White-Box Implementations for Hash-Based Signatures and One-Time Passwords

Kemal Bicakci[1,3], Kemal Ulker[2,3], Yusuf Uzunay[3], Halis Taha Şahin[1,4], and Muhammed Said Gündoğan[4]

[1] Informatics Institute, Istanbul Technical University, Istanbul, Türkiye, kemalbicakci@itu.edu.tr
[2] Department of Computer Engineering, TOBB University of Economics and Technology, Ankara, Türkiye, kemal.lkr@gmail.com
[3] Securify Information Tech. and Security Training Consulting Ltd., Ankara, Türkiye, yusuf.uzunay@securify.com.tr
[4] Informatics and Information Security Research Center (BİLGEM), TÜBİTAK, Kocaeli, Türkiye
{halis.sahin,said.gundogan}@tubitak.gov.tr

Abstract. White-box cryptography challenges the assumption that the end points are trusted and aims at providing protection against an adversary more powerful than the one in the traditional black-box cryptographic model. Most existing white-box implementations focus on symmetric encryption. In particular, we are not aware of any previous work on general-purpose quantum-safe digital signature schemes also secure against white-box attackers. We present white-box implementations for hash-based signatures so that the security against white-box attackers depends on the availability of a white-box secure pseudorandom function (in addition to a general one-way function). We also present a hash tree-based solution for one-time passwords secure in a white-box attacker context. We implement the proposed solutions and share our performance results.

Keywords: white-box cryptography · digital signature · white-box signature · quantum-safe signature · hash chain · one-time password · hash tree.

1 Introduction

The standard cryptographic model (black-box model) assumes the end points are trusted hence the secret keys in cryptographic implementations cannot be observed while they are in use. The first work that challenged this assumption is by Chow et al. [1] in 2002. The authors proposed an implementation of AES algorithm to prevent secret key extraction even when an attacker has a full access to the execution environment. Although their specific implementation was later broken, their core idea of building up a key-dependent lookup table(s) from which the encryption (and decryption) could be performed without a need for the cryptographic key remains highly relevant. Later, some dedicated white-box ciphers have been designed with the same philosophy, which are not broken till now e.g., SPACE [2] and SPNbox [13].

White-box implementations have the objective of preventing extraction of cryptographic keys useful on a different platform. Instead of using keys directly, an attacker may attempt to isolate the complete implementation code from the environment, carry it to his own device and directly use it like a larger key. These so-called code lifting attacks are assumed to be mitigated by the help of the notion so called *space-hardness* (security against code lifting is quantified by the amount of code that needs to be extracted from the implementation). We also note that with additional software protection techniques such as device binding and code obfuscation, the use of software may not be possible in other hardware devices.

Up to now, white-box cryptography is mostly studied in symmetric encryption context. As detailed in Related Work section, proposals for white-box implementation of digital signature algorithms are rare and not sufficiently analyzed from security point of view. In our work, we present simple and elegant designs for white-box implementation of hash-based signatures and cryptographic primitives desirable in authentication protocols. Although known for a long time, hash-based signatures have received a new surge of interest due to their ability to remain post-quantum safe [3]. We contribute to the literature by presenting implementations for hash-based digital signatures where the security against white-box attacker depends not more than the availability of a white-box secure pseudo-random function (in addition to a general one-way function). We also show a hash tree based alternative to the hash chain primitive (useful for entity authentication) to remain secure against white-box attackers on an untrusted environment.

2 Lamport's Signature Scheme

Lamport's construction of one-time signature (OTS) is the first scheme which relies solely on one-way (hash) functions for its security [4]. Although the efficiency of this scheme has been improved in subsequent studies, for pedagogical reasons, we prefer to use it to explain our core idea for making hash-based signatures strong against white-box attackers.

As always, there are three algorithms defining Lamport's one-time signature scheme:

Let f be a one-way hash function with an output length of N.

Key Generation:
Input: *Parameters* L, N
L: *the length of random numbers*
$2N$: *total number of random numbers*
Output:

For one-time private key, generate:
$2N$ *L-bit random numbers* $r_1, r_2, \ldots, r_{2N}$
As one-time public key, compute:
$p_k = f(r_k)$ *for* $1 \leq k \leq 2N$ *(Distribute the public key securely as usual).*
As another more useful notation, random numbers (pre-images) and hash values (hash-images) could be indexed as follows, respectively:
$r_{i,j}(1 \leq i \leq N$ *and* $1 \leq j \leq 2)$ *and* $p_{i,j}(1 \leq i \leq N$ *and* $1 \leq j \leq 2)$

Signing:
Input: M
M: *message to be signed*
$h = f(M)$ *(h has a length of N)*
Output:
for $1 \leq s \leq N$ */* index value for bits of h */*
if $h_s = 0$ *reveal* $r_{s,1}$
else reveal $r_{s,2}$
as part of the signature

