', x') \neq y'$ (i.e. $z' \neq z$); or
  (b) There is i, such that $\perp \neq (z', *) = \mathcal{E}(\alpha_0, y_i)$, $h^P(z', 0^n) = y$, and $h^P(z', x') = y'$; but for first such i, we have $z' \neq z$.
(iv) $|\{y : \mathcal{E}(y, \alpha) \neq \perp\}| \leq |\alpha|$.

Let δ_2 be the probability that (A_1, A_2) succeeds while $|\{y: \mathcal{E}(y,\alpha) \neq \perp\}| > |\alpha|$ (i.e. the condition (iv) does not hold). It is easy to see that the logical statement

$$[(a) \vee (b) \vee (c)] \vee [(i) \wedge (ii) \wedge (iii) \wedge (iv)] \vee \overline{(iv)}$$

is a tautology, and hence, $\delta = \delta_0 + \delta_1 + \delta_2$. Let α_1 be the query string that consists all the P-queries made by B while simulating A_1.

- If $\mathcal{E}(\alpha_1, y) \neq (z, 0^n)$ or $\mathcal{E}(\alpha_1, y') \neq (z, x')$, then B fools $\mathcal{E}$ at steps 3,4. Hence, we may assume that $\mathcal{E}(\alpha_1, y) = (z, 0^n)$ and $\mathcal{E}(\alpha_1, y') = (z, x')$. Then (i) and (ii) imply $\mathcal{E}(\alpha_0, y) = \mathcal{E}(\alpha_0, y') = \perp$. Hence, in the next analysis, considering (iv), we may assume that y and y' are the only outputs, for which $\mathcal{E}(\alpha_0, y) = \perp = \mathcal{E}(\alpha_0, y')$ and $\mathcal{E}(\alpha_1, y) \neq \perp \neq \mathcal{E}(\alpha_1, y')$.
- From (iii,b) it follows that there is i such that $\perp \neq (z', *) = \mathcal{E}(\alpha_0, y_i)$, $h^P(z', 0^n) = y$, and $h^P(z', x') = y'$; and $z' \neq z$ for first such i, which means that B fools the extractor at step 6.
- If (A_1, A_2) successfully finds (z, x) with $x' \neq x \neq 0^n$ and $h^P(z, x) = y_i$, then $y \neq y_i \neq y'$ and hence $\mathcal{E}(\alpha_0, y_i) = \mathcal{E}(\alpha_1, y_i)$. From (iii, a) then follows that $\mathrm{Ex}(y_i) = \mathcal{E}(\alpha_1, y_i) = \mathcal{E}(\alpha_0, y_i) \neq (z, *)$, which means that B fools the extractor at step 7.

Therefore, $\textsc{Adv}^{\mathrm{BPrA}}(h^P, B) \geq \delta_1$. As $\frac{t^2}{2^n} \geq \textsc{Adv}(\mathcal{E}, Q^P) \geq \delta_2$, and

$$\delta \leq \textsc{Adv}^{\mathrm{NG}}(h, A) + \textsc{Adv}^{\mathrm{BPrA}}(h^P, B) + \textsc{Adv}(\mathcal{E}, Q^P) \leq 2\frac{t}{S} + \frac{t^2}{2^n} \leq 2\frac{t}{S} + \frac{t}{2^{n/2}} \leq (2+\alpha)\frac{t}{S} \ ,$$

it follows that h^P is $\frac{S}{2+\alpha}$-secure weakly non-malleable. □

Procedure $\mathsf{ExTree}^{\mathrm{Ex}}(y)$:
  If $y \notin \mathcal{T}$ and $N > 0$, **then**
    $\mathcal{T} := \mathcal{T} \cup \{y\}$
    $N := N - 1$
    If $\perp \neq \mathrm{Ex}(y) = (y_0, y_1)$ **then**
      Define y_0, y_1 as children of y
      $\mathsf{ExTree}^{\mathrm{Ex}}(y_0)$
      $\mathsf{ExTree}^{\mathrm{Ex}}(y_1)$.
    endif
  endif

Procedure $\mathsf{ExForest}^{\mathrm{Ex}}(\mathcal{R})$:
  $\mathcal{T} := \emptyset$
  $N :=$ 2 × "a time bound for A_1"
  For all $r \in \mathcal{R}$ **do**
    $\mathsf{ExTree}^{\mathrm{Ex}}(r)$.

