Off-Line/On-Line Signatures: Theoretical aspects and Experimental Results*

Dario Catalano[1], Mario Di Raimondo[1], Dario Fiore[1], and Rosario Gennaro[2]

[1] Dipartimento di Matematica e Informatica – Università di Catania
Viale Andrea Doria 6, 95125 Catania, Italy.
{catalano,diraimondo,fiore}@dmi.unict.it.
[2] IBM Research – Yorktown Heights, NY, USA
rosario@watson.ibm.com

Abstract. This paper presents some theoretical and experimental results about off-line/on-line digital signatures. The goal of this type of schemes is to reduce the time used to compute a signature using some kind of preprocessing. They were introduced by Even, Goldreich and Micali and constructed by combining regular digital signatures with efficient one-time signatures. Later Shamir and Tauman presented an alternative construction (which produces shorter signatures) by combining regular signatures with chameleon hash functions.
We first unify the Shamir-Tauman and Even *et al.* approaches by showing that they can be considered different instantiations of the same paradigm. We do this by showing that the one-time signatures needed in the Even *et al.* approach only need to satisfy a weak notion of security. We then show that chameleon hashing are in effect a type of one-time signatures which satisfy this weaker security notion.
In the process we study the relationship between one-time signatures and chameleon hashing, and we prove that a special type of chameleon hashing (which we call *two-trapdoor*) is a fully secure one-time signature.
Finally we ran experimental tests using OpenSSL libraries to test the difference between the two approaches. In our implementation we make extensive use of the observation that off-line/on-line digital signatures do not require collision-resistant hash functions to compress the message, but can be safely implemented with universal one-way hashing in both the off-line and the on-line step. The main application of this observation is that both the steps can be applied to shorter digests. This has particular relevance if block-ciphers or hash functions based one-time signatures are used since these are very sensitive to the length of the message. Interestingly, we show that (mostly due to the above observation about hashing), the two approaches are comparable in efficiency and signature length.

1 Introduction

Off-line/On-line digital signatures were introduced by Even, Goldreich and Micali in [12]. In these signatures the signing process is divided in two parts. First

* An extended abstract of this paper appears in the proceedings of PKC 2008

a computationally intensive part is performed off-line, i.e. before the message being signed is known. This off-line part produces some temporary data which is stored and then used at the time the message to be signed is known. At that point, the computation of the actual signature requires very little effort.

The original construction in [12] was based on combining two different types of digital signatures: many-times (or "regular") signatures and one-time signatures [24, 21, 2, 22, 26]. While the former can be used to sign a polynomial number of messages, in the latter a private key can be used to sign only a single message. Because of this limitation, one-time signatures can be constructed more efficiently. The construction in [12] goes as following. The signer generates a pair (VK, SK) of keys for a regular signature scheme: she publishes VK and keeps SK as a secret. In the off-line part she generates vk a one-time public verification key, and signs it with SK: let S be the resulting signature. Then when the message m is available, the signer computes its signature s with the one-time signing key sk. The final signature is (vk, S, s).

The construction in [12] utilizes one-way functions based one-time signatures, such as the ones introduced by Lamport [20]. While these signatures are very fast to compute and verify, the signature string can be very long, and it grows quadratically with the length of the message being signed.

To address these issues Shamir and Tauman in [27] offered an alternative construction which combines regular signatures with chameleon hashing [18]. A chameleon hash function is defined by a public key pk and a secret trapdoor tk. The function $\mathsf{C}_{pk}(\cdot, \cdot)$ takes two arguments a message m and a random string r. The function is collision-resistant, unless one knows the trapdoor tk. But knowledge of tk allows to find arbitrary collisions, i.e. given $c = \mathsf{C}_{pk}(m, r)$ and an arbitrary different message m', the holder of the trapdoor can find r' such that $c = \mathsf{C}_{pk}(m', r')$. For many chameleon hash functions, this collision-finding procedure is very efficient, requiring only a single modular multiplication. The Shamir-Tauman idea is to construct off-line/on-line signatures as follows. The signer's public key is VK, like before, and pk. The off-line part would consists of computing $c = \mathsf{C}_{pk}(a, r')$ for some arbitrary a, r' and then computes S the signature of c using SK. On input the actual message m the signer (who knows the trapdoor tk as part of the signing key) computes r such that $c = \mathsf{C}_{pk}(m, r)$ and outputs (S, r). The verifier re-computes c as $\mathsf{C}_{pk}(m, r)$ and verifies S on it. As we will see later in the examples of chameleon hashing, the length of r grows only linearly in the length of the message m, so the Shamir-Tauman approach provides shorter signatures.

1.1 Our Contributions

This work was motivated by two basic questions:

1. Is the Shamir-Tauman approach conceptually different from the Even *et al.* approach, or are they really two different instantiations of the same paradigm?
2. In practical implementations, for today's security levels, which approach is preferable, in terms of speed, memory and ease of implementation?

This paper presents some theoretical and experimental results about off-line/on-line digital signatures which are aimed at answering the above questions.

We first show that conceptually the Shamir-Tauman construction is not different from the Even *et al.* one. Indeed we present a unifying paradigm which encompasses both the Shamir-Tauman and the Even *et al.* approaches. We do this by showing that a chameleon hash function can also be seen as a one-time signature with a very weak security property. As already observed in [12], this weak property is sufficient to prove the security of the Even *et al.* approach. In the process of exploring the relationship between one-time signatures and chameleon hashing, we discovered that fully secure one-time signatures can be obtained from a special type of chameleon hashing that we call *two-trapdoor chameleon hashing*.

Finally we ran experimental tests using OpenSSL libraries to test the difference between the two approaches. In our implementation we make extensive use of the observation that off-line/on-line digital signatures do not require collision-resistant hash functions to compress the message, but can be safely implemented with universal one-way hashing in both the off-line and the on-line step. The main application of this observation is that both the steps can be applied to shorter digests. This has particular relevance if block-ciphers or hash functions based one-time signatures are used since these are very sensitive to the length of the message. Surprisingly, we show that (mostly due to the above observation about hashing), the two approaches are comparable in efficiency and signature length.

Related work. As we pointed out, Even *et al.* introduced the notion of off-line/on-line signatures in [12] and constructed them combining regular signatures with efficient one-time signatures. However the length of the signatures is an issue in this approach. Shorter signatures can be obtained by using chameleon hashing [18] combined with regular signatures as pointed out by Shamir and Tauman [27]. Off-line/On-Line digital signatures can also be obtained by applying the Fiat-Shamir heuristic to a variety of identification protocols known as Σ-protocols. Example of such schemes are [13, 30, 11, 28]. However such schemes are proved secure in the random oracle model [3, 23]; our paper is focused on schemes which are secure in the standard model.

2 Preliminaries

In the following, with $\mathbb{N}$ we denote the set of integers and with $\mathbb{R}$ the set of real numbers. We denote the security parameter with ℓ. A function $f : \mathbb{N} \to \mathbb{R}$ is said to be negligible if for any $c > 0$, there exists an index $\ell_c \in \mathbb{N}$ such that $f(\ell) < \ell^{-c}$ for all $\ell > \ell_c$.

2.1 Hash functions

For lack of space the definitions of Collision resistant hash functions and Target collision resistant hash functions, are postponed to Appendix D

Target Division Intractable (TDI) hash functions: Consider a family $\mathcal{H} = \{h^{\mathsf{tdi}}(k,\cdot)\}_k$ (we are making explicit the fact that an element of the family is parametrized by a key k) with $poly(\ell)$-bit input and ℓ bit output. We say that $\mathcal{H}$ is *target division intractable* if it is hard for the attacker to win the following game:

1. the attacker chooses polynomially many inputs $x_1, x_2, \ldots$;
2. a random key k is chosen;
3. the attacker outputs $y \neq x_i$ such that $h^{\mathsf{tdi}}(k,y)$ divides the product of the $h^{\mathsf{tdi}}(k,x_i)$'s.

This notion was introduced in a stronger variant[3] by Gennaro *et al.* [14]. They conjectured that a random oracle with approximately 600 bits of output would be a safe choice for a DI function. Later, Coron and Naccache [7] described an attack that disproves such a conjecture and forces one to use functions with much longer outputs (see [7] for details).

Recently, Kurosawa and Schmidt-Samoa [19] introduced the notion of *weak division intractability* (wDI). Informally, wDI formalizes a weaker (i.e. with respect to the notions discussed above) notion of division intractability. Here, the adversary A should be unable to find $y \neq x_1, \ldots x_n$, such that $h^{\mathsf{tdi}}(k,y)$ divides the product of the $h^{\mathsf{tdi}}(k,x_i)$'s, when the x_i's are chosen at random (i.e. and thus are not of A's choice). Kurosawa and Schmidt-Samoa showed that this property is sufficient to prove the random-message security (see Section 2.3) of the GHR signature scheme. Very informally, this is because the attack of Coron and Naccache crucially relies on the attacker choosing the x_i's.

Notice that, by a similar reasoning, the same attack cannot be applied if one uses a TDI function. This is because, in such a case, the adversary does not know the key (of the hash function) when choosing the x_i's. This is why in our constructions we only require the GHR scheme to be obliviously secure and the underlying hash function to be target division intractable.

