

# Time Capsule Signature

Yevgeniy Dodis<sup>1</sup> and Dae Hyun Yum<sup>1,2</sup>

<sup>1</sup>Dept. of Computer Science, New York University, NY, USA

<sup>2</sup>Dept. of Electronic and Electrical Eng., POSTECH, Pohang, Korea  
{dodis, dhyum}@cs.nyu.edu

**Abstract.** We introduce a new cryptographic problem called *time capsule signature*. Time capsule signature is a ‘future signature’ that becomes valid from a specific future time  $t$ , when a trusted third party (called *Time Server*) publishes some trapdoor information associated with the time  $t$ . In addition, time capsule signature should satisfy the following properties:

- (1) If the signer wants, she can make her time capsule signature effective before the pre-defined time  $t$ .
- (2) The recipient of ‘future signature’ can verify right away that the signature will become valid no later than at time  $t$ .
- (3) Time Server need not contact any user at any time, and in fact does not need to know anything about the PKI employed by the users.
- (4) Signatures completed by the signer before time  $t$  are indistinguishable from the ones completed using the Time Server at time  $t$ .

We provide the rigorous definition of time capsule signature and the generic construction based on another new primitive of independent interest, which we call *identity-based trapdoor hard-to-invert relation* (ID-THIR). We also show an efficient construction of ID-THIRs (and, hence, time capsule signatures) in the random oracle model, and a less efficient construction in the standard model.

If the time  $t$  is replaced by a specific event, the concept of time capsule signature can be generalized to *event capsule signature*.

## 1 Introduction

### 1.1 Time Capsule Signature

In an ordinary signature scheme, the validity of a signature value is determined at the point of signature generation and never changes (unless the signer’s public key is revoked). Users cannot generate the so-called ‘future signature’ which is not currently valid but becomes valid from a future time  $t$ . A naive way to achieve this is signing with a statement such as ‘the signature of message  $m$  becomes valid from time  $t$ .’ This, however, has several drawbacks. First, and least serious, the verifier is required to be aware of the current time. When time is generalized to arbitrary events (i.e., ‘the signature of  $m$  becomes valid if the event  $e$  happens’), this becomes even more problematic. More seriously, however, in the naive solution the signer herself loses control over the validity of the future

signature, i.e., even the real signer cannot make her signature valid before time  $t$ . This means that either the signer has to wait until time  $t$  — which could be undesirable in certain situations (e.g., if the borrower wants to quickly repay her debt before the actual due date to improve her credit history) — or the signer can issue a new, independent signature of  $m$  before time  $t$ . The latter solution, however, can also be undesirable in certain situations. First, in case the message  $m$  carries some monetary value, the signer needs to make sure that no “double spending” occurs (i.e., to somehow revoke the original signature, so that it does not become valid at time  $t$ ). Second, the verifier now knows whether the message  $m$  was signed in the ‘future’ or ‘regular’ way, which seems to be unnecessary in most situations.

Therefore, we would like a solution where the signer can issue a future signature so that at least the following properties are satisfied:

- (1) At the time of creation, the recipient is sure that the signature will become valid by time  $t$ , even if the signer refuses to cooperate after she produces the future signature.
- (2) The legal signer can make the future signature valid at any time after the initial creation.
- (3) Irrespective of whether the signer validated the signature earlier, or it became “automatically valid” at time  $t$ , the resulting signatures are indistinguishable. In other words, the verifier after time  $t$  cannot tell the lower-level details of how the signature became valid.

Of course, it is also crucial to specify the mechanism under which the signature can be “automatically” completed at time  $t$  (which we call “hatching” as opposed to “pre-hatching” which can be done by the signer at any time). As we remarked, we cannot just make it valid at time  $t$ , since this requires the verifier to “know” the current time, and, more importantly, will not make the hatching indistinguishable from pre-hatching. Another option would be to use some “time-release” primitive, such as timed signature [9], where the verifier knows that by investing some intensive computation, he can complete a future signature within some pre-specified time, even if the signer refuses to cooperate. However, this option is only approximate (i.e., the verifier can open the signature roughly by time  $t$  depending on its computational capabilities), and, more importantly, forces the verifier to invest a considerable computational effort the moment the future signature was generated.

Finally, we can follow the approach of optimistic fair exchange protocols [1, 2, 14], where an “off-line” arbitrator (a trusted third party) can complete the signer’s partial signature into the full signature, shall the signer refuse to cooperate. In particular, so called verifiably committed signatures of [14] seem to be ideally suited for this task. The main drawback of this solution is that the arbitrator, although only involved if the signer refuses to cooperate (say, before time  $t$ ), has to be involved in a message-by-message manner. Thus, in our scenario of future signatures, — where by default the signer will not pre-hatch her signature, — the arbitrator would have to literally complete almost every

signature separately. The latter, of course, makes the arbitrator quite “on-line” and whole paradigm very unattractive for our application.

Instead, we introduce *time capsule signatures*, where the arbitrator (which we call the *Time Server*)

- (1) Does not ever need to contact users, know about the particular format of their signature, or be involved in any signature resolution protocols.
- (2) At the beginning of time period  $t$ , outputs a single message  $Z_t$ , which automatically allows anybody to complete any future signature set to hatch at time  $t$ .

More specifically, time capsule signature is a ‘future signature’ that becomes valid from a specific future time  $t$ , when a trusted third party (called *Time Server*) publishes some trapdoor information associated with the time  $t$ . When Alice gives Bob her time capsule signature  $\sigma'_t$  for a future time  $t$ , Bob can verify that Alice’s time capsule signature will become valid from the time  $t$ . In addition, if Alice wishes, she can make her time capsule signature effective before the pre-defined time  $t$ . The assumption on Time Server is minimal, in that Time Server only publishes some information at the beginning of each time period and need not contact any user at any time. Finally, the concept of time capsule signature can be generalized to *event capsule signature*, where *Event Server* issues the notification information of specific events. The event capsule signature becomes valid if a specific event happens or the signer makes valid before the event occurs.

## 1.2 Our Contribution

We provide the rigorous definition of time (or event) capsule signature and the generic construction based on another new primitive of independent interest, which we call *identity-based trapdoor hard-to-invert relation* (ID-THIR). Intuitively, ID-THIR is given by a family  $\mathcal{R}$  of relations  $R_{id}$ , where (1) it is easy to sample a random pair  $(c, d) \in R_{id}$  and verify if the pair  $(c, d)$  belongs to  $R_{id}$ ; (2) for each identity  $id$ , there exists a trapdoor  $\mathbf{td}_{id}$ , which allows one to compute a random  $d$  corresponding (w.r.t.  $R_{id}$ ) to any given  $c$  (The trapdoor  $\mathbf{td}_{id}$ ’s can be efficiently computed from a single “master key”  $\mathbf{mtd}_{\mathcal{R}}$ ); (3) without the trapdoor  $\mathbf{td}_{id}$ , it is hard to find a matching  $d$  corresponding (w.r.t.  $R_{id}$ ) to a randomly sampled  $c$ , even if one knows many trapdoors corresponding to identities  $id' \neq id$ .