Fig. 7. Procedures for extracting the set $\mathcal{T}$ from the published hash database $\mathcal{R}$.

Theorem 8. *For any $\alpha > 0$ and $S = \alpha 2^{n/2}$, if a rank 1 hash function $h^P: \{0,1\}^{2n} \to \{0,1\}^n$ (queries P only once) is S-secure BPrA and S-secure non-guessable, then it is $\frac{S}{2+\alpha}$-secure strongly unforgeable against adversaries with running time $t \leq 2^{n/2}$.*

Proof. Suppose there is an adversary (A_1, A_2) with running time $t \leq 2^{\frac{n}{2}}$ that breaks strong unforgeability ability with success δ, and assume also that h^P is BPrA. We construct three adversaries: a guesser A, a BPrA-adversary $B^{\mathsf{P},\mathsf{Ex}}$, and a query-list adversary Q^P.

The guesser $A^{h^P(z,\cdot),P}(y)$:

1. Simulates $(\mathcal{R}, a) \leftarrow A_1^{h^P(z,\cdot),P}(y)$;
2. Creates an extraction forest $\mathcal{T}_0$ by simulating the $\mathsf{ExForest}^{\mathrm{Ex}}(\mathcal{R})$ procedure (Fig. 7) so that the oracle calls are simulated using $\mathcal{E}(\alpha_0, \cdot)$, where α_0 is the query string that consists of all P-queries made by A_1. Note that α_0 may not contain the single P-query of $y' = h^P(z, x')$ of A_1 and of $y_0 = h^P(z, 0^n)$. We may assume without loss of generality that $x' \neq 0^n$.
3. If $\perp \neq \mathcal{E}(\alpha_0, y) = (z', *)$, returns z';
4. If $\perp \neq \mathcal{E}(\alpha_0, y') = (z', *)$, returns z';
5. Find a triple $(z', *; *)$ in $\mathcal{T}_0$ such that $h^P(z', 0^n) = y$ and $h^P(z', x') = y'$, and return z'; If there is no such triple in $\mathcal{T}_0$, return $\perp$.

The PrA adversary B:

1. Picks $z \leftarrow \{0,1\}^n$ and simulates $(\mathcal{R}, a) \leftarrow A_1^{h^P(z,\cdot)}(y)$, so that the P-calls are made through the P-oracle, where $y = h^P(z, 0^n)$ and $y' = h^P(z, x')$ is the oracle call of A_1.
2. Creates an extraction forest $\mathcal{T}_1$ by simulating the $\mathsf{ExForest}^{\mathrm{Ex}}(\mathcal{R})$ procedure (Fig. 7) using the oracle Ex. Note that this time Ex uses $\mathcal{E}(\alpha_1, \cdot)$, where α_1 is the query string that consists of all P-queries made by A_1 and also the P-queries of $y' = h^P(z, x')$ and $y_0 = h^P(z, 0^n)$.
3. If $\mathrm{Ex}(y_0) \neq (z, 0^n)$, outputs $(z, 0^n)$ and stops.
4. If $\mathrm{Ex}(y') \neq (z, x')$, outputs (z, x') and stops.
5. Finds a triple $(z', *; *)$ in $\mathcal{T}_1$ for which $h^P(z', 0^n) = y$, and $h^P(z', x') = y'$, and outputs $(z', 0^n)$ and stops if $z' \neq z$.
6. If there are no such triples in $\mathcal{T}_1$, simulates $(x, c) \leftarrow A_2^P(a, z)$.
7. If $y = h^P(z, x) \in \{y_0, y'\}$ then outputs (z, x) and stops;
8. If $y \notin \{y_0, y'\}$, tries to find the first triple $c_i = (x_i, x_i'; y_i)$ in c so that c_i is not a subtree of $\mathcal{T}_1$, but $y_i \in \mathcal{T}_1$. Note that if the triple $(z, x; y)$ is not in $\mathcal{T}_1$ the triple c_i is the point where c enters into $\mathcal{T}_1$. If there is such a triple, B outputs (x_i, x_i') and stops. Otherwise, B returns $\perp$.