2.2 Chameleon Hashing

Definition 1. *A chameleon hash function (also known as trapdoor commitment scheme) is a triplet of polynomial-time algorithms:*

$\mathsf{CKG}(1^\ell)$**:** *a probabilistic algorithm which, on input a security parameter 1^ℓ, outputs a pair of matching public/private keys (pk, tk);*

$\mathsf{C}_{pk}(m,r)$**:** *the evaluation algorithm which, on input the public key pk, a message m and a random nonce r, outputs a hashed value;*

$\mathsf{Coll}(tk,m,m',r)$**:** *the collision finding algorithm which, on input the private trapdoor key tk, two messages m, m' and a nonce r, outputs a nonce r' such that $\mathsf{C}_{pk}(m,r) = \mathsf{C}_{pk}(m',r')$.*

[3] In such a variant, called division intractability (DI), the adversary is allowed to choose the x_i's after having seen the hash function

As required in [18], the public key defines a particular hash function which, however, takes a random input additionally to the message. The security properties of this function are as follows:

Collision Resistance Without knowledge of the associated trapdoor, this function is collision resistant, i.e. it is infeasible to find two different pairs $(m, r), (m', r')$ such that $\mathsf{C}_{pk}(m, r) = \mathsf{C}_{pk}(m', r')$;

Distribution of Collisions For every m, m', and a random r, the distribution of $r' = \mathsf{Coll}(tk, m, m', r)$ is uniform, even when given pk, $c = \mathsf{C}_{pk}(m, r)$, m and m'. This implies that the chameleon hashing function is also a information-theoretically hiding commitment.

Two efficient constructions of chameleon hash function follow: the first is due to Boyar *et al.* [4] and its security is based on the Discrete Log problem difficulty; the second [8, 10] relies on the RSA assumption. Both these constructions are given in Appendix E.

2.3 Signature schemes

We recall the definition of secure signature scheme from [15].

Definition 2. *A signature scheme is a triplet* $(\mathsf{KG}, \mathsf{Sign}, \mathsf{Ver})$ *of PPT algorithms:*

- *the key generation algorithm* $\mathsf{KG}(1^\ell)$ *outputs a pair* (vk, sk) *of matching public/private keys;*
- *the signing algorithm* $\mathsf{Sign}(sk, m)$ *takes as input the private key and a message* m *and produces a signature* σ*;*
- *the verification algorithm* $\mathsf{Ver}(vk, m, \sigma)$ *takes as input the public key, a message and an alleged signature* σ *and outputs a single bit.*

For every possible output (vk, sk) of KG, and every m, it is required that $\mathsf{Ver}(vk, m, \mathsf{Sign}(sk, m)) = 1$. We say that a signature scheme is secure against adaptive chosen message attack (or in short "secure") if a forger after asking for the signature on several adaptively chosen messages will not be able to produce a valid signature on a message he had not previously requested.

Definition 3. $(\mathsf{KG}, \mathsf{Sign}, \mathsf{Ver})$ *is a* secure signature scheme *if for every efficient forger* $\mathcal{F}$*, the following*

$$\Pr\left[\begin{array}{l} (vk, sk) \leftarrow \mathsf{KG}(1^\ell)\ ; \\ \textbf{for } i = 1 \textbf{ to } k \\ \qquad M_i \leftarrow \mathcal{F}(vk, M_1, \sigma_1, \ldots, M_{i-1}, \sigma_{i-1})\ ; \\ \qquad \sigma_i \leftarrow \mathsf{Sign}(sk, M_i)\ ; \\ (M, \sigma) \leftarrow \mathcal{F}(vk, M_1, \sigma_1, \ldots, M_k, \sigma_k)\ ; \\ \mathsf{Ver}(vk, M, \sigma) = 1 \quad \textit{and} \quad M \neq M_i \end{array}\right]$$

is negligible in ℓ*.*

We say that a signature scheme is *obliviously secure* if in the game above the adversary chooses the messages M_i before seeing the public key. Also we say that a signature scheme is *random-message secure* if the above holds for messages M_i chosen randomly in the message space, rather than adaptively and adversarially chosen. Similar security definitions apply to one-time signatures if the above hold for $k = 1$.

Examples of one-time signatures Lamport *et al.* [20] proposed a method to construct a one-time signature scheme from one-way functions. Later Even, Goldreich and Micali [12] suggested an improved method to shorten the length of keys and signatures. In the follow we recall their ideas and also describe a technique due to Jakobsson [17] to speedup the signature phase.

- **Lamport's scheme:** let M be the m-bit message to sign and $f : \{0,1\}^\ell \to \{0,1\}^\ell$ be a one-way function. We choose $2m$ ℓ-bit strings $x_1^0, x_1^1, \ldots, x_m^0, x_m^1$ at random as the signing key. The verification key is computed applying f to each x_i^0, x_i^1 for $i = 1, \ldots, m$: $f(x_1^0), f(x_1^1), \ldots, f(x_m^0), f(x_m^1)$.
  To sign a message $M = \mu_1, \cdots, \mu_m$ the signer reveals $x_1^{\mu_1}, \ldots, x_m^{\mu_m}$. Given a message M and its signature $s = s_1, \cdots, s_m$, the verifier applies f to the values $s_1, \cdots, s_m$ from the signature and checks if they are equal to the corresponding images in the verification key.
  This simple scheme is proved to be secure if f is a one-way function; it is really fast but has the drawback of quite large keys and signatures.
- **Shortening length of keys and signatures (Even *et al.*'s):** let M be the m-bit message, we partition the message in blocks of t bits, where $t|m$. Let f be a one-way function as before[4]. We choose at random $\frac{m}{t} + 1$ ℓ-bit strings $x_0, x_1, \ldots, x_{m/t}$ as the signing key. The corresponding verification key is:
  $$y_0 = f^{(2^t-1)m/t}(x_0); \quad y_1 = f^{2^t-1}(x_1), \ldots, y_{m/t} = f^{2^t-1}(x_{m/t})$$
  To sign a message $M = \mu_1, \ldots, \mu_{m/t}$, whose t-bit blocks μ_i are interpreted as integers, the signer outputs:
  $$s_0 = f^{\sum_{i=1}^{m/t} \mu_i}(x_0),\ s_1 = f^{2^t-1-\mu_1}(x_1), \ldots, s_{m/t} = f^{2^t-1-\mu_{m/t}}(x_{m/t})$$
  Given a message $M = \mu_1, \ldots, \mu_{m/t}$ and a signature $s_0, s_1, \ldots, s_{m/t}$ the verifier applies f to each signature component the proper times and compares the resulting values with the verification key elements. Namely, it checks:
  $$y_0 \stackrel{?}{=} f^{(2^t-1)m/t-\sum_{i=1}^{m/t} \mu_i}(s_0); \quad y_1 \stackrel{?}{=} f^{\mu_1}(s_1), \ldots, y_{m/t} \stackrel{?}{=} f^{\mu_{m/t}}(s_{m/t})$$
  It is interesting to note the trade-off: a small t makes the signature computation more efficient (because the hash chains are shorter), but makes the signature longer (because the number of blocks m/t is bigger).

[4] As explained in Appendix A, the proof of security requires a stronger assumption than the inverting infeasibility: the *quasi-inverting assumption* has to hold on f. Also in Appendix A we make a concrete security analysis of this assumption compared to the assumption of basic one-wayness.

– **Speedup the signature step (Jakobsson's):** in the previous scheme the length of the hash chains is exponential in the size of the block; this makes the signature and verification steps computationally expensive for big blocks. The optimization for one-way hash chains traversal proposed by Jakobsson [17] can be applied here: the idea is to store not only the first and last value of the chains, but also some intermediate elements (called *pebbles*) that permit in the signature procedure to speedup the traversal originating the iterative computation from the nearer pebble in the chain. In [17] it is stated that keeping $O(\log n)$ number of pebbles, where n is the chain length, the traversal time becomes $O(\log n)$; in our case, the storage and the running time become $O(t)$, where t is the size of the block.

Examples of obliviously secure signatures In this section we recall two signature schemes: one is due to Gennaro *et al.* [14] and the other to Cramer and Shoup [10]. Their security is based on the Strong RSA Assumption, and they are the most efficient signature schemes in the literature whose security can be proved without using the random oracle model.

We present simplified versions of these schemes which can be proved to be obliviously secure since that's all we need later.

– **Simplified GHR Signature:** This scheme uses a target division-intractable hash function $h^{\mathsf{tdi}}(\cdot,\cdot)$.
  - **Key generation:** let $N = pq$ be an RSA modulus where p, q are safe primes of identical sizes; select a random element s in $\mathbb{Z}_N^*$ and a key k for the TDI hash function $h^{\mathsf{tdi}}(\cdot,\cdot)$; the public key is (N, s, k) and the secret key is $\phi(N) = (p-1)(q-1)$.
  - **Signature algorithm:** given a message m to sign, compute $e = h^{\mathsf{tdi}}(k, m)$ and $d = e^{-1} \bmod \phi(N)$ and outputs the signature $\sigma = s^d \bmod N$.
  - **Verification algorithm:** on input the public key (N, s, k) and the message/signature pair m, σ, compute the value $e = h^{\mathsf{tdi}}(k, m)$ and check if $\sigma^e = s \bmod N$.
– **Simplified CS Signature:**
  - **Key generation:** generate an RSA modulus $N = pq$ as in GHR (safe primes), select two random elements s, t in $\mathbb{Z}_N^*$ and draw a random key k for a TCR hash function $h^{\mathsf{tcr}}(\cdot,\cdot)$; the public key is (N, s, t, k) and the secret key is $\phi(N)$.
  - **Signature algorithm:** given an arbitrary long message m to sign, generate a random 161-bit prime e and compute $d = e^{-1} \bmod \phi(N)$ and $\sigma = (st^{h^{\mathsf{tcr}}(k,m)})^d \bmod N$. The signature is (e, σ).
  - **Verification algorithm:** on input the public key (N, s, t, k) and the message/signature pair $m, (e, \sigma)$, check if $\sigma^e = st^{h^{\mathsf{tcr}}(k,m)} \bmod N$.