Our construction of time (or event) capsule signatures from ID-THIR is very natural: the future signature of  $m$  is  $(\mathbf{Sig}(m||c||t), c)$ , while the full hatched signature is  $(\mathbf{Sig}(m||c||t), c, d)$ , where ‘ $\mathbf{Sig}$ ’ is any ordinary signature, ‘ $m$ ’ a message, ‘ $||$ ’ the concatenation and  $(c, d)$  a random lock/proof pair corresponding to the “identity” equal to time  $t$  (or event  $e$ ). The legal signer would sample  $(c, d)$  and remember  $d$  for pre-hatching, while the Time (or Event) Server would periodically publish the trapdoors  $\mathbf{td}_t$  (or  $\mathbf{td}_e$ ) which would allow anyone to hatch the signature by computing the corresponding  $d$  from  $c$ . Moreover, hatching and pre-hatching would look the same, by the security properties of the ID-THIR scheme.

Finally, we give a generic construction of ID-THIR (and, therefore, time/event capsule signature). In the standard model, the construction is mainly of theoretical interest, as it relied on non-interactive witness indistinguishable proofs of knowledge. Nevertheless, it shows that our primitives exist if trapdoor one-way permutations exist. On a practical front, we give several very efficient implementations of ID-THIR in the random oracle model. Generically, we show that in the random oracle model one can construct our primitives from mere one-way functions. Concretely, we show very efficient instantiations based on RSA or discrete log in the Gap Diffie-Hellman groups [24, 8].

### 1.3 Related Work

As we pointed out, there are two main lines of work related to time capsule signature, depending on whether or not the trusted third party is involved.

The first approach, which is that of timed-release cryptography, is to ensure that the encryption, commitment or signature can be opened in a brute force way by solving some time-consuming, but computationally feasible problem. For example, Dwork and Naor [15] used such moderately hard functions in order to deter abuse of resources, such as spamming. Bellare and Goldwasser [3, 4] suggested “(verifiable) time capsules”<sup>1</sup> for key escrowing in order to deter widespread wiretapping. There, the main issue is the verification at escrow time that the right key will be recovered. Rivest, Shamir and Wagner [27] suggested “time-lock puzzle,” where the goal is to design “inherently sequential puzzles” which are resistant to parallel attacks. However, they did not address verifiability at escrow time. The latter was formally addressed by Boneh and Naor [9], who defined (verifiable) timed commitments. As one of their applications, they get an analog of our time capsule signature (termed “timed signature”), where the future signature can either be opened by the signer, or by the recipient — the latter if the recipient solves a moderately hard problem. More recent advances were made by [19, 20].

The second approach, based on the trusted third party, has two main flavors: optimistic fair exchange of digital signatures, and identity-based future encryption. In the former case, the server needs to resolve all individual signatures where the signer refused to validate the signature (say, by a given time  $t$ ).<sup>2</sup> Representative examples include [1, 2, 10, 14]. In contrast, in our model we insist that users do not communicate ever with the trusted server.

In the case of future encryption [7, 6, 25], the main problem addressed was that the sender wants to ensure that the message would remain hidden before the Time Server would publish the corresponding trapdoor. However, this is orthogonal to our model, where we want to “encrypt” a *signature* on a *public* message. Thus, we do not need to hide the message (and can even leak partial

<sup>1</sup> Which should not be confused with our time capsule signatures, which are totally different.

<sup>2</sup> In fact, in some of the solutions the clients additionally need to either register their keys with the server, or have an interactive resolution protocol.

information about the full signature, as long as the full signature is hidden). On the other hand, we have to resolve two crucial complications not present in the above scenario: (1) the future signature has to be verifiable right away, to ensure the recipient it will be successfully completed at time  $t$ ; (2) the sender can pre-hatch the signature in a manner indistinguishable from the regular hatching at time  $t$ . Not surprisingly, the solutions above all utilized some kind of identity-based encryption, and do not seem to be useful for our time-capsule signatures.

## 2 Primitives

### 2.1 $\Sigma$ -protocol

A  $\Sigma$ -protocol [13] is an efficient 3-round two-party protocol between the prover and the verifier on a common input  $x \in \mathcal{L}_R$ , where  $\mathcal{L}_R$  is a language for an NP relation  $R$ . Besides  $x$ , a valid NP-witness  $w$  for  $x$  is also given to the prover as a private input. The prover first sends a commitment message  $a$  to the receiver. After receiving the commitment message  $a$ , the verifier sends a challenge message  $b$  to the prover. Finally, the prover sends a response message  $z$  to the verifier who decides to output 1 (accept) or 0 (reject) based on the input  $x$  and the transcript  $\pi = \{a, b, z\}$ . The transcript  $\pi$  is valid if the verifier outputs 1 (accept). A binary  $\Sigma$ -protocol is a special case of  $\Sigma$ -protocol where the challenge message takes only a binary value (0 or 1).

A  $\Sigma$ -protocol should satisfy three properties: correctness, special soundness, and special (honest-verifier) zero-knowledge. Correctness property states that for all  $x \in \mathcal{L}_R$  and all valid witnesses  $w$  for  $x$ , if the prover and the verifier follow the protocol honestly, the verifier must output 1 (accept). Special soundness property says that there is an efficient extraction algorithm (called a knowledge extractor)  $\text{Ext}$  that on input  $x \in \mathcal{L}_R$  and two valid transcripts  $\pi_1, \pi_2$  with the same commitment message outputs  $z$  such that  $(x, z) \in R$ . Special zero-knowledge property says that there is an efficient simulation algorithm (called a simulator)  $\text{Sim}$  that on input  $x \in \mathcal{L}_R$  and any challenge message  $b$ , outputs a valid transcript  $\pi' = \{a', b, z'\}$ . Moreover, the distribution of  $(a', z')$  is computationally indistinguishable from the corresponding distribution on  $(a, z)$  produced by the prover knowing a valid witness  $w$  for  $x$  and the verifier. This is true even if the distinguisher knows the witness  $w$ .

A function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  is a one-way function, if there exists a polynomial time algorithm which computes  $f(x)$  correctly for all  $x$  and the following probability is negligible for all PPT (Probabilistic Polynomial Time) algorithm  $A$ :  $\Pr(f(x') = y \mid x \leftarrow \{0, 1\}^k; y = f(x); x' \leftarrow A(y, 1^k))$ . It is known that any language in NP has a  $\Sigma$ -protocol if one-way functions exist [21, 16]. Of course, specific languages can have much more efficient  $\Sigma$ -protocols. A  $\Sigma$ -protocol can also be transformed into a signature scheme by using the Fiat-Shamir heuristic [18]. To sign a message  $m$ , the legal signer produces a valid transcript  $\pi = \{a, b, z\}$  of the  $\Sigma$ -protocol, where  $b = H(a, m)$  and  $H(\cdot)$  is a cryptographic hash function modelled as a random function. The signature scheme obtained by applying the Fiat-Shamir heuristic to the  $\Sigma$ -protocol is secure in the random oracle model [5,

28]. It is also known that the Fiat-Shamir heuristic gives a non-interactive proof of knowledge in the random oracle model (i.e., the witness can be extracted by rewinding the adversary).

