WOTSwana: A Generalized $\mathcal{S}_{\text{leeve}}$ Construction for Multiple Proofs of Ownership

David Chaum[1], Mario Larangeira[2], and Mario Yaksetig[1]

[1] xx network, Cayman Islands
{david, mario, will}@xx.network
[2] Tokyo Institute of Technology and IOHK, Japan
mario@c.titech.ac.jp/mario.larangeira@iohk.io

Abstract. The $\mathcal{S}_{\text{leeve}}$ construction proposed by Chaum et al. (ACNS'21) introduces an extra security layer for digital wallets by allowing users to generate a "back up key" securely nested inside the secret key of a signature scheme, *i.e.*, ECDSA. The "back up key", which is secret, can be used to issue a "proof of ownership", *i.e.*, only the real owner of this secret key can generate a *single* proof, which is based on the WOTS+ signature scheme. The authors of $\mathcal{S}_{\text{leeve}}$ proposed the formal technique for a *single* proof of ownership, and only informally outlined a construction to generalize it to multiple proofs. This work identifies that their proposed construction presents drawbacks, *i.e.*, varying of signature size and signing/verifying computation complexity, limitation of linear construction, etc. Therefore we introduce *WOTSwana*, a generalization of $\mathcal{S}_{\text{leeve}}$, which is, more concretely, a more general scheme, *i.e.* an extra security layer that generates *multiple* proofs of ownership, and put forth a thorough formalization of two constructions: (1) one given by a linear concatenation of numerous WOTS+ private/public keys, and (2) a construction based on tree like structure, *i.e.*, an underneath Merkle tree whose leaves are WOTS+ private/public key pairs. Furthermore, we present the security analysis for multiple proofs of ownership, showcasing that this work addresses the early mentioned drawbacks of the original construction. In particular, we extend the original security definition for $\mathcal{S}_{\text{leeve}}$. Finally, we illustrate an alternative application of our construction, by discussing the creation of an encrypted group chat messaging application.

Keywords: Hash-based signatures · Post-quantum cryptography · ECDSA.

1 Introduction

The ECDSA based wallets have been target to intensive exposure given its wide use in cryptopocurrencies, *e.g.*, Bitcoin [21], Ethereum [24] and Ouroboros [2,12,16], which has driven the research community to channel its efforts to propose new attacks to the signature scheme/wallets [1,23]. The solution proposed by Chaum et al. [7], *i.e.* $\mathcal{S}_{\text{leeve}}$, is a signature based new cryptographic primitive designed to mitigate damages during massive leaks of private information of wallets. In a

nutshell, the construction in [7] allows the rightful user to prove its ownership in the face of its secret key becoming public. Important to note, that proving the knowledge of the correct secret key, via zero knowledge protocols for example, is of no use as, potentially, anyone could generate such a proof during a massive leak. The main technique of $\mathcal{S}_{\text{leeve}}$ is to leverage the regular ECDSA scheme by having a nested "back up key" to generate the proof of ownership, or even to fully discard the ECDSA scheme for a (post-quantum) signature scheme; a hash based signature scheme.

The Single Proof of Ownership of $\mathcal{S}_{leeve}$ The most significant novelty of [7] is the introduction of a *second layer of security* by allowing the user to verify the correct ownership of the leaked keys, which would be impossible otherwise. However this feature, as fully presented in [7], is rather limited given that only *a single* proof can be issued. The main construction relies on a variant of the WOTS+ [15], therefore it can be used only once. The authors of [7] mitigate it by presenting the sketch of a general construction which concatenates several instances of the WOTS+ like scheme in order to generate multiple proofs. Unfortunately, the description is rather informal and the construction seemingly introduces an unusual feature: the signature has varying size and sign/verification times depending on how many proofs of ownership were previously issued.

By closer inspection of their described construction, one would realize that the varying time and size of the proofs are due to the level of the linear sequence of WOTS+ like *ladders*, *i.e.*, sequence of hash function executions. Given that each proof generation uses one instance of the signature, the issuer has to keep the state, *i.e.* the "height" position, in the sequence of ladders and add it to the signature. Moreover the presented security analysis does not cover the case that the adversary can sample several proof of ownership in order to come up with a forgery. Their security proofs focus on the single proof case.

Related Works. The work by Chaum et al. [7] seems to be unique in the sense that introduces a novel strategy in adding extra layer of security to wallets. It, for example, sharply contrasts with hot/cold wallet strategies used before, as for example[11,14,9]. A follow up work by Chaum et al. [6] introduces a much more robust security analysis. Whereas the original construction relies on the L-Tree data structure, as it was introduced by Hülsing [15], the work in [6] updates the original Sleeve construction to support Tweakable Hash functions as introduced by Bernstein et al. [3]. The latter puts forth a more module design and therefore it is more desirable approach, which is use in our constructions.

Our Contribution This work tackles the early mentioned limitations of [7]. That is, we introduce the first generalization of $\mathcal{S}_{\text{leeve}}$ by proposing two constructions for "multiple generations of proof of ownership": the linear and the tree constructions. The main difference between them is how the $\mathcal{S}_{\text{leeve}}$ back up key is kept underneath the ECDSA secret key. We recall the $\mathcal{S}_{\text{leeve}}$ introduces an extra key, the early mentioned "back up key", in the key generation algorithm in addition to the verification and secret keys. Whereas the verification and secret

key are used as a regular signature key pair, the back up key remains secret and is only used when issuing the proof of ownership.

Novel Constructions. The proof of ownership of the original $\mathcal{S}_{\text{leeve}}$ is a variant of the WOTS+ signature scheme [15] by Hülsing, named Extended WOTS+ (eWOTS+). Briefly, the proof generated by [7] is an eWOTS+ signature, whose the single respective private key is kept nested into the ECDSA secret key, *i.e.*, thus used as the "back up key". This work reviews this design and proposes *Wotswana* with two constructions: (1) a linear construction, under a similar design to the one outlined in [7], and (2) a tree based construction, *i.e.*, the nested key pairs are kept under a Merkle Tree nested inside the ECDSA secret key. Both constructions rely on the regular WOTS+ signature scheme, instead of the eWOTS+.

In our (1) *linear construction*, blocks of WOTS+ keys are concatenated linearly. Briefly, the "deepest" WOTS+ verification key is used to derive the next WOTS+ secret key, up to the point that the uppermost WOTS+ verification key is converted into the ECDSA secret/verification key pair. This construction naturally extends the original [7] by adding more underlying WOTS+ key pairs as the set of backup keys to be used to issue proofs of ownership. Providing varying verification/generation time complexity as the outlined construction of [7]. The (2) *tree construction* does not compose the early described "sequence" of WOTS+ blocks. Instead, it organizes them as leaves of a Merkle Tree, such that the root of the tree is converted into the ECDSA key pair. This design provides the advantage that, although both constructions keep a state, *i.e.*, the WOTS+ pair to be used for the proof of ownership, the tree construction does not require the verification routine to transverse the whole structure (as in (1) linear construction). For comparison, we remark that while verifying the linear construction proof of ownership, the routine transverses the linear structure from the point in the structure upwards until the ECDSA verification key. The Merkle Tree structure prevents that in the tree based construction. Furthermore, our linear and tree based constructions rely on the original WOTS+ construction, instead of eWOTS+ from [7], because they are based on Tweakable Hash Functions (while [7] relies on L-Trees [10]). This brings the advantage of using a more well established signature scheme.

Multiple Proofs and Fallback. Furthermore we extend the original proof of ownership [7] security definition which is suitable for *only* a single proof. Concretely, our proposed security definition allows the adversary to have access to a "proof generation oracle". Therefore, for a given Wotswana scheme with capability of t proofs of ownership, the adversary can query the oracle for at most $t - 1$ proofs, before attempting to generate a forgery of its own. Yet we highlight that while the original constructions reveal the ECDSA secret in order to prove ownership, our constructions keep the full secret-key undisclosed even after numerous proofs of ownership generations.

Our construction does not rely on any type of publishing on public data structures (e.g., a blockchain). This allows for a cleaner and more secure design that takes place completely off-chain during the key generation phase of the

users. For example, if users posted a signed hash of the secret key using the fallback key instead, there is the risk of a block producer taking advantage of the miner extractable value (MEV) and front-running these different transactions and posting such data with a malicious signature instead. As a result, solutions that rely on posting on a public board, require indistinguishability assumptions of the underlying and potentially different post-quantum signature schemes. It is a conservative approach to avoid that sort of additional assumption. We highlight that our system is front-running resistant in a setting where there is an ongoing proof-of-ownership stage (e.g., traditional transactions are halted). Therefore, a miner extracting the user's private key from a proof-of-ownership signature should not be able to steal the user's money.