  Cramer and Shoup in [10] suggest an efficient method for the generation of small primes of 161 bits. This operation is critical for the performance of the scheme since a fresh 161 bit prime number is necessary to sign a message.

2.4 Off-line/On-line digital signatures

In this section we recall the Even *et al.* and Shamir-Tauman approaches to construct off-line/on-line signatures.

Using one-time signatures The idea is to combine a random-message secure signature scheme with a one-time signature. In the off-line step a pair of keys for a one-time signature is generated and the public key of this scheme is signed using the long-term signing key. During the on-line phase, given the message to sign, its signature is computed using the one-time secret key.

We call this scheme the EGM scheme. A more detailed description follows: let $(\mathsf{KG}, \mathsf{Sign}, \mathsf{Ver})$ be a signature scheme, $(\mathsf{KG}^{\mathsf{ot}}, \mathsf{Sign}^{\mathsf{ot}}, \mathsf{Ver}^{\mathsf{ot}})$ a one-time signature scheme. The combined off-line/on-line signature works as follows:

- **Key generation**: this step coincides with the key generation of the ordinary scheme; run $\mathsf{KG}(1^\ell)$ to obtain a pair of long-term keys (VK, SK); the public component VK is announced, while SK is kept secret.
- **Off-line Signature**: in this phase a fresh pair of keys (vk, sk) for a one-time signature is generated using $\mathsf{KG}^{\mathsf{ot}}(1^\ell)$. The verification key vk is signed with the long-term signing key SK as $\pi = \mathsf{Sign}(SK, vk)$. The token (vk, sk, π) is kept as part of the signer's state.
- **On-line Signature**: given the message m to sign, a precomputed token (vk, sk, π) is retrieved; the message m is signed using the one-time scheme as $\sigma = \mathsf{Sign}^{\mathsf{ot}}(sk, m)$ and the complete signature is the triple (vk, π, σ).
- **Verification**: given a message m and its purported signature (vk, π, σ), the master verification key VK is used as follows. First, the algorithm Ver is used to check that π is indeed a valid signature of vk with respect of the long-term verification key VK. Next, the tag σ is verified to be a (one-time) signature of m using vk; namely, the verification consists in evaluating the following predicate:

$$\mathsf{Ver}(VK, vk, \pi) \wedge \mathsf{Ver}^{\mathsf{ot}}(vk, m, \sigma)$$

The following Theorem appears in [12].

Theorem 1 (EGM [12]). *If $(\mathsf{KG}, \mathsf{Sign}, \mathsf{Ver})$ is a "regular" signature scheme and $(\mathsf{KG}^{\mathsf{ot}}, \mathsf{Sign}^{\mathsf{ot}}, \mathsf{Ver}^{\mathsf{ot}})$ is a one-time signature scheme and both the schemes are secure (as in Definition 3) then the EGM scheme described above is secure in the standard sense.*

Using Chameleon hash functions This construction is also known as the "hash-sign-switch" paradigm: in the off-line phase, the signer hashes an arbitrary message m' with a chameleon hash. It then signs the results. When, during the on-line phase, he is given the message m the signer uses its knowledge of the chameleon hash trapdoor to find a second preimage and "switches" m with the arbitrary m' used in the off-line phase.

We call this the ST scheme. Let $(\mathsf{KG}, \mathsf{Sign}, \mathsf{Ver})$ be a signature scheme and $(\mathsf{CKG}, \mathsf{C}, \mathsf{Coll})$ a chameleon hash function family. Given a security parameter ℓ, an off-line/on-line signature scheme can be constructed as follows:

– **Key generation**: a pair of keys (VK, SK) is generated using the signature key generation algorithm $\mathsf{KG}(1^\ell)$; furthermore, a specific chameleon hash function is selected in the family using the trapdoor key generation algorithm as $(pk, tk) = \mathsf{CKG}(1^\ell)$. The signing key is (SK, tk) and the verification key is (VK, pk).
– **Off-line Signature**: an arbitrary message m' is chosen together with a random string r'. The hash value $\delta = \mathsf{C}_{pk}(m', r')$ is computed and signed with SK, to compute $\sigma = \mathsf{Sign}(SK, \delta)$; the token (m', r', σ) is kept in the signer's internal state.
– **On-line Signature**: given the message m to sign, a precomputed token (m', r', σ) is retrieved; use Coll with the trapdoor key tk to find r such that $\mathsf{C}_{pk}(m, r) = \delta = \mathsf{C}_{pk}(m', r')$; the signature given in output is (r, σ).
– **Verification**: given a message m and a signature (r, σ), first compute $\delta = \mathsf{C}_{pk}(m, r)$ and then verify the signature σ on it using $\mathsf{Ver}(VK, \mathsf{C}_{pk}(m, r))$.

Theorem 2 (ST [27]). *If* $(\mathsf{CKG}, \mathsf{C}, \mathsf{Coll})$ *is a chameleon hash function and* $(\mathsf{KG}, \mathsf{Sign}, \mathsf{Ver})$ *is an obliviously secure signature scheme then the* ST *scheme described above is a secure signature scheme.*

3 A unifying paradigm

In this section we show that the Even, Goldreich, Micali [12] construction and the Shamir, Tauman [27] solution can be seen as two special cases of the same methodology. This would be immediate if we could show that chameleon hashing is a form of secure one-time signatures. Unfortunately that is not true in general, though in the next subsection, we describe a sufficient condition on chameleon hashing to be a secure one-time signature. Nevertheless, for a general statement, we must follow a different approach.

Our starting point, is the observation (originally made in [12]) that the Even, Goldreich, Micali construction remains secure even if the underlying one-time and regular signature schemes are obliviously secure. Next, we show that chameleon hash functions are a form of oblivious one-time signatures. This shows an unifying paradigm that encompasses both the Even *et al.* and the Shamir-Tauman approach.

Informally an oblivious one-time signature is guaranteed to be secure only against an adversary which chooses the (one) message for which she is allowed to see a valid signature, *before* seeing the public key. Notice that this level of security is indeed sufficient for the EGM approach since, in the off-line/on-line EGM signature, the keys of the one-time signatures are chosen *independently* from the message being signed (i.e. the adversary does not see the keys of the one-time signature when she submits a message to be signed).

Definition 4. $(\mathsf{KG}, \mathsf{Sign}, \mathsf{Ver})$ *is an* obliviously secure one-time signature *if for every efficient forger* $\mathcal{F}$*, the following probability is negligible in* ℓ*.*

$$\Pr\left[\begin{array}{l}(M, \mathtt{state}) \leftarrow \mathcal{F};\ (vk, sk) \leftarrow \mathsf{KG}(1^\ell);\ \sigma \leftarrow \mathsf{Sign}(sk, M)\ ; \\ (M', \sigma') \leftarrow \mathcal{F}(vk, M, \sigma, \mathtt{state})\ : \\ \mathsf{Ver}(vk, M', \sigma') = 1 \quad \textit{and} \quad M' \neq M\end{array}\right]$$

We state the following

Theorem 3 (EGM [12]). *If* $(\mathsf{KG}, \mathsf{Sign}, \mathsf{Ver})$ *is an obliviously secure signature scheme and* $(\mathsf{KG}^{\mathsf{ot}}, \mathsf{Sign}^{\mathsf{ot}}, \mathsf{Ver}^{\mathsf{ot}})$ *is an obliviously secure one-time signature scheme then the* EGM *scheme described above is a secure signature scheme.*

Now we show that an oblivious one-time signature scheme can be implemented using a chameleon hash function. The construction Cham-Sig is as follows.

Key Generation. On input a security parameter ℓ, run $\mathsf{CKG}(1^\ell)$. Then it chooses a message α and a nonce r and computes $c = \mathsf{C}_{pk}(\alpha, r)$. The public key is (pk, c), the signing key is (tk, α, r).

Signature Algorithm. On input a message m the signer uses his knowledge of the trapdoor to compute a nonce s such that, $c = \mathsf{C}_{pk}(m, s)$. The signature is then (m, s).

Verification. On input a purported signature (m, s), the verifier checks whether $c = \mathsf{C}_{pk}(m, s)$. If this is the case the signature is accepted as valid, otherwise it is rejected.

The proof of the following theorem appears in Appendix C.1.

Theorem 4. *The scheme presented above is an obliviously secure one-time signature scheme assuming that the underlying primitive is a chameleon hash function.*

3.1 Double trapdoor Chameleon hash function

In the previous section we showed that a chameleon hash function is an obliviously secure one-time signature. It is not hard to see why it fails to be a (fully) secure one-time signature. In the oblivious case, the adversary commits to the message she wants to be signed before seeing the public key: this allows us to "prepare" the public key as a commitment to that specific message. In the adaptive case, when we prepare the public key we do not know the message, so when the adversary asks us for a signature we do not know how to produce it.