If there are two  $\Sigma$ -protocols, i.e.,  $\Sigma_1$  for  $R_1$  and  $\Sigma_2$  for  $R_2$ , we can construct another  $\Sigma$ -protocol  $\Sigma_{OR}$  (called OR-proof) [13] which allows the prover to show that given two inputs  $x_1, x_2$ , he knows  $w$  such that either  $(x_1, w) \in R_1$  or  $(x_2, w) \in R_2$  without revealing which is the case (called the witness indistinguishability property [17]). By applying the Fiat-Shamir heuristic to the OR-proof  $\Sigma_{OR}$ , we get a signature scheme (called OR-signature) secure in the random oracle model such that a valid signature can be generated by the signer who knows a valid witness  $w$  corresponding to either of the two inputs  $x_1, x_2$ . It is known that the Fiat-Shamir heuristic does not affect the witness indistinguishability property of the  $\Sigma$ -protocol.

## 2.2 Identity-Based Trapdoor Hard-to-Invert Relation

A (binary) relation  $R$  is a subset of  $\{0, 1\}^* \times \{0, 1\}^*$  and the language  $\mathcal{L}_R$  is the set of  $\alpha$ 's for which there exist  $\beta$  such that  $(\alpha, \beta) \in R$ , i.e.,  $\mathcal{L}_R = \{\alpha \mid \exists \beta [(\alpha, \beta) \in R]\}$ . We assume that (1) there is an efficient algorithm to decide whether  $\alpha \in \mathcal{L}_R$  or not, (2) if  $(\alpha, \beta) \in R$ , then the length of  $\beta$  is polynomially bounded in  $|\alpha|$ , and (3) there exists a short description  $D_R$  which specifies the relation  $R$ .

We also assume that the membership in  $f(X)$  can be efficiently determined for a (trapdoor) one-way function  $f : X \rightarrow Y$ .

**Definition 1.** An identity-based trapdoor hard-to-invert relation (ID-THIR) is a set of relations  $\mathcal{R} = \{R_{id} \mid id \in I_{\mathcal{R}}\}$ , where each relation  $R_{id}$  is trapdoor hard-to-invert relation (i.e., sampling a random lock/proof pair  $(c, d) \in R_{id}$  is easy but finding a proof for a given lock is difficult without knowing the trapdoor  $\text{td}_{id}$ ) and there is a master trapdoor  $\text{mtd}_{\mathcal{R}}$  for extracting the trapdoor  $\text{td}_{id}$  of each relation  $R_{id}$ . ID-THIR can also be specified by 5-tuple of PPT algorithms  $(\text{Gen}, \text{Sample}, \text{Check}, \text{Extract}, \text{Invert})$  such that:

- **Gen.** This algorithm is used to generate  $\mathcal{R} = \{R_{id} \mid id \in I_{\mathcal{R}}\}$ , where  $I_{\mathcal{R}}$  is a finite set of indices.  $\text{Gen}(1^k)$  returns  $D_{\mathcal{R}}$  (the description of  $\mathcal{R}$ ) and  $\text{mtd}_{\mathcal{R}}$  (the master trapdoor).
- **Sample.** This sampling algorithm takes  $(D_{\mathcal{R}}, id)$  as input and  $\text{Sample}_{D_{\mathcal{R}}}(id)$  returns a random lock/proof pair  $(c, d) \in R_{id}$ .
- **Check.** This algorithm is used to check the validity of the proof. If  $(c, d) \in R_{id}$ , then  $\text{Check}_{D_{\mathcal{R}}, id}(c, d)$  returns 1 (accept). Otherwise, it returns 0 (reject).
- **Extract.** This algorithm is used to extract the trapdoor of each relation by using  $\text{mtd}_{\mathcal{R}}$ .  $\text{Extract}_{\text{mtd}_{\mathcal{R}}}(id)$  returns the trapdoor  $\text{td}_{R_{id}}$  of the relation  $R_{id}$ .
- **Invert.** This algorithm is used to find a proof  $d$  for a given  $c \in \mathcal{L}_{R_{id}}$  by using the trapdoor  $\text{td}_{R_{id}}$ . If  $c \in \mathcal{L}_{R_{id}}$ , then  $\text{Invert}_{\text{td}_{R_{id}}}(c)$  returns a proof  $d$  such that  $(c, d) \in R_{id}$ .

Let  $(c, d) \leftarrow \text{Sample}_{D_{\mathcal{R}}}(id)$  and  $\tilde{d} \leftarrow \text{Invert}_{\text{td}_{R_{id}}}(c)$ . Correctness property states that  $\text{Check}_{D_{\mathcal{R}}, id}(c, d) = 1$  and  $\text{Check}_{D_{\mathcal{R}}, id}(c, \tilde{d}) = 1$ , and ambiguity property

states that  $(c, d)$  and  $(c, \tilde{d})$  are computationally indistinguishable, even if the distinguisher knows the master key  $\text{mtd}_{\mathcal{R}}$ . Let  $O_{\text{Extract}}$  be the oracle simulating the trapdoor extraction procedure  $\text{Extract}$  and  $\text{Query}(A, O_{\text{Extract}})$  the set of queries an algorithm  $A$  asked to  $O_{\text{Extract}}$ . One-wayness property states that the following probability is negligible for all PPT algorithm  $A = (A_1, A_2)$ :

$$\Pr[\text{Check}_{D_{\mathcal{R}}, id}(c, \hat{d}) = 1 \wedge id \notin \text{Query}(A, O_{\text{Extract}}) \mid (D_{\mathcal{R}}, \text{mtd}_{\mathcal{R}}) \leftarrow \text{Gen}(1^k); (id, h) \leftarrow A_1^{O_{\text{Extract}}}(D_{\mathcal{R}}); (c, d) \leftarrow \text{Sample}_{D_{\mathcal{R}}}(id); \hat{d} \leftarrow A_2^{O_{\text{Extract}}}(D_{\mathcal{R}}, c, h)]$$

Soundness property states that the following probability is negligible for all algorithm  $B$ :

$$\Pr[R_{id} \in \mathcal{R} \wedge c \in \mathcal{L}_{R_{id}} \wedge \text{Check}_{D_{\mathcal{R}}, id}(c, \tilde{d}) = 0 \mid (D_{\mathcal{R}}, \text{mtd}_{\mathcal{R}}) \leftarrow \text{Gen}(1^k); (c, id) \leftarrow B(D_{\mathcal{R}}); \text{td}_{R_{id}} \leftarrow \text{Extract}_{\text{mtd}_{\mathcal{R}}}(id); \tilde{d} \leftarrow \text{Invert}_{\text{td}_{R_{id}}}(c)]$$

If ID-THIR satisfies these four properties, we say that ID-THIR is secure.