The motivation for this design is to allow users to rollover to a quantum-secure blockchain—using the fallback key—and then use the multiple proofs-of-ownership as a main mechanism to perform transactions in the new chain. Therefore, the construction can potentially act as a replacement for hash-based signature schemes such as SPHINCS [3] or XMSS [5] as it results in constant-sized signatures at the cost of linear verification time. The chain, however, can feature potential improvements as it can act as a state-keeping layer.

We also carried out a formal method analysis of Wotswana using Verifpal [19,17]. Our formal analysis shows that our design preserves the confidentiality of the underlying keys.

In summary our contributions are:

- **Two Wotswana Constructions:** Section 3 presents the two designs , the *linear* and the *tree* constructions. Both keep a state, meaning how many proofs of ownership were issued. However the main difference is how the WOTS+ keys are kept internally. Namely, in linear or tree fashion;
- **Security Analysis:** Section 4 introduces the security analysis of our protocol with respect to the Key Derivation Function (KDF), the component used in our construction to internally concatenate WOTS+ blocks, and our two constructions for Wotswana;
- **Formal Methods:** Next, in Section 5, after a brief description of the main tool used, *i.e.*, Verifpal, we describe our proposed model for the Wotswana Protocol. We end our formal methods approach by discussion the interpretation of our results;
- **Application:** Finally, Section 6 discusses Encrypted Group Chat as an application for Wotswana. In particular, we discuss the applicability of multiple proofs of ownership.

2 Preliminaries

It is convenient to quickly review the WOTS+ signature construction from [15], the Tweakable Hash proposal from [3], the original $\mathcal{S}_{\text{leeve}}$ definitions [7], and the definitions for Key Derivation Function (KDF) [8,20] used in our constructions. We also denote Probabilistic Polynomial Time algorithms as PPT.

2.1 The WOTS+ Signature Scheme

Here we review the original WOTS+ signature scheme. The Wotswana later constructions, linear and tree, rely on the standard WOTS+ construction.

Definition 1 (Family of Functions). *Given the security and the Winternitz parameters, respectively, $\lambda \in \mathbb{N}$ and $w \in \mathbb{N}, w > 1$, let a family of functions $\mathcal{H}_\lambda$ be $\{h_k : \{0,1\}^\lambda \to \{0,1\}^\lambda | k \in \mathcal{K}_\lambda\}$ with key space $\mathcal{K}_\lambda$.*

Definition 2 (Chaining Function). *Given a family of functions $\mathcal{H}_\lambda$, $x \in \{0,1\}^\lambda$, an iteration counter $i \in \mathbb{N}$, a key $k \in \mathcal{K}_\lambda$, for j $\lambda-$bit strings $\mathbf{r} = (r_1, \ldots, r_j) \in \{0,1\}^{\lambda \times j}$ with $j \geq i$, then we have the chaining function as follows*

$$c_k^i(\mathbf{r}, x) = \begin{cases} h_k(c_k^{i-1}(\mathbf{r}, x) \oplus r_i), \ 1 \leq i \leq j; \\ x, \ i = 0. \end{cases}$$

We rely on the same setting from [15], that is this work assumes the chaining function uses a family of functions $\mathcal{F}_n : \{f_k : \{0,1\}^n \to \{0,1\}^n | k \in \mathcal{K}_n\}$ with a key space $\mathcal{K}_n$. Additionally, we review the notation for the subset of randomness vector $\mathbf{r} = (r_1, \ldots, r_\ell)$, and denote by $\mathbf{r}_{a,b}$ the subset of $(r_a, \ldots, r_b)$.

Definition 3 (W-OTS$^+$). *Given the security parameter λ, a chaining function c, and $k \leftarrow \mathcal{K}$ from the key space $\mathcal{K}$, the W-OTS$^+$ signature scheme is the tuple $(\mathsf{Gen}_W, \mathsf{Sign}_W, \mathsf{Verify}_W)$, defined as in Table 1.*

The Security of WOTS+. The standard security notion for digital signature schemes is existential unforgeability under adaptive chosen message attacks (EU-CMA) which is defined using the following experiment. By DSS(1^λ), we denote the digital signature scheme (DSS) with security parameter λ, then we model the security by defining the security experiment $\mathsf{Exp}^{\mathsf{EU\text{-}CMA}}_{\mathsf{DSS}(1^\lambda)}(\mathcal{A})$, as follows:

Experiment $\mathsf{Exp}^{\mathsf{EU\text{-}CMA}}_{\mathsf{DSS}(1^\lambda)}(\mathcal{A})$
$(\mathsf{sk}, \mathsf{pk}) \longleftarrow \mathsf{keygen}(1^\lambda)$
$(M^*, \sigma^*) \longleftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(\mathsf{pk})$
Let $\{M_i, \sigma_i\}_1^q$ be the query-answer pairs of $\mathsf{Sign}(sk, \cdot)$
Return 1 iff $\mathsf{Verify}(pk, M^*, \sigma^*) = 1$ and $M^* \notin \{M_i\}_1{}^q$

We define the success probability of the adversary $\mathcal{A}$ in the above EU-CMA experiment as $\mathsf{Succ}^{\mathsf{EU\text{-}CMA}}_{\mathsf{DSS}(1^\lambda)}(\mathcal{A}) = \mathsf{Pr}[\mathsf{Exp}^{\mathsf{EU\text{-}CMA}}_{\mathsf{DSS}(1^\lambda)}(\mathcal{A}) = 1]$.

Definition 4 (EU-CMA). *For a polynomial $poly(\cdot)$, let $\lambda, t, q \in \mathbb{N}, t, q = poly(\lambda)$, DSS a digital signature scheme. It is said the DSS is EU-CMA-secure, if the maximum success probability $\mathsf{InSec}^{EU\text{-}CMA}(DSS(1^\lambda); t, q)$ of all possibly probabilistic adversaries $\mathcal{A}$, running in time $\leq t$, making at most q queries to Sign in the above experiment, is negligible in λ, $\mathsf{InSec}^{EU\text{-}CMA}(DSS(1^\lambda); t, q) = max\ \{\mathsf{Succ}^{EU\text{-}CMA}_{DSS(1^\lambda)}(\mathcal{A})\} = negl(\lambda)$.*

$\mathsf{Gen}^k_W(1^\lambda)$	$\mathsf{Sign}^k_W(\mathsf{m},\mathsf{sk})$
Pick $(\ell+w-1)$ λ-bit strings r_i	Compute $\mathsf{m} \to (\mathsf{m}_1,\ldots,\mathsf{m}_{\ell_1})$,
Set $\mathsf{sk}_i \leftarrow r_i$, for $1 \le i \le \ell$	for $\mathsf{m}_i \in \{0,\ldots,w-1\}$
Set $\mathsf{sk} = (\mathsf{sk}_1,\ldots,\mathsf{sk}_\ell)$	Compute checksum $C = \sum_{i=1}^{\ell_1}(w-1-\mathsf{m}_i)$,
Set $\mathbf{r} = (r_{\ell+1},\ldots,r_{\ell+w-1})$	and its base w representation $(C_1,\ldots,C_{\ell_2})$,
Set $\mathsf{vk}_0 = (\mathbf{r},k)$	for $C_i \in \{0,\ldots,w-1\}$
Set $\mathsf{vk}_i = c_k^{w-1}(\mathbf{r},\mathsf{sk}_i)$, $1 \le i \le \ell$	Parse $B = \mathsf{m}\|C$ as $(b_1,\ldots,b_{\ell_1+\ell_2})$
Set $\mathsf{vk} = (\mathsf{vk}_0,\mathsf{vk}_1,\ldots,\mathsf{vk}_\ell)$	Set $\sigma_i = c_k^{b_i}(\mathbf{r},\mathsf{sk}_i)$, for $1 \le i \le \ell_1+\ell_2$
Return $(\mathsf{sk},\mathsf{vk})$	Return $\sigma = (\sigma_1,\ldots,\sigma_{\ell_1+\ell_2})$

$\mathsf{Verify}^k_W(\mathsf{m},\mathsf{vk},\sigma)$
Compute $\mathsf{m} \to (\mathsf{m}_1,\ldots,\mathsf{m}_{\ell_1})$,
for $\mathsf{m}_i \in \{0,\ldots,w-1\}$
Compute checksum $C = \sum_{i=1}^{\ell_1}(w-1-\mathsf{m}_i)$,
and the base w representation $(C_1,\ldots,C_{\ell_2})$,
for $C_i \in \{0,\ldots,w-1\}$
Parse $B = \mathsf{m}\|C$ as $(b_1,\ldots,b_{\ell_1+\ell_2})$
Return 1, if the following equations hold
$\mathsf{vk}_0 = (\mathbf{r},k)$
$\mathsf{vk}_i = c_k^{w-1-b_i}(\mathbf{r}_{b_i+1,w-1},\sigma_i)$ for $1 \le i \le \ell_1+\ell_2$

Table 1: The main idea of the W-OTS$^+$ construction is to create "ladders" of hash function executions, via the Chaining Function $c_k^i(\cdot,\cdot)$, from the secret keys sk_i to the verification keys vk_i.