In order to get a fully adaptively secure one-time signature from chameleon hashing, a possible way is to compose two different hash functions (i.e. apply one function over the output of the other). Conceptually this is not surprising as it corresponds to a chain of length two in the [15] signature scheme (in that scheme a chain of length two, instead of a full binary tree, gives a one-time signature).

In some cases we can do better. If a chameleon hashing admits the "double trapdoor" property (described below) then we can obtain the same effect as composing two hash functions, but more efficiently.

A double trapdoor chameleon hash function scheme generalizes the notion of chameleon hash by allowing the existence of two independent trapdoors. Knowing either of the two trapdoors, one can can easily find collisions. More formally:

Definition 5. *Let ℓ be a security parameter. A double trapdoor chameleon hash function is composed of the following, polynomial-time, algorithms:*

$\mathsf{CKG}(1^\ell)$**:** *a probabilistic algorithm which, on input the security parameter* 1^ℓ*, outputs a triplet of public/private keys* (pk, tk_0, tk_1)

$\mathsf{TCKG}(1^\ell, i)$**:** *a probabilistic algorithm which, on input the security parameter* 1^ℓ *and a bit* i *outputs a pair of public/private keys* (pk, tk)*.*

$\mathsf{C}_{pk}(m, r)$**:** *the evaluation algorithm which, on input the public key* pk*, a message* $m \in M$ *and a random nonce* $r \in R$*, outputs a hashed value;*

$\mathsf{Coll}(tk_i, m, m', r)$**:** *the collision finding algorithm which, on input one of the two private trapdoor keys* tk_i*, two messages* m*,* m' *and a nonce* r*, outputs a nonce* r' *such that* $\mathsf{C}_{pk}(m, r) = \mathsf{C}_{pk}(m', r')$*.*

We make the following security requirements

Distribution of Keys Let $\overline{\mathsf{CKG}}(1^\ell, i)$ the algorithm that executes $\mathsf{CKG}(1^\ell)$ and restricts its output to (pk, tk_i). We require that the distribution of the output of $\mathsf{TCKG}(1^\ell, i)$ is identical to the distribution of the output of $\overline{\mathsf{CKG}}(1^\ell, i)$.

Collision Resistance Let $(pk, tk_0, tk_1) = \mathsf{CKG}(1^\ell)$.

1. For every $i = 0, 1$, given pk and tk_i it is infeasible to find $tk_{i\oplus 1}$.
2. Moreover there exists an, efficient, algorithm A that on input the public key pk and a collision m, r, m', r' finds at least one of the trapdoors tk_i.

As a consequence, it is infeasible to find collisions without at least one of the trapdoors tk_i.

Distribution of Collisions For every m, m', and a random r, and for every $i = 0, 1$, the distribution of $r' = \mathsf{Coll}(tk_i, m, m', r)$ is uniform, even when given pk, $c = \mathsf{C}_{pk}(m, r)$, m and m'. As in the case of the 'regular' chameleon hashing, this implies that the function is an information-theoretically hiding commitment. Moreover it implies that the distributions of the openings are the same no matter what trapdoor one uses.

Double trapdoor chameleon hashing leads to a very simple construction of a fully secure one-time signature scheme (rather than just an obliviously secure signature scheme as it is the case when using standard chameleon hash functions). The construction given a two-trapdoor chameleon hash $(\mathsf{CKG}, \mathsf{C}, \mathsf{Coll})$ is as follows.

Key Generation. On input a security parameter ℓ, run $\mathsf{CKG}(1^\ell) = (pk, tk_0, tk_1)$. Then it chooses a message α and a nonce r and computes $c = \mathsf{C}_{pk}(\alpha, r)$. The public key is (pk, c), the signing key is (tk_0, tk_1, α, r).

Signature Algorithm. On input a message m the signer uses his knowledge of either trapdoor to compute a nonce s such that, $c = \mathsf{C}_{pk}(m, s)$. The signature is then (m, s).

Verification. On input a purported signature (m, s), the verifier checks whether $c = \mathsf{C}_{pk}(m, s)$. If this is the case the signature is accepted as valid, otherwise it is rejected.

Theorem 5. *If* $(\mathsf{CKG}, \mathsf{C}, \mathsf{Coll})$ *is a two-trapdoor chameleon hash function then the scheme presented above is a secure one-time signature scheme.*

The proof appears in Appendix C.2.

CONSTRUCTION. The notion of double trapdoor commitment scheme was proposed (even though not explicitly defined) in [6]. There they present a scheme based on the discrete logarithm problem and they show how to use such a construction to build threshold on-line off-line digital signature schemes. In Appendix F we briefly recall the double trapdoor commitment scheme given in [6]

4 Experimental results

As we said in the introduction, this work was motivated by two basic questions about the relationship between the EGM and the Shamir-Tauman approach to build off-line/on-line signatures. In the previous section we showed that, at least conceptually, the Shamir-Tauman approach is really an instantiation of the EGM paradigm. In this section we set out to discuss a *practical* comparison between the two approaches in terms of their efficiency. To achieve that, an extensive work of implementation was carried out. We implemented all the schemes presented in the previous sections, in order to directly measure their real efficiency. To get objective values, all the implementations share the same level of optimization and all the tests were iterated hundreds of times on a reference hardware: an Intel Pentium 4 CPU running at 2.80 GHz. We implemented the algorithms in C using OpenSSL[29] as the underlying library for large number manipulations[5].

4.1 Implementation details

The different types of hash functions (see Section 2.1) required in our constructions were implemented as follows:

- **FCR hashing**: we use SHA-1 [9] with its full 160-bit output;
- **TCR hashing**: it is implemented using SHA-1 as follows[16]: given a message x and the key k, the function is computed as $h^{\mathsf{tcr}}(k, x) = Trunc_\ell(\mathsf{SHA\text{-}1}(x \oplus k'))$, where k' is the concatenation of copies of k until k' and x have the same length. $Trunc_\ell(\cdot)$ is a function that outputs the first ℓ bits of its input.
- **TDI hashing**: as practical construction we use the one suggested in [14] with SHA-1 as the underling tool, but with an additional randomizing key. Given a message x and a key k:

$$h^{\mathsf{tdi}}(k, x) = Set_{msb}(Set_{lsb}(\mathsf{SHA\text{-}1}(x \circ 1 \oplus k') \circ \cdots \circ \mathsf{SHA\text{-}1}(x \circ 4 \oplus k')))$$

  where $\circ$ is the concatenation operator, k' is the concatenation of copies of k until k' and x have the same length and $Set_{msb}(\cdot)$, $Set_{lsb}(\cdot)$ are functions that force the most-significant-bit (resp. least-significant-bit) to be 1; this function takes arbitrarily long inputs and outputs of 640-bit integers.

[5] The sources of the tests are available upon request to the authors.

One-Time Signatures. In our tests we implemented the one-time signature proposed by Even *et al.* with the option to apply Jakobsson's speedup (see Section 2.3). The one-way function used in the implementation is:
$f(x) = Trunc_\ell(\mathsf{SHA\text{-}1}(x))$ with different values for the security parameter ℓ.

The public key in this scheme is composed of m/t strings $y_0, y_1, \ldots, y_{m/t}$, but it can also be replaced with its hash value $y = h^{\mathsf{fcr}}(y_0, y_1, \ldots, y_{m/t})$, which is what we do in our implementation, in order to keep keys shorter (the price to pay is an extra computation of h^{fcr} at verification time).

4.2 Using target-collision resistant hash

USING TCR HASHING IN THE ON-LINE STEP. When signing messages one usually hashes them down with a FCR function to shorten them. It is well known that one can uses a TCR function provided that the key of the hash function is signed together with the message digest and sent as part of the signature. One of the advantages of using TCR functions is that the message digest may be shorter, but this advantage is usually off-set by the need to sign the key as well.

However in the case of off-line/on-line signatures, the advantage of using TCR functions can be substantial. Indeed one can 'prepare in advance' the key k for the TCR function to be used in the on-line step, and sign it during the off-line step with the "regular" signature scheme.

In the EGM construction, this results in a substantial efficiency gain, since the one-way functions based one-time signatures are very sensitive to the length of the message being signed. Indeed the size of the signature grows quadratically in the length. Since the key k of the TCR function is signed in the off-line step, the one-time signature is only applied to the digest, resulting in a substantially shorter signature.

Similarly in the Shamir-Tauman approach, using a TCR function to hash the message in the online case can improve the efficiency. For example if we use the RSA-based chameleon hash in Appendix E it will be possible to use a shorter public exponent e.

USING TCR HASH IN THE OFF-LINE STEP. As we pointed in the previous section, the quantities signed in the off-line step are *not* under the control of the adversary, and they are actually random quantities (the verification key of the one-time signature or of the chameleon hashing, and the key of the TCR function). For this reason it is also possible to use a TCR function to compress them, rather than a FCR one. In this case the key k is chosen once and for all and made part of the public key.

4.3 Test settings

As we said above we performed implementation of all the schemes described above. With OTS we denote the "one-time signature" based on one-way functions described in Section 2.3.