The query-list adversary Q^P picks $z \leftarrow \{0,1\}^n$ and simulates $(\mathcal{R}, a) \leftarrow A_1^{h^P(z,\cdot)}(h^P(z, 0^n))$ and outputs the list α_1 of all the P-queries made by A_1, $h^P(z, 0^n)$ and $h^P(z, x')$.

Let δ_0 be the probability that (A_1, A_2) succeeds, while either: (a) $(z, *) \leftarrow \mathcal{E}(\alpha_0, y)$; or (b) $(z, *) \leftarrow \mathcal{E}(\alpha_0, y')$; or (c) there are triples $(z', *; *)$ in $\mathcal{T}_0$ for which $h^P(z', 0^n) = y_0$ and $h^P(z', x') = y'$, and $z = z'$ for all such triples. By definition of A, we have $\mathrm{Adv}^{NG}(h, A) \geq \delta_0$.

Let δ_1 be the probability that (A_1, A_2) succeeds, while all the following conditions hold:

(i) $\mathcal{E}(\alpha_0, y) \neq (z, *)$ and $\mathcal{E}(\alpha_0, y') \neq (z, *)$;
(ii) Either:
  (a) there is a triple $(z', *; *)$ in $\mathcal{T}_0$ such that $h^P(z', 0^n) = y_0$ and $h^P(z', x') = y'$, but $z' \neq z$; or
  (b) threre are no triples $(z', *; *)$ in $\mathcal{T}_0$ such that $h^P(z', 0^n) = y_0$ and $h^P(z', x') = y'$;
(iii) $|\{y\colon \mathcal{E}(\alpha_1, y) \neq \perp\}| \leq |\alpha|$.

If either $\mathrm{Ex}(y_0) \neq (z, 0^n)$ or $\mathrm{Ex}(y') \neq (z, x')$, then B fools $\mathcal{E}$ at steps 3, 4. Hence, we may assume that $\mathcal{E}(\alpha_1, y_0) = (z, 0^n)$ and $\mathcal{E}(\alpha_1, y') = (z, x')$. From (i) it now follows that $\mathcal{E}(\alpha_0, y) = \perp = \mathcal{E}(\alpha_0, y')$ and hence from (iii) it follows that y_0 and y' are the only outputs which $\mathcal{E}(\alpha_1, \cdot)$ can extract, but $\mathcal{E}(\alpha_0, \cdot)$ doesn't. If (ii,a), then B fools the axtractor at step 5. Otherwise, (ii,b) must hold. As $x \neq 0^n$ and $x \neq x'$, then if $y \in \{y_0, y'\}$, the adversary B fools $\mathcal{E}$ at step 7. If $y \notin \{y_0, y'\}$ then all y_0, y', and y are all different and y_0 and y' were the only outputs that differentiate $\mathcal{E}(\alpha_0, \cdot)$ and $\mathcal{E}(\alpha_1, \cdot)$,

we conclude using (ii,b) that the triple $(z, x; y)$ is not in the tree $\mathcal{T}_1$, which means that at step 6, the adversary B successfully fools the extractor. Hence, $\textsc{Adv}^{\mathrm{PrA}}(h^P, B) \geq \delta_1$.

Let δ_2 be the probability that (A_1, A_2) succeeds while $|\{y\colon \mathcal{E}(y, \alpha) \neq \perp\}| > |\alpha|$ (i.e. the condition (iii) does not hold). It is easy to see that the logical statement

$$[(\mathrm{a}) \vee (\mathrm{b}) \vee (\mathrm{c})] \vee [(\mathrm{i}) \wedge (\mathrm{ii}) \wedge (\mathrm{iii})] \vee \overline{(\mathrm{iii})}$$

is a tautology, and hence, $\delta = \delta_0 + \delta_1 + \delta_2$. Therefore, $\textsc{Adv}^{\mathrm{BPrA}}(h^P, B) \geq \delta_1$. As $\frac{t^2}{2^n} \geq \textsc{Adv}(\mathcal{E}, Q^P) \geq \delta_2$, and

$$\delta \leq \textsc{Adv}^{\mathrm{NG}}(h, A) + \textsc{Adv}^{\mathrm{BPrA}}(h^P, B) + \textsc{Adv}(\mathcal{E}, Q^P) \leq 2\frac{t}{S} + \frac{t^2}{2^n} \leq 2\frac{t}{S} + \frac{t}{2^{n/2}} \leq (2+\alpha)\frac{t}{S} \ ,$$

it follows that h^P is $\frac{S}{2+\alpha}$-secure strongly unforgeable. □

5 Generalization to rank k Hash Functions

It turns out that the following general notion of k-grade non-forgeability is useful for proving the security of BLT where higher grade hash functions are used, i.e. if $h^P\colon \{0,1\}^{2n} \to \{0,1\}^n$ calls P up to k times.