2.2 Tweakable Hash Functions

Tweakable hash functions allow for a better abstraction of hash-based signature scheme description. By decoupling the computations of hash chains, hash trees, and nodes, protocol designers can separate the analysis of the high-level construction from exactly how the computation is done. Therefore abstracting the computation away in hash-based schemes only requires analyzing the hashing construction. The standard definition is as follows.

Definition 5 (Tweakable Hash Function). *For a security parameter λ and a polynomial $n(\lambda)$, a tweakable hash function has three inputs: a public parameter $P \in \mathcal{P}$, a tweak $T \in \mathcal{T}$ and a message $M \in \{0,1\}^\alpha$. The hash produces an output digest $\mathcal{MD} \in \{0,1\}^{n(\lambda)}$: Let $\mathcal{P}$ the public parameters space, $\mathcal{T}$ the tweak space, and $n, \alpha \in \mathbb{N}$. A Tweakable Hash Function is an efficient function mapping an α-bit message M to an n-bit hash value MD using a function key called public parameter $P \in \mathcal{P}$ and a tweak $T \in \mathcal{T}$. Therefore, we have $\mathsf{Th} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^\alpha \to \{0,1\}^{n(\lambda)}$, $\mathsf{MD} \leftarrow \mathsf{Th}(P,T,M)$.*

For later sections, we may omit the security parameter out of polynomial $n(\lambda)$.

A tweakable hash function takes public parameters P and context information in the form of a tweak T in addition to the message. The public parameters

might be thought of as a function key or index. The tweak might be interpreted as a nonce. We use the term public parameter for the function key to emphasize that it is intended to be public.

2.3 The Security of $\mathcal{S}_{\text{leeve}}$

The $\mathcal{S}_{\text{leeve}}$ primitive is composed by the tuple $(\mathsf{Gen}_{\pi_{\mathcal{S}_{\text{leeve}}}}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Verify\text{-}Proof})$. The generation algorithm outputs the pairs of keys, vk and sk, and the backup key bk. The first pair is the regular verification key, used for verifying a signature, and the secret-key used for issuing a signature. While the last key is used to issue the *Proof of Ownership* $\pi_{\mathcal{S}_{\text{leeve}}}$, with respect to vk as follows.

Definition 6 ($\mathcal{S}_{\text{leeve}}$ [7]). *A fallback scheme* $\mathcal{S}_{leeve} = (\mathsf{Gen}_{\pi_{\mathcal{S}_{leeve}}}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Verify\text{-}Proof})$ *is a set of PPT algorithms:*

- $\mathsf{Gen}_{\pi_{\mathcal{S}_{leeve}}}(1^\lambda)$ *on input of a security parameter* λ *outputs a private signing key* sk*, a public verification key* vk *and the back up key* bk*;*
- $\mathsf{Sign}(\mathsf{sk}, \mathsf{m})$ *outputs a signature* σ *under* sk *for a message* m *using the designated main signature scheme, in our example this is an ECDSA signature;*
- $\mathsf{Verify}(\mathsf{vk}, \sigma, \mathsf{m})$ *outputs* 1 *iff* σ *is a valid signature on* m *under* vk*;*
- $\mathsf{Proof}(\mathsf{bk}, c)$ *on input of the backup information* bk *and the challenge* c*, it outputs the ownership proof* $\pi_{\mathcal{S}_{leeve}}$*. In our example, this is a WOTS+ signature on the challenge* c *using the fallback key* bk*;*
- $\mathsf{Verify\text{-}Proof}(\mathsf{vk}, \mathsf{sk}, \pi_{\mathcal{S}_{leeve}}, c)$ *is a deterministic algorithm that on input of a public-key* vk*, secret-key* sk*, a ownership proof* $\pi_{\mathcal{S}_{leeve}}$ *and a challenge* c*, it outputs either* 0*, for an invalid proof, or* 1 *for a valid one.*

The two main security properties of $\mathcal{S}_{\text{leeve}}$ are (1) the capability of issuing a proof to confirm the ownership of the secret key, even in the face of a massive leakage, when the secret key becomes public, and (2) the capability to smoothly switch to another signature scheme, namely a quantum resistant one. Briefly, we formally review both properties.

Definition 7 (Single Proof of Ownership [7]). *For any PPT algorithm* $\mathcal{A}$ *and security parameter* λ*, it holds* $\Pr[(\mathsf{vk},\mathsf{sk},\mathsf{bk}) \leftarrow \mathsf{Gen}_{\pi_{\mathcal{S}_{leeve}}}(1^\lambda) : (c^*, \pi^*_{\mathcal{S}_{leeve}}) \leftarrow \mathcal{A}(\mathsf{sk}, \mathsf{vk}) \wedge \mathsf{Verify\text{-}Proof}(\mathsf{vk}, \mathsf{sk}, \pi^*_{\mathcal{S}_{leeve}}, c^*) = 1] < negl(\lambda)$ *for all the probabilities are computed over the random coins of the generation and proof verification algorithms and the adversary.*

Definition 8 (Fallback [7]). *We say that the scheme* $(\mathsf{Gen}_{\pi_{\mathcal{S}_{leeve}}}, \mathsf{Sign}, \mathsf{Verify})$*, with secret and verification key respectively* sk *and* vk *such that* $\mathsf{Gen}_{\pi_{\mathcal{S}_{leeve}}}(1^\lambda) \rightarrow (\mathsf{vk}, \mathsf{sk}, \mathsf{bk})$*, has fallback if there are sign and verification algorithms* $\mathsf{Sign}_{\pi_{\mathcal{S}_{leeve}}}$ *and* $\mathsf{Verify}_{\pi_{\mathcal{S}_{leeve}}}$ *such that* sk *and* bk *can be used as verification and secret*

keys respectively, along with $\mathsf{Sign}_{\pi_{\mathcal{S}_{leeve}}}$ *and* $\mathsf{Verify}_{\pi_{\mathcal{S}_{leeve}}}$ *as fully independent signature scheme.*

2.4 Key Derivation Functions

The Key Derivation Function KDF [8,20] is a cryptographic component that takes as input an initial source of entropy, or initial keying material, and allows for the derivation of one (or more) cryptographically secure key values. This input is not necessarily uniformly distributed and the adversary may have partial knowledge of such input. The adversary, however, should not be able to distinguish an output from a random uniform string of the same length, and a KDF output should not leak information on any of the other generated bits. We note that these KDF output values are not necessarily exclusive for secret key derivation and may optionally be made public depending on the cryptographic use case.

Definition 9 (KDF [20]). *A* KDF *accepts as input four arguments: a value* σ *sampled from a source of keying material, a length value* ℓ*, a salt value* r *defined over a set of possible salt values, and a context variable* c*. The latter two values are both optional. As a result, these values can either be null, or assigned a constant value. In this setting, the source of keying material* Σ *is a two-valued probability distribution* (σ, α) *generated by an efficient probabilistic algorithm. The resulting* KDF *output is a string of* ℓ *bits.*

In this model, the adversary $\mathcal{A}$ should be given the value pair (σ, α) to model the "partial knowledge" of the input entropy. Later in our constructions we rely on the use of $\mathsf{KDF}(\sigma, \ell, r, c)$, for fixed values of ℓ and c. In particular, we fix $r = null$. Hence, we drop two arguments in the later descriptions of the Wotswana by denoting the KDF as a two-value function $\mathsf{KDF}(\sigma, c) = \mathsf{KDF}(\sigma, \ell_{fix}, null, c)$, for a fixed length ℓ_{fix}.

Definition 10 (KDF One-wayness). *A* KDF *is* $(t_{\mathcal{A}}, q, \epsilon)$*-one-way secure if no adversary* $\mathcal{A}$ *running in time* $t_{\mathcal{A}}$ *and making at most* q *queries produces the input entropy* Σ*, when given the output value* σ *and the partial knowledge* α*, with probability* $p > \frac{q}{2^n} + \epsilon$*, where* n *is the length of the input entropy* $|\Sigma|$*.*

3 The WOTSwana Versions

We present two constructions; a linear and a tree based construction. While the former the WOTS+ ladders are concatenated in a *linear* fashion. Each new back up key has nested another one inside "deeper" in the linear structure. The verification of each new signature includes the generation of the previous back up key. The latter construction, *i.e.* the tree based, does not have this feature as each new back up key is located in a different branch. Figure 1 illustrates a simplified outline of both constructions.

