On Universal Composable Security of Time-Stamping Protocols

Toshihiko Matsuo and Shin'ichiro Matsuo

R&D Headquarters, NTT DATA Corporation,
1-21-2, Shinkawa, Chuo-ku, Tokyo 104-0033, Japan
{matsuotsh,matsuosn}@nttdata.co.jp

Abstract. Time-stamping protocols, which assure that a document was existed at a certain time, are applied to some useful and practical applications such as electronic patent applications and so on. There are two major time-stamping protocols, the simple protocol and the linking protocol. In the former, a time-stamp authority issues a time-stamp token that is the digital signature of the concatenated value of a hashed message and the present time. In the latter, the time-stamp authority issues a time-stamp token that is the hash value of the concatenated value of a hashed message and the previous hash value. Although security requirements and analysis for above time-stamping protocols has been discussed, there are no strict cryptographic security notions for them. In this paper, we reconsider the security requirements for time-stamping protocols and define security notions for them, in a universally composable security sense, which was proposed by Canetti. We also show that these notions can be achieved using combinations of a secure key exchange protocol, a secure symmetric encryption scheme, and a secure digital signature scheme.
Keyword: time-stamping protocol, universal composable security

1 Introduction

Opportunities for creating a lot of digital documents and distributing them over digital networks are growing rapidly. They include not only various kinds of private documents but also important documents such as formal applications and business contracts. It is thus important to be able to prove that a digital document existed at a certain time. For example, contracts are dated and the validity of the date may later have to be proven. For patents, the exact date when it was applied must be shown, and it must be possible to prove the validity of the date. In stock trading, the exact time when a buy/sell order was placed must be shown since the gain/loss is directly affected by the time.

Two major time-stamping protocols, the simple protocol and the linking protocol [ISO,HS91,ACPZ01] that satisfy these requirements have been developed. The former uses a digital signature as a time-stamp token [ISO,ACPZ01]. A time stamp authority (TSA) signs the concatenation of the hashed message and the present time. by using its private signing key. Since the validity of the time-stamp token depends on that of the digital signature scheme itself, the TSA

must be a trusted third party. In the latter, a time-stamp token is the hash value of the concatenation of the present hashed message and the previous hash value [HS91]. A verifier can check the validity of the token by using published values for the hash chain. In this case, the TSA is not necessarily a trusted third party. Several variants of linking protocols have been proposed such as [BLLV98,BLS00]. These two major protocols are already being used by actual time-stamping services such as Surety [Sure].

1.1 Security requirements of time-stamping protocols

There are two major types of attacks on time-stamping protocols. The first type is that an adversary may try to back-date the valid time-stamp. This is a fatal attack for applications in which the priority is based on descendent time order. [1] This type of attack is called a "*back-dating attack*." The adversary may corrupt the TSA and may try to create a forged but valid time-stamp token. The simple protocol is clearly not secure against TSA corruption, but the linking protocol is since a verifier can check the validity by computing the chain of hash values using published hash values.

In the other type of attack, an adversary may try to forward-date the time-stamp without the approval of the valid requester. This is a fatal attack for applications in which the priority is based on ascendent time order. [2] This type of attack is called a "*forward-dating attack*."

Although there have been several studies of the security of time-stamping protocols [HS91,Just98,UM02], there are no strict security notions for them in a cryptographic (computational) sense.

1.2 Universally composable security

Canetti proposed a framework for defining the security of cryptographic protocols that he called *universally composable security* (UC security) [C01]. In this framework, the ideal functionality that achieves a certain service, the set of parties and the adversary are denoted as $\mathcal{F}$, $\widetilde{P}$, and $\mathcal{S}$, respectively. Each party does not communicate directly with the others, and the adversary can corrupt any party at any time. On the other hand, the actual protocol that achieves the service, the set of parties, and the adversary are denoted as π, P, and $\mathcal{A}$, respectively. Then, we assume the existence of an environment $\mathcal{Z}$ which communicates all parties and $\mathcal{A}$. Each party can communicate with the others and $\mathcal{A}$ can control all communication, meaning that $\mathcal{A}$ can read or alter all messages among the parties. $\mathcal{A}$ can also corrupt any party at any time. In Canetti's framework, protocol π securely realizes $\mathcal{F}$ if, for $\forall\mathcal{A}$ and $\forall\mathcal{Z}$, there exists an adversary $\mathcal{S}$ which makes $\mathcal{Z}$ difficult to distinguish whether she accesses $\widetilde{P}$ and $\mathcal{S}$ or P and $\mathcal{A}$. This framework helps us to prove security of large cryptographic protocol due to following two properties

[1] For example, intellectual property rights protection is the case.
[2] For example, digital will is the case.

Composition Theorem: The key advantage of UC security is that we can create a complex protocol from already-designed sub-protocols that securely achieves the given local tasks. This is very important since complex systems are usually divided into several sub-systems, each one performing a specific task securely. Canetti presented this feature as the composition theorem [C01]. This theorem assures that we can generally construct a large size "UC-secure" cryptographic protocols by using sub-protocols which is proven as secure in UC-secure manner.

