Securing data accountability in decentralized systems

Ricardo Corin[1] and David Galindo[2] and Jaap-Henk Hoepman[2]

[1] University of Twente, Enschede, The Netherlands `corin@cs.utwente.nl`
[2] Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands {`d.galindo,jhh`}`@cs.ru.nl`

Abstract. We consider a decentralized setting in which agents exchange data along with usage policies. Agents may violate the policies, although later on auditing authorities may verify the agents' accountability with respect to the used data. Using (provably secure) time-stamping and signature schemes, we design and analyze an efficient cryptographic protocol that generates the sufficient communication evidences which auditing authorities need to carry out their auditing. Finally, we describe an extension providing policy confidentiality against eavesdroppers time-stamping servers.

1 Introduction

In many situations, there is a need to share data between potentially untrusted parties while ensuring the data is used according to given policies. For example, Alice may be interested in sending her e-mail address to Bob, but also attaching a non-disclosure policy, so that Bob may not disclose Alice's email to anyone else.

Of course, a priori nothing guarantees that Bob will actually follow the policy, making the enforcement of such policies a difficult problem. *Access and usage control* [JSSB97,SS94,PS02,BCFP03] are exemplary enforcement mechanisms. Here, a trusted access control service restricts data access at the moment the request happens, something that can be sometimes overly restrictive and expensive.

Recently, a more flexible approach has been proposed [CEdH$^+$04,CCD$^+$05]. In this approach, after Bob receives Alice's e-mail address, Bob is free to violate the policy, for instance by sending Alice's e-mail to Charlie and including a "free-to-be-spammed" policy. However, it could happen that later on Bob is *audited* by an authority that requests a convincing proof of Bob's permission to disclose Alice's e-mail address. Auditing authorities are not fixed and pre-established; they may be formed dynamically by (groups of) agents that observe actions in the system. For example, consider that Alice starts getting tons of spam from Charlie. Alice may switch to "auditing authority" mode, and enquiry Charlie for a proof of permission to use Alice's e-mail address. A convincing proof exonerating Charlie would be evidence of the communication with Bob which happened before the spam, in which Charlie got Alice's e-mail address along with a "free-to-be-spammed" policy. Upon seeing this, Alice can go to Bob and request a convincing proof he was allowed to disclose Alice's e-mail to Charlie, which Bob can not provide and hence is found guilty of policy violation. (In this case, we say that Bob violates the *datum accountability* of Alice's e-mail address.)

Following [CEdH$^+$04,CCD$^+$05], we allow the target system to be decentralized, and it may very well be the case that Alice, Bob and Charlie are registered in different domains, which are not even synchronized in their local clocks; even worse, each domain's clock may not even be keeping real time, but may be simply a logical clock (e.g. a counter). Still, an auditing authority would need to recognize events (e.g. communications) that happened before others. For example, the fact that Charlie sent spam to Alice only after having received the permission from Bob exonerates Charlie. Also, an auditing authority needs to be able to recognize valid communication evidence from fake ones; for example, Bob must not be able to repudiate the communication with Charlie.

CONTRIBUTIONS The works of [CEdH$^+$04,CCD$^+$05] focus on designing a high-level formal proof system to build authorization permissions starting from the atomic communication evidences, which are abstracted away and assumed to be sound (this is analogous to Dolev-Yao models [DY83,CES06] which assume "perfect encryption" in the analysis of security protocols). However, from the above discussion it is clear that it is not trivial to design an effective and secure cryptographic protocol to achieve valid communication evidences to let auditing authorities do their work, as we are in a decentralized setting in which different agents may collude in order to fool the auditing authorities.

In this paper, we design such a cryptographic protocol, which allows agents to exchange usage policies among them in such a way that valid communication evidence is generated after the exchange. Our protocol is efficient, and it's based on standard cryptographic building blocks, as detailed next.

BUILDING BLOCKS We use two building blocks in our protocol: time-stamping and signature schemes. We choose to model different agents' domains running different local clocks, which may be logical, as described above. Our model is quite flexible, and in particular brings the following advantages:

– First, we allow the protocol to work in decentralized environments, in which clocks domains do not need to interact between them;
– Second, it is possible to use both digital-based and hash-based time-stamping schemes (see [ABSW01] for an overview), the choice depending on the use the implementor has in mind. For instance, if the implementor prefers to regard the time-stamping servers as non-trusted, auditable time-stamping schemes can be used without loss of security. Time-stamping is used to order the events within the domain, and enables our cryptographic protocol to safely logs the different domains' times of the sender and the receiver into the communication evidence. At a later stage, the auditing authority can use this information for the auditing procedure. Signatures by the agents

are used to provide integrity, authentication and non-repudiability to the communication evidences.