3.1 The Linear Construction

Here we introduce a construction for the generation of t proofs of ownership. The main idea is to concatenate t blocks of a variant of the WOTS+ signature. Additionally, we review the notation for the subset of the randomness vector $\mathbf{r} = (r_1, \ldots, r_\ell)$. We denote by $\mathbf{r}_{a,b}$ the subset of $(r_a, \ldots, r_b)$, and our constructions to be presented next rely on the KDF [8,20].

The Auxiliary Blocks: Ladder and Block. Given the security parameter λ, a chaining function c, and $k \leftarrow \mathcal{K}$ from the key space $\mathcal{K}$, Table 2 defines the auxiliary procedures, namely *Ladder* and *Block*. The former is used to derive the new internal "internal public key" $(\mathsf{v}_0, \mathsf{v}_1, \ldots, \mathsf{v}_\ell)$ from the "internal secret key" $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$. While the latter uses the former, as an internal routine, to derive the secret key to the new "internal secret key", *i.e.* the key one level above in the linear structure. These auxiliary routines are used in the linear and tree based constructions in further sections.

$Ladder_w^k(\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$	$Block_w^{P,T,k}(\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$
Set $r_p \leftarrow \mathsf{KDF}(\mathsf{sk}_1 \| \ldots \| \mathsf{sk}_\ell, p), 1 \leq p \leq w-1$	$Ladder_w^k(\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell) \rightarrow (\mathsf{v}_0, \mathsf{v}_1, \ldots, \mathsf{v}_\ell)$
Set $\mathbf{r} = (r_1, \ldots, r_{w-1})$	Set $\mathsf{v} \leftarrow (\mathsf{v}_0, \mathsf{v}_1, \ldots, \mathsf{v}_\ell)$
Set $\mathsf{v}_0 = (\mathbf{r}, k)$	Set $\mathsf{seed} = \mathsf{Th}(P \| T \| \mathsf{v})$
Set $\mathsf{v}_i = c_k^{w-1}(\mathbf{r}, \mathsf{sk}_i)$, $1 \leq i \leq \ell$	Set $\mathsf{sk}'_i \leftarrow \mathsf{KDF}(\mathsf{seed}, i), 1 \leq i \leq \ell$
Return $\mathsf{v} = (\mathsf{v}_0, \mathsf{v}_1, \ldots, \mathsf{v}_\ell)$	Return $(\mathsf{sk}'_1, \ldots, \mathsf{sk}'_\ell)$

Table 2: The Ladder procedure performs the sequence of the hashes, *i.e.* the "ladders", from the secret key, in order to output a intermediate key, *i.e.* similar to WOTS+ public key. The *Block* procedure concatenates each of the secret key generation block to the next one.

From now we present the three main algorithms for key generation, and proof generation and verification for both constructions: Linear-Gen, Linear-Proof, in Table 3, and Linear-Verify-Proof in Table 4.

$\mathsf{Linear\text{-}Gen}_w^{k,t}(\lambda)$	$\mathsf{Linear\text{-}Proof}_w^{k,t,\mathsf{st}}(c,\mathsf{bk})$
Pick $P \xleftarrow{\$} \{0,1\}^{n(\lambda)}$, $\mathcal{X} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$	Parse $\mathsf{bk} \rightarrow (\mathbb{T}[\mathsf{st}], \mathsf{bk}_1^{(t)}, \ldots, \mathsf{bk}_\ell^{(t)})$
Pick $\mathbb{T}[t] \xleftarrow{\$} \{0,1\}^{n(\lambda)}$, random value a	Set $r_p \leftarrow \mathsf{KDF}(\mathsf{bk}_1^{(t)} \| \ldots \| \mathsf{bk}_\ell^{(t)} \| p), 1 \le p \le w-1$
Set $\mathsf{bk}_i^{(t)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}, 1 \le i \le \ell$	Set $\mathbf{r} = (r_1, \ldots, r_{w-1})$
For $t \ge i \ge 2$	For $t \ge i \ge \mathsf{st}$
$(\mathsf{bk}_1^{(i-1)}, \ldots, \mathsf{bk}_\ell^{(i-1)})$	$\mathbb{T}[i-1] \leftarrow \mathsf{KDF}(\mathbb{T}[i], i)$
$\leftarrow Block_w^{P,\mathbb{T}[i],k}(\mathsf{bk}_1^{(i)}, \ldots, \mathsf{bk}_\ell^{(i)})$	$(\mathsf{bk}_1^{(i-1)}, \ldots, \mathsf{bk}_\ell^{(i-1)})$
$\mathbb{T}[i-1] \leftarrow \mathsf{KDF}(\mathbb{T}[i], i)$	$\leftarrow Block_w^{P,\mathbb{T}[i],k}(\mathsf{bk}_1^{(i)}, \ldots, \mathsf{bk}_\ell^{(i)})$
Set $(\mathsf{bk}_1^{(1)}, \ldots, \mathsf{bk}_\ell^{(1)})$	Compute $c \rightarrow (c_1, \ldots, c_{\ell_1})$,
$\leftarrow Block_w^{P,\mathbb{T}[2],k}(\mathsf{bk}_1^{(2)}, \ldots, \mathsf{bk}_\ell^{(2)})$	for $c_i \in \{0, \ldots, w-1\}$
Set $\mathsf{v}^{(1)} \leftarrow Ladder(\mathsf{bk}_1^{(1)}, \ldots, \mathsf{bk}_\ell^{(1)})$	Compute checksum $C = \sum_{i=1}^{\ell_1}(w-1-c_i)$,
Set $W = \mathsf{Th}(P \| \mathbb{T}[1] \| \mathsf{v}^{(1)})$	and its base w representation $(C_1, \ldots, C_{\ell_2})$
Set $\mathsf{sk} = a \cdot \mathsf{Th}(P \| \mathcal{X} \| W)$	Parse $B = c \| C$ as $(b_1, \ldots, b_{\ell_1+\ell_2})$, $\ell = \ell_1 + \ell_2$
Set $\mathsf{vk} \leftarrow g^{\mathsf{sk}}$	Set $\sigma_i = c_k^{b_i}(\mathbf{r}, \mathsf{sk}_i)$, $1 \le i \le \ell$
Set $\mathsf{bk} \leftarrow (\mathbb{T}[t], \mathsf{bk}_1^{(t)}, \ldots, \mathsf{bk}_\ell^{(t)})$	Set $\sigma_0 \leftarrow (\mathbb{T}[\mathsf{st}], st, \mathbf{r}, k), h \leftarrow g^a$
Return $(\mathsf{bk}, \mathsf{sk}, \mathsf{vk})$	Return $\pi_{\mathcal{S}_{\text{leeve}}} = (\sigma_0, \sigma_1, \ldots, \sigma_\ell, h)$

Table 3: The generation procedure selects the random secret key $(\mathsf{sk}_1^{(t)}, \ldots, \mathsf{sk}_\ell^{(t)})$, and interactively, by executing *Ladder* and *Block* algorithms, creates t signatures while keeping the tweaks in the list $\mathbb{T}$. The generation of the proofs works by traversing the linear construction for each new proof of ownership from t to 1.

$\mathsf{Linear\text{-}Verify\text{-}Proof}_w^{k,t,\mathsf{st}}(\mathsf{vk}, \pi_{\mathcal{S}_{\text{leeve}}}, c)$
Parse $\pi_{\mathcal{S}_{\text{leeve}}} \rightarrow (T, st, \mathbf{r}, k, \sigma_1, \ldots, \sigma_\ell, h)$
Compute $c \rightarrow (c_1, \ldots, c_{\ell_1})$, $c_i \in \{0, \ldots, w-1\}$
Compute checksum $C = \sum_{i=1}^{\ell_1}(w-1-c_i)$,
and its base w representation $(C_1, \ldots, C_{\ell_2})$
Parse $B = c \| C$ as $(b_1, \ldots, b_{\ell_1+\ell_2})$ and $\ell = \ell_1 + \ell_2$
Set $\mathsf{v}_0 = (\mathbf{r}, k)$
Set $\mathsf{v}_i = c_k^{w-1-b_i}(\sigma_i, \mathbf{r}_{b_i+1,w-1})$, $1 \le i \le \ell$
Set $\mathsf{v} = (\mathsf{v}_0, \ldots, \mathsf{v}_\ell)$
For $st \ge j > 1$
$W^{(j)} = \mathsf{Th}(P \| T \| \mathsf{v}^{(j)})$
Set $(\mathsf{bk}_1^{(j)}, \ldots, \mathsf{bk}_\ell^{(j)}) \leftarrow \mathsf{KDF}(W^{(j)}, i)$, $1 \le i \le \ell$
Set $T \leftarrow \mathsf{KDF}(T, j)$
Set $\mathsf{v}^{(j-1)} \leftarrow Ladder(\mathsf{bk}_1^{(j)}, \ldots, \mathsf{bk}_\ell^{(j)})$
$W^{(1)} = \mathsf{Th}(P \| T \| \mathsf{v}^{(1)})$
$\mathsf{sk}' = \mathsf{Th}(P \| \mathcal{X} \| W^{(1)})$
If $\mathsf{vk} = h^{\mathsf{sk}'}$: Output 1
Else: Output 0

Table 4: The intuition of the verification procedure is that given the state st, *i.e.* the height in the linear structure, and execute the list of hashes and generation through the ladders up until the verification key on the upmost position.