Hybrid model: In order to state above theorem and to formalize the notion of an actual protocol with access to multiple copies of an ideal functionality, Canetti also introduced the hybrid model which is identical to the actual model with the following. On top of sending messages to each other, the parties may send messages to and receive messages from an unbounded number of copies of an ideal functionality $\mathcal{F}$. The copies of $\mathcal{F}$ are differentiated using their session identifier SIDs. All messages addressed to each copy and all messages sent by each copy carry the corresponding SID.

There are various studies on the sense of UC security. Although several ideal functionalities of cryptographic primitives have been proposed [3] [C01,CF01,CK02,C04], there is no definition of functionality of time-stamping protocol.

1.3 Our contribution

In this paper, we consider the security notions of the time-stamping protocol and define its functionality based on the UC framework. Our definition follows that of the signature functionality, $\mathcal{F}_{SIG}$, defined by Canetti [C04] because the required properties of the time-stamping protocol are similar to those of the digital signature scheme. However, a time-stamping protocol requires unique security properties, so we have to newly define its functionality.

In addition, we describe the construction of a secure time-stamping protocol using the key exchange functionality $\mathcal{F}_{KE}$ [C01] and the signature functionality. Briefly speaking, a time-stamp token requester and a TSA exchange a session key by using $\mathcal{F}_{KE}$, and then the requester encrypts the message by using the session key and sends it to the TSA. Then, the TSA time-stamps the received message by using $\mathcal{F}_{SIG}$ so as to include the requester's ID and returns the token to the requester. Since the message the requester wants to be stamped is encrypted, an adversary can not obtain a time-stamp token ahead of a valid requester. The requester's ID prevents the adversary from claiming her legitimacy with a time-stamp token she acquires by observing the transaction.

Organization of this paper is as follows. In section 2, we give our definition of time-stamping protocols, and describe their security requirements and ideal functionality. Then we show the construction of a UC secure time-stamping protocol and its security proof in section 3. In section 4, we discuss a simpler

[3] Digital signature, public-key encryption, key exchange, bit commitment etc.

construction and its security. We conclude our study in section 5 with a brief summary.

2 Time-Stamping Protocols

2.1 Definition

In this paper, we define a time-stamping protocol as follows.

- Let k be a security parameter. A TSA obtains time-stamping key δ and verification key θ by executing key generation protocol or algorithm $SetUp$, which outputs δ and θ on input 1^k.
- Each time-stamp token requester executes a time-stamp token generation protocol and acquires the time-stamp token σ for document d and time t from the TSA.
- A verifier verifies σ by executing time-stamp token verification protocol or verification algorithm Ver. He verifies σ with θ and auxiliary information ρ.

In the following, we denote a time-stamping protocol by π_{TS}.

2.2 Security requirements

We let an adversary for a time-stamping protocol be an interactive Turing machine (ITM) [4] [C04]. The adversary (we sometimes denote an adversary by $\mathcal{A}$) can do anything to the communication between any two parties. $\mathcal{A}$ can also corrupt any party at anytime. The security requirements for a time-stamping protocol are similar to those for a digital signature; however, we have to take the following requirements into consideration.

1. $\mathcal{A}$ may initiate a man-in-the-middle attack because a time-stamp requester can not issue a time-stamp token by herself; the requester has to communicate with a TSA.
2. To protect "forward-dating attack," the time-stamp token should contain the requester's ID to make the protocol secure [MO04].
3. In the linking protocol, the verification algorithm needs not only a verification key but published hash values to enable σ to be verified.

Therefore, we define the security requirements for a time-stamping protocol as follows.

Definition 1. *Let k be a security parameter and $\varepsilon(\cdot)$ be a negligible function on k. Let δ be a time-stamping key, θ be a verification key, and ID be a unique identifier of the requester. We say that a time-stamping protocol satisfies the security requirements if the following properties hold.*

[4] We sometimes denote an ITM entity by using a calligraphic font.

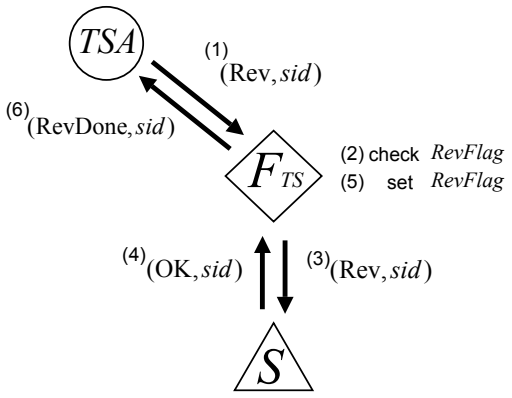

Fig. 1. Key Revocation process

Completeness *For any document d and valid time-stamp token σ stamped at time t,*

$$\Pr_{\delta}[(\delta, \theta) \leftarrow SetUP(1^k); 0 \leftarrow Ver(\theta, \rho, t, d, ID, \sigma)] \leq \varepsilon(k),$$

where ρ is valid auxiliary information generated during execution of the protocol.