Verifying:
Input: *Parameters* $M', r'_{s,j}, p_{i,j}, h'$
M': *message received*
$r'_{s,j}$: *signature received* $(1 \leq s \leq N)$
$p_{i,j}$: *public key* $(1 \leq i \leq N$ *and* $1 \leq j \leq 2)$
$h' = f(M')$
Output:
"Accept" if for each $1 \leq s \leq N$
if $h'_s = 0$ $\quad h(r'_{s,1}) = p_{s,1}$
else $\quad h(r'_{s,2}) = p_{s,2}$
"Reject" otherwise

3 White-Box Implementation of Lamport's Scheme

Instead of storing all the random numbers constituting the one-time private key, one can use a cryptographically secure pseudo random function (PRF) to generate all the random numbers using a single secret (private) key. Instead of using a general-purpose PRF (practically implemented using standard block ciphers such as AES), we now introduce an implementation of Lamport's scheme secure in a (weak) white-box model [2] assuming that there is a white-box attack-resistant (secure) block cipher which also behaves as a PRF.

Let f *be a one-way hash function with an output length of* N.

Let E_K *be a white-box secure block cipher (e.g., SPNbox [13]).* E_K *is represented as one big key-dependent lookup table denoted as* $WBT\text{-}E_K$. *We assume key* K *is securely erased after* $WBT\text{-}E_K$ *is ready. For simplicity, we assume the block length of* E_K *is also* L.

Key Generation:
Input: *Parameters* L, N, IP
L: *the length of random numbers (as well as block length of* E_K*)*
$2N$: *total number of random numbers*
IP: *(randomly generated and stored) initial plaintext for* $WBT\text{-}E_K$
Output:
For one-time private key, generate $2N$ L*-bit pseudo-random numbers:*
for $1 \leq k \leq 2N \quad r_k = WBT\text{-}E_K\ (IP + k)$
As the one-time public key, compute:
$p_k = f(r_k)$ *for* $1 \leq k \leq 2N$ *(distribute the public key securely as usual).*
For a more useful notation, random numbers (pre-images)
and hash values (hash-images) could be indexed as follows, respectively:
$r_{i,j}(1 \leq i \leq N$ *and* $1 \leq j \leq 2)$ *and* $p_{i,j}(1 \leq i \leq N$ *and* $1 \leq j \leq 2)$
After the public key is generated, r_k *values are securely erased.*

Signing:
Input: M
M: *message to be signed*
$h = f(M)$ *(*h *has a length of* N*)*
Output:
for $1 \leq s \leq N$ */* index value for bits of* h **/*
if $h_s = 0$ *compute and reveal* $r_{s,1} = WBT\text{-}E_k(IP + 2s - 1)$
else compute and reveal $r_{s,2} = WBT\text{-}E_k(IP + 2s)$
as part of the signature.

Verifying:
Same as the original case (nothing changed).

4 Signing Multiple Messages with a Single Public Key

Lamport's OTS scheme is only useful for signing a single message per single public key hence its utility is quite limited. The problem of extending Lamport's OTS for multiple messages has already been extensively studied in the literature. Most of the proposed schemes are variations of the early work by Merkle [5].

In Merkle's original scheme, the tree is built for certification of additional OTS public keys i.e., every node has three public keys, one for the message itself, one for the left child node, and one for right child node. With this scheme, an infinite number of messages could be signed using a single root one-time public key. Another, more popular implementation of Merkle's scheme is adopting a bottom-up approach rather than top-to-bottom one to sign multiple but finite pre-determined number of messages. Here, first, the leaf N nodes (one-time private keys and corresponding public keys) are prepared. Then, using hash values of public keys, a binary tree is built. The final public key is the root of the node. See Fig. 1 for an example of Merkle three with 8 leaf nodes.

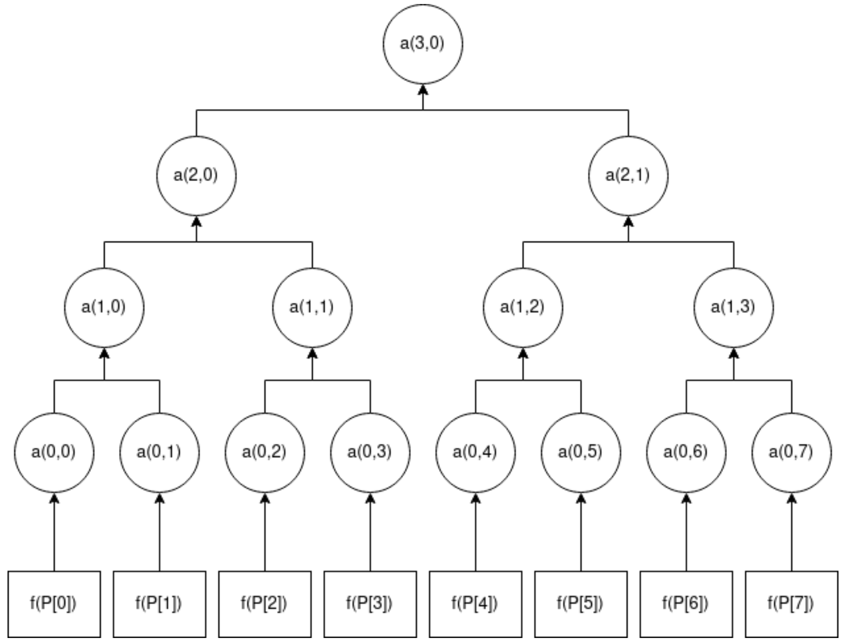

Fig. 1. Merkle tree to sign 8 messages.