Definition 21 (k-grade Strong Non-Forgeability, SNF_k). *For $k > 0$, we say that a hash function $h\colon \{0,1\}^* \to \{0,1\}^n$ is S-secure k-grade strongly non-forgeable, if for every t-time $A = (A_1, A_2)$ (that queries $h(z, \cdot)$ only once and not with m):*

$$\mathsf{Pr}\left[z \leftarrow \{0,1\}^n,\ (\mathcal{R}, a) \leftarrow A_1^{h(z,\cdot)},\ (m, c) \leftarrow A_2(a, z)\colon\ h(z, m) \overset{c}{\rightsquigarrow} \mathcal{R}, |c| \leq k-1\right] \leq \frac{t}{S} \ ; \qquad (5)$$

and for $k = 0$, $\mathsf{Pr}\left[z \leftarrow \{0,1\}^n,\ (\mathcal{R}, a) \leftarrow A_1^{h(z,\cdot)}\colon z \in \mathcal{R}\right] \leq \frac{t}{S}$.

Note that SNF_1 is equivalent to weak non-malleability.

Theorem 9. *For any $\alpha > 0$ and $S = \alpha 2^{n/2}$, if a rank k hash function $h^P\colon \{0,1\}^{2n} \to \{0,1\}^n$ (queries P no more than k times) is S-secure BPrA and S-secure non-guessable, and $2(k-1)$-grade non-forgeable, then it is $\frac{S}{3+\alpha}$-secure strongly unforgeable against adversaries with running time $t \leq 2^{n/2}$.*

Proof. Suppose there is an adversary (A_1, A_2) with running time $t \leq 2^{\frac{n}{2}}$ that breaks strong unforgeability ability with success δ, and assume also that h^P is BPrA. We construct four adversaries: a guesser G, a BPrA-adversary $B^{\mathsf{P},\mathsf{Ex}}$, an SU-adversary $\underline{A} = (\underline{A}_1^{h^P(z,\cdot),P}, \underline{A}_2^P)$ and a query-list adversary Q^P.

The guesser $G^{h^P(z,\cdot),P}(y)$:

1. Simulates $(\mathcal{R}, a) \leftarrow A_1^{h^P(z,\cdot),P}(y)$. Let α_0 is the query string that consists of all P-queries made by A_1. Note that α_0 may not contain the single P-query of $y' = h^P(z, x')$ of A_1 and of $y_0 = h^P(z, 0^n)$. We may assume without loss of generality that $x' \neq 0^n$.
2. If $\perp \neq \mathcal{E}(\alpha_0, y) = (z', *)$, returns z';

3. If $\perp \neq \mathcal{E}(\alpha_0, y') = (z', *)$, returns z';

The PrA adversary B:

1. Picks $z \leftarrow \{0,1\}^n$, and calls $\mathrm{Ex}(z)$.
2. Simulates $(\mathcal{R}, a) \leftarrow A_1^{h^P(z,\cdot)}(y)$, so that the P-calls are made through the P-oracle, where $y = h^P(z, 0^n)$ and $y' = h^P(z, x')$ is the oracle call of A_1.
3. Creates an extraction forest $\mathcal{T}_1$ by simulating the $\mathsf{ExForest}^{\mathrm{Ex}}(\mathcal{R})$ procedure (Fig. 7) using the oracle Ex. Note that this time Ex uses $\mathcal{E}(\alpha_1, \cdot)$, where α_1 is the query string that consists of all P-queries made by A_1 and also the P-queries of $y' = h^P(z, x')$ and $y_0 = h^P(z, 0^n)$.
4. If $\mathrm{Ex}(y_0) \neq (z, 0^n)$, outputs $(z, 0^n)$ and stops.
5. If $\mathrm{Ex}(y') \neq (z, x')$, outputs (z, x') and stops.
6. Simulates $(x, c) \leftarrow A_2^P(a, z)$ and if $y = h^P(z, x) \in \{y_0, y'\}$ then outputs (z, x) and stops;
7. If $y \notin \{y_0, y'\}$, finds the first triple $(x_i, x_i'; y_i)$ in c so that $\mathrm{Ex}(y_i) \neq (x_i, x_i')$, and outputs (x_i, x_i') or if $\mathrm{Ex}(y) \neq (z, x)$ outputs (z, x).