3.2 The Tree Construction

The next construction (Tables 5 and 6) makes use of Merkle Tree, which is described as $(\mathsf{MT.Gen}, \mathsf{MT.Proof}, \mathsf{MT.Verify})$. That is, given 2^t strings $s^{(1)}, \ldots, s^{(2^t)}$, the root generation is given by $\mathsf{MT.Gen}(s^{(1)}, \ldots, s^{(2^t)}) = \mathsf{M}$, such that for any $s^{(i)}$, $\mathsf{MT.Proof}(\mathsf{M}, s^{(i)}) = \pi_{\mathsf{M}}^{(i)}$. The generated proof $\pi^{(i)}$, can be verified as the verification $\mathsf{MT.Verify}(\mathsf{M}, s^{(i)}, \pi_{\mathsf{M}}^{(i)}) = 1$. The proof fails to verify if the output is $\mathsf{MT.Verify}(\mathsf{M}, s^{(i)}, \pi_{\mathsf{M}}^{(i)}) = 0$.

$\mathsf{Tree\text{-}Gen}_w^{k,t}(\lambda)$	$\mathsf{Tree{-}Proof}_w^{k,\mathsf{st}}(c, \mathsf{bk})$
Pick $P \xleftarrow{\$} \{0,1\}^{n(\lambda)}$, random value a	Parse $\mathsf{bk} \rightarrow (\mathsf{M}, u, \ldots, (\mathsf{bk}_1^{(\mathsf{st})}, \ldots, \mathsf{bk}_\ell^{(\mathsf{st})}), \ldots)$
For $1 \leq i \leq t$	Set $r_p \leftarrow \mathsf{KDF}(\mathsf{bk}_1^{(\mathsf{st})} \| \ldots \| \mathsf{bk}_\ell^{(\mathsf{st})}, p), 1 \leq p \leq w-1$
Pick $\mathsf{bk}_j^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}, 1 \leq j \leq \ell$	Set $\mathbf{r} = (r_1, \ldots, r_{w-1})$
Set $T \leftarrow \mathsf{KDF}(\mathsf{bk}_1^{(i)} \| \ldots \| \mathsf{bk}_\ell^{(i)}, i)$	Set $(\mathsf{y}_1^{(\mathsf{st})}, \ldots, \mathsf{y}_\ell^{(\mathsf{st})}) \leftarrow Block_w^{P,T,k}(\mathsf{bk}_1^{(\mathsf{st})}, \ldots, \mathsf{bk}_\ell^{(\mathsf{st})})$
Set $(\mathsf{y}_1^{(i)}, \ldots, \mathsf{y}_\ell^{(i)})$ $\leftarrow Block_w^{P,T,k}(\mathsf{bk}_1^{(i)}, \ldots, \mathsf{bk}_\ell^{(i)})$	Set $\mathsf{v}^{(\mathsf{st})} \leftarrow Ladder(\mathsf{y}_1^{(\mathsf{st})}, \ldots, \mathsf{y}_\ell^{(\mathsf{st})})$
Set $\mathsf{v}^{(i)} \leftarrow Ladder(\mathsf{y}_i^{(i)}, \ldots, \mathsf{y}_\ell^{(i)})$	Set $T \leftarrow \mathsf{KDF}(\mathsf{y}_1^{(\mathsf{st})} \| \ldots \| \mathsf{y}_\ell^{(\mathsf{st})}, \mathsf{st})$
Set $W^{(i)} = \mathsf{Th}(P \| T \| \mathsf{v}^{(i)})$	Set $W = \mathsf{Th}(P \| T \| \mathsf{v}^{(\mathsf{st})})$
Set $\mathsf{M} = \mathsf{MT.Gen}(W^{(1)}, \ldots, W^{(t)})$	Set $\mathsf{MT.Proof}(W, \mathsf{M}) = \pi_{\mathsf{M}}$
Pick $u \xleftarrow{\$} \{0,1\}^{n(\lambda)}$	Compute $c \rightarrow (c_1, \ldots, c_{\ell_1})$, for $c_i \in \{0, \ldots, w-1\}$
Set $\mathsf{sk} \leftarrow a \cdot \mathsf{Th}(P \| \mathsf{M} \| u)$	Compute checksum $C = \sum_{i=1}^{\ell_1}(w-1-c_i)$ and its base w representation $(C_1, \ldots, C_{\ell_2})$
Set $\mathsf{vk} \leftarrow g^{\mathsf{sk}}$	Parse $B = c \| C$ as $(b_1, \ldots, b_{\ell_1+\ell_2})$ and $\ell = \ell_1 + \ell_2$
Set $\mathsf{bk} \leftarrow (\mathsf{M}, u, (\mathsf{bk}_1^{(1)}, \ldots, \mathsf{bk}_\ell^{(1)}),$ $\ldots, (\mathsf{bk}_1^{(t)}, \ldots, \mathsf{bk}_\ell^{(t)}))$	Set $\sigma_i = c_k^{b_i}(\mathbf{r}, \mathsf{bk}_i)$, for $1 \leq i \leq \ell$
	Set $\sigma_0 \leftarrow (\mathsf{M}, u, \pi_{\mathsf{M}}, \mathsf{st}, \mathbf{r}, k)$, $h \leftarrow g^a$
Return $(\mathsf{bk}, \mathsf{sk}, \mathsf{vk})$	Return $\pi_{\mathcal{S}_{\text{leeve}}} = (\sigma_0, \sigma_1, \ldots, \sigma_\ell, h)$

Table 5: Differently from the linear construction, the sets of values for each back up key $(\mathsf{bk}_1^{(i)}, \ldots, \mathsf{bk}_\ell^{(i)})$ in a Merkle Tree whose root is given by its root M. The root of the Merkle-tree M is a component of the signature, therefore it is only revealed when generating a proof of ownership. Namely, it is not disclosed while using the ECDSA signature.

4 Security Analysis

We assume an adversary attempting to forge a single WOTSwana proof-of-ownership, hence this section discusses the unforgeability of such proofs for our linear and tree based constructions, respectively Sections 4.2 and 4.3. Given that both constructions allow multiple proofs, we start by defining an extended security game, in comparison to the one reviewed in Section 2.3, where we model

$\mathsf{Tree\text{-}Verify\text{-}Proof}_w^{k,\mathsf{st}}(\mathsf{vk}, \pi_{\mathcal{S}_{\text{leeve}}}, c)$
Parse $\pi_{\mathcal{S}_{\text{leeve}}} \rightarrow (\sigma_0, \sigma_1, \ldots, \sigma_\ell, h)$
Parse $\sigma_0 \rightarrow (\mathsf{M}, u, \pi_{\mathsf{M}}, \mathsf{st}, \mathbf{r}, k)$
Compute $c \rightarrow (c_1, \ldots, c_{\ell_1})$, $c_i \in \{0, \ldots, w-1\}$
Compute checksum $C = \sum_{i=1}^{\ell_1}(w - 1 - c_i)$,
and its base w representation $(C_1, \ldots, C_{\ell_2})$
Parse $B = c\|C$ as $(b_1, \ldots, b_{\ell_1+\ell_2})$ and $\ell = \ell_1 + \ell_2$
Set $y_i = c_k^{w-1-b_i}(\mathbf{r}_{b_i+1,w-1}, \sigma_i)$, $1 \leq i \leq \ell$
Compute $(\mathsf{v}_0, \ldots, \mathsf{v}_\ell) \leftarrow Ladder(y_1, \ldots, y_\ell)$
Set $T \leftarrow \mathsf{KDF}(\mathsf{v}_0\| \ldots \|\mathsf{v}_\ell, \mathsf{st})$
Compute $W = \mathsf{Th}(P\|\|T\|\|(\mathsf{v}_0, \ldots, \mathsf{v}_\ell))$
If the following equations hold, return 1
$\mathsf{MT.Verify}(\mathsf{M}, W, \pi_{\mathsf{M}}) = 1$
$\mathsf{vk} = h^{\mathsf{Th}(P\|\|\mathsf{M}\|\|u)}$

Table 6: The verification of the proof of ownership $\pi_{\mathcal{S}_{\text{leeve}}}$ depends directly on a two-step verification: (1) the used back up key is part of the Merkle Tree given in the proof, and (2) and the obtained public key vk is correct.

the adversary accessing multiple proofs before issuing its forgery. Next we prove the security of our constructions in the light of such a security game.