Below, we first describe an insecure implementation of Merkle's scheme and then show how to make it secure in a white-box model.

Let f be a one-way hash function with an output length of N. Let E_K be a white-box secure block cipher. E_K is represented as one big key-dependent lookup table denoted as WBT-E_K. We assume key K is securely erased after WBT-E_K is ready. We assume the block length of E_K is also L.

Key Generation:
Input: *Parameters* L, N, IP, T
L: *the length of random numbers (as well as block length of* E_K*)*
$2N$: *total number of random numbers*
IP: *(randomly generated and stored) initial plaintext for* WBT-E_K
T: *number of messages to be signed* ($T = 2^n$) *(n is the height of the tree)*
Output:
a. *for* $0 \leq t \leq T - 1$
for $1 \leq k \leq 2N$
generate $2N$ L*-bit pseudo-random numbers:*
$r_{k,t} = WBT\text{-}E_K(IP + t * 2N + k)$
compute $p_{k,t} = f(r_{k,t})$
For a more useful notation, random numbers (pre-images)

and hash values (hash-images) could be indexed as follows, respectively:
$r_{i,j,t}(1 \leq i \leq N$ *and* $1 \leq j \leq 2$ *and* $0 \leq t \leq T-1)$
$p_{i,j,t}(1 \leq i \leq N$ *and* $1 \leq j \leq 2$ *and* $0 \leq t \leq T-1)$

b. *Generate hashes of public keys as follows:*
for $0 \leq t \leq T-1$

$P[t] = p_{1,1,t}||p_{1,2,t}|| \ldots ||p_{N,1,t}||p_{N,2,t}$
$a(0,t) = f(P[t])$

c. *Generate the root public key and distribute it securely (note that only a single hash value constitutes the public key here):*
As an example, consider the case given in Figure 1:

$$a(3,0) = f(f(f(a(0,0)||a(0,1))||f(a(0,2)||a(0,3)))||f(f(a(0,4)||a(0,5))||f(a(0,6)||a(0,7))))$$

We note that a naive implementation either requires the random numbers to be stored for later use or erase all data (except IP*) for later generation once needed (soon, we will show why both of these are insecure options).*

Signing:
Input: *Parameters* M, t

M: *message to be signed*
$t =$ *index of the leaf node for signing* $(0 \leq t \leq T-1)$

Output:

for $1 \leq s \leq N$ /* *index value for bits of* h */

compute (if not already stored):
$r_{s,1,t} = WBT\text{-}E_K(IP + t * 2N + 2s - 1)$
$r_{s,2,t} = WBT\text{-}E_K(IP + t * 2N + 2s)$

if $h_s = 0$
reveal $r_{s,1,t}$ *and compute and reveal* $f(r_{s,2,t})$
else
reveal $f(r_{s,1,t})$ *and compute and reveal* $r_{s,2,t}$

as part of the signature.

Also additional nodes (hash values) up to the root node should be sent as auxiliary information to make it possible to compute and verify the root public key) (for instance $a(0,1), a(1,1)$ *and* $a(2,1)$ *should be sent for* $t = 0$ *in the example shown in Figure 1).*

Verifying:
Input:

M': *message received*
Signature received: $r'_{s,j,t}$ *or* $f'(r_{s,j,t})(1 \leq s \leq N)(0 \leq t \leq T-1)$ *and auxiliary information received e.g.,* $a'(0,1)$, $a'(1,1)$, $a'(2,1)$
Public key: $a(3,0)$
$h' = f(M')$

Output:

for each $1 \le s \le N$
 if $h'_s = 0$
  compute $f'(r'_{s,1,t}) = p_{s,1,t}$
 else
  compute $f'(r'_{s,2,t}) = p_{s,2,t}$

compute $a'(0,t) = f(p_{1,1,t}||p_{1,2,t}||\dots||p_{N,1,t}||p_{N,2,t})$
Accept if $a(3,0) = f(f(f(a'(0,t)||a'(0,1))||a'(1,1))||a'(2,1))$ *(for t=0)*
"Reject" otherwise

Now, we show that the above scheme is not secure in a white-box model. The underlying reason is that all the random numbers are computed (or already stored) during signature generation although not revealed as part of the signature. This is required in order to compute the hash values for the random numbers not revealed. Hash values are sent as part of the signature to let the verifier to compute the value of $a'(0,t)$ and verify the signature. However, a white-box attacker having the ability to observe the internal state information could identify and extract all the random numbers and use them to forge a signature for any message he wants. Below, we show a slight change in the implementation to make it secure against white-box attacker.