The SU adversary $\underline{A} = (\underline{A}_1, \underline{A}_2)$ is defined as follows:

1. $\underline{A}_1^{h^P(z,\cdot)}(y)$ first simulates $(\mathcal{R}, a) \leftarrow A_1^{h^P(z,\cdot)}(y)$, then extracts $\mathcal{T}_0$ and outputs $(\underline{\mathcal{R}}, a')$, where $\underline{\mathcal{R}} = \mathcal{T}_0 \cup \{y_0, x', y', 0^n\}$, and $a' = (a, \overline{\mathcal{R}})$.
2. $\underline{A}_2^P(z, a')$ parses $a' = (a, \mathcal{R})$ and executes $(x, c) \leftarrow A_2(a, z)$. If A_2 is successful, then it finds a shortest subchain c' of $h^P(z, x) \overset{c}{\rightsquigarrow} \mathcal{R}$, such that $h^P(z, x) \overset{c'}{\rightsquigarrow} \underline{\mathcal{R}}$.

The query-list adversary Q^P:

1. Picks $z \leftarrow \{0,1\}^n$ and if $\mathcal{E}((), z) \neq \perp$, stops. (BPrA condition implies that this never happens!)
2. Simulates $(\mathcal{R}, a) \leftarrow A_1^{h^P(z,\cdot)}(h^P(z, 0^n))$ and outputs the list α_1 of all the P-queries made by A_1, $h^P(z, 0^n)$ and $h^P(z, x')$.

We define the following list of logical conditions (Tab. 1), everyone of which implies the success of either G, B, or Q. Hence, it remains to show that $\overline{\mathcal{A}} \wedge \wedge \overline{\mathcal{B}} \wedge \overline{\mathcal{C}} \wedge \overline{\mathcal{D}} \wedge \overline{\mathcal{E}}$ implies the success of $\underline{A} = (\underline{A}_1, \underline{A}_2)$.

Table 1. Logical conditions, everyone of which implying the success of either G, B, or Q.

Name	Condition	Implication
$\mathcal{A}$	$\mathcal{E}(\alpha_0, y_0) = (z, 0^n)$ or $\mathcal{E}(\alpha_0, y') = (z, x')$	G guesses z (steps 2, 3)
$\mathcal{B}$	$\mathcal{E}(\alpha_1, y_0) \neq (z, 0^n)$ or $\mathcal{E}(\alpha_1, y') \neq (z, x')$	B fools $\mathcal{E}$ (steps 4, 5)
$\mathcal{C}$	$y = h^P(z, x) \in \{y_0, y'\}$	B fools $\mathcal{E}$ (step 6)
$\mathcal{D}$	$\exists (x_i, x_i'; y_i) \in c$: $\mathrm{Ex}(y_i) \neq (x_i, x_i')$	B fools $\mathcal{E}$ (step 7)
$\mathcal{E}$	$\lvert \{y \colon \mathcal{E}(\alpha_1, y) \neq \perp\} \rvert > \lvert \alpha \rvert$	Q breaks $\mathcal{E}$

Indeed, $\overline{\mathcal{A}}$ and $\overline{\mathcal{B}}$ implies that $\mathcal{E}(\alpha_0, y_0) = \perp = \mathcal{E}(\alpha_0, y')$ and hence, y_0 and y' are two values that $\mathcal{E}$ can extract with α_1, but not with α_0. It follows from $\overline{\mathcal{E}}$, that there no more than $2k$ of such outputs. It also follows from $\overline{\mathcal{A}}$ and $\overline{\mathcal{B}}$ that $y_0 \neq y'$ because $x' \neq 0^n$ and hence $(z, 0^n) = \mathcal{E}(\alpha_1, y_0) \neq \mathcal{E}(\alpha_1, x') = (z, x')$. As $0^n \neq x \neq x'$, it follows from $\overline{\mathcal{C}}$ that all y_0, y', y are all different.