**CONSTRUCTION.** Each trapdoor hard-to-invert relation  $R_{id}$  in ID-THIR  $\mathcal{R} = \{R_{id} \mid id \in I_{\mathcal{R}}\}$  looks like a trapdoor one-way function. However, there is an important difference: we can sample a random lock/proof pair  $(c, d) \in R_{id}$  but may not necessarily be able to compute a lock  $c$  for a given proof  $d$ . Therefore, we can show that a trapdoor one-way function implies a trapdoor hard-to-invert relation but cannot prove the reverse direction. While the concept of ID-THIR also seems very general, the construction is not trivial. For example, it is not obvious whether identity-based encryption (IBE) [32, 7] implies ID-THIR or not, since IBE does not automatically guarantee the ambiguity property of ID-THIR.<sup>3</sup> Now, we provide our general construction of ID-THIR.

**Theorem 1.** *If there is a one-way function, there exists a secure ID-THIR in the random oracle model.*

**Proof:** Assume that there is a one way function  $f : X \rightarrow Y$ . We can build a secure signature scheme (**Set**, **Sig**, **Ver**) from the one-way function  $f$ , since secure signatures exist if and only if one-way functions exist [29]. Let  $\Sigma_{\text{Sig}}$  be the  $\Sigma$ -protocol for the knowledge of a signature value  $\text{Sig}(m)$  on a common input  $m \in M$  and  $\Sigma_f$  for the knowledge of a pre-image of a common input  $f(x) \in f(X)$ . If we denote by  $\Sigma_{OR}$  the OR-proof for  $\Sigma_{\text{Sig}}$  or  $\Sigma_f$ , we can obtain an OR-signature scheme (**Set**<sup>OR</sup>, **Sig**<sup>OR</sup>, **Ver**<sup>OR</sup>) by applying the Fiat-Shamir heuristic to  $\Sigma_{OR}$ . The OR-signature is secure in the random oracle model and **Set**<sup>OR</sup> can be implicitly defined by  $\Sigma_{\text{Sig}}$  and  $\Sigma_f$ .

Now, we define the identity-based trapdoor hard-to-invert relation  $\mathcal{R}_{OR} = \{R_m \mid m \in M\}$  where  $R_m = \{(y, \pi) \mid y = f(x) \text{ for } x \in X, \pi \text{ is an OR-signature on } m \parallel f(x) \text{ for the knowledge of a pre-image of } f(x) \text{ or } \text{Sig}(m)\}$  and the algorithms

<sup>3</sup> The decryption algorithm of IBE does not necessarily recover the temporary random number used in the encryption algorithm. For example, see [7].

(Gen, Sample, Check, Extract, Invert) as follows; Gen chooses  $(pk, sk) \leftarrow \mathbf{Set}(1^k)$  and outputs  $D_{\mathcal{R}_{OR}} = pk, \text{mtd}_{\mathcal{R}_{OR}} = sk$  (technically,  $D_{\mathcal{R}_{OR}}$  should also contain the one-way function  $f$ ). We assume that message space  $M$  is known implicitly. On input  $id = m$ , Sample randomly chooses  $x \in X$  and generates an OR-signature  $\pi$  for the knowledge of a pre-image of  $f(x)$  or  $\mathbf{Sig}(m)$ . Sample outputs a lock/proof pair  $(c, d) = (f(x), \pi)$ . For a given  $(id, c, d) = (m, f(x), \pi)$ , Check verifies whether  $\pi$  is a valid OR-signature for a pre-image of  $f(x)$  or  $\mathbf{Sig}(m)$ . Extract takes as input  $id = m$  and outputs  $\text{td}_{R_m} = \mathbf{Sig}(m)$ . On input  $(id, c) = (m, f(x))$ , Invert knowing  $\text{td}_{R_m} = \mathbf{Sig}(m)$  generates an OR-signature  $\pi$  for the knowledge of a pre-image of  $f(x)$  or  $\mathbf{Sig}(m)$ .

Correctness property is obvious and ambiguity property results from the fact that the OR-proof  $\Sigma_{OR}$  is witness indistinguishable. Now, consider the one-wayness property. The attacker  $A$  against  $\mathcal{R}_{OR}$  gets  $D_{\mathcal{R}_{OR}} = pk$  as input and has access to the signing oracle  $O_{\mathbf{Sig}}$ .  $A$  wins if it comes up with  $m$  which was not queried to  $O_{\mathbf{Sig}}$  such that for a given lock  $f(x) \in \mathcal{L}_{R_m}$ ,  $A$  can find an OR-signature  $\pi$  for the knowledge of a pre-image of  $f(x)$  or  $\mathbf{Sig}(m)$ . However, the Fiat-Shamir proof is actually proof of knowledge and the ability to come up with a valid proof implies that we can extract a valid witness which is either a new signature value or a pre-image of the one-way function. Therefore, if  $A$  succeeds, we can either forge an ordinary signature or invert the one-way function, both of which easily lead to contradiction to the security of the underlying signature scheme and one-way function. Finally, the soundness property can be checked from the correctness property of the OR-proof  $\Sigma_{OR}$ .  $\square$

*Remark 1. ( $\Sigma$ -PROTOCOLS)* The  $\Sigma$ -protocol  $\Sigma_f$  for the knowledge of a pre-image of a one-way function and  $\Sigma_{\mathbf{Sig}}$  for the knowledge of a signature value can be constructed in generic ways [16]. However, there exist very efficient  $\Sigma$ -protocols for specific cases. For example,  $\Sigma$ -protocol in [22] can be used for the RSA function or the FDH signature scheme [5], and  $\Sigma$ -protocol in [31] can be applied to the discrete logarithm function or the BLS signature scheme [8]. While  $\Sigma$ -protocols for the knowledge of a signature value in [22, 31] require the random oracle model,  $\Sigma$ -protocols for the knowledge of a signature value without the random oracle model can be founded in [11, 12].

*Remark 2. (ALTERNATIVE TO THE FIAT-SHAMIR PROOF – I)* Notice that the proof of Theorem 1 only requires the following properties from the Fiat-Shamir proof: (1) witness indistinguishability and (2) proof of knowledge. Therefore, we can use the straight-line extractable WI proof [26] instead of the Fiat-Shamir proof. Like the Fiat-Shamir proof, the construction of the straight-line extractable WI proof starts with  $\Sigma$ -protocol but the length of the resulting proof is much longer. However, non-programmable random oracle can be used and better exact security is obtained. Therefore, the choice depends on the tradeoff between efficiency and exact security.

*Remark 3. (ALTERNATIVE TO THE FIAT-SHAMIR PROOF – II)* Instead of the Fiat-Shamir proof, we can also use non-interactive witness indistinguishable



proofs of knowledge (for ‘I know the pre-image of  $f(x)$ ’ or ‘I know the signature value  $\mathbf{Sig}(m)$ ’). In this case, we do not need the random oracle and can use instead a common reference string (which can be included in the public key  $pk$ ). However, the best known way of constructing non-interactive witness indistinguishable proofs of knowledge requires the existence of trapdoor one-way permutations [30] and is extremely inefficient. Nevertheless, this observation leads to the following corollary.