The change we require is to prepare all the hash values required to build the signature once the message is ready, without a need to generate the random numbers not required as part of the signature itself [5]. For this purpose, after all the component of one-time public key are computed by $p_{k,t} = f(r_{k,t})$ for $1 \le k \le 2N$ and $0 \le t \le T-1$, the values of $p_{k,t}$ are stored on the signer side. Once a signature is required, signing algorithm is changed slightly as follows:

for $1 \le s \le N$ */* index value for bits of h */*
 if $h'_s = 0$
  compute $r_{s,1,t} = WBT\text{-}E_K(IP + t * 2N + 2s - 1)$
  reveal $r_{s,1,t}$ *and* $f(r_{s,2,t})$ */** $f(r_{s,2,t})$ *has been previously stored */*
 else
  compute $r_{s,2,t} = WBT\text{-}E_K(IP + t * 2N + 2s)$
  reveal $f(r_{s,1,t})$ *and* $r_{s,2,t}$ */** $f(r_{s,1,t})$ *has been previously stored */*

With this change, a white-box attacker could not observe a random number required to forge a signature for any message different than the message the user has already signed.

To summarize, an implementation of hash-based signatures is not secure in a white-box model if any pre-image(s) not used in the signature itself is generated during signature generation or it is already stored after key generation is completed. While this might be just an implementation choice in some of the

[5] We also note that to improve computational efficiency of signature generation auxiliary information could also be pre-computed.

schemes (e.g., the above Merkle's scheme) (without any white-box security concern, one might still prefer different variations of the basic scheme for leveraging different storage-computation tradeoffs), in some others secure implementations are not possible at all (e.g., Winternitz scheme and other hash-chain based approaches [10]). Leaving the analysis of every scheme in the rich literature of hash-based signatures in this context as a future work, we provide a white-box alternative to a generic crypto primitive (i.e., Lamport's hash chain) in the next section.

5 A White-Box Alternative to Hash Chains and T/Key

A hash chain (proposed also by Lamport [6]) is a useful cryptographic primitive where a single shared (public) value is sufficient to verify securely the authenticity of a finite (but potentially large) number of different values. Besides a number of other applications, elements of hash chains could simply be used as one-time passwords (OTPs) for user authentication.

For a better grasp of the advantage of using a hash chain, let us first consider the case where a number of independent one-time passwords are generated on the user (client) side and the hash value of each is sent to the server as part of the initialization. Each OTP is sent one by one during normal operation to be verified by the server using hash values. The disadvantage here ıs that the storage requirement both on the server and client side increases linearly with the number of OTPs. (Besides, the risk is evident on the client side, any (white-box) attacker having access to the untrusted client machine could easily intercept all the OTPs to be used for later impersonation.)

Could we eliminate some of these problems with hash chains? A hash chain of length m is simply obtained by iteratively applying a one-way (hash) function to a randomly generated seed value for m times:

$$f^m(s) = \underbrace{(f \circ f \circ \cdots \circ f)}_{m\ times}(s)$$

The final value $f^m(s)$ is sent to the server for initialization(registration). The first OTP used for authentication is the element just before the final value: $f^{m-1}(s)$. In this reverse order, in total $m-1$ OTPs could be generated and used while requiring only a single value on the server side for verification. On the client side, there are two options for the storage:

- The client could choose to generate and store all the elements in the hash chain. Later, once one of them is to be used, there will not be any need to do computation.
- The client chooses to store only the seed value and generates the required OTP by iteratively doing the required number of hash computations [6].

[6] An amortization technique could also be used to reduce memory-times-computation complexity [11].

It is evident that the first option is not secure against a white-box attacker. It is also easy to see that the second option is similarly insecure simply because on the untrusted machine either the seed value itself (while one of the elements of the hash chain is generated) or other elements prior to a particular element could be intercepted by the white-box attacker for later use. We require a solution where OTPs could be generated independently so that white-box attacker could not gain any advantage even when he could fully observe the internal state of the client-side software. Below, we illustrate a solution for achieving this. In fact, the white-box implementation of Merkle's tree discussed in the previous section could be tailored to serve for our purposes i.e., each leaf node corresponds to the hash of a single random number rather than $2N$ random numbers.

Below, we only show the initialization phase (generation and verification of OTPs are skipped for the sake of brevity).

