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

Kemal Bicakci[1,3], Kemal Ulker[2,3], and Yusuf Uzunay[3]

[1] Informatics Institute, Istanbul Technical University, Istanbul, Turkey, kemalbicakci@itu.edu.tr
[2] Department of Computer Engineering, TOBB University of Economics and Technology, Ankara, Turkey, kemal.lkr@gmail.com
[3] Securify Information Tech. and Security Training Consulting Ltd., Ankara, Turkey, yusuf.uzunay@securify.com.tr

Abstract. White-box cryptography aims at providing protection against a powerful adversary which is in complete control of the execution environment of the cryptographic operation. Most existing white-box implementations focus on symmetric encryption. In particular, we are not aware of any previous work on general-purpose digital signature schemes secure against white-box attackers. We present white-box implementations for hash-based signatures so that the security against white-box attackers depends not 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.

Keywords: white-box cryptography · digital signature · white-box 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 algorithm [2].

White-box implementations have a modest objective i.e., preventing extraction of cryptographic keys useful on a different platform. By accepting that there are natural limits for achievable security in such an extreme environment, using directly the cryptographic software on the targeted device is not of concern.

Additionally, 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 addressed by additional software protection techniques, such as device binding and code obfuscation, so that the use of software is not 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 is the first scheme which relies solely on one-way (hash) functions for its security. 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 to make 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 $L, 2N$
    $L$: 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 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: Parameters $M, h$
    $M$: message to be signed
    $h = f(m)$ ($h$ has an output 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 $h'_s = 0$    $h(r'_{s,1}) = p_{s,1}$
        else             $h(r'_{s,2}) = p_{s,2}$
    "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 white-box attack context [1] 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., SPACE [2]). 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. For simplicity, we assume the block length of E_K is also L.

```
Key Generation:
Input: Parameters $L, 2N, IP$
    $L$: 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 $WBT$-$E_K$
Output:
    For one-time private key, generate $2N$ $L$-bit pseudo-random numbers:
```

```
    for $1 \leq k \leq 2N$   $r_k = WBT\text{-}E_K\ (IP + k)$
    As the one-time public key, compute:
    $p_k = f(r_k)$ for $1 \leq k \leq 2N$ (distribute 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: *Parameters* M, h

```
    $M$: message to be signed
    $h = f(m)$ ($h$ has an output 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 Informal Security Analysis and Extensions for Multiple Messages

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 (code lifting attacks and other side concerns outside the threat model is no different than the common white-box assumptions in a symmetric encryption setting). We leave the formal proof as a future work.

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. Below, we show that a single look-up table could be used to sign multiple messages, but there are some caveats that implementations should take into account.

4.1 Implementing Merkle's One-Time Signatures Secure against White-Box Attackers

The problem of extending Lamport's one-time signatures for multiple messages has already been extensively studied in the literature. Most of the proposed schemes are variations of the early work by Merkle [4].

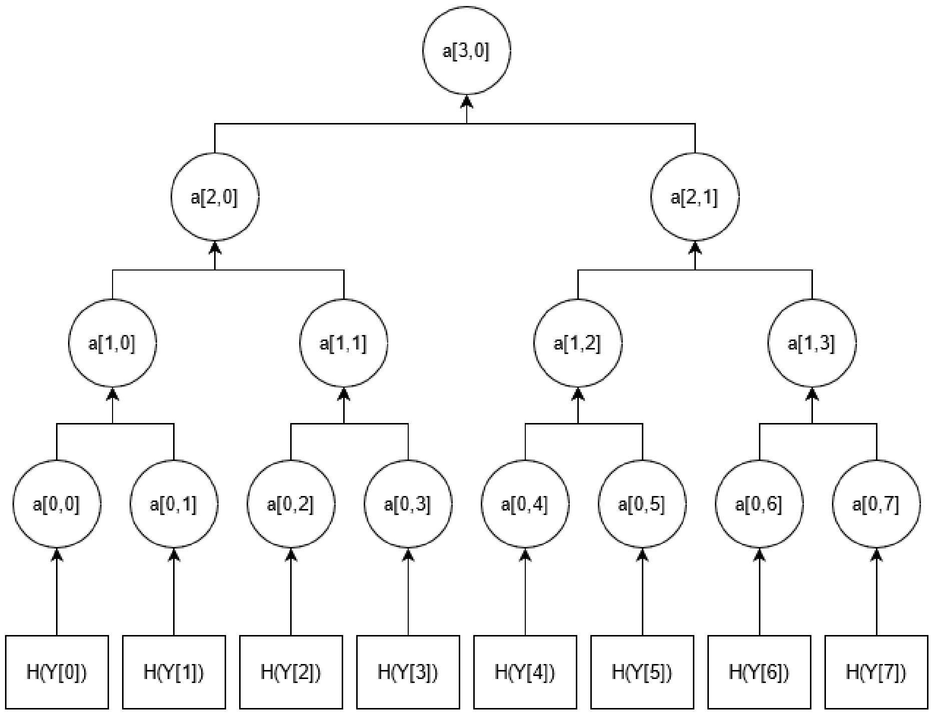

Fig. 1. Merkle tree to sign 8 messages.

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.