4.1 The Extended Security Definition for WOTSwana

The next definition provide the adversary with access to the Ownership Proof Oracle $\mathcal{O}_{\mathsf{Proof}}(\mathsf{bk}, \cdot)$, since the original scheme has the capacity of only a *single* proof, extending it to multiple ones. The adversary can sample up to $t-1$ proofs by querying the oracle with challenges c_i of its choice, assuming the scheme with capacity of t proofs of ownership.

Definition 11 (Multiple Proofs of Ownership). *For any PPT $\mathcal{A}$, which can query the Ownership Proof Oracle $\mathcal{O}_{\mathsf{Proof}}(\mathsf{bk}, \cdot)$ for challenge-proofs pairs $(c_i, \pi^i_{\mathcal{S}_{leeve}})$, and a list of queried pairs $\mathcal{C}$ initially empty, on a polynomial number of queries, it holds*

$$\Pr[(\mathsf{vk},\mathsf{sk},\mathsf{bk}) \leftarrow \mathsf{Gen}_{\pi_{\mathcal{S}_{leeve}}}(1^\lambda) : (c^*, \pi^*_{\mathcal{S}_{leeve}}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Proof}}(\mathsf{bk},\cdot)}(\mathsf{sk}, \mathsf{vk}) \\ \wedge (c^*, \pi^*_{\mathcal{S}_{leeve}}) \notin \mathcal{C} \wedge \mathsf{Verify\text{-}Proof}(\mathsf{vk},\mathsf{sk},\pi^*_{\mathcal{S}_{leeve}}, c^*) = 1] < negl(\lambda)$$

for all the probabilities are computed over the random coins of the generation and proof verification algorithms and the adversary.

4.2 The Unforgeability of the (Linear) Proof of Ownership

To produce a forgery and subvert the security of our construction, $\mathcal{A}$ may attempt to explore different attack vectors. For example, $\mathcal{A}$ may attempt to invert

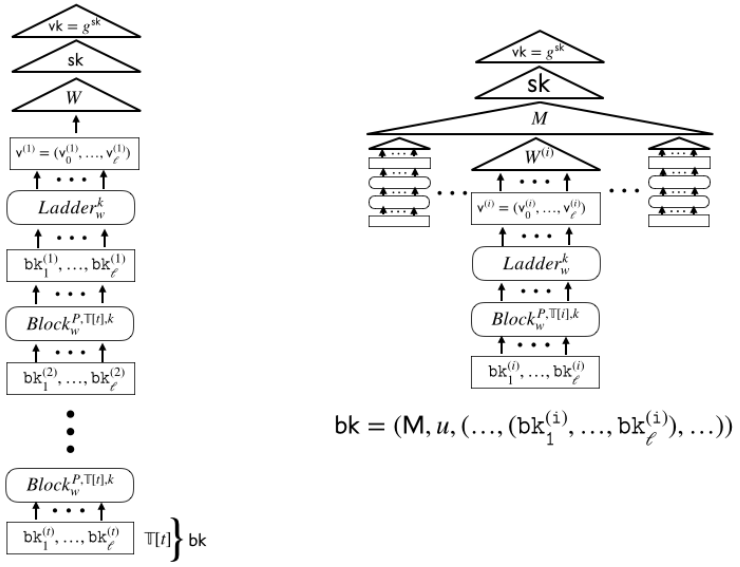

Fig. 1: Outline of the linear (left) and the tree (right) based constructions of WOTSwana. Note that the linear constructions back up key is the lowest part of the data structure. Whereas the tree based construction, the back up key is given by the Merkle Tree root M, the random value u and the each individual WOTS+ key $\mathtt{bk}^{(i)} = (\mathtt{bk}_1^{(i)}, \ldots, \mathtt{bk}_\ell^{(i)})$.

the used KDF, find a collision in the used tweakable hash function $\mathsf{Th}(\cdot)$, break the unforgeability of the used signature scheme (i.e., WOTS+), or even forge a proof of inclusion for the Merkle tree used in the construction. These scenarios represent the different attack vectors we identified and model in this section.

Theorem 1. *Consider the construction in Table 3, and assume a secure* KDF*, as in Definition 10, is used as in the Block and Ladder routines, given by Table 2 with security parameter* λ *and a* b*-bit long string as seed, then a PPT adversary* $\mathcal{A}$*, with running in time at most* $t_{\mathcal{A}}$*, produces a forged proof of ownership for any state* $\mathsf{st} = j$ *with success probability at most* $\epsilon \leq 1/2^b$ *by performing an exhaustive search over all possible seed values.*

Proof: (Sketch.) We consider that the security of a key derivation function is measured by the amount of work required to distinguish the output of the KDF from a truly uniformly distributed bit string of the same length, under the assumption that the seed is the only unknown input to the KDF. We know that the security upper bound for the subversion of the KDF is defined by the exhaustive search over all the possible seed values, which can be recovered in (at least) 2^b attempts, where b is the bit-length of the seed. This bound holds if the output key is sufficiently long (*i.e.*, no less than b bits).

This is true for our construction in an arbitrary state $\mathsf{st} = j$, as the key generation uses the WOTS+ of the level below as a secret seed to generate the next upper level. From this seed, which has a total size of b bits, the KDF produces a total of ℓ secret key values acting as the WOTS+ secret keys, each with b bits. Therefore, since the output size of the KDF is a total of a bit stream of size $\ell \cdot b$, the expected KDF security level is $\lambda \leqslant b$. □

Theorem 2. *Consider the construction in Table 3. Given the consecutive public-key* pk_j *and* σ_j, $j \in \{1, \ldots, t-1\}$ *and* $\mathsf{pk}_j = (\mathsf{vk}_1^{(j)}, \ldots, \mathsf{vk}_\ell^{(j)})$, *where* t *is the total number of public-key levels (*i.e.*, states) of the linear construction. If* $\mathcal{H}_n$ *is the function used in the chaining function, and* Th *is from a second preimage resistant hash-function family, then an adversary* $\mathcal{A}$ *with running in time at most* $t_{\mathcal{A}}$ *has negligible success probability of producing a proof of ownership forgery for any level* j.

Proof: (Sketch.) In order to prove the theorem, consider a game between an adversary $\mathcal{A}$, and a challenger which provides access to a proof of ownership oracle $\mathcal{O}$. The oracle receives a challenge c and a state j such that $j \in \{1, \ldots, t-1\}$, and returns a proof of ownership $\pi_{\mathcal{S}_{\text{leeve}}}$ and the WOTS+ verification key internally generated by the *Ladder* procedure of Table 2, *i.e.* $(\mathsf{vk}_0, \mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$. The adversary goal is to output a proof of ownership forgery $\pi'_{\mathcal{S}_{\text{leeve}}}$ for an arbitrary challenge of its choice $c_\prime$.

Since WOTS+ uses a family of functions $\mathcal{F}_n : \{f_k : \{0,1\}^n \to \{0,1\}^n | k \in \mathcal{K}_n\}$ with a key space $\mathcal{K}_n$, and we know from [15] that, to attack the EU-CMA property, an adversary $\mathcal{A}$ must break the security level λ_1, such that $\lambda_1 \geqslant n - \log_2(w^2\ell + w)$, given the WOTS+ parameter w.

Assume $\mathcal{A}$ succeeds. Then $\mathcal{A}$ produces a forgery for the challenge c' such that $c \neq c'$, for previously queried challenges. Then $\mathcal{A}$ breaks the unforgeability of the WOTS+ scheme, which is considered infeasible as long as the scheme is instantiated with an appropriate security parameter.