**Corollary 1.** *If there is a trapdoor one-way permutation, there exists a secure ID-THIR in the standard model.*

### 3 Time Capsule Signature

#### 3.1 Definition

**Definition 2.** *A time capsule signature scheme is specified by an 8-tuple of PPT algorithms  $(\text{Setup}^{\text{TS}}, \text{Setup}^{\text{User}}, \text{TSig}, \text{TVer}, \text{TRelease}, \text{Hatch}, \text{PreHatch}, \text{Ver})$  such that:*

- $\text{Setup}^{\text{TS}}$ . *This setup algorithm is run by Time Server. It takes a security parameter as input and returns a public/private time release key pair  $(\text{TPK}, \text{TSK})$ .*
- $\text{Setup}^{\text{User}}$ . *This setup algorithm is run by each user. It takes a security parameter as input and returns the user’s public/private key pair  $(\text{PK}, \text{SK})$ .*
- $\text{TSig}$ . *The time capsule signature generation algorithm  $\text{TSig}$  takes as input  $(m, \text{SK}, \text{TPK}, t)$ , where  $t$  is the specific time from which the signature becomes valid. It outputs a time capsule signature  $\sigma'_t$ .*
- $\text{TVer}$ . *The time capsule signature verification algorithm  $\text{TVer}$  takes as input  $(m, \sigma'_t, \text{PK}, \text{TPK}, t)$  and outputs 1 (accept) or 0 (reject).*
- $\text{TRelease}$ . *The time release algorithm  $\text{TRelease}$  is run by Time Server and takes as input  $(t, \text{TSK})$ . At the beginning of each time period  $t$ , Time Server publishes  $Z_t = \text{TRelease}(t, \text{TSK})$ . Note that Time Server does not contact any user at any time and need not know anything about the users.*
- $\text{Hatch}$ . *This algorithm is run by any party and is used to open a valid time capsule signature which became mature. It takes as input  $(m, \sigma'_t, \text{PK}, \text{TPK}, Z_t)$  and returns the hatched signature  $\sigma_t$ .*
- $\text{PreHatch}$ . *This algorithm is run by the signer and used to open a valid time capsule signature which is not mature yet. It takes as input  $(m, \sigma'_t, \text{SK}, \text{TPK}, t)$  and returns the pre-hatched signature  $\sigma_t$ .*
- $\text{Ver}$ . *This algorithm is used to verify a hatched (or pre-hatched) signature.  $\text{Ver}$  takes as input  $(m, \sigma_t, \text{PK}, \text{TPK}, t)$  and returns 1 (accept) or 0 (reject).*

*Correctness property states that*

- $\text{TVer}(m, \text{TSig}(m, \text{SK}, \text{TPK}, t), \text{PK}, \text{TPK}, t) = 1$  and
- $\text{Ver}(m, \sigma_t, \text{PK}, \text{TPK}, t) = 1$ , where  $\sigma_t = \text{Hatch}(m, \text{TSig}(m, \text{SK}, \text{TPK}, t), \text{PK}, \text{TPK}, Z_t)$  or  $\sigma_t = \text{PreHatch}(m, \text{TSig}(m, \text{SK}, \text{TPK}, t), \text{SK}, \text{TPK}, t)$ .

*Ambiguity property states that*

- The “hatched signature”  $\sigma_t = \text{Hatch}(m, \text{TSig}(m, \text{SK}, \text{TPK}, t), \text{PK}, \text{TPK}, Z_t)$  is computationally indistinguishable from the “pre-hatched signature”  $\sigma_t = \text{PreHatch}(m, \text{TSig}(m, \text{SK}, \text{TPK}, t), \text{SK}, \text{TPK}, t)$ , even if the distinguisher knows TSK.

The security of time capsule signatures consists of ensuring three aspects: security against the signer Alice, security against the verifier Bob, and security against Time Server. In the following, the oracle simulating the time capsule signature generation algorithm TSig is denoted by  $O_{\text{TSig}}$ , the oracle simulating the time release algorithm TRelease by  $O_{\text{TR}}$ , and the oracle simulating PreHatch by  $O_{\text{PreH}}$ . The oracle  $O_{\text{TSig}}$  takes  $(m, t)$  as input and returns Alice’s time capsule signature  $\sigma'_t$ .<sup>4</sup> The oracle  $O_{\text{PreH}}$  takes  $(m, t, \sigma'_t)$  as input and returns Alice’s pre-hatched signature  $\sigma_t$ .

**SECURITY AGAINST ALICE.** We require that any PPT adversary  $A$  succeeds with at most negligible probability in the following experiment.

$$\begin{aligned} \text{Setup}^{\text{TS}}(1^k) &\rightarrow (\text{TPK}, \text{TSK}) \\ (m, t, \sigma'_t, \text{PK}) &\leftarrow A^{O_{\text{TR}}}(\text{TPK}) \\ Z_t &\leftarrow \text{TRelease}(t, \text{TSK}) \\ \sigma_t &\leftarrow \text{Hatch}(m, \sigma'_t, \text{PK}, \text{TPK}, Z_t) \\ \text{success of } A &= [\text{TVer}(m, \sigma'_t, \text{PK}, \text{TPK}, t) \stackrel{?}{=} 1 \wedge \text{Ver}(m, \sigma_t, \text{PK}, \text{TPK}, t) \stackrel{?}{=} 0] \end{aligned}$$

In other words, Alice should not be able to produce a time capsule signature  $\sigma'_t$ , where  $\sigma'_t$  looks good to Bob but cannot be hatched into Alice’s full signature by the honest Time Server.

**SECURITY AGAINST BOB.** We require that any PPT adversary  $B$  succeeds with at most negligible probability in the following experiment.

$$\begin{aligned} \text{Setup}^{\text{TS}}(1^k) &\rightarrow (\text{TPK}, \text{TSK}) \\ \text{Setup}^{\text{User}}(1^k) &\rightarrow (\text{PK}, \text{SK}) \\ (m, t, \sigma_t) &\leftarrow B^{O_{\text{TSig}}, O_{\text{TR}}, O_{\text{PreH}}}(\text{PK}, \text{TPK}) \\ \text{success of } B &= [\text{Ver}(m, \sigma_t, \text{PK}, \text{TPK}, t) \stackrel{?}{=} 1 \wedge t \notin \text{Query}(B, O_{\text{TR}}) \\ &\quad \wedge (m, t, \cdot) \notin \text{Query}(B, O_{\text{PreH}})] \end{aligned}$$

where  $\text{Query}(B, O_{\text{TR}})$  is the set of queries  $B$  asked to the time release oracle  $O_{\text{TR}}$ , and  $\text{Query}(B, O_{\text{PreH}})$  is the set of *valid* queries  $B$  asked to the oracle  $O_{\text{PreH}}$  (i.e.,  $(m, t, \sigma'_t)$  such that  $\text{TVer}(m, \sigma'_t, \text{PK}, \text{TPK}, t) = 1$ ). In other words, Bob should not be able to open a pre-mature time capsule signature without help of the singer or Time Server. Notice that Bob can make any time release query to

<sup>4</sup> We assume that the adversary attacks an honest user Alice. The adversary can collude with all other (dishonest) users.