`GHR-OTS` **setting**: This implementation uses the GHR scheme for the off-line step, and the OTS scheme for the on-line case. As pointed above we use TCR

hashing to compress the message, the key is signed in the off-line part. The `GHR-OTS` setting can be configured with various parameters. In all our experiments we set the size of the RSA modulus to 1024. We varied the other parameters as you can see in the Table. These parameters are: `ots_l` $= 80, 96, 112$ the size of the TCR output, i.e. the size of the digest being signed in the OTS scheme; `ots_t` $= 4, 8, 10, 12, 16$ the size of the blocks in the OTS scheme; and `ots_p` $= 1, 5, 8, 10, 12, 16$ the number of memorized pebbles in Jakobsson's optimization (1 means that it is disabled).

`GHR-DL` **setting**: This implementation uses the GHR scheme for the off-line step, and the discrete-log based chameleon hashing for the on-line case. Here we use FCR hashing to compress the message in the on-line step. We implemented the group G as the subgroup of order q in Z_p^* where p, q are primes such that $q|(p-1)$. The parameters of the `GHR-DL` setting are: the size of the GHR modulus N and the sizes of the primes p and q. We only ran experiments with $|N| = |p| = 1024$ and $|q| = 160$.

`GHR-RSA` **setting**: This implementation uses the GHR scheme for the off-line step, and the RSA based chameleon hashing for the on-line case. Here we use FCR hashing to compress the message in the on-line step. The parameters of the `GHR-RSA` setting are: the size of the GHR modulus N (which can be used also as the modulus for the chameleon hash) and the size of the exponent e for the chameleon hash. We only ran experiments with $|N| = 1024$ and $|e| = 160$.

`GHR-DL2` **setting**: This the same as `GHR-DL` but use TCR hashing to compress the message in the on-line step. The key of the TCR hash is signed in the off-line step. This results in the shortening of some of the exponents used to compute the chameleon hash. The parameters are the same of `GHR-DL` with an extra one: `tcr_bits` $= 80, 96$ the length of the output of the TCR hash function.

`GHR-RSA2` **setting**: the same as `GHR-RSA` but using TCR hashing to compress the message in the on-line step. The key of the TCR hash is signed in the off-line step. This results in the shortening of the public exponent e used to compute the chameleon hash. In this case the parameter `tcr_bits` $= 80, 96$ denotes the length of the output of the TCR hash function and of the exponent e (actually $|e| =$ `tcr_bits` $+ 1$).

`CS-OTS`, `CS-DL`, `CS-RSA`, `CS-DL2` **and** `CS-RSA2` **settings**: they are analogous to the previous settings, but here we use the CS signature scheme instead of the GHR one. As before the CS signature modulus was always chosen as a 1024-bit one. The other parameters are the same, as in the above cases.

Table 4.3 reports the results of our experiments. The second column reports which scheme was used and which parameters. The next columns are the times in milliseconds elapsed for: key generation, off-line phase, on-line phase and verification. The last two columns report the size in bits of the temporary token that must be secretly stored between the off-line and the on-line steps, and the size of the final signature.

4.4 Analysis of the results

In this section we summarize what we learned from our experimental results.

EGM construction *vs.* ST construction. The use of TCR hashing in the EGM settings, results in experimental results which are comparable to the ST measurements. For example if we focus on the time to perform the on-line step (arguably the most important measure in off-line/on-line signatures), we see that for the ST construction this is minimized with a time of about 0.03 ms by using the Discrete Log based chameleon hashing (no matter if the GHR and CS signature is used in the off-line step, of course) Nevertheless the setting `GHR-OTS` reaches a comparable on-line signing time of 0.47 ms when istantiated with similar security levels (row 1). The drawback is a longer signature, though the difference is not huge: 2944 bits versus the 1184 bits of `GHR-DL`. It is possible to shrink the EGM signature size to 2144 using bigger blocks and applying Jakobsson's technique. The on-line signature time continues to be competitive (1.27 ms) at the cost of a bigger temporary storage (8304 bits). It is important to note that the hash chain traversals in the verification step do not enjoy the benefit of the Jakobsson's technique as the pebbles must be kept secret.

GHR *vs.* CS. The GHR signature scheme outperforms the CS signature scheme in almost all parameters: off-line and on-line signature time, and signature size. The CS scheme is faster only in verification time, as to be expected since the GHR must use a longer public exponent, because of the division-intractability assumptions.

Chameleon hashing: DL-based *vs.* RSA-based. The time required for the hash evaluation step is comparable in both the schemes but the DL-based one has a notable advantage in the collision finding step. This operation is fundamental in the off-line/on-line signature construction, so it is the optimal choice for the ST construction.

Use of TCR hashing. As we pointed out above the use of TCR hashing has a dramatic impact on the efficiency of the OTS schemes. The experiments also point out that TCR hashing improves also the Shamir-Tauman approach, as it reduces the size of some of the exponents used in the exponentiations. A more pronounced improvement is obtained when using the RSA-based chameleon hashing: as in this construction the use of TCR hashing reduces the size of two exponents, rather than one as in the Discrete Log based one.

5 Conclusions

This paper presents some theoretical results about off-line/on-line digital signatures. We showed that the Shamir-Tauman approach of composing a regular signature with a chameleon hashing is conceptually just a different instantiation of the generic EGM paradigm that composes regular signatures with one-time signatures. We did this by proving that the EGM paradigm requires weaker security properties from its components and then showing that such properties

are satisfied by chameleon hash functions. We also showed that some type of chameleon hash functions can be used as full-fledged one-time signatures. We performed extensive implementation results to see what approach is preferable. Surprisingly we found that for appropriate choices of security parameters the ST and EGM approaches are comparable. Our experiments also showed that the Gennaro-Halevi-Rabin signature scheme is preferable to the Cramer-Shoup one on all respects except verification time.

Acknowledgments We thank the anonymous reviewers for their useful comments.

References

1. N. Barić and B. Pfitzmann, Collision-free Accumulators and Fail-stop Signature Schemes without Trees, *Advances in Cryptology – proceedings of EUROCRYPT '97, LNCS 1233*, Springer-Verlag, pp. 480–494, 1997.
2. M. Bellare and S. Micali, How To Sign Given Any Trapdoor Function, *Proceedings of STOC 88*, ACM, pp. 32–42, 1988.
3. M. Bellare and P. Rogaway, Random Oracles Are Practical: a Paradigm for Designing Efficient Protocols, *proceedings of* 1^{st} *ACM Conference on Computer and Communications Security (CCS 1993)*, ACM Press, pp. 62–73, 1993.
4. J.F. Boyar, S. A. Kurtz and M.W. Krentel, A Discrete Logarithm Implementation of Perfect Zero-Knowledge Blobs, *Journal of Cryptology*, vol. 2, n. 2, pp. 63–76, 1990.
5. G. Brassard, D. Chaum and C. Crépeau, Minimum disclosure proofs of knowledge, *Journal of Computer and System Sciences,* vol. 37, n. 2, pp. 156–189, Oct. 1988.
6. E. Bresson, D. Catalano and R. Gennaro, Improved On-Line/Off-Line Threshold Signatures, *Proceedings of Public Key Cryptography '07, LNCS 4450*, Springer-Verlag, pp. 217–232, 2007.
7. J. Coron and D. Naccache, Security analysis of the Gennaro-Halevi-Rabin Signature Scheme, *Advances in Cryptology – proceedings of EUROCRYPT '99, LNCS 1807*, Springer Verlag, pp. 91–101, 1999.
8. R. Cramer and I. Damgard, New Generation of Secure and Practical RSA-based Signatures, *Advances in Cryptology – proceedings of CRYPTO '96, LNCS 1109*, Springer-Verlag, pp. 173–185, 1996.
9. D. Eastlake and P. Jones, US Secure Hash Algorithm 1 (SHA1), RFC, RFC Editor.
10. R. Cramer and V. Shoup, Signature Scheme based on the Strong RSA Assumption, *Proceedings of* 6^{th} *ACM Conference on Computer and Communications Security (CCS 1999)*, ACM Press, pp. 46–51, 1999.
11. T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory* vol. 31, issue 4, pp. 469–472, 1985.
12. S. Even, O. Goldreich and S. Micali, On-Line/Off-Line Digital Signatures, *Journal of Cryptology*, vol. 9, n. 1, pp. 35–67, Springer, 1996.
13. A. Fiat and A. Shamir, How to Prove Yourself: Practical Solutions of Identification and Signature Problems, *Advances in Cryptology – proceedings of CRYPTO '86, LNCS 263*, Springer-Verlag, pp. 187–194, 1976.
14. R. Gennaro, S. Halevi and T. Rabin, Secure Hash-and-Sign Signatures without the Random Oracle, *Advances in Cryptology – proceedings of EUROCRYPT '99, LNCS 1592*, Springer-Verlag, pp. 123–139, 1999.