Consistency *For any document d and valid time-stamp token σ stamped at time t, the probability that $Ver(\theta, \rho, t, d, ID, \sigma)$ generates two different outputs in two independent invocations is smaller than $\varepsilon(k)$, where ρ is valid auxiliary information generated during execution of the protocol.*

Unforgeability $\Pr_{\delta}[(\delta, \theta) \leftarrow SetUP(1^k); \{(t_0, d_0, ID_0, \sigma_0, \rho_0), (t_1, d_1, ID_1, \sigma_1, \rho_1)\} \leftarrow \mathcal{A}^{\pi_{TS}}(\theta);\ b \in \{0,1\}; 1 \leftarrow Ver(\theta, \rho_b, t_b, d_b, ID_b, \sigma_b)] \leq \varepsilon(k)$,

where ρ_b is valid auxiliary information generated during execution of the protocol. Furthermore, (1) either σ_0 or σ_1 is not generated by TSA or (2) $d_0 = d_1$, $ID_0 = ID_1$, and $t_0 \neq t_1$.

2.3 Ideal functionality and security condition

In the UC framework, all entities are interactive Turing machines [C04]. Each entity has a session-identifier (SID) that represents the session to which the entity belongs. It also has a party identifier (PID) that represents the role of the entity in the protocol instance. The pair $sid = (PID, SID)$ is guaranteed to

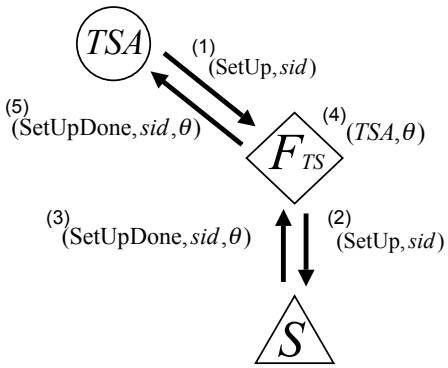

Fig. 2. Key generation process

be unique in the system. We define the functionality $\mathcal{F}_{TS}$ of the time-stamping protocol as follows. For simplicity, we assume that a TSA manages only one time-stamping key at one time.

Key revocation (Fig. 1):

1. $\mathcal{TSA}$ sends (**Rev**, sid) to $\mathcal{F}_{TS}$.
2. If $sid = (\mathcal{TSA}, sid')$ for some sid' and the corresponding revocation flag $RevFlag$ equals to 1, go to the next step. Otherwise ignore the request.
3. $\mathcal{F}_{TS}$ sends (**Rev**, sid) to $\mathcal{S}$.
4. $\mathcal{S}$ sends (**OK**, sid) to $\mathcal{F}_{TS}$.
5. $\mathcal{F}_{TS}$ sets $RevFlag \leftarrow 0$, erases the corresponding record $(\mathcal{TSA}, \theta)$, and sends (**RevDone**, sid) to $\mathcal{TSA}$.

Key generation (Fig. 2):

1. $\mathcal{TSA}$ sends (**SetUp**, sid) to $\mathcal{F}_{TS}$.
2. If $sid = (\mathcal{TSA}, sid')$ for some sid' and $RevFlag = 0$, go to the next step. Otherwise ignore the request.
3. $\mathcal{F}_{TS}$ sends (**SetUp**, sid) to $\mathcal{S}$.
4. $\mathcal{S}$ sends (**SetUpDone**, sid, θ) to $\mathcal{F}_{TS}$.
5. $\mathcal{F}_{TS}$ records $(\mathcal{TSA}, \theta)$ and then sends (**SetUpDone**, sid, θ) to $\mathcal{TSA}$. $\mathcal{F}_{TS}$ sets the corresponding flag $RevFlag \leftarrow 1$.

Time-stamp token generation (Fig. 3):