$O_{\text{TR}}$  except the target time  $t$ . Therefore, the above experiment requires strong security guaranteeing both forward and backward secrecy.

**SECURITY AGAINST TIME SERVER.** We require that any PPT adversary  $C$  succeeds with at most negligible probability in the following experiment.

$$\begin{aligned} & \text{Setup}^{\text{TS}^*}(1^k) \rightarrow (\text{TPK}, \text{TSK}^*) \\ & \text{Setup}^{\text{User}}(1^k) \rightarrow (\text{PK}, \text{SK}) \\ & (m, t, \sigma_t) \leftarrow C^{O_{\text{TSig}}, O_{\text{PreH}}}(\text{PK}, \text{TPK}, \text{TSK}^*) \\ & \text{success of } C = [\text{Ver}(m, \sigma_t, \text{PK}, \text{TPK}, t) \stackrel{?}{=} 1 \wedge (m, \cdot) \notin \text{Query}(C, O_{\text{TSig}})] \end{aligned}$$

where  $\text{Setup}^{\text{TS}^*}$  denotes the run of  $\text{Setup}^{\text{TS}}$  with a dishonest Time Server (run by  $C$ ),  $\text{TSK}^*$  is  $C$ 's state after this run, and  $\text{Query}(C, O_{\text{TSig}})$  is the set of queries  $C$  asked to the time capsule signature generation oracle  $O_{\text{TSig}}$  (i.e.,  $(m, t') \notin \text{Query}(C, O_{\text{TSig}})$  for all  $t'$ ). In other words, Time Server should not be able to produce a valid hatched or pre-hatched signature on  $m$  of Alice without explicitly asking Alice to produce a time capsule signature on  $m$ .

### 3.2 Generic Construction Based on ID-THIR

**THE SCHEME.** Let  $(\mathbf{Set}, \mathbf{Sig}, \mathbf{Ver})$  be an ordinary signature scheme and  $(\mathbf{Gen}, \mathbf{Sample}, \mathbf{Check}, \mathbf{Extract}, \mathbf{Invert})$  be the procedures for ID-THIR.

- $\text{Setup}^{\text{TS}}$ . Time Server chooses  $(D_{\mathcal{R}}, \text{mtd}_{\mathcal{R}})$  by running  $\mathbf{Gen}(1^k)$  and sets  $(\text{TPK}, \text{TSK}) = (D_{\mathcal{R}}, \text{mtd}_{\mathcal{R}})$ .
- $\text{Setup}^{\text{User}}$ . Each user chooses  $(pk, sk)$  by running  $\mathbf{Set}(1^k)$  and sets  $(\text{PK}, \text{SK}) = (pk, sk)$ .
- $\text{TSig}$ . To generate a time capsule signature on a message  $m$  for time  $t$ , the signer gets a random lock/proof pair  $(c, d)$  from  $\mathbf{Sample}_{D_{\mathcal{R}}}(t)$  and computes  $s = \mathbf{Sig}_{sk}(m||c||t)$ . The time capsule signature value  $\sigma'_t$  is  $(s, c)$  and the signer stores the proof  $d$  for later use.
- $\text{TVer}$ . For a given time capsule signature  $\sigma'_t = (s, c)$ , the verifier checks that  $c \in \mathcal{L}_{R_t}$  and  $s$  is a valid signature on  $m||c||t$  by running  $\mathbf{Ver}_{pk}(m||c||t, s)$ .
- $\text{TRelease}$ . For a given time value  $t$ , Time Server computes  $\text{td}_{R_t} = \mathbf{Extract}_{\text{mtd}_{\mathcal{R}}}(t)$  and publishes  $Z_t = \text{td}_{R_t}$ .
- $\text{Hatch}$ . To open a mature time capsule signature  $\sigma'_t = (s, c)$ , a party computes  $\tilde{d} = \mathbf{Invert}_{\text{td}_{R_t}}(c)$  and returns the hatched signature  $\sigma_t = (s, c, \tilde{d})$ .
- $\text{PreHatch}$ . To open a valid pre-mature time capsule signature  $\sigma'_t = (s, c)$ , the signer returns the pre-hatched signature  $\sigma_t = (s, c, d)$  where the proof  $d$  is a stored value in the stage of  $\text{TSig}$ .
- $\text{Ver}$ . For a given hatched (or pre-hatched) signature  $\sigma_t = (s, c, d)$ , the verifier checks the lock/proof pair by running  $\mathbf{Check}_{D_{\mathcal{R}}, t}(c, d)$ . Then, he verifies that  $s$  is a valid signature on  $m||c||t$  by running  $\mathbf{Ver}_{pk}(m||c||t, s)$ .

The correctness property and the ambiguity property of the scheme are obvious from the properties of ID-THIR. We now analyze its security.

**Theorem 2.** *The time capsule signature scheme presented above is secure if the underlying ordinary signature scheme and the ID-THIR are secure.*

**Proof:** We prove the security against Alice, Bob, and Time Server.

**SECURITY AGAINST ALICE.** Security against Alice follows unconditionally. A valid time capsule signature  $\sigma'_t = (s, c)$  satisfies that  $c \in \mathcal{L}_{R_t}$  and  $\mathbf{Ver}_{pk}(m||c||t, s) = 1$ . If Time Server releases  $\mathbf{td}_t = \mathbf{Extract}_{\text{mtd}_{\mathcal{R}}}(t)$ , any party can obtain a proof  $\tilde{d} = \mathbf{Invert}_{\mathbf{td}_t}(c)$  for the lock  $c \in \mathcal{L}_{R_t}$ . By the correctness property of ID-THIR,  $\mathbf{Check}_{D_{\mathcal{R}}, t}(c, \tilde{d}) = 1$  always holds. Therefore, the hatched signature  $\sigma_t = (s, c, \tilde{d})$  passes the verification algorithm  $\mathbf{Ver}$ .

**SECURITY AGAINST BOB.** To show security against Bob, we convert any attacker  $B$  that attacks our time capsule signature scheme into an inverter  $Inv$  of ID-THIR. Recall that  $Inv$  gets  $D_{\mathcal{R}}$  as input and has access to the trapdoor extraction oracle  $O_{\mathbf{Extract}}$ .  $Inv$  wins if it comes up with  $id$  which was not queried to  $O_{\mathbf{Extract}}$  s.t. for a given lock  $c \in L_{R_{id}}$ ,  $Inv$  can find a proof  $d$  for  $c$ . On the other hand,  $B$  expects  $(\mathbf{PK}, \mathbf{TPK})$  as input and has access to  $O_{\mathbf{TSig}}, O_{\mathbf{TR}}, O_{\mathbf{PreH}}$ .  $B$  wins if it forges a hatched (or pre-hatched) signature  $\sigma_t$  of some message  $m$  without asking  $t$  to  $O_{\mathbf{TR}}$  or  $(m, t, \sigma'_t)$  to  $O_{\mathbf{PreH}}$ . Let  $(m_B, t_B, \sigma_{t_B})$  be the successful forgery of the attacker  $B$ . We can assume that  $B$  obtained the corresponding time capsule signature  $\sigma'_{t_B}$  from  $O_{\mathbf{TSig}}$ , since the underlying ordinary signature scheme  $(\mathbf{Set}, \mathbf{Sig}, \mathbf{Ver})$  is existentially unforgeable against chosen message attacks.