Alternatively, $\mathcal{A}$ may attempt to subvert the underlying tweakable hash function Th used for public-key compression and find a different set of top ladder values or verification keys that result in a public-key collision. Therefore, to successfully perform this attack, $\mathcal{A}$ must find a colliding set of verification keys $(\mathsf{vk}'_1, \ldots, \mathsf{vk}'_\ell) \neq (\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$ and a potentially different tweak T', such that $\mathsf{Th}(P \,||\, T' \,||\, \mathsf{vk}'_1, \ldots, \mathsf{vk}'_\ell) = \mathsf{Th}(P \,||\, T \,||\, \mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$. We know from [3] that to find the described collision, $\mathcal{A}$ must break the second-preimage resistance property of Th. This break corresponds to an amount of work of $\lambda_2 = 2^n$, where n is the digest output of the used tweakable hash function. As a result, the security level λ of our construction against an attacker attempting to produce a forgery is equivalent to the attack that requires the less amount of work. Consequently, $\lambda = \mathsf{min}\{\lambda_1, \lambda_2\}$.

We note that, however, previous proofs-of-ownership are trivially forgeable, as the public key of the level j is used as a seed for the secret key generation of the level $j - 1$. □

Theorem 3. *The adversary $\mathcal{A}$ running in time at most $t_{\mathcal{A}}$, that issues arbitrary challenges c_j, and receives consecutive public-key and proofs of ownership $(\mathsf{pk}_j, \pi_{\mathcal{S}_{leeve}}{}^{(j)})$, $j \in \{1, \ldots, t-1\}$ and $\mathsf{pk}_j = (\mathsf{vk}_1^{(j)}, \ldots, \mathsf{vk}_\ell^{(j)})$, has negligible probability of producing a proof of ownership forgery for any of the unrevealed levels $j + \delta$, s.t. $\delta \geq 1$, if both $\mathcal{H}_n$ and Th are from a second preimage resistant hash-function family.*

Proof: (Sketch.) We start by highlighting that this proof is an extension of the previous proof where $\mathcal{A}$ queries the oracle for a proof of ownership on an arbitrary challenge using the key of index $j \in \{1, \ldots, h\}$. In this setting, however, the oracle $\mathcal{O}$ receives a set of consecutive index queries and challenge pairs (j, c_j), and releases the corresponding proofs $\pi^{(j)}_{\mathcal{S}_{\text{leeve}}}$ for each of the queried levels. The previous proof is easily applicable to this stronger adversarial setting, where $\mathcal{A}$ starts by setting the index to query to the first level of the construction ($j = 1$), creates a challenge of their choosing, and sends it to the oracle. Upon every response from the oracle, $\mathcal{A}$ successively increments the index j by 1, creates a new challenge for that index, and sends the challenge/index pair to the $\mathcal{O}$. To succeed, $\mathcal{A}$ must produce a forgery of the proof-of-ownership for level $j + 1$.

In this case, $\mathcal{A}$ must perform the exact same work and subvert the described security level λ. Initially, the reader may expect that the release of different key material associated with different indexes could provide $\mathcal{A}$ with additional knowledge or even additional attack vectors (e.g., multi-target attacks). The used primitives, however, are instantiated in a manner that is resistant against multi-target attacks and therefore require the adversary to actually break the second preimage of the used functions. □

4.3 Attacking the Tree Construction

The previous proofs are not exclusive to the linear construction and also apply to the tree construction as the main introduction in the latter approach is the use of a tree to aggregate string values. We now review the security of the added data structure.

Theorem 4. *Given a merkle root, a string, and a proof of inclusion $(\mathsf{M}, s^{(i)}, \pi^{(i)}_{\mathsf{M}})$ for a specific level $i \in \{1, \ldots, t\}$, the adversary has negligible probability of producing a proof forgery $\pi'^{(i)}_{\mathsf{M}}$ for the tree construction, if the hash function used for Merkle Tree generation is from a second preimage resistant hash function family.*

Proof: (Sketch.) In this setting, $\mathcal{A}$ attempts to prove inclusion of a value that is not in the original Merkle tree data structure generated by the signer. Since the algorithm $\mathsf{MT.Verify}(\mathsf{M}, s^{(i)}, \pi^{(i)}_{\mathsf{M}}) = 0$ if a proof $\pi^{(i)}_{\mathsf{M}}$ fails to verify, and the value M is fixed upon the generation of the key material, $\mathcal{A}$ must produce a value $s'^{(i)} \neq s^{(i)}$ and use the algorithm $\mathsf{MT.Proof}(\mathsf{M}, s^{(i)})$ to generate a colliding $\pi^{(i)}_{\mathsf{M}}$. Alternatively, $\mathcal{A}$ may attempt to generate a different tree with a different set of values that results in a tree with the same Merkle root. Both settings are

equivalent to finding a different second preimage, such that the malicious values result in a collision with the initial signer generated values. □

5 Formal Methods Analysis

In this section, we define the Verifpal [19,17] model used to analyze Wotswana along with the description of some of the technical challenges inherent to the model development process. The purpose of this section is to confirm the early analytical proofs of the construction and verify whether or not both approaches for security analyses provide the same results.

5.1 Verifpal and Modeling Challenges

Verifpal is a software that allows for the verification of the security of cryptographic protocols and is particularly oriented towards real-world practitioners attempting to integrate formal verification into their line of work. Moreover, this tool supports advanced security properties such as forward secrecy or key compromise impersonation. We note that Verifpal has been used to verify security properties of widely deployed tools, such as Signal [18] and TLS 1.3 [4].

Challenges to Modelling Wotswana in Verifpal. Symbolic model protocol verifiers, typically face a problem when analysing complex protocols: the space of the user states and different combinations of variables the verifier must assess, quickly becomes too large for the verifier to terminate in a reasonable time. Verifpal attempts to optimize for this challenge by separating the analysis into a number of stages in which it gradually allows the increasing modification of states. The different variable combinations quickly becoming too large is a challenge we faced while a ladder, where each level contains a WOTS+ keypair, and the tool constantly issued memory fault errors when starting to perform the hash ladder iterations for the key generation processed. These memory errors resulted in the stopping of the verification process in a faulty manner. Additionally, we highlight the lack of existence of the *XOR* logical function in the verification tool, which leads to initial design attempts with a slightly changed variant of the chaining function used in WOTS+.

Verifpal Model of Wotswana. To avoid the memory fault issues derived from iterating different attack scenarios involving a high number of hash function calls, we model a simpler Lamport signature scheme instead of WOTS+. For simplicity of the model and readability of the code, we simulate a setting with only two ladder levels, each containing a hash-based key pair. This code, containing the Verifpal model, is open-sourced and published in [25].

Participants. We model two participants: a signer and a verifier. The signer generates an initial Lamport keypair $(\mathsf{sk}, \mathsf{pk})$ to sign a single bit $b \in \{0, 1\}$. Upon generating this first hash-based key pair, the signer compresses the first public key value using the tweakable hash function and then uses a hash-based key derivation function (HKDF) to generate the second Lamport keypair. This HKDF receives the first Lamport public key, now compressed into a single value,

as key material to be expanded and outputs a second Lamport keypair. This second public key is also compressed using a tweakable hash function. For consistency with the protocol specification from this work, we compress the Lamport public keys using a tweakable hash function. We note, however, that this step is purely for readability as the main purpose of using tweakable hash functions in Wotswana is to mitigate multi-target attacks, which are out of scope of the results produced by the tool.

Attacker model and message flow. We assume the Dolev-Yao model [13] where the adversary is in charge of delivering the messages. Therefore, all the transmitted messages go through the adversary first or are in fact delivered by $\mathcal{A}$. In our model, the signer starts by sending to the verifier the following values: a signature on a 1 bit from one level lower than the top one, and a public key for verification. We note that all the public key values transmitted between both parties are authenticated using the guarded constant feature from Verifpal, which allows the model to ensure that the public key used to verify the signature is authenticated. Therefore, man-in-the-middle or impersonation attacks are out of the scope of the analysis. We consider this approach as the scheme is designed to be used in a blockchain setting where the public keys are openly available on the distributed ledger.

We assume that after signing a message, and submitting the signature to the network, the signer exposes the next public key on the ladder to inform the network of what the next verification key is. This exposure is achieved using the *leaks* command present in the Verifpal tool, which fully exposes a variable to the adversary. The goal of this step is to simulate the adversary $\mathcal{A}$ from the security game described in the formal security proof, where $\mathcal{A}$ cannot invert a hash function, yet is capable of looking at messages sent to the network before anyone else.

5.2 Modelling Results

After performing these steps, we run four confidentiality queries and request the tool to perform an analysis on whether or not the adversary can break the confidentiality of the two individual Lamport secret keys $(\mathsf{sk}_0, \mathsf{sk}_1)$ for each of the ladder levels. The tool output a positive result for two of these confidentiality queries. Therefore, the adversary can fully obtain the secret key values for the top Lamport key pair and produce forgeries on additional messages. The lower two secret values, however, remain confidential. Therefore, the adversary $\mathcal{A}$ is not expected to be able to produce a forgery with the necessary key, which matches our results obtained in the security proof.