Initialization Phase:

Input: *Parameters* L, IP, T

L: *the length of random numbers (as well as block length of* E_K*)*

IP: *(randomly generated and stored) initial plaintext for* $WBT\text{-}E_K$

T: *number of OTPs* ($T = 2^n$) *(n is the height of the tree)*

Output:

a. *for* $0 \leq t \leq T - 1$

$r_t = WBT\text{-}E_K(IP + t)$ /*generate L-bit pseudo-random numbers*/

$p_t = f(r_t)$ /* compute hash of the random numbers */

The values of p_t *also correspond to the leaf nodes of the tree (no need to compute the hash of* p_t *values) i.e.,* $p_t = a(0, t)$

b. *Generate the root node and distribute it securely.*

To summarize, a Merkle's tree in which the leaf nodes are the hash values of a single random number could be a viable alternative to hash chains if white-box attackers are also of concern. Table 1 compares our proposed scheme with hash chain based OTPs and independent OTPs.

6 White-Box Resistant Time-based OTPs

The idea of using a hash chain for one-time passwords was developed and implemented under the name of S/KEY [7]. As noted in [8], S/KEY has a number of undesirable properties. In particular, the scheme is vulnerable to an attack where the client reveals OTP(s) to attackers for future abuses by various means such as social engineering or by impersonating the server. The need and difficulty for synchronization of the chain between server and client is another concern (which element is the next one?).

A widely used solution for OTPs is implemented in the TOTP standard [9]. Here, the server and the client shares a secret key. Using the current timestamp (usually in a granularity of 30 seconds) as an implicit challenge of the server, this

Table 1. Comparison of schemes (for use of T OTPs).

	Proposed Scheme	w/ Hash Chains	Independent OTPs
White-box resistant	√	No	No
Client-side computation per OTP	$O(1)$ $WBT\text{-}E_K$ $+O(log\,T)$ hash	None (if elements are stored)	None
Storage on client	$O(T)$	$O(T)$ (if elements are stored)	$O(T)$
Storage on server	$O(1)$	$O(1)$	$O(T)$
Communication cost per authentication	$O(log\,T)$	$O(1)$	$O(1)$
Initialization cost	$O(T)$ $WBT\text{-}E_K$ $+O(T)$ hash	$O(T)$ hash	$O(T)$ hash

standard actually implements a simple challenge-response protocol. The client computes the MAC of the challenge and transmits the output (actually part of it) as the OTP response. The same computation could be done on the server side if a loose time synchronization is present. On the down side, TOTP depends on a secret stored both on the client and server, hence it is open to attacks on both sides.

To solve this problem for the server side, Kogan et al. proposed T/Key, a time-based OTP scheme [8]. The key idea in T/Key is to map each element of a hash chain to a specific time period so that OTPs are now time dependent [7]. However although no secret is stored on the server side, T/Key is vulnerable to a white-box attacker having access to the client side implementation. Below, we will show that the scheme we proposed in previous section could easily be made time-dependent just like TOTP and T/Key.

In fact, making our proposed scheme time-dependent requires no more than a mapping of each OTP (leaf node of the tree) to a pre-determined specific time period. For instance we could prefer a mapping from left to right. Suppose we choose such a mapping in the example of Merkle tree with 8 leaf nodes shown in Fig. 1. Suppose also the tree is already built and the root node is shared with the server. Then, the time-dependent OTP is generated as follows:

Time-dependent OTP Generation:

Input: *Parameters* I, t_{init}, t

I: *time slot length*

[7] Additionally, in order to make the chain birthday-attack resistant, to generate each of its element, independent hash functions from a single hash function is obtained using the idea of domain separation.

t_{init}: setup time t (measured in slots of length I) (it has to be shared with the server side during registration)

t: current time (measured in slots of length I)

Output:

$r_t = WBT\text{-}E_K(IP + t - t_{init})$ /* generate time-based OTP */

(In addition, nodes (hash values) up to the root node should be computed and sent as auxiliary information to make it possible to compute and verify the root public key) (for instance $a(0,1), a(1,1)$ and $a(2,1)$ should be sent for $t = t_{init}$ in the example shown in Figure 1).

On the down side, as compared to T/Key, one major efficiency drawback of the proposed scheme is that for a binary tree with 1×10^6 leaf nodes (valid approximately for one year), initialization (set up) requires 1×10^6 white-box encryption operations besides the hash operations. This drawback might be addressed by using a lightweight white-box encryption primitive (as further discussed in the Implementation section).

On the other hand, we argue that additional communication cost for OTP transmissions in our proposed scheme is less of a concern especially in a use case where OTPs are sent to the online server without manual entry (with only with a simple confirmation tap in a mobile authenticator application). Note that the use of a traditional digital signature scheme in this use case might not be preferable due to a white-box attacker threat.

If manual entry of OTPs is performed using QR-codes (the phone displays a QR code containing OTP and the user scans it using his laptop camera) as proposed by T/Key inventors [8], our proposed scheme still seems a viable approach. It requires an OTP length of less than 1 KB for 128-bit security and 32 years of authentication period, which does not exceed the maximum capacity of QR codes [12] (see Table 6).