When  $Inv$  receives  $D_{\mathcal{R}}$  from an ID-THIR challenger  $\mathcal{C}$ , it begins simulating the attack environment of  $B$ .  $Inv$  picks a random public/private key pair  $(pk, sk)$  by running  $\mathbf{Set}(1^k)$ , sets  $\mathbf{PK} = pk$ ,  $\mathbf{SK} = sk$ ,  $\mathbf{TPK} = D_{\mathcal{R}}$ , and gives  $(\mathbf{PK}, \mathbf{TPK})$  to  $B$ .  $Inv$  manages a list  $L = \{(m_i, t_i, s_i, c_i, d_i)\}$  to answer  $B$ 's queries to  $O_{\mathbf{PreH}}$ . Let  $q_{\mathbf{TSig}}$  be the total number of  $O_{\mathbf{TSig}}$  queries made by  $B$  and  $r$  be a random number chosen by  $Inv$  in the interval of  $\{1, 2, \dots, q_{\mathbf{TSig}}\}$ . Now,  $Inv$  knowing  $\mathbf{SK} = sk$  responds to the  $i$ -th  $O_{\mathbf{TSig}}$  query  $(m_i, t_i)$  of  $B$  as follows;

- If  $i = r$ ,  $Inv$  outputs  $t_r$  to the challenger  $\mathcal{C}$  and receives a random lock  $c \in R_{t_r}$  from the challenger.  $Inv$  sets  $c_r = c$  and computes  $s_r = \mathbf{Sig}_{sk}(m_r||c_r||t_r)$ .  $Inv$  returns  $\sigma'_{t_r} = (s_r, c_r)$  to  $B$  and stores the element  $(m_r, t_r, s_r, c_r, \perp)$  in the list  $L$ .
- If  $i \neq r$ ,  $Inv$  picks a random lock/proof pair  $(c_i, d_i)$  from  $\mathbf{Sample}_{D_{\mathcal{R}}}(t_i)$  and computes  $s_i = \mathbf{Sig}_{sk}(m_i||c_i||t_i)$ .  $Inv$  returns  $\sigma'_{t_i} = (s_i, c_i)$  to  $B$  and stores the element  $(m_i, t_i, s_i, c_i, d_i)$  in the list  $L$ .

To simulate  $O_{\mathbf{TR}}$  to the query  $t_i$  of  $B$ ,  $Inv$  simply asks  $t_i$  to its own trapdoor extraction oracle  $O_{\mathbf{Extract}}$  and gets  $\mathbf{td}_{R_{t_i}}$ . If  $t_i = t_r$ ,  $Inv$  aborts. Otherwise,  $Inv$  returns  $Z_{t_i} = \mathbf{td}_{R_{t_i}}$  to  $B$ .

To simulate  $O_{\mathbf{PreH}}$  to the query  $(m_i, t_i, s_i, c_i)$ ,  $Inv$  checks whether the query is in the list  $L$  or not (by considering only the first four components of an element in  $L$ ). If  $(m_i, t_i, s_i, c_i)$  is in the list  $L$  and equal to  $(m_r, t_r, s_r, c_r)$ ,  $Inv$  aborts. If  $(m_i, t_i, s_i, c_i)$  is in the list  $L$  and not equal to  $(m_r, t_r, s_r, c_r)$ ,  $Inv$  obtains a proof  $d_i$  from the list  $L$  and give a pre-hatched signature  $\sigma_{t_i} = (s_i, c_i, d_i)$  to

$B$ . If  $(m_i, t_i, s_i, c_i)$  is not in the list  $L$  (i.e., the time capsule signature was not generated by  $Inv$  and therefore the query is invalid with very high probability),  $Inv$  answers randomly to  $B$ .

The probability that  $Inv$  does not abort during the simulation is at least  $1/q_{\text{TSig}}$ , since  $r \in \{1, \dots, q_{\text{TSig}}\}$  is randomly chosen and a secure ID-THIR satisfies the ambiguity property. When  $B$  outputs the forgery  $(m_B, t_B, s_B, c_B, d_B)$ ,  $Inv$  verifies that the forgery passes the verification algorithm  $\text{Ver}$  and  $(m_B, t_B, s_B, c_B) = (m_r, t_r, s_r, c_r)$ . If so,  $Inv$  outputs the proof  $d_B$ . Otherwise,  $Inv$  chooses a proof  $d_{Inv}$  randomly and outputs  $d_{Inv}$ . Therefore, if  $B$  forges with a probability  $\epsilon$ ,  $Inv$  succeeds in breaking the one-wayness of ID-THIR with a probability  $\epsilon' \geq \epsilon/q_{\text{TSig}}$ .

**SECURITY AGAINST TIME SERVER.** To show security against Time Server, we convert any attacker  $C$  that attacks our time capsule signature scheme into a forger  $F$  for the underlying ordinary signature. Recall that  $F$  gets  $pk$  as an input, and has access to the signing oracle  $O_{\text{Sig}}$ . On the other hand,  $C$  expects  $(\text{PK}, \text{TPK}, \text{TSK})$  as input and has access to  $O_{\text{TSig}}$  and  $O_{\text{PreH}}$ .  $C$  wins if it forges a hatched (or pre-hatched) signature  $\sigma_t$  of some message  $m$  without obtaining a time capsule signature on  $m$  from  $O_{\text{TSig}}$ .

So here is how  $F$  simulates the run of  $C$ . To choose ID-THIR,  $F$  runs  $\text{Gen}(1^k)$  and obtains  $(D_{\mathcal{R}}, \text{mtd}_{\mathcal{R}})$ . Then,  $F$  gives  $(\text{PK}, \text{TPK}, \text{TSK}) = (pk, D_{\mathcal{R}}, \text{mtd}_{\mathcal{R}})$  to  $C$ .  $F$  can respond to  $O_{\text{TSig}}$  queries  $(m_i, t_i)$  of  $C$  by choosing a random lock/proof pair  $(c_i, d_i)$  from  $\text{Sample}_{D_{\mathcal{R}}}(t_i)$  and getting an ordinary signature  $s_i$  on  $m_i||c_i||t_i$  from its own signing oracle  $O_{\text{Sig}}$ .  $F$  stores  $(m_i, c_i, d_i, t_i)$  in the list  $L = \{(m_i, c_i, d_i, t_i)\}$  to answer  $C$ 's queries to  $O_{\text{PreH}}$ . To simulate  $O_{\text{PreH}}$  to the queries  $(m_i, t_i, s_i, c_i)$ ,  $F$  verifies that  $s_i$  is a valid signature on  $m_i||c_i||t_i$ .