Plan of the paper We introduce preliminaries and define the security requirements of the protocol in Section 2. The building blocks of the protocol (i.e. time-stamping services and signature schemes) are introduced in Section 3. We present the protocol in Section 4, along with an (informal) security analysis in Section 5. Section 6 discusses extensions, and finally Section 7 concludes the paper.

2 The setting

Potentially untrusted agents $a, b, c \in \mathcal{G}$ are sharing data objects $d \in \mathcal{D}$ which must be used according to policies $\phi \in \Phi$. The agents are not enforce to comply with the policies, but they can be audited at any time by an Authorization Authority AA which will check if they were allowed to perform a certain action on a given piece of data d at a certain moment in time. To be *accountable*, agents must show a policy ϕ allowing them to perform the action under investigation. If an agent c is the *creator* of the data, he is allowed to perform any action on the data; otherwise, the agent must show that another agent b sent him the policy ϕ and therefore the permission to perform the actions specified therein. Eventually, the AA could iterate the process starting from agent b, and ideally the sequence should led to the owner/creator of the piece of data.

We do not study how to build evidences for the creation of data. For the purpose of this paper, certifying that an agent owns/creates a piece of data needs the interaction with several organizations external to the system, since it involves legal and societal aspects (as an example, an organization issuing patents). Therefore, we assume there exists a public function

$$\textsc{Owner} : \mathcal{D} \longrightarrow \mathcal{G}. \tag{1}$$

stating to whom data belongs.

In our setting, $\mathsf{comm}(t_1, t_2, a \Rightarrow b, \phi)$ provides a proof that a string ϕ was sent from a to b with respect to some local logical time values t_1, t_2, where t_1 refers to the sender local time and t_2 refers to the receiver local time. The properties that such a proof should satisfy are as follows:

Meaningfulness. $\mathsf{comm}(t_1, t_2, b \Rightarrow c, \phi)$ provides a proof that a string ϕ has been communicated from b to c.

The reason for this property is trivial. We want to be sure that a communication evidence between Bob and Charlie can only be generated if Bob has communicated ϕ to Charlie.

Unforgeability. It is not feasible to create an evidence $\mathsf{comm}(t_3, t_4, a \Rightarrow b, \phi')$ either when the local logical clock for b is $t_4' > t_4$ or when the local logical clock for a is $t_3' > t_3$.

Assume the auditing authority contacts Bob at Bob's logical time t'_4 and asks him for a permission for having sent Alice's e-mail address to Charlie. The unforgeability property implies Bob can not provide an exonerating evidence even in the unlikely case he colludes with Alice, since exoneration would require an evidence of the form $\mathsf{comm}(t_3, t_4, a \Rightarrow b, \texttt{spread Alice's e-address})$ to be created at time t'_4 with $t_4 < t'_4$.

Liability. A valid $\mathsf{comm}(t_1, t_2, b \Rightarrow c, \phi)$ implies b has committed itself at a logical time t_1 to send ϕ to c.

That is, if Charlie shows an evidence $\mathsf{comm}(t_1, t_2, b \Rightarrow c, \phi)$ showing that Bob sent him a permission to spam Alice at Bob's logical time t_1, then Bob is liable for showing he had permission to communicate this policy before time t_1.

Comparability. Any pair of communication evidences $\mathsf{comm}(t_1, t_2, b \Rightarrow c, \phi)$ and $\mathsf{comm}(t_3, t_4, a \Rightarrow b, \phi')$, with a, b, c agents, should be comparable with respect to b's local time, i.e. communication evidences with origin or destination b are totally ordered with respect to b's local logical time.

With this property, we ensure Bob can not show an evidence allowing him to execute a certain action, such that the order of action's permission reception and action/s execution is undetermined.

3 Building blocks

3.1 Time-stamping schemes

Time-stamping is an important data integrity protection mechanism the main objective of which is to prove that electronic records existed at a certain time. Two major time-stamping protocols, the absolute (hash-and-sign) time stamps protocol and the linking protocol have been developed. In the former a time-stamping authority (TSA) signs the concatenation of the hashed message and the present time by using its private key. Therefore the TSA is completely trusted in this case.

In the linking protocol, a time-stamp token is the hash value of the concatenation of the present hashed message and the previous hash value. 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 trusted, and auditing techniques can be used to detect if the TSA eventually cheated