From $\overline{\mathcal{D}}$ it follows that all triples of c belong to $\mathcal{T}_1$ as subtrees; and that c' does not contain triples $(x_i, x_i'; z)$, because as the $\mathrm{Ex}(z)$ call was made by B before performing any other oracle calls, we have that $\mathrm{Ex}(z) = \mathcal{E}((), z) = \perp \neq (x_i, x_i')$.

As c' is reduced and is the shortest hash chain from $y = h^P(z, x)$ to $\underline{\mathcal{R}} = \mathcal{T}_0 \cup \{y_0, x', y', 0^n\}$, we conclude that the output values of all the triples in c' are different and hence $|\,c'\,| \leq 2k$. Moreover, none of the three triples $(z, 0^n; y_0)$, $(z, x'; y')$, $(z, x; y)$ can be in c'. Indeed, $(z, x; y)$ cannot be in c' because then we could omit this triple together with all the preceding triples and the rest of the chain still goes from y to $\underline{\mathcal{R}}$. The triples $(z, 0^n; y_0)$, $(z, x'; y')$ could only be the final links of c' because $y_0, y' \in \underline{\mathcal{R}}$ and c' is reduced. But if $(z, x'; y')$ was the final link of c', the previous link should have output either z or y' which is impossible. If $(z, 0^n; y_0)$ was the final link, then the previous triple (link) must have output either z or 0^n which is impossible. Therefore, $|\,c'\,| \leq 2k - 3 = 2(k-1) - 1$, which means that $\underline{A}$ successfully breaks the $2(k-1)$-grade non-forgeability. As

$$\begin{aligned}\delta &\leq \mathrm{Adv}^{\mathrm{NG}}(h^P, G) + \mathrm{Adv}^{\mathrm{BPrA}}(h^P, B) + \mathrm{Adv}^{\mathrm{kNF}}(h^P, \underline{A}) + \mathrm{Adv}(\mathcal{E}, Q^P) \\ &\leq 3\frac{t}{S} + \frac{t^2}{2^n} \leq 3\frac{t}{S} + \frac{t}{2^{n/2}} \leq (3+\alpha)\frac{t}{S}\ ,\end{aligned}$$

it follows that h^P is $\frac{S}{3+\alpha}$-secure strongly unforgeable. □

6 Conclusions and Discussion

We have shown that if a rank 1 hash function $h^P\colon \{0,1\}^{2n} \to \{0,1\}^n$ is BPrA, and additionally $h^P(z, \cdot)$ is PRF and $h^P(\cdot, 0^n)$ is quasi-balanced and output one-way, then the BLT scheme with h^P is secure. All these assumptions are natural and are expected to hold for most of the hash function constructions. From this, it follows that the PGV-constructions [19] of block-cipher-based hash functions are all suitable for the BLT scheme.

We generalized the result to higher rank constructions (where h^P uses P more than once), by showing that if a rank k (where $k \geq 2$) hash function h^P is BPrA, $h^P(z, \cdot)$ is PRF and $h^P(\cdot, 0^n)$ is quasi-balanced and output one-way, and if additionally h^P is $2(k-1)$-grade non-forgeable, then the BLT scheme with h^P is secure. But for example for $k = 2$, we have to show that h^P is 2-grade non-forgeable but this is not a standard security property and has not been studied so far. It is an open question how the Merkle-Damgård construction behaves in terms of BLT, if more than one block is needed to construct a two-to-one hash function. This might indeed be the case if an additional padding is added to the two n-bit inputs. So, it would be desirable to study the two-block MD-hash functions.

Not all PrA constructions of hash functions are suitable for the BLT scheme. For example, the Dodis-Pietrzak-Punyia (DPP) [13] hash function $h(m, v) = H^{f_1, f_2}(m, v) = f_1(m) \oplus f_2(v)$ is malleable, i.e. is not WNM. Let (A_1, A_2) be the following adversary. The first stage $A_1^{h(z,\cdot)}$ (with random $z \leftarrow \{0,1\}^n$) makes a call m (any n-bit constant) to the oracle and obtains $h(z, m) = f_1(z) \oplus f_2(m)$, then it makes an f_2-call to obtain $f_2(m)$ and by combining $h(z, m)$ and $f_2(m)$ finds $f_1(z)$. It then makes a second call to f_2 to obtain $f_2(m')$, where $m' \neq m$ and outputs $r = f_1(z) \oplus f_2(m') = h(z, m')$. The advice a given to the second stage A_2 consists of m'. The second stage $A_2(z, a)$ just parses a to get m' and outputs (z, m'). Obviously, (A_1, A_2) breaks the WNM property with probability 1. This also implies that the DPP hash function is not strongly unforgeable, and one can also show that DPP is insecure in the BLT scheme. It would be interesting to know whether the Shrimpton-Stam hash function [20] is suitable for BLT, but as it is of rank 3, we have to show that it is 4-grade non-forgeable.