15. S. Goldwasser, S. Micali and R.L. Rivest, A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks, *SIAM Journal on Computing*, vol. 17, n. 2, pp. 281–308, 1988.
16. S. Halevi and H. Krawczyk. Strengthening Digital Signatures Via Randomized Hashing. *Advances in Cryptology – proceedings of CRYPTO 2006, LNCS 4117*, Springer-Verlag, pp. 41–59, 2006.
17. M. Jakobsson, Fractal Hash Sequence Representation and Traversal, *Proceedings of IEEE International Symposium on Information Theory, 2002, ISIT '02*, pp. 437, 2002.
18. H. Krawczyk and T. Rabin, Chameleon Hashing and Signatures, *Proceedings of Network and Distributed Systems Security Symposium (NDSS) 2000*, Internet Society, pp. 143–154, 2000.
19. K. Kurosawa and K. Schmidt-Samoa, New Online/Offline Signature Schemes Without Random Oracles, *Proceedings of Public Key Cryptography 2006, LNCS 3958*, Springer-Verlag, pp. 330–346, 2006.
20. L. Lamport, Constructing digital signatures from a one-way function, *Technical Report SRI-CSL-98, SRI International Computer Science Laboratory*, Oct. 1979.
21. R. C. Merkle, A digital signature based on a conventional encryption function, *Advances in Cryptology – proceedings of CRYPTO'87, LNCS 293*, Springer-Verlag, pp. 369–378, 1987.
22. M. Naor and M. Yung, Universal One-Way Hash Functions and Their Cryptographic Application, *Proceedings of STOC 89*, ACM, pp. 33–43, 1989.
23. D. Pointcheval and J. Stern, Security Arguments for Digital Signatures and Blind Signatures, *Journal of Cryptology*, vol. 13, n. 3, pp. 361–396, 2000.
24. M. O. Rabin, Digital Signatures, *Foundations of Secure Computation, R. A. DeMillo et al. (eds.)*, Academic Press, pp. 155–168, 1978.
25. R. Rivest, A. Shamir and L. Adelman, A Method for Obtaining Digital Signature and Public Key Cryptosystems, *Communications of the ACM*, vol. 21, n. 2, pp. 120–126, 1978.
26. J. Rompel, One-Way Functions Are Necessary and Sufficient for Secure Signatures, *Proceedings of STOC 1990*, pp. 387–394, 1990.
27. A. Shamir and Y. Tauman, Improved On-line/Off-line Signature Schemes, *Advances in Cryptology – proceedings of CRYPTO'01, LNCS 2139*, Springer-Verlag, pp.355–367, 2001.
28. C.P. Schnorr, Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, vol. 4, n. 3, pp. 161–174, 1991.
29. OpenSSL Project, `http://www.openssl.org`.
30. National Institute for Standards and Technology, Digital Signature Standard (DSS), *Technical Report 169*, August 30 1991.

A On one-time signatures

In Section 2.3 we presented two one-time signature schemes: Lamport's and Even *et al.*'s. The former is faster, but produces long signatures and keys. The latter allows for an efficiency trade-off between the signature/key sizes and time required to generate and to verify a signature tag.

Security. Lamport's scheme is proved secure under the assumption that one way functions exist. Even *et al.* solution relies on a seemingly stronger assumption:

Definition 6 (Quasi-Inverting). *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. Given an image, y, the task of* quasi-inverting *f on y is to find an x and an $i = poly(|y|)$ so that $f^{i+1}(x) = f^i(y)$. (For $i = 0$, the standard notion of inverting is regained.)*

Concrete security analysis. Here we focus on the security of the two one-time signature schemes presented in Section 2.3. In particular we analyze the efficiency (in terms of signature/key length) of Even *et al.*'s scheme with respect to Lamport's one, under the additional requirement that the two schemes should achieve the same security level.

Let $f : \{0,1\}^\ell \to \{0,1\}^\ell$ be a one-way function. In both schemes we assume to sign messages of length m. In the scheme of Even *et al.*, t represents the block length.

Let $\mathcal{A}$ be an adversary that breaks Lamport's one-time signature scheme with probability ϵ. It is possible to prove that this leads to an adversary $\mathcal{B}$ that inverts f with probability $\frac{\epsilon}{2m}$.
Similarly, if $\mathcal{A}'$ is an adversary that breaks Even *et al.* scheme with probability ϵ', this leads to an adversary $\mathcal{B}'$ that quasi-inverts f with probability $\frac{\epsilon'}{(m/t)2^{t+1}}$ (see [12], for details).

In what follows, we restrict to the case where f is a one way permutation (so that quasi-inverting f is equivalent to inverting f). We assume that no adversary can invert f with probability better than $1/2^\ell$. For the case of Lamport's scheme, this leads to $\frac{\epsilon}{2m} = \frac{1}{2^\ell}$ which means that one cannot forge signatures with probability better than $\epsilon = \frac{2m}{2^\ell}$. Similarly for Even *et al.*'s scheme we have that $\frac{\epsilon'}{(m/t)2^{t+1}} = \frac{1}{2^{\ell'}}$, implies a security for the signature scheme which is $\epsilon' = \frac{m2^{t+1-\ell'}}{t}$. Thus, in order for the two schemes to achieve the same security level, it has to be the case that $\epsilon' = \epsilon$, which means $\frac{2m}{2^\ell} = \frac{m2^{t+1-\ell'}}{t}$.

Thus, to achieve the same security level, for the two schemes, one has to consider a larger security parameter for the Even *et al.* scheme.

$$\ell' = \ell + t - log(t) \tag{1}$$

Signature length. In Lamport's scheme signatures have length $d = m\ell$. In Even *et al.*'s, on the other hand, the signature length is $d' = ((m/t)+1)\ell'$. From Equation (1) we get:

$$d' = \frac{m\ell}{t} + m + \ell + t - (\frac{m}{t} + 1)log(t) \tag{2}$$

Now, we want to establish for which choice of t we have $d' < d$. That is, for which choice of t Even *et al.* signatures are shorter than Lamport's ones.
From $\frac{m\ell}{t} + m + \ell + t - (\frac{m}{t} + 1)log(t) < m\ell$ one easily derives that if $m, \ell > 2$ then $t > 1$ is the required condition.

Experimental results. The relation among the variables involved in Equation (2) is analyzed through the tabulation of realistic values. We fix the security

parameter for the Lamport's scheme $\ell = 80$ and we assume to deal with messages of $m = 2\ell$ bits length. For different values of t we determine the corresponding values for the Even *et al.*'s parameters ℓ', d' using the above relations. All these values are reported in Table A; the signature length gain $(d - d')$ obtained using the EGM scheme instead of the Lamport's one is emphasized too (a negative value means that the use of the EGM construction is self-defeating).

We observe that the EGM solution is a winning solution for each real cases: the necessary augment of the security parameter ℓ' is minimal.

B Hard problems

B.1 The Discrete logarithm problem

Let $\mathcal{G}$ be an algorithm that takes in input a security parameter ℓ and outputs an integer q (such that $|q| = k$) and a generator g of a cyclic group G of order q. Let $\mathcal{A}$ be a an algorithm that outputs an integer in $\mathbb{Z}_q$. Let we consider the following experiment:

Experiment $\mathbf{Exp}^{dl}_{\mathcal{G}}(\mathcal{A}, \ell)$
$(q, g) \leftarrow \mathcal{G}(1^{\ell})$
$x \stackrel{\$}{\leftarrow} \mathbb{Z}_q; X \leftarrow g^x$
$\bar{x} \leftarrow \mathcal{A}(X)$
If $g^{\bar{x}} = X$ then return 1 else return 0.

The *dl*-advantage of $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{dl}_{\mathcal{G},\mathcal{A}}(\ell) = Pr[Exp^{dl}_{\mathcal{G}}(\mathcal{A}, \ell) = 1]$$

Definition 7. *The discrete logarithm problem is said to be* hard *in G if the dl-advantage of any polynomial time adversary is negligible in ℓ.*

B.2 The RSA and Strong RSA Problems

Let N be the product of two primes, $N = pq$. With $\phi(N)$ we denote the Euler function of N, i.e. $\phi(N) = (p-1)(q-1)$. With Z^*_N we denote the set of integers between 0 and $N-1$ and relatively prime to N. Let e be an integer relatively prime to $\phi(N)$. The RSA Assumption [25] states that it infeasible to compute e-roots in Z^*_N. I.e. given a random element $s \in_R Z^*_N$ it is hard to find x such that $x^e = s \bmod N$. The Strong RSA Assumption (introduced in [1]) states that given a random element s in Z^*_N it is hard to find $x, e \neq 1$ such that $x^e = s \bmod N$. The assumption differs from the traditional RSA assumption in that we allow the adversary to freely choose the exponent e for which she will be able to compute e-roots.

C Proofs

C.1 Proof of theorem 4

For the sake of contradiction assume the statement does not hold. This means that there exists an adversary F that breaks the oblivious security of the proposed one time signature. We show how to build an efficient B that, uses F to find collisions in the chameleon hash function. B goes as follows. On input the public key material pk, of a chameleon hash function, it waits F to make its unique signature request. When F asks for a message m to be signed, B chooses a random nonce r and computes $c = \mathsf{C}_{pk}(m, r)$. Next, it publishes the public key (pk, c) for the oblivious one time signature scheme F is supposed to attack, and hands F the signature (m, r) for m. If F manages to produce a valid forgery (m', s) for a message $m \neq m'$, B outputs (m, m', r, s) as the required collision.