1. Time-stamp token requester $\mathcal{P}$ sends (**StampReq**, sid, d) to $\mathcal{F}_{TS}$.

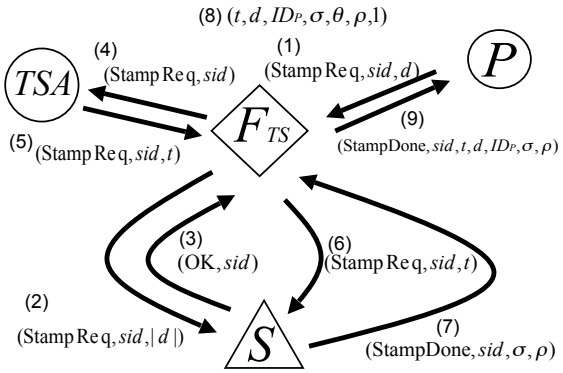

Fig. 3. Time-stamp token generation process

2. If $sid = (\mathcal{TSA}, sid')$ for some sid', $\mathcal{F}_{TS}$ sends $(\mathbf{StampReq}, sid, |d|)$ to $\mathcal{S}$. Otherwise ignore the request.
3. $\mathcal{S}$ sends $(\mathbf{OK}, sid)$ to $\mathcal{F}_{TS}$.
4. $\mathcal{F}_{TS}$ sends $(\mathbf{StampReq}, sid)$ to $\mathcal{TSA}$.
5. $\mathcal{TSA}$ chooses t and then sends $(\mathbf{StampReq}, sid, t)$ to $\mathcal{F}_{TS}$, where t is an increasing value.
6. $\mathcal{F}_{TS}$ sends $(\mathbf{StampReq}, sid, t)$ to $\mathcal{S}$.
7. $\mathcal{S}$ sends $(\mathbf{StampDone}, sid, \sigma, \rho)$ to $\mathcal{F}_{TS}$.
8. $\mathcal{F}_{TS}$ records $(t, d, ID_{\mathcal{P}}, \sigma, \theta, \rho, 1)$ and then sends $(\mathbf{StampDone}, sid, t, d, ID_{\mathcal{P}}, \sigma, \rho)$ to $\mathcal{P}$.

Time-stamp token verification (Fig. 4):
Let $f, \phi \in \{0, 1\}$.

1. Verifier $\mathcal{V}$ sends $(\mathbf{Ver}, sid, \alpha)$ to $\mathcal{F}_{TS}$, where $\alpha = (t, d, ID_{\mathcal{P}}, \sigma, \widetilde{\theta}, \widetilde{\rho})$.
2. $\mathcal{F}_{TS}$ sends $(\mathbf{Ver}, sid, \alpha)$ to $\mathcal{S}$.
3. $\mathcal{S}$ sends $(\mathbf{VerDone}, sid, \alpha, \phi)$ to $\mathcal{F}_{TS}$.
4. $\mathcal{F}_{TS}$ executes the following. (1) If $(\widetilde{\theta}, \widetilde{\rho}) = (\theta, \rho)$ and $\mathcal{F}_{TS}$ has already recorded $(\alpha, 1)$, $f \leftarrow 1$. (2) If $(\widetilde{\theta}, \widetilde{\rho}) = (\theta, \rho)$, $\mathcal{S}$ does not corrupt the $\mathcal{TSA}$, and $\mathcal{F}_{TS}$ has not recorded $(t, d, ID_{\mathcal{P}}, \sigma', \theta, \rho, 1)$ for $\forall \sigma'$, $f \leftarrow 0$ and $\mathcal{F}_{TS}$ records $(\alpha, 0)$. (3) If $(\widetilde{\theta}, \widetilde{\rho}) \neq (\theta, \rho)$ and $\mathcal{F}_{TS}$ has already recorded $(\alpha, \widetilde{f})$, $f \leftarrow \widetilde{f}$. (4) Otherwise, $f \leftarrow \phi$ and $\mathcal{F}_{TS}$ records (α, ϕ).
5. $\mathcal{F}_{TS}$ sends $(\mathbf{VerDone}, sid, t, d, ID_{\mathcal{P}}, f)$ to $\mathcal{V}$.

We use Canetti's definition of the signature functionality $\mathcal{F}_{SIG}$ [C04] to define $\mathcal{F}_{TS}$.

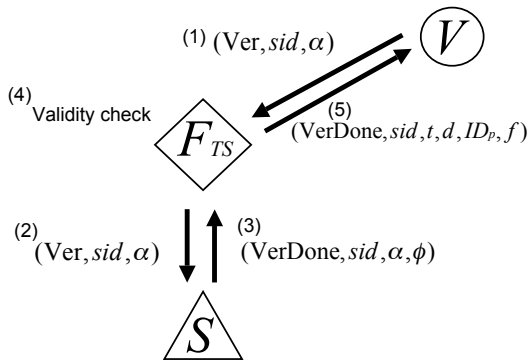

Fig. 4. Time-stamp token verification process

In the key generation, we define that $\mathcal{S}$ can choose the verification key θ. This is because the security requirements in Def. 1 do not restrict the distribution of θ. Similarly, we define that $\mathcal{S}$ can choose time-stamp token σ and auxiliary information ρ in the time-stamp token generation.

In the token generation, if $\mathcal{S}$ obtains document d itself during protocol execution, it is clear that she can acquire its time-stamp token ahead of a valid time-stamp requester. That is, as soon as $\mathcal{S}$ gets d, she can ask the TSA to issue the time-stamp token for d as a valid requester, ahead of the valid time-stamp requester. This is a fatal attack in applications like electronic patent applications. Therefore, d must be kept secret until protocol execution ends.

We define the UC security condition of a time-stamping protocol as follows.

Definition 2. *Let $\mathcal{F}_{TS}$ be an ideal time-stamping functionality, $\widetilde{P}$ be the set of dummy parties, and $\mathcal{S}$ be an ideal adversary with access to $\mathcal{F}_{TS}$. Let π_{TS} be the actual time-stamping protocol, P be the set of actual parties, $\mathcal{A}$ be the actual adversary with access to π_{TS}, and $\mathcal{Z}$ be an environment which communicates P and $\mathcal{A}$. If for any $\mathcal{A}$ and $\mathcal{Z}$, there exists $\mathcal{S}$ such that $\mathcal{Z}$ can not distinguish which entities she accesses, we say that π_{TS} securely realizes $\mathcal{F}_{TS}$.*

3 Construction of UC secure time-stamping protocol

Canetti proposed key exchange functionality $\mathcal{F}_{KE}$ and basic signature functionality $\mathcal{F}_{SIG}$ [C01,C04]. In this section, we describe a construction of UC-secure time-stamp protocol π_{TS} that is based on $\mathcal{F}_{KE}$ and $\mathcal{F}_{SIG}$. For simplicity, we omit the key revocation procedure.