References

1. Bayer, D., Haber, S., Stornetta, W.-S.: Improving the efficiency and reliability of digital time-stamping. 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'93, 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 '96. LNCS 1070, pp. 399–416. Springer, Heidelberg (1996)
4. Buldas, A., Kroonmaa,A., Laanoja,R.: Keyless signatures infrastructure: How to build global distributed hash-trees. In: H. Riis Nielson, H.R., Gollmann, D. (eds.): NordSec 2013. LNCS 8208, pp. 313–320. Springer, Heidelberg (2013)
5. Buldas, A., Laanoja, R.: Security proofs for hash tree time-stamping using hash functions with small output size. In: Boyd, C., Simpson, L. (eds.): ACISP 2013. LNCS 7959, pp. 235–250. Springer, Heidelberg (2013)
6. Buldas, A., Laanoja, R., Laud, P., Truu, A.: Bounded pre-image awareness and the security of hash-tree keyless signatures. In: Chow, S.M., Liu, J.K. (eds.): ProvSec 2014. (to appear)
7. Buldas, A., Laanoja, R., Truu, A.: Efficient quantum-immune keyless signatures with identity. Cryptology ePrint Archive 2014/321 (2014)
8. Buldas, A., Laanoja, R., Truu, A.: Efficient Implementation of Keyless Signatures with Hash Sequence Authentication. Cryptology ePrint Archive 2014/689 (2014)
9. Buldas, A., Niitsoo, M.: Optimally tight security proofs for hash-then-publish time-stamping. In: Steinfeld, R., Hawkes, P. (eds.): ACISP 2010. LNCS 6168, pp. 318–335. Springer, Heidelberg (2010)
10. Buldas, A., Saarepera, M.: On provably secure time-stamping schemes. In: Lee, P.J. (ed.): ASIACRYPT 2004. LNCS 3329, pp. 500–514. Springer, Heidelberg (2004)
11. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. JACM 51 (4), 557–594 (2004)
12. Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgard revisited: How to construct a hash function. In: Shoup, V. (ed.): CRYPTO'05. LNCS 3621, pp. 430–448. Springer, Heidelberg (2005)
13. Dodis, Y., Pietrzak, K., Puniya, P.: A new mode of operation for blockciphers and length-preserving MACs. In: Smart, N. (ed.): EUROCRYPT 2008. LNCS 4965, pp. 198–219, Springer (2008)
14. Dodis, Y., Ristenpart, T., Shrimpton, T.: Salvaging Merkle-Damgård for practical applications. In: Joux, A. (ed.): Eurocrypt 2009. LNCS 5479, pp. 371–388. Springer, Heidelberg (2009)
15. Haber, S., Stornetta, W.-S.: How to time-stamp a digital document. Journal of Cryptology 3(2), 99–111 (1991)
16. Luby, M.: Pseudorandomness and Cryptographic Applications. Princeton University Press, Princeton (1996)
17. Maurer, U., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Naor, M. (ed.): TCC'04. LNCS 2951, pp. 21–39. Springer, Heidelberg (2004)
18. Merkle, R.C.: Protocols for public-key cryptosystems. In: Proceedings of the 1980 IEEE Symposium on Security and Privacy, pp. 122–134 (1980)
19. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: s synthetic approach. In: Stinson, D.R. (ed.): CRYPTO'93. LNCS 773, pp. 368–378. Springer (1993)
20. Shrimpton, T., Stam, M.: Building a collision-resistant compression function from non-compressing primitives. In: Aceto, L. et al (Eds.): ICALP 2008, Part II. LNCS 5126, pp. 643–654. Springer Heidelberg (2008)
21. Stam, M.: Blockcipher-based hashing revisited. In: Dunkelman, O. (Ed.): FSE 2009. LNCS 5665, pp. 67–83. Springer (2009)