In summary, Verifpal returned that there is no breach in the confidentiality of any of the secret key variables that must remain private for the construction to achieve its security goals, thus outputting a formal positive result about our design. We note that this formal methods analysis does not find attacks involving structural weaknesses of the used cryptographic primitives. For example, the use of a hash function with a small security parameter is not in the scope of the

attack model of the tool. Therefore, implementations of this construction must take into account and appropriate choice of security parameters.

6 Use Case: Encrypted Group Chat

We now expose an alternative use-case for this work, namely an approach where secure messaging apps can use our construction to achieve constant-sized messaging in an encrypted group chat setting, while preserving fundamental security properties for secure messaging (e.g., deniability). We start by exposing the main existing approaches along with its associated communication complexity. Finally, we showcase a Wotswana -based encrypted group chat.

Trivial Client fan-out. One scheme is for Bob to encrypt the message with every participant's key. In a group chat with 30 other participants, Bob sends the message 30 times, encrypting each message for the intended reader. An advantage of client-side fan-out is that it reuses the same protocol used for two-person conversations. This approach, however, quickly becomes prohibitive if the group is big or the network bandwidth is small.

Improved Client fan-out. Alternatively, Bob can encrypt a message for a global group chat shared key and attach an authentication tag (i.e., MAC) for each of the group participants. In a group chat with 30 other participants, Bob sends one message and 30 tags for the intended readers. We call this scheme "improved client fan-out" as it improves on the communication complexity of the trivial fan-out approach. An advantage of this approach is the bandwidth savings as this only results in the linearly increasing of the number of sent authentication tags and constant-sized number of messages, which in this case is only one.

Signed Server fan-out. A final possible approach is for Bob to encrypt a single message for a global group chat shared key along with a digital signature. In a group chat with 30 other participants, Bob sends a single message along with one digital signature for the server, which then fans out the same message for the total set of 30 participants. This scheme can be called "signed server fan-out". This approach can even feature an optimization where the server does not fan out and instead simply relays the message to a message fetching service. Later in time, clients in this group chat can contact this fetching service and obtain the corresponding message(s) associated with this group chat.

FFS-based Encrypted Group Chat. We now discuss the approach that relies on forward-forgeable signatures [22] to achieve linear complexity and preserve deniability. Bob encrypts a single message for a global group chat shared key and attaches a single forward-forgeable digital signature. In a group chat with 30 other participants, Bob sends a single message along with one FFS for the server, which then fans out the same message for the total set of 30 participants. This scheme can be called "FFS-based group chat" and features a potentially optimal communication complexity as the sender simply sends a single message, regardless of the total number of participants in the group, which results in bandwidth savings and, unlike the previous approach, preserves the deniability property

as for each newly signed message or a specific time window, Bob removes the non-repudiation property of the previously sent message. Therefore, Bob is able to deny sending specific group messages. Table 7 illustrates a communication complexity comparison of the different approaches.

Group Chat Approach	**Communication Complexity**
Trivial Client fan-out	$\mathcal{O}((N-1)\cdot(\lvert m\rvert+\lvert t\rvert)$
Improved Client fan-out	$\mathcal{O}(\lvert m\rvert+(N-1)\cdot\lvert t\rvert)$
Signed Server fan-out	$\mathcal{O}(\lvert m\rvert+\lvert\sigma\rvert)$
Wotswana-based [This Work]	$\mathcal{O}(\lvert\mathbf{m}\rvert+\lvert\sigma'\rvert)$

Table 7: Communication complexity comparison for the different approaches. Where $|\sigma|$, $|t|$ and $|m|$ are the sizes of signatures, tags and message, and N is the number of group participants.

7 Conclusion

The recently introduced $\mathcal{S}_{\text{leeve}}$ primitive adds an extra layer of security for cryptocurrency wallets. It is specifically designed to provide means for the users to assure the ownership of the cryptographic keys in the event of a massive leak. A wallet with the $\mathcal{S}_{\text{leeve}}$ design provides the user with a *back up key* which can be used to generate a single proof of ownership; a clear limitation of the original design.

This work extends the security guarantees of $\mathcal{S}_{\text{leeve}}$ by introducing a new design named Wotswana, and its main feature is the capability of issuing multiples proofs of ownership. This novel capability naturally extends the original security definition for $\mathcal{S}_{\text{leeve}}$. Furthermore, we propose two constructions for Wotswana and in both cases the back up keys provided by the Sleeve design are kept two types of data structures: (1) a linear and (2) a binary tree.

Finally, we prove the security of both constructions given an extended security notion adapted from the single proof of ownership, *i.e.* multiple proofs of ownership. Moreover we analyse the security of our constructions based on formal methods, *i.e.* Verifpal. We introduced practical use cases for our design and initiated the process of contacting development groups to analyze the possibility of integrating this construction into some of their services. We hope this design helps the community, and raises awareness about the importance of preparing for the eventual integration of quantum secure solutions in commercial applications.

References

1. Diego F. Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, and Yuval Yarom. Ladderleak: Breaking ecdsa with less than one bit of nonce leakage. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 225–242, New York, NY, USA, 2020. Association for Computing Machinery.

2. Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 913–930, Toronto, ON, Canada, October 15–19, 2018. ACM Press.
3. Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2129–2146. ACM Press, November 11–15, 2019.
4. K. Bhargavan, B. Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the tls 1.3 standard candidate. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 483–502, 2017.
5. Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss - a practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, pages 117–129, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
6. David Chaum, Mario Larangeira, and Mario Yaksetig. Tweakable sleeve: A novel sleeve construction based on tweakable hash functions. *The 3rd International Conference on Mathematical Research for Blockchain Economy (MARBLE)*, 2022.
7. David Chaum, Mario Larangeira, Mario Yaksetig, and William Carter. Wots+ up my sleeve! a hidden secure fallback for cryptocurrency wallets. In *International Conference on Applied Cryptography and Network Security*, pages 195–219. Springer, 2021.
8. Lily Chen. Recommendation for key derivation using pseudorandom functions-revision 1. *NIST special publication*, 2021. Accessed: 2022-02-20.
9. Nicolas T. Courtois, Pinar Emirdag, and Filippo Valsorda. Private key recovery combination attacks: On extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor RNG events. Cryptology ePrint Archive, Report 2014/848, 2014. `http://eprint.iacr.org/2014/848`.
10. Erik Dahmen, Katsuyuki Okeya, Tsuyoshi Takagi, and Camille Vuillaume. Digital signatures out of second-preimage resistant hash functions. In Johannes Buchmann and Jintai Ding, editors, *Post-quantum cryptography, second international workshop, PQCRYPTO 2008*, pages 109–123, Cincinnati, Ohio, United States, October 17–19 2008. Springer, Heidelberg, Germany.
11. Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 651–668. ACM Press, November 11–15, 2019.
12. Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.
13. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
14. Chun-I Fan, Yi-Fan Tseng, Hui-Po Su, Ruei-Hau Hsu, and Hiroaki Kikuchi. Secure hierarchical bitcoin wallet scheme against privilege escalation attacks. *International Journal of Information Security*, pages 1–11, 2019.

15. Andreas Hülsing. W-OTS+ - shorter signatures for hash-based signature schemes. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *AFRICACRYPT 13: 6th International Conference on Cryptology in Africa*, volume 7918 of *Lecture Notes in Computer Science*, pages 173–188, Cairo, Egypt, June 22–24, 2013. Springer, Heidelberg, Germany.
16. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.
17. Nadim Kobeissi. Verifpal: Cryptographic Protocol Analysis for Students and Engineers. `https://verifpal.com`, 2021. [Online; accessed 05-March-2022].
18. Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 435–450, 2017.
19. Nadim Kobeissi, Georgio Nicolas, and Mukesh Tiwari. Verifpal: Cryptographic protocol analysis for the real world. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, CCSW'20, page 159, New York, NY, USA, 2020. Association for Computing Machinery.
20. Hugo Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 631–648, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
21. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.
22. Michael A. Specter, Sunoo Park, and Matthew Green. Keyforge: Non-attributable email from forward-forgeable signatures. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1755–1773. USENIX Association, 2021.
23. Trinity attack incident part 1: Summary and next steps. `https://blog.iota.org/trinity-attack-incident-part-1-summary-and-next-steps-8c7ccc4d81e8`. Accessed: 2020-09-22.
24. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.
25. xx network. Wotswana verifpal model. `https://github.com/xx-labs/wotswana`.