3.1 Preliminaries

We use $\mathcal{F}_{SIG}$ and $\mathcal{F}_{KE}$ as proposed by Canetti [C01,C04] in our construction. Canetti defined $\mathcal{F}_{SIG}$ as follows.

Key generation:

1. Signer $\mathcal{P}$ sends $(\textbf{KeyGen}, sid)$ to $\mathcal{F}_{SIG}$.
2. $\mathcal{F}_{SIG}$ verifies that $sid \stackrel{?}{=} (\mathcal{P}, sid')$ for some sid'. If it does not hold , $\mathcal{F}_{SIG}$ ignores the request. Else, $\mathcal{F}_{SIG}$ sends $(\textbf{KeyGen}, sid)$ to $\mathcal{S}$.
3. $\mathcal{S}$ sends $(\textbf{VerificationKey}, sid, \theta)$ to $\mathcal{F}_{SIG}$.
4. $\mathcal{F}_{SIG}$ records $(\mathcal{P}, \theta)$ and then sends $(\textbf{VerificationKey}, sid, \theta)$ to $\mathcal{P}$.

Signature generation:

1. $\mathcal{P}$ sends $(\textbf{Sign}, sid, d)$ to $\mathcal{F}_{SIG}$.
2. $\mathcal{F}_{SIG}$ verifies that $sid \stackrel{?}{=} (\mathcal{P}, sid')$ for some sid'. If it does not hold , $\mathcal{F}_{SIG}$ ignores the request. Else, $\mathcal{F}_{SIG}$ sends $(\textbf{Sign}, sid, d)$ to $\mathcal{S}$.
3. $\mathcal{S}$ sends $(\textbf{Signature}, sid, d, \sigma)$ to $\mathcal{F}_{SIG}$.
4. $\mathcal{F}_{SIG}$ looks for the record $(d, \sigma, \theta, 0)$. If it is found, $\mathcal{F}_{SIG}$ sends an error message to $\mathcal{P}$ and halts. Else, $\mathcal{F}_{SIG}$ sends $(\textbf{Signature}, sid, d, \sigma)$ to $\mathcal{P}$ and then records $(d, \sigma, \theta, 1)$.

Signature verification:

1. Verifier $\mathcal{V}$ sends $(\textbf{Verify}, sid, d, \sigma, \widetilde{\theta})$ to $\mathcal{F}_{SIG}$.
2. $\mathcal{F}_{SIG}$ sends $(\textbf{Verify}, sid, d, \sigma, \widetilde{\theta})$ to $\mathcal{S}$.
3. $\mathcal{S}$ sends $(\textbf{Verified}, sid, d, \phi)$ to $\mathcal{F}_{SIG}$.
4. Upon receiving $(\textbf{Verified}, sid, d, \phi)$, $\mathcal{F}_{SIG}$ works as follows.
   1. If $\widetilde{\theta} = \theta$ and there exists the record $(d, \sigma, \theta, 1)$, $f \leftarrow 1$.
   2. If $\widetilde{\theta} = \theta$, $\mathcal{P}$ has not yet been corrupted by $\mathcal{S}$, and there exists no record such that $(d, \widetilde{\sigma}, \theta, 1)$ for $\forall\widetilde{\sigma}$, $f \leftarrow 0$.
   3. If $\widetilde{\theta} \neq \theta$ and there exists the record $(d, \sigma, \widetilde{\theta}, \widetilde{f})$, $f \leftarrow \widetilde{f}$.
   4. Else, $f \leftarrow \phi$, then records $(d, \sigma, \widetilde{\theta}, \phi)$.
5. $\mathcal{F}_{SIG}$ sends $(\textbf{Verified}, sid, d, f)$ to $\mathcal{V}$.

Canetti also defined $\mathcal{F}_{KE}$ as follows.

Functionality $\mathcal{F}_{KE}$:

1. Let $\mathcal{P}_i$ and $\mathcal{P}_j$ be two parties who want to share a key. $\mathcal{P}_i$ sends $(\textbf{exchange}, sid, \mathcal{P}_i, \mathcal{P}_j, \beta)$ to $\mathcal{F}_{KE}$.
2. $\mathcal{P}_j$ sends $(\textbf{exchange}, sid, \mathcal{P}_i, \mathcal{P}_j, \beta')$ to $\mathcal{F}_{KE}$.
3. Upon receiving both messages, $\mathcal{F}_{KE}$ works as follows.
   1. If $\beta = \beta' = \perp$, $\kappa \leftarrow \{0,1\}^k$.
   2. If $\beta \neq \perp$, $\kappa \leftarrow \beta$.
   3. Else, $\kappa \leftarrow \beta'$.
4. $\mathcal{F}_{KE}$ sends $(\textbf{Key}, sid, \kappa)$ to $\mathcal{P}_i$ and $\mathcal{P}_j$, and sends $(\textbf{Key}, sid, \mathcal{P}_i, \mathcal{P}_j)$ to $\mathcal{S}$.