C.2 Proof of theorem 5

We prove the theorem by *reductio ad absurdum*. We assume that the proposed construction is not a secure one time signature scheme. This means that there exists an efficient forger F that breaks the (one time) existential unforgeability of the signature scheme and we show how to build an efficient algorithm B, out of F, that breaks the collision resistance of the underlying double trapdoor chameleon hash function (DTCH, for short). B works as follows. On input a security parameter 1^ℓ it chooses a random bit i and runs the algorithm $\mathsf{TCKG}(1^\ell, i)$, thus obtaining a couple (pk, tk). Next, it chooses a random message a in the message space, a random nonce s and sets $c = \mathsf{C}_{pk}(a, s)$. B, outputs (pk, c) as the public key for a one time signature scheme. Notice that, such a public key, is distributed as it should, because of the fact that the public keys produced by TCKG is perfectly indistinguishable, with respect to those generated by CKG.

Now, whenever F asks a signature query on some message m (which is different from a with overwhelming probability), B uses its knowledge of (one) trapdoor to generate a random nonce r such that $c = \mathsf{C}_{pk}(m, r)$. Moreover, being such an r generated using one a valid trapdoor, it is correctly distributed. If, at some point, F outputs a valid forgery m', r' (with $r \neq r'$), B uses this forgery to break the collision resistance of the hash function as follows. First, notice that, by property 2 of collision resistance for double trapdoor chameleon hashing, there exists an efficient extractor A, that receiving in input a valid collision outputs one of the two trapdoors. B invokes such an algorithm, on input (m, m', r, r') and obtains back a trapdoor tk_j, which is different from tk_i with probability $1/2$. This is because, the collision produced by F is independent from the trapdoor held by B in a strong information theoretic sense (simply because, the distribution of collision property of DTCHs imposes that the distribution of the openings is the uniform one, no matter what trapdoor one uses).

D Hash functions

Fully Collision Resistant (FCR) hash functions: Consider a family $\mathcal{H} = \{h^{\mathsf{fcr}}(\cdot)\}$ with arbitrary input and ℓ bit output. We say that $\mathcal{H}$ is *fully collision resistant* if for a random element of the family $h^{\mathsf{fcr}}(\cdot) \in_R \mathcal{H}$ it is hard to find $x \neq y$ such that $h^{\mathsf{fcr}}(x) = h^{\mathsf{fcr}}(y)$.

Target Collision Resistant (TCR) hash functions[6]: Consider a family $\mathcal{H} = \{h^{\mathsf{tcr}}(k, \cdot)\}_k$ (we are making explicit the fact that an element of the family is parametrized by a key k) with arbitrary input and ℓ bit output. We say that $\mathcal{H}$ is *target collision resistant* if it is hard for the attacker to win the following game:

1. the attacker choose a message x;
2. a random key k is chosen;
3. the attacker outputs $y \neq x$ such that $h^{\mathsf{tcr}}(k, x) = h^{\mathsf{tcr}}(k, y)$.

E Examples of chameleon hashing

- **Discrete Log based:** This construction is from [4]. Let G be a group of prime order q where membership test and multiplication can be performed efficiently, and in which the discrete logarithm is hard.
  - **Key setup:** choose g at random in G and compute $h = g^x \bmod p$, with x chosen at random in $\mathbb{Z}_q$; the public key is $pk = (g, h)$, the secret key is x;
  - **Function evaluation:** given a message m and randomness r in $\mathbb{Z}_q$, the function is computed as $\mathsf{C}_{pk}(m, r) = g^m h^r$;
  - **Chameleon property:** given a commitment $c = g^m h^r$, the message m, the input randomness r, the trapdoor key x and a different message $m' \neq m$, we have that $m + xr = m' + xr' \bmod q$, so we can compute $r' = r + (m - m')x^{-1} \bmod q$.

  As it can be seen, the collision-finding algorithm requires the computation of a single multiplication (once one stores directly the value $x^{-1} \bmod q$).
  A typical implementation of the group G is to consider a subgroup of order q in Z_p^* where p, q are primes such that $q|(p-1)$. Reasonable security parameters are $|p| = 1024$ and $|q| = 160$.
- **RSA based:** This construction is from [8, 10]. Let N be the product of two large primes p, q (reasonable security parameter is $|p| = |q| = 512$); we consider the group $\mathbb{Z}_N$.
  - **Key setup:** let e be a prime number relatively prime to $\phi(N) = (p-1)(q-1)$, and s a random element of $\mathbb{Z}_N^*$. Compute $d = e^{-1} \bmod \phi(N)$; the public key is $pk = (N, s, e)$, the secret key is $\sigma = s^d \bmod N$;

[6] They are also known as universal one-way or second preimage collision resistant hash functions

- **Function evaluation:** given a message $m \in [1..e-1]$ and randomness r in $\mathbb{Z}_N^*$, the function is computed as $\mathsf{C}_{pk}(m,r) = s^m r^e \bmod N$;
- **Chameleon property:** given a commitment $c = s^m r^e$, the message m, the input randomness r, the trapdoor key σ and a different message $m' \neq m$, we have that $s^m r^e = s^{m'} r'^e \bmod N$, so we can compute $r' = r\sigma^{m-m'} \bmod N$.

F A discrete log-based double trapdoor commitment scheme

Here we briefly recall the double trapdoor commitment scheme given in [6] and then we discuss further applications of such a scheme.

Key Generation. Consider a cyclic group G of prime order q (with $|q| = \ell$ the security parameter) like before. Next, denoting with g a generator of G, choose two random values $x, y \in \mathbb{Z}_q$ and sets $h_1 = g^x$ and $h_2 = g^y$. The public key is (G, q, g, h_1, h_2) the private key is (x, y).

The Commitment Function. To commit to a message $m \in \mathbb{Z}_q$, we use two random values $r, s \in_R \mathbb{Z}_q^*$ and set

$$\mathsf{C}(m, r, s) = g^m h_1^r h_2^s$$

Theorem 6. *Under the assumption that computing discrete logarithms is hard, the above function* C *is a double trapdoor commitment scheme.*

Proof. We prove this theorem by showing that the three main properties of double trapdoor chameleon hash functions are satisfied.

Distribution of keys. Here we show the details of the TCKG algorithm. On input 1^ℓ and a bit i, it chooses two random generators $g, h_{i\oplus 1} \in G$, a random $tk_i \in \mathbb{Z}_q^*$ and sets, $h_i = g^{tk_i}$. The public key is set as (G, q, g, h_1, h_2) the trapdoor is tk_i. It is trivial to verify that all the required properties are satisfied.

Collision resistance. We prove this by contradiction. We assume there exists an adversary $\mathcal{A}$ that can find a collision in the proposed double trapdoor commitment scheme with non-negligible probability ϵ. Then we show how to build a simulator $\mathcal{B}$ that can solve the Discrete Logarithm (DLog) problem with non-negligible probability at least $\epsilon/6$. $\mathcal{A}$ finds a collision if, given the public key pk, it outputs two triples $(m, r, s), (m', r', s')$ with $m \neq m'$ such that $\mathsf{C}_{pk}(m, r, s) = \mathsf{C}_{pk}(m', r', s')$. We observe that at least one of the following conditions must hold: (1) $r \neq r'$ or (2) $s \neq s'$. We can distinguish between three types of collisions:

Type I $m \neq m', r \neq r', s \neq s'$
Type II $m \neq m', r = r', s \neq s'$
Type III $m \neq m', r \neq r', s = s'$

Thus $\mathcal{A}$ outputs a collision of either type I, type II or type III with probability at least $\epsilon/3$. Now we describe a simulator $\mathcal{B}$ that uses such collisions to solve the DLog problem.
In the first phase $\mathcal{B}$ receives in input two primes p, q such that $q|p-1$, a generator g of a cyclic subgroup G of $\mathbb{Z}_p^*$ of order q and an element $X \in G$. The aim of $\mathcal{B}$ is to output $x \in \mathbb{Z}_q^*$ such that $g^x = X$.
$\mathcal{B}$ has to construct the public key for the double trapdoor commitment scheme. First it flips a binary coin β. If $\beta = 0$ $\mathcal{B}$ bets on the fact that $\mathcal{A}$ will provide a collision of type I or III (where condition 1 holds true). Otherwise if $\beta = 1$ it bets on the fact that the received collision is of type I or II (it satisfies condition 2). $\mathcal{B}$ chooses random $y \xleftarrow{\$} \mathbb{Z}_q^*$. If $\beta = 0$ it sets $h_1 = X, h_2 = g^y$, otherwise it sets $h_1 = g^y, h_2 = X$. It gives $PK = (G, q, g, h_1, h_2)$ to $\mathcal{A}$. Then $\mathcal{A}$ produces a collision $(m, r, s), (m', r', s')$. Now we distinguish between the three types of collisions described above.

TYPE I COLLISION. In this case $\mathcal{B}$ can solve the DLog problem with non-negligible probability $\epsilon/3$. Indeed if $\beta = 0$ $\mathcal{B}$ outputs $x = \frac{m'-m+y(s'-s)}{r-r'} \bmod q$ as the discrete logarithm of X. Otherwise if $\beta = 1$ $\mathcal{B}$ outputs $x = \frac{m'-m+y(r'-r)}{s-s'} \bmod q$.