7 Security Analysis and Space-Hardness of the Schemes

Specifically, a white-box attacker's goal against the implementation of Lamport's scheme described above is to obtain the private key (or part of it) to generate a signature for a message not intended to be signed by the legitimate user. This corresponds to any r_k values not revealed as part of the signature. The attacker has two options for achieving this:

- He could try to invert at least one of the hash images (the hash values in the public key).
- He could try to generate at least one of the unrevealed pseudo-random numbers using the stored IP value.

The first option is not possible due to the one-way property of the hash function used. Similarly, the second option is out of reach if a secure white-box block cipher is available which prevents to extract the key K from the lookup table. (the concern of code lifting will be addressed by the space-hardness analysis of the proposed schemes.)

What if the attacker accesses the implementation environment before or while the key-dependent look-up table is built? Since the encryption key K is available in memory at that time, access to this key brings the ability to generate the whole one-time private key itself. We remind that this is also a legitimate concern for the symmetric encryption case but with a subtle difference. For encryption, the cryptographic key (therefore the key-dependent lookup table) is prepared to encrypt potentially infinite amount of plaintext messages. Hence the time window of vulnerability against a white-box attacker is short and acceptable (other precautions such as building the tables while the untrusted device is offline could be considered). On the other hand, if the lookup table is only used for signing a single message and building a second table is required thereafter, the risk against white-box attacker is significantly increased. In previous sections, we have already shown that a single look-up table could be used to sign multiple (potentially infinite) messages but there are some caveats that implementations should take into account.

For symmetric ciphers, a weak white-box model is defined in [2]:

Definition 1. *A cipher is said to be (M, Z)-space hard if it infeasible for an adversary to encrypt (decrypt) a randomly chosen plaintext with probability more that 2^{-Z} given code (table) size less than M.*

Similarly, we define the space-hardness for signatures as follows:

Definition 2. *A signature scheme is said to be (M, Z)-space hard if it infeasible for an adversary to forge a (signed) message with probability more that 2^{-Z} given code (table) size less than M.*

According to the White-Box Attack Context (WBAC) definition by Chow et al. [1], it is assumed that "internal details of cryptographic algorithms are both completely visible and alterable at will". This model is not applicable for the signature schemes as the attacker can change the message to be signed. That means each signature scheme is forgeable according to WBAC. As a result, we think Definition 2 makes more sense for signature schemes. Here, adversary has read-only access to software where the size of data accessed is bounded by M.

We also note that the signature schemes we proposed in our work are not stateless. Although the stateful hash based signatures can easily be broken when the attacker has ability to change the state, it is not possible in our security model as the adversary has read-only access.

Theorem 1. *Let S be the signing scheme described above with $T = 2^n$ number of messages to be signed, N is the message length and one-way hash function f with an output length of N. If the S uses (M, Z)-space hard cipher $WBT\text{-}E_K$, then the scheme (M, W)-space hard where*

$$W = Z - \log_2(N) - n - 1$$

provided that N is not too small.

In the signature scheme, to forge a message, an adversary needs to find at least one of the preimages $f(r_{s,i,t})$ where $1 \le s \le N$, $1 \le i \le 2$ and $0 \le t \le T-1$. So there is $2N \cdot 2^n$ preimages which are encrypted by $WBT\text{-}E_k$. By the assumption each value can be obtained by the adversary with probability less than 2^{-Z}. Then the probability that at least one of them can be obtained by the adversary is

$$2N \cdot 2^n \cdot 2^{-Z} = 2^{-W}.$$

Here, an adversary can also try to find the an inverse of the hash function f by directly taking random hashes or using some weakness in hash function. Because of this, we assume the scheme uses a secure hash function with enough output length. For example, if SHA256 is used, then the adversary needs to take about $\frac{2^{256}}{2N \cdot 2^n}$ to find at least one of preimages $f(r_{s,i,t})$. For the white-box ciphers such as SPACE or SPNbox, the space hardness level is generally taken as $Z = 64$ or $Z = 128$. So, the security bottleneck of this scheme is explosion of the white box cipher.

For our white-box resistant OTP scheme, a similar calculation can be done.

Theorem 2. *A white-box resistant OTP scheme using $T = 2^n$ number of passwords is $(M, Z-n)$-space hard if an (M, Z)-space hard cipher $WBT\text{-}E_k$ is used, provided that the scheme is using secure hash function with enough output length.*

We know that SPNbox is $(M, -10 \cdot t \cdot \log_2(M/I))$-space hard for any given M where I is the size of the input-output table of the inner cipher and $t = 16, 8, 5, 4$ for SPNbox-8, -16, -24, -32 respectively [13]. Using the theorems above we can calculate space-hardness of the signature and OTP schemes. We assume the signature scheme with $T = 2^{20}$ number of messages to be signed and OTP with $T = 2^{20}$ number of passwords. For $Z = 64$ and $Z = 128$, (M, Z)-space hardness of the schemes with SPNbox instantiations calculated in Table 2.

Table 2. (M, Z)-space hardness of the proposed schemes with SPNbox instantiations.