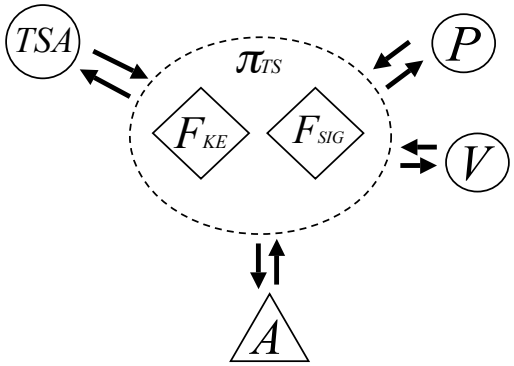

Fig. 5. Hybrid construction of time-stamping protocol π_{TS}

3.2 Construction of π_{TS}

Let $Enc(\cdot,\cdot)$ be an ideal symmetric encryption algorithm and $Dec(\cdot,\cdot)$ be a corresponding decryption algorithm. We denote the concatenation of a and b by $a \cdot b$. We construct π_{TS} by applying hybrid-model as follows (Fig. 5).

Key generation:

1. $\mathcal{TSA}$ sends (**Key**, sid) to $\mathcal{F}_{SIG}$ and then executes the key generation process of $\mathcal{F}_{SIG}$.
2. $\mathcal{TSA}$ obtains (**VerificationKey**, sid, θ) and then outputs (**SetUpDone**, sid, θ).

Time-stamp token generation:

1. Time-stamp requester $\mathcal{P}$ sends (**Start**, $sid, \mathcal{P}, \mathcal{TSA}$) to $\mathcal{TSA}$.
2. $\mathcal{P}$ and $\mathcal{TSA}$ share session key κ with $\mathcal{F}_{KE}$.
3. $\mathcal{P}$ computes $C = Enc(\kappa, d)$ and then sends it to $\mathcal{TSA}$.
4. Let $M = t \cdot C \cdot ID_{\mathcal{P}}$. $\mathcal{TSA}$ executes the signature generation process of $\mathcal{F}_{SIG}$ with input (**Sign**, sid, M) and then obtains (**Signature**, sid, M, σ).
5. $\mathcal{TSA}$ sends (**Signature**, sid, M, σ) to $\mathcal{P}$.
6. $\mathcal{P}$ parses M as $M = (t', C', ID'_{\mathcal{P}})$. If $C' = C$ and $ID'_{\mathcal{P}} = ID_{\mathcal{P}}$, $\mathcal{P}$ accepts σ' as the signature. Else, $\mathcal{P}$ rejects it.

Time-stamp token verification:

1. Verifier $\mathcal{V}$ sends (**Verify**, $sid, t \cdot C \cdot ID_{\mathcal{P}}, \sigma, \widetilde{\theta}$) to $\mathcal{F}_{SIG}$, and $\mathcal{V}$ executes the verification process of $\mathcal{F}_{SIG}$.

2. $\mathcal{V}$ obtains $(\textbf{Verified}, sid, t{\cdot}C{\cdot}ID_{\mathcal{P}}, f)$ and then $\mathcal{V}$ outputs f if $C = Enc(\kappa, d)$. Otherwise it outputs 0.

In the following, we show that π_{TS} securely realizes $\mathcal{F}_{TS}$ in the UC secure sense.

Theorem 1. *In the $(\mathcal{F}_{KE}, \mathcal{F}_{SIG})$-hybrid model, π_{TS} securely realizes $\mathcal{F}_{TS}$ in the UC secure sense for any adversary.*

Proof. Let $\mathcal{A}$ be an adversary that interacts with entities running π_{TS}. We construct a simulator $\mathcal{S}$ such that the view of any environment $\mathcal{Z}$ of an interaction with $\mathcal{A}$ and π_{TS} is distributed identically to its view of an interaction with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{TS}$. As usual, simulator $\mathcal{S}$ runs an internal copy of $\mathcal{A}$ and of each of the involved parties. All messages from $\mathcal{Z}$ to $\mathcal{A}$ are written to $\mathcal{A}$'s input tape. In addition, $\mathcal{S}$ does the followings.

Simulating key generation:
On receiving message $(\textbf{SetUp}, sid)$ from $\mathcal{F}_{TS}$, $\mathcal{S}$ simulates the key generation protocol of π_{TS}. That is,

1. $\mathcal{S}$ sends $(\textbf{KeyGen}, sid)$ to $\mathcal{A}$ and then obtains its return $(\textbf{Verificationkey}, sid, \theta)$.
2. $\mathcal{S}$ records $(\mathcal{TSA}, \theta)$ and sends $(\textbf{SetUpDone}, sid, \theta)$ to $\mathcal{F}_{TS}$.