- If  $s_i$  is a valid signature on  $m_i||c_i||t_i$ ,  $F$  checks whether  $(m_i, c_i, t_i)$  is in the list  $L$  or not. If it is in the list,  $F$  can give the corresponding proof  $d_i$  from the list  $L$ . Otherwise,  $s_i$  is a new signature value and  $F$  succeeds in producing a new forgery  $s_i$  on  $m_i||c_i||t_i$ .  $F$  stops the simulation.
- If  $s_i$  is not a valid signature on  $m_i||c_i||t_i$ ,  $F$  answers randomly.

When  $C$  outputs the forgery  $(\widehat{m}, \widehat{t}, \widehat{\sigma}_t)$  where  $\widehat{\sigma}_t = (\widehat{s}, \widehat{c}, \widehat{d})$ ,  $F$  outputs an ordinary signature  $\widehat{s}$  on a message  $\widehat{m}||\widehat{c}||\widehat{t}$ . Therefore, if  $C$  succeeds with a probability  $\epsilon$ ,  $F$  succeeds in producing a new forgery with a probability  $\epsilon' \geq \epsilon$ .  $\square$

**Theorem 3.** *If there is a one-way function, there exists a secure time capsule signature scheme in the random oracle model.*

**Proof:** Secure signatures exist if and only if one-way functions exist [29]. Together with Theorem 1 and Theorem 2, we obtain Theorem 3.  $\square$

**Theorem 4.** *If there is a trapdoor one-way permutation, there exists a secure time capsule signature scheme in the standard model.*

**Proof:** Secure signatures exist if and only if one-way functions exist [29]. Together with Corollary 1 and Theorem 2, we obtain Theorem 4.  $\square$

*Remark 4.* (EVENT CAPSULE SIGNATURE) In the definition and construction of time capsule signature, we did not use any characteristic of the real time. Actually,  $t$  need not be a time value and any index works for  $t$ . Therefore, the definition and construction of time capsule signature can be efficiently converted to those of event capsule signature.

## 4 On Trapdoor Hard-to-Invert Relation

A trapdoor hard-to-invert relation (THIR) is a specific elementary relation  $R_{id}$  in ID-THIR  $\mathcal{R} = \{R_{id} \mid id \in I_R\}$ . The definition of THIR can be derived from that of ID-THIR and the construction becomes even simpler as a signature on one identity is simply a one-way function. Notice that THIR is also very easily constructed without the random oracle model (unlike ID-THIR) if trapdoor one-way permutations exist (for details, refer to the full version).

However, it is interesting to ask whether THIR (primitive simpler than ID-THIR) can be built from one-way functions (or even one-way permutations) in the standard model. We leave this as an open problem. However, we comment that it is highly unlikely that a special case of THIR — so called deterministic THIR where only one proof  $d$  exists for a given lock  $c$  — can be constructed from one-way permutations. Indeed, one can easily see that the existence of a deterministic THIR implies that of a secure key agreement scheme. However, it is known that there exists no block-box reduction from one-way permutations to secure key agreement schemes [23].

## Acknowledgments

We would like to thank Leonid Reyzin for several very insightful discussions. The work of the second author was partially supported by Brain Korea 21 project.

## References

1. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *EUROCRYPT 1998*, LNCS 1403, pp. 591–606, Springer, 1998.
2. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE J. Select. Areas Commun*, 18(4), pp. 593–610, 2000.
3. M. Bellare and S. Goldwasser. Encapsulated key escrow. MIT Laborator for Computer Science Technical Report 688, 1996.
4. M. Bellare and S. Goldwasser. Verifiable partial key escrow. In *the 4th ACM CCS*, pp. 78–91, 1997.
5. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *the 1st ACM CCS*, pp. 62–73, 1993.
6. I. Blake and A. Chan. Scalable, server-passive, user-anonymous timed release public key encryption from bilinear pairing. <http://eprint.iacr.org/2004/211/>.
7. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *CRYPTO 2001*, LNCS 2139, pp. 213–229, Springer, 2001.

8. D. Boneh, B. Lynn and H. Shacham. Short signatures from the Weil pairing. In *ASIACRYPT 2001*, LNCS 2248, pp. 514–532, Springer, 2001.
9. D. Boneh and M. Naor. Timed commitments. In *CRYPTO 2000*, pp. 236–254.
10. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT 2003*, pp. 416–432.
11. J. Camenisch and A. Lysyanskaya. Signature schemes with efficient protocols. In *SCN 2002*, LNCS 2576, pp. 268–289, Springer, 2002.
12. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO 2004*, LNCS 3152, pp. 56–72, Springer, 2004.
13. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO 1994*, pp. 174–187.
14. Y. Dodis and L. Reyzin. Breaking and repairing optimistic fair exchange from PODC 2003. In *Digital Rights Management 2003*, pp. 47–54, 2003.
15. C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *CRYPTO 1992*, LNCS 740, pp. 139–147, Springer, 2004.
16. U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In *CRYPTO 1989*, LNCS 435, pp. 526–544, Springer, 1989.
17. U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *the 22nd Annual ACM Symposium on Theory of Computing*, pp. 416–426, 1990.
18. A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO 1986* LNCS 263, pp. 186–194, Springer, 1986.
19. J. Garay and M. Jakobsson. Timed release of standard digital signatures. In *Financial Cryptography 2002*, LNCS 2357, pp. 168–182, Springer, 2002.
20. J. Garay and C. Pomerance. Timed fair exchange of standard signatures. In *Financial Cryptography 2003*, LNCS 2742, pp. 190–207, Springer, 2003.
21. O. Goldreich, S. Micali and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3), pp. 691–729, 1991.
22. L. Guillou and J.J. Quisquater. A “paradoxical” identity-based signature scheme resulting from zero-knowledge. In *CRYPTO 1988*, pp. 216–231.
23. R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *the 21st STOC*, pp. 44–61, 1989.
24. A. Joux and K. Nguyen. Separating decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. <http://eprint.iacr.org/2001/003/>.
25. I. Osipkov, Y. Kim and J. Cheon. New approaches to timed-release cryptography. <http://eprint.iacr.org/2004/231/>.
26. R. Pass. On deniability in the common reference string and random oracle model. In *CRYPTO 2003*, LNCS 2729, pp. 316–337, Springer, 17–21 Aug. 2003.
27. R. Rivest, A. Shamir, and D. Wagner. Time lock puzzles and timed release cryptography. Technical report, MIT/LCS/TR-684.
28. D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT 1996*, LNCS 1070, pp. 387–398, Springer, 1996.
29. J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *the 22nd Annual ACM Symposium on Theory of Computing*, pp. 387–394, 1990.
30. A. De Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction. In *the 33rd FOCS*, pp. 427–436, 1992.
31. C. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO 1989*, LNCS 435, pp. 239–252, Springer, 1989.
32. A. Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO 1984*, LNCS 196, pp. 47–53, Springer, 1984.