$WBT\text{-}E_K$	Table Size U	Signature Space-hardness		OTP Space-hardness	
		$Z = 64$	$Z = 128$	$Z = 64$	$Z = 128$
SPNbox-8	256 B	$U/2^{0.58}$	$U/2^{0.98}$	$U/2^{0.53}$	$U/2^{0.93}$
SPNbox-16	132 KB	$U/2^{1.16}$	$U/2^{1.96}$	$U/2^{1.05}$	$U/2^{1.85}$
SPNbox-24	50.3 MB	$U/2^{1.86}$	$U/2^{3.14}$	$U/2^{1.68}$	$U/2^{2.96}$
SPNbox-32	17.2 GB	$U/2^{2.33}$	$U/2^{3.93}$	$U/2^{2.10}$	$U/2^{3.70}$

8 Implementation

We implement our proposed scheme using the SPNbox algorithm [13]. SPNbox algorithm is considered secure when evaluated in terms of various adversary models such as cache timing, key extraction and space-hardness. We have implemented the SPNbox algorithm for different configurations, the results are shown in Table 3 and Table 4 below (The results of SPNbox-32 are omitted due to high memory requirements).

While implementing the SPNbox algorithm, we aimed to increase the performance by creating ready-made tables for matrix multiplications in the nonlinear and linear layers. We have 4,5,8 and 16 independent table lookups in SPNbox-32, 24, 16 and 8, respectively. As a result, we achieved 15% performance improvement compared to other available implementations [14] (parallel instruction sets were not used).

Table 3. Software performance of the SPNbox cipher family on the Intel platform in a white-box setting. Numbers are given in cycles per byte (cpb).

Algorithm	Rounds (outer)	Table Size	Performance [cpb]
SPNbox-24	10	50.3 MB	3323
SPNbox-16	10	132 KB	833
SPNbox-8	10	256 B	1030

Table 4. Software performance of the SPNbox cipher family in a black-box setting on Intel. Numbers are given in cycles per byte (cpb).

Algorithm	Rounds (outer)	Rounds (inner)	Performance [cpb]
SPNbox-24	10	20	6545
SPNbox-16	10	32	6069
SPNbox-8	10	64	9873

While implementing the signature scheme, instead of producing 2N preimages and hash-images for N-bit hash length, we preferred an optimized message mapping algorithm [15], thereby we achieved almost 50% percent of improvement in time and memory. Considering the time and memory usage performance, we decided to use the SPNbox-16 configuration in our signature and OTP implementations. Performance results of our signature and OTP implementations are given in Table 5 and Table 6, respectively.

As a future work, we intend to further improve our performance results using AVX (Intel) and NEON (ARMv8) instructions.

9 Related Work

Joye pointed out that one of the potential applications of white-box cryptography is to transform a MAC into a digital signature [16]. Here, “MAC verification”

Table 5. Performance results of the one-time signature scheme for different number of leaf nodes. Results obtained with the Apple M1(ARM) processor.(L-bit = 128, N-bit = 128, E_K = SPNbox-16)

Number of Leaf Node	Root Hash Generation	Signing	Verifying	Signature Size (B)
2^{10}	0.129 sec	191 μs	153 μs	2272
2^{15}	4.14 sec	258 μs	155 μs	2352
2^{20}	129.05 sec	262 μs	155 μs	2432

Table 6. Performance results of the one-time password scheme for different number of leaf nodes. Results obtained with the Apple M1(ARM) processor. (L-bit = 128, N-bit = 256, E_K = SPNbox-16)

Number of Leaf Node	Root Hash Generation	Generation	Verification	Size of OTP
2^{15}	0.041 sec	4 μs	6 μs	496 byte
2^{20}	1.308 sec	12 μs	9 μs	656 byte
2^{25}	44.11 sec	91 μs	11 μs	816 byte

algorithm is assumed to have a "certified" white-box implementation. Since the cryptographic key could not be extracted and cannot be used for generation of a MAC, the implementation is only useful for verification of (supposedly) a digital signature. However, this implementation choice is restricted in the sense that only those who have obtained a certified white-box implementation could perform the signature verification.

Zhang et. al. presented a white-box implementation of the identity-based signature scheme in the IEEE P1363 standard [17]. Feng et. al. proposed white-box implementation for the classical Shamir's identity based signature scheme [18]. In a recent work, Dottax et. al provided a deeper comprehension on the challenges of white-box ECDSA implementations [19]. Ma introduced a white-box Schnorr signature scheme [20] but provided only a limited security analysis.

Up to our best knowledge, our paper is the first study presenting a general-purpose quantum-safe digital signature algorithm in a white-box security model [8].