Simulating time-stamp token generation:
On receiving message$(\textbf{StampReq}, sid, |d|)$ from $\mathcal{F}_{TS}$ where the requester is $\mathcal{P}$, $\mathcal{S}$ simulates the token generation protocol of π_{TS} before step 4. That is,

1. $\mathcal{S}$ simulates $\mathcal{F}_{KE}$, generates a random session key κ, and sends $(\textbf{Key}, sid, \mathcal{TSA}, \mathcal{P})$ to $\mathcal{A}$.
2. $\mathcal{S}$ chooses $C \xleftarrow{R} \{0,1\}^*$ and records the tuple $((\textbf{StampReq}, sid, |d|), \kappa, C)$ in its list. $\mathcal{S}$ returns $(\textbf{OK}, sid)$ to $\mathcal{F}_{TS}$.
3. On receiving message $(\textbf{StampReq}, sid, t)$ from $\mathcal{F}_{TS}$, $\mathcal{S}$ simulates the token generation of π_{TS} after step 3. That is, $\mathcal{S}$ sets $M = t \cdot C \cdot ID_{\mathcal{P}}$ and sends $(\textbf{Signature}, sid, M)$ to $\mathcal{A}$. On receiving the tuple $(\textbf{Signature}, sid, M, \sigma)$ from $\mathcal{A}$, $\mathcal{S}$ sends $(\textbf{StampDone}, sid, M, \sigma, \kappa, C)$ to $\mathcal{F}_{TS}$.

Simulating time-stamp token verification:
On receiving message $(\textbf{Ver}, sid, \alpha)$ from $\mathcal{F}_{TS}$, where $\alpha = (t, d, ID_{\mathcal{P}}, \sigma, \widetilde{\theta}, \widetilde{k}, \widetilde{C})$, $\mathcal{S}$ simulates the verification protocol. That is,

1. $\mathcal{S}$ sends $(\textbf{Verify}, sid, \widetilde{M}, \sigma, \widetilde{\theta})$ to $\mathcal{A}$ and obtains its return, $(\textbf{Verified}, sid, \widetilde{M}, \phi)$, where $\widetilde{M} = t \cdot \widetilde{C} \cdot ID_{\mathcal{P}}$.
2. $\mathcal{S}$ simulates $\mathcal{F}_{SIG}$, verifies the signature, and records $(\widetilde{M}, \sigma, \widetilde{\theta}, f)$ in its list. $\mathcal{S}$ returns $(\textbf{VerDone}, sid, \alpha, \phi)$ to $\mathcal{F}_{TS}$.

Simulating requester corruption:
When $\mathcal{A}$ corrupts a requester, $\mathcal{S}$ corrupts that requester in the ideal process, and obtains the set of documents $\{d\}$ that held by the requester. $\mathcal{S}$ sends the

documents to $\mathcal{A}$.

Since we assume that $Enc(\cdot,\cdot)$ is an ideal symmetric encryption, the distribution of ciphertext C of (κ, d) in the simulation is identical to the actual one. It is obvious that $\mathcal{S}$ can obtain all secret information of a requester if $\mathcal{A}$ corrupts it; therefore, any $\mathcal{Z}$ can not distinguish which adversary and requesters ($\mathcal{A}$ and $\mathcal{P}$ / $\mathcal{S}$ and $\mathcal{F}_{TS}$) she accesses. This concludes the proof.

4 Discussion

In the time-stamp token verification of π_{TS}, verifier $\mathcal{V}$ first verifies signature σ and then checks the validity of ciphertext C. Since we defined that the time-stamp token verification of $\mathcal{F}_{TS}$ follows the verification of $\mathcal{F}_{SIG}$, we need this setting to prove Theorem 1. If verifier $\mathcal{V}$ firstly checks the validity of C, $\mathcal{V}$ can reject the signature without activating the verification of $\mathcal{F}_{SIG}$ in a case of C is invalid; however, the proof fails with this setting. This is because $\mathcal{S}$ can not check the validity of C when $\mathcal{V}$ (i.e., $\mathcal{Z}$) sends a verification query without activating $\mathcal{F}_{SIG}$ (i.e., $\mathcal{A}$).

However, it is natural that $\mathcal{V}$ firstly checks the validity of the ciphertext and then verifies the signature. To implement this setting, we slightly modify the definition of the time-stamp token verification of $\mathcal{F}_{TS}$ as follows.

Time-stamp token verification:
Let $f \in \{0, 1, *\}$, $\phi \in \{0, 1\}$.