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

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., SPACE [2]). 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. We assume the block length of E_K is also L.

```
Key Generation:
Input: Parameters $L, 2N, IP, T$
    $L$: 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 $WBT\text{-}E_K$
    $T$: number of messages to be signed ($T = 2^n$) ($n$ is the height of the tree)
Output:
    a. for $0 \le t \le T-1$
        for $1 \le k \le 2N$
            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:
    $r_{i,j,t} (1 \le i \le N \text{ and } 1 \le j \le 2 \text{ and } 0 \le t \le T-1)$
    $p_{i,j,t} (1 \le i \le N \text{ and } 1 \le j \le 2 \text{ and } 0 \le t \le T-1)$

    b. Generate hashes of public keys as follows:
    for $0 \le t \le T-1$:    $a(0,t) = f(p_{1,1,t}||p_{1,2,t}||\ldots||p_{N,1,t}||p_{N,2,t})$

    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 $M, h, t$
    $M$: message to be signed
    $h = f(m)$ (h has an output length of $N$)
    $t$ = index of the leaf nodes $(0 \leq t \leq T-1)$
**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 $f(r_{s,2,t})$
        else
            reveal $f(r_{s,1,t})$ and $r_{s,2,t}$

    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 $aux_1$, $aux_2$ and $aux_3$ e.g., $a(0,1)$, $a(1,1)$, $a(2,1)$
    Public key: $a(3,0)$
    $h' = f(M')$
**Output:**
    for each $1 \leq s \leq N$
        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 $a'(0,t) = f(p_{1,1,t}||p_{1,2,t}|| \ldots ||p_{N,1,t}||p_{N,2,t})$
    Accept if $a(3,0) = f(f(f(a'(0,t)||aux_1)||aux_2)||aux_3)$
    "Reject" otherwise
```

Now, we will show that the above scheme is not secure in a white-box attacker context. 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 ultimately to 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 auxiliary data 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. For this purpose, after all the component of one-time public key are computed by $p_{k,t} = f(r_{k,t})$ for $1 \leq k \leq 2N$ and $0 \leq t \leq T-1$, the values of $p_{k,t}$ are stored on the client side. Once a signature is required, signature generation algorithm is changed slightly as follows:

```
for $1 \leq s \leq N$ /* 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-attacker could not observe a random number required to forge a signature for any message different than the message legitimate user has already signed.

To summarize, an implementation of hash-based signatures is not secure for our purposes 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 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 [8]). Leaving the analysis of every scheme in the rich literature of hash-based signatures in this context as a future work, we provide the underlying reason using a generic crypto primitive (i.e., Lamport's hash chain) in the next section.

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

A hash chain (also proposed by Lamport [5]) 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 authentication purposes.

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 client side and the hash values 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 stored. 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:

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

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 any computation.
- The client chooses to store only the seed value and generates the required OTP by iteratively doing the required number of hash computations ([4]).

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 or (while one of the elements of the hash chain is generated) 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 an alternative 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 so that 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* $WBT\text{-}E_K$

[4] An amortization technique could also be used to reduce memory-times-computation complexity of $O(n)$ [9]

```
    $T$: *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.*
```

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

	Proposed Scheme	w/ Hash Chains	Independent OTPs
White-box resistant	$\surd$	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	$O(T)$	$O(T)$	$O(T)$
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

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 [6]. As noted in [7], 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 (which element is the next?) is another concern.

A widely used solution for OTPs is implemented in the TOTP standard. Here, the server and the client shares a secret key. By using the current timestamp (usually in a granularity of 30 seconds) as an implicit challenge of the server, this 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 [7]. 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 [5]. However although no secret is stored on the server side, T/Key is vulnerable to a white-box attacker having a full 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. We could either prefer a mapping from left to right or right to left. Suppose we choose a left to right mapping in the example of Merkle three 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
    $t_{init}$: setup time $t$ (measured in slots of length I)
    $t$: 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 2×10^6 leaf nodes (valid for the next two years), initialization (set up) requires 2×10^6 white-box encryption operations besides the hash operations. This drawback might be addressed by using a lightweight white-box encryption primitive.

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 user involvement except a

[5] 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.

simple confirmation tap in a mobile authenticator application. Note that the use of a digital signature in this use case might not be preferable due to a white-box attacker threat. Even when a manual entry of OTP 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 [7], our proposed scheme still seems a viable approach. It requires an OTP length of 352 B ($log_2(2 \times 10^6) + 1) \times 16$ B) for 128-bit security and 2 years of authentication period, which does not exceed the maximum capacity of QR codes [10].

7 Related Work

Joye pointed out that one of the potential applications of white-box cryptography is to transform a MAC into a digital signature [11]. Here, "MAC verification" 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 [12]. Feng et. al. proposed white-box implementation for the classical Shamir's identity based signature scheme [13]. Up to our best knowledge, our paper is the first study presenting a general-purpose digital signature algorithm secure in a white-box attacker context.

8 Concluding Remarks

To motivate our research, we consider a mobile authenticator application supporting multifactor user authentication. In this scenario, digital signatures and 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, as a white-box resistant digital signature solution, we presented a tweak for hash-based signatures and as white-box resistant one-time passwords, we presented a tweak for hash trees. The proposed simple and elegant techniques 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. 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.
5. Lamport, L. (1981). Password authentication with insecure communication. Communications of the ACM, 24(11), 770-772.
6. 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.
7. 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).
8. 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.
9. Jakobsson, M. (2002, June). Fractal hash sequence representation and traversal. In Proceedings IEEE International Symposium on Information Theory, (p. 437). IEEE.
10. QR Codes. (2020). https://github.com/ricmoo/QRCode/blob/master/README.md
11. Joye, M. (2008). On white-box cryptography. Security of Information and Networks, 7-12.
12. 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.
13. 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.