10 Concluding Remarks

To motivate our research, we consider a mobile authenticator application supporting multifactor user authentication. In this scenario, digital signatures and

[8] After the first version of our paper appeared in IACR ePrint in 2021, a white-box signature scheme based on multivariate polynomials was published [21]. We note that this new scheme requires 256 GB memory and 62 MB public keys for 80-bit security. We believe our scheme is more practical since it requires 16 B for public key and the memory requirement is only for a white-box implementation of a symmetric cipher.

hash chains are preferable constructions since no secret information is required to be stored on the verification (server) side. On the other hand, a typical mobile authenticator application is installed on an untrusted client device vulnerable to attacks and therefore should also be considered in a more sophisticated yet realistic threat model. To protect software implementations in such an environment, in this paper, we presented white-box resistant solutions for hash-based signatures and one-time passwords. We implemented these schemes and showed that these schemes are feasible in practice. The proposed simple and elegant schemes address critical challenges and provides an important step in white-box cryptography.

Acknowledgments

This research (patent pending) is funded by TUBITAK (The Scientific and Technological Research Council of Turkey) under the grant No: 3191520. We thank Assoc. Prof. Dr. Murat CENK for helpful comments and discussion.

References

1. Chow, S., Eisen, P., Johnson, H., & Van Oorschot, P. C. (2002, August). White-box cryptography and an AES implementation. In International Workshop on Selected Areas in Cryptography (pp. 250-270). Springer, Berlin, Heidelberg.
2. Bogdanov, A., & Isobe, T. (2015, October). White-box cryptography revisited: Space-hard ciphers. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (pp. 1058-1069).
3. Cooper, D. A., Apon, D. C., Dang, Q. H., Davidson, M. S., Dworkin, M. J., & Miller, C. A. (2020). Recommendation for stateful hash-based signature schemes. NIST Special Publication, 800, 208.
4. Lamport, L. (1979). Constructing digital signatures from a one way function.
5. Merkle, R. C. (1987, August). A digital signature based on a conventional encryption function. In Conference on the theory and application of cryptographic techniques (pp. 369-378). Springer, Berlin, Heidelberg.
6. Lamport, L. (1981). Password authentication with insecure communication. Communications of the ACM, 24(11), 770-772.
7. N. Haller. 1995. The S/KEY One-Time Password System. RFC 1760. Internet Engineering Task Force. 1–12 pages. https://doi.org/10.17487/RFC1760.
8. Kogan, D., Manohar, N., & Boneh, D. (2017, October). T/key: second-factor authentication from secure hash chains. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (pp. 983-999).
9. M'Raihi, D., Machani, S., Pei, M., Rydell, J. (2011). Totp: Time-based one-time password algorithm. Internet Request for Comments, 685E.
10. Merkle, R. C. (1989, August). A certified digital signature. In Conference on the Theory and Application of Cryptology (pp. 218-238). Springer, New York, NY.
11. Jakobsson, M. (2002, June). Fractal hash sequence representation and traversal. In Proceedings IEEE International Symposium on Information Theory, (p. 437). IEEE.

12. QR Codes. (2020). https://github.com/ricmoo/QRCode/blob/master/README.md, Last accessed on 2/28/2022.
13. Bogdanov, Andrey & Isobe, Takanori & Tischhauser, Elmar. (2016). Towards Practical Whitebox Cryptography: Optimizing Efficiency and Space Hardness. 126-158.10.1007/978-3-662-53887-6_5.
14. Liu, J., Rijmen, V., Hu, Y. et al. WARX: efficient white-box block cipher based on ARX primitives and random MDS matrix. Sci. China Inf. Sci. 65, 132302 (2022). https://doi.org/10.1007/s11432-020-3105-1
15. Kemal Bicakci, Gene Tsudik, and Brian Tung. 2003. How to construct optimal one-time signatures. Comput. Netw. 43, 3 (22 October 2003), 339–349. DOI:https://doi.org/10.1016/S1389-1286(03)00285-8
16. Joye, M. (2008). On white-box cryptography. Security of Information and Networks, 7-12.
17. Zhang, Y., He, D., Huang, X., Wang, D., Choo, K. K. R., & Wang, J. (2020). White-box implementation of the identity-based signature scheme in the IEEE P1363 standard for public key cryptography. IEICE TRANSACTIONS on Information and Systems, 103(2), 188-195.
18. Feng, Q., He, D., Wang, H., Kumar, N., & Choo, K. K. R. (2019). White-box implementation of Shamir's identity-based signature scheme. IEEE Systems Journal, 14(2), 1820-1829.
19. Dottax, E., Giraud, C., & Houzelot, A. (2021, October). White-Box ECDSA: Challenges and Existing Solutions. In International Workshop on Constructive Side-Channel Analysis and Secure Design (pp. 184-201). Springer, Cham.
20. Ma, Tianchen. "White-box Schnorr Signature for Internet of Things Security." 2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE). IEEE, 2020.
21. Galissant, P., Goubin, L. (2022). Resisting Key-Extraction and Code-Compression: a Secure Implementation of the HFE Signature Scheme in the White-Box Model. Cryptology ePrint Archive.