1. Verifier $\mathcal{V}$ sends $(\mathbf{Ver}, sid, \alpha)$ to $\mathcal{F}_{TS}$, where $\alpha = (t, d, ID_{\mathcal{P}}, \sigma, \widetilde{\theta}, \widetilde{\rho})$.
2. $\mathcal{F}_{TS}$ executes the following. (1) If $(\widetilde{\theta}, \widetilde{\rho}) = (\theta, \rho)$ and $\mathcal{F}_{TS}$ has already recorded $(\alpha, 1)$, $f \leftarrow 1$. (2) If $(\widetilde{\theta}, \widetilde{\rho}) = (\theta, \rho)$, $\mathcal{S}$ does not corrupt $\mathcal{TSA}$, and $\mathcal{F}_{TS}$ has not recorded $(t, d, ID_{\mathcal{P}}, \sigma', \theta, \rho, 1)$ for $\forall\sigma'$, $f \leftarrow 0$ and $\mathcal{F}_{TS}$ records $(\alpha, 0)$. (3) If $(\widetilde{\theta}, \widetilde{\rho}) \neq (\theta, \rho)$ and $\mathcal{F}_{TS}$ has already recorded $(\alpha, \widetilde{f})$, $f \leftarrow \widetilde{f}$. (4) Otherwise, $f \leftarrow *$.
3. $\mathcal{F}_{TS}$ sends $(\mathbf{Ver}, sid, \alpha, f)$ to $\mathcal{S}$.
4. $\mathcal{S}$ sends $(\mathbf{VerDone}, sid, \alpha, \phi)$ to $\mathcal{F}_{TS}$.
5. If $f = *$, then $\mathcal{F}_{TS}$ sets $f = \phi$ and records (α, ϕ).
6. $\mathcal{F}_{TS}$ sends $(\mathbf{VerDone}, sid, t, d, ID_{\mathcal{P}}, f)$ to $\mathcal{V}$.

With this setting, $\mathcal{S}$ knows the validity of signature σ even if it does not know the corresponding document d. Therefore, $\mathcal{S}$ can simulate the verification.

This modified setting does not provide any advantage to the adversary since she can record all message-signature pairs related to the verification key that she chooses; therefore, she can verify the signatures herself. In most standard digital signature schemes, the verification does not require extraneous communication; therefore, this indeed corresponds to our intuitive notion of a signature verification process. We can apply this to the definition of the verification of $\mathcal{F}_{SIG}$.

In this study, we separate the management of an accurate time t (or a counter value, a hash value, etc.) from the issuing of a time-stamp token, and we defined functionality $\mathcal{F}_{TS}$ for the token issuing. Hence, $\mathcal{F}_{TS}$ does not check the validity of t. If TSA manages both t and token issuing, $\mathcal{F}_{TS}$ is defined such that it checks the validity of t by comparing it with the latest t' recorded in its register. In the linking protocol, a verifier can verify t even if an adversary corrupts the TSA. On the other hand, in the time-stamping protocol based on a digital signature scheme, such as the simple protocol, it is an open question of how to implement the functionality needed for verifying of t.

5 Conclusion

In this paper, we reconsidered the security notions of the time-stamping protocol and defined its functionality based on the UC framework. Our definition follows that of the signature functionality defined by Canetti. In addition, we described the construction of a secure time-stamping protocol using the key exchange functionality and the signature functionality. We also showed security proof of our proposed protocol in UC framework.

References

[ACPZ01] C. Adams, P. Cain, D. Pinkas and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)," IETF RFC3161.

[BLLV98] A Buldas, P. Laud, H. Lipmaa and J. Villemson, "Time-stamping with Binary Linking Schemes," In Proc. of *CRYPTO98*, LNCS 1462, pp.486-501, Springer-Verlag, 1998.

[BLS00] A. Buldas, H. Lipmaa and B. Schoenmakers, "Optimally Efficient Accountable Time-stamping," In Proc. of *PKC 2000*, LNCS 1751, pp.293-305, Springer-Verlag, 2000.

[C01] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," available at http://eprint.iacr.org/2001

[C04] R. Canetti, "Universally Composable Signatures, Certification, and Authentication," In Proc. of *the 17th Computer Security Foundations Workshop (CSFW'04).*

[CF01] R.Canetti and M.Fischlin, "Universally Composable Commitments," Extended version of the paper that appeared at *CRYPTO 2001.*

[CK02] R.Canetti and H.Krawczyk, "Universally Composable Notions of Key Exchange and Secure Channels," Extended version of the paper that appeared at *EUROCRYPT 2002*, pages 337-351.

[HS91] S. Haber and W. S. Stornetta, "How to Time-stamp a Digital Document," *Journal of Cryptology: the Journal of the International Association for Cryptologic Research 3, 2 (1991)*, pages 99-111.

[ISO] ISO/IEC 18014-1, 18014-2 and 18014-3, Information technology – Security techniques – Time-stamping services – Part 1, Part 2, and Part 3.

[Just98] M. Just, "Some Timestamping Protocol Failures", In Proc. of *the Symposium on Network and Distributed Security (NDSS98)*, San Diego, CA, USA, Mar. 1998, Internet Society.

[MO04] S. Matsuo and H. Oguro, "User-side Forward-dating Attack on Time-stamping Protocol," In Proc. of *the 3rd International Workshop for Applied Public Key Infrastructure (IWAP'04)*, pages 72-83.

[Sure] http://www.surety.com

[UM02] M. Une, and T. Matsumoto, "A Framework to Evaluate Security and Cost of Time Stamping Schemes," *IEICE Transactions on Fundamentals*, EA85-A, No.1, pp. 125-139, 2002.