TYPE II COLLISION. In this case if $\beta = 0$ $\mathcal{B}$ loses its initial bet and fails. Otherwise if $\beta = 1$ it computes $x = \frac{m'-m}{s-s'} \bmod q$. Thus with probability at least $\frac{\epsilon}{3}\frac{1}{2}$ $\mathcal{B}$ solves the DLog problem.

TYPE III COLLISION. This case is similar to the previous. If $\beta = 1$ $\mathcal{B}$ loses its initial bet and fails. Otherwise if $\beta = 0$ it computes $x = \frac{m'-m}{r-r'} \bmod q$. Thus with probability at least $\frac{\epsilon}{3}\frac{1}{2}$ the simulator can find the discrete logarithm of X.

Distributions of Collisions. We consider the two distributions:

$$\{m, m', r, s \leftarrow \mathbb{Z}_q^*, : \mathsf{Coll}(tk_1, m, m', r, s)\}$$

$$\{m, m', r, s \leftarrow \mathbb{Z}_q^*, : \mathsf{Coll}(tk_2, m, m', r, s)\}$$

In the first distribution Coll outputs a value (r', s) such that $r' = \frac{m-m'}{x} + r \bmod q$. We observe that s is uniformly distributed in $\mathbb{Z}_q^*$ and if r is uniformly distributed in $\mathbb{Z}_q^*$, then also r' is uniformly distributed in $\mathbb{Z}_q^*$. In the second distribution Coll outputs a pair (r, s') such that $s' = \frac{m-m'}{y} + s \bmod q$. If s is uniform in $\mathbb{Z}_q^*$, then also s' is uniform in $\mathbb{Z}_q^*$. Thus, both the two distributions are perfectly indistinguishable from uniform in $\mathbb{Z}_q^*$.

Table 1. Results of the experimental tests

N.	Settings	KeyGen time	Off-line sign time	On-line sign time	Verif. time	Temp. data size	Sign. size
1	GHR-OTS $ots\_\{l,t,p\} = 80, 4, 1$	5901.603	12.841	0.469	7.986	2944	2944
2	GHR-OTS $ots\_\{l,t,p\} = 80, 8, 1$	5895.344	17.813	2.860	10.353	2144	2144
3	GHR-OTS $ots\_\{l,t,p\} = 80, 8, 8$	5512.062	17.832	1.266	10.383	8304	2144
4	GHR-OTS $ots\_\{l,t,p\} = 80, 10, 1$	6202.157	30.351	8.866	16.278	1984	1984
5	GHR-OTS $ots\_\{l,t,p\} = 80, 10, 5$	4929.411	30.405	7.057	16.306	4864	1984
6	GHR-OTS $ots\_\{l,t,p\} = 80, 10, 10$	5076.828	30.381	3.791	16.270	8464	1984
7	GHR-OTS $ots\_\{l,t,p\} = 96, 4, 1$	4893.636	13.009	0.532	8.027	3680	3680
8	GHR-OTS $ots\_\{l,t,p\} = 96, 8, 1$	6123.729	19.037	3.431	10.983	2528	2528
9	GHR-OTS $ots\_\{l,t,p\} = 96, 8, 8$	5386.401	19.032	1.509	11.153	11264	2528
10	GHR-OTS $ots\_\{l,t,p\} = 96, 12, 1$	6318.599	85.410	35.432	42.556	2144	2144
11	GHR-OTS $ots\_\{l,t,p\} = 96, 12, 6$	5076.328	85.484	28.273	42.613	6464	2144
12	GHR-OTS $ots\_\{l,t,p\} = 96, 12, 12$	6109.271	85.495	14.990	42.548	11648	2144
13	GHR-OTS $ots\_\{l,t,p\} = 112, 4, 1$	5687.215	13.173	0.594	8.112	4544	4544
14	GHR-OTS $ots\_\{l,t,p\} = 112, 8, 1$	4670.010	20.148	3.971	11.498	2976	2976
15	GHR-OTS $ots\_\{l,t,p\} = 112, 8, 8$	4831.965	20.186	1.759	11.446	14736	2976
16	GHR-OTS $ots\_\{l,t,p\} = 112, 16, 8$	5204.409	1029.727	390.909	493.273	8464	2192
17	GHR-OTS $ots\_\{l,t,p\} = 112, 16, 16$	5951.515	1029.443	200.213	492.931	15632	2192
18	GHR-DL $tc\_\{p,q\}\_bits = 1024, 160$	7102.520	14.267	0.033	11.551	1184	1184
19	GHR-RSA $tc\_\{n,e\}\_bits = 1024, 160$	6037.322	14.305	11.864	11.624	2048	2048
20	GHR-DL2 $tc\_\{p,q\}\_bits = 1024, 160, tcr\_bits = 80$	5563.434	14.227	0.157	10.775	1264	1264
21	GHR-DL2 $tc\_\{p,q\}\_bits = 1024, 160, tcr\_bits = 96$	5989.609	14.291	0.157	11.050	1280	1280
22	GHR-RSA2 $tc\_\{n,e\}\_bits = 1024, 80, tcr\_bits = 80$	5479.807	13.779	11.977	9.805	2128	2128
23	GHR-RSA2 $tc\_\{n,e\}\_bits = 1024, 96, tcr\_bits = 96$	5758.465	13.957	11.972	10.206	2144	2144
24	CS-OTS $tcr\_bits = 80, ots\_\{l,t,p\} = 80, 4, 1$	4457.142	17.880	0.466	3.616	3105	3105
25	CS-OTS $tcr\_bits = 80, ots\_\{l,t,p\} = 80, 8, 1$	5167.754	22.902	2.857	5.970	2305	2305
26	CS-OTS $tcr\_bits = 80, ots\_\{l,t,p\} = 80, 8, 8$	6389.489	22.939	1.296	5.950	8465	2305
27	CS-OTS $tcr\_bits = 80, ots\_\{l,t,p\} = 80, 10, 1$	5076.968	35.298	8.847	11.886	2145	2145
28	CS-OTS $tcr\_bits = 80, ots\_\{l,t,p\} = 80, 10, 5$	5314.612	35.290	7.080	11.858	5025	2145
29	CS-OTS $tcr\_bits = 80, ots\_\{l,t,p\} = 80, 10, 10$	5313.972	35.375	3.747	11.872	8625	2145
29	CS-OTS $tcr\_bits = 96, ots\_\{l,t,p\} = 96, 4, 1$	7192.007	18.192	0.547	3.834	3841	3841
30	CS-OTS $tcr\_bits = 96, ots\_\{l,t,p\} = 96, 8, 1$	5670.858	24.185	3.425	6.719	2689	2689
31	CS-OTS $tcr\_bits = 96, ots\_\{l,t,p\} = 96, 8, 8$	5006.019	24.040	1.526	6.709	11425	2689
32	CS-OTS $tcr\_bits = 96, ots\_\{l,t,p\} = 96, 12, 1$	5466.489	90.280	35.339	38.353	2305	2305
33	CS-OTS $tcr\_bits = 96, ots\_\{l,t,p\} = 96, 12, 6$	5536.198	90.348	28.190	38.417	6625	2305
34	CS-OTS $tcr\_bits = 96, ots\_\{l,t,p\} = 96, 12, 12$	5340.548	91.437	14.571	38.392	11809	2305
35	CS-OTS $tcr\_bits = 112, ots\_\{l,t,p\} = 112, 4, 1$	4736.900	18.580	0.600	4.102	4705	4705
36	CS-OTS $tcr\_bits = 112, ots\_\{l,t,p\} = 112, 8, 1$	5894.944	25.416	3.975	7.524	3137	3137
37	CS-OTS $tcr\_bits = 112, ots\_\{l,t,p\} = 112, 8, 8$	5210.748	25.881	1.759	7.512	14897	3137
38	CS-OTS $tcr\_bits = 112, ots\_\{l,t,p\} = 112, 16, 8$	4789.552	1035.058	387.609	491.539	8625	2353
39	CS-OTS $tcr\_bits = 112, ots\_\{l,t,p\} = 112, 16, 16$	5795.599	1036.105	204.137	491.250	15793	2353
40	CS-DL $tcr\_bits = 96, tc\_\{p,q\}\_bits = 1024, 160$	7551.952	19.415	0.034	7.349	1345	1345
41	CS-RSA $tcr\_bits = 96, tc\_\{n,e\}\_bits = 1024, 160$	6150.725	19.539	11.875	7.366	2209	2209
42	CS-DL2 $tcr\_bits = 80, tc\_\{p,q\}\_bits = 1024, 160$	5758.045	19.186	0.150	6.354	1425	1425
43	CS-DL2 $tcr\_bits = 96, tc\_\{p,q\}\_bits = 1024, 160$	6768.931	19.265	0.150	6.696	1441	1441
44	CS-RSA2 $tcr\_bits = 80, tc\_\{n,e\}\_bits = 1024, 80$	5732.029	18.310	11.965	5.421	2289	2289
45	CS-RSA2 $tcr\_bits = 96, tc\_\{n,e\}\_bits = 1024, 96$	6076.316	18.586	11.967	5.987	2305	2305

t	ℓ'	d'	d	gain $(d - d')$
1	81	13041	12800	-241
2	81	6561	12800	6239
3	81.4150	4423.6	12800	8376.4
4	82	3362	12800	9438
5	82.6781	2728.4	12800	10071.6

Table 2. Experimental results with parameters $\ell = 80, m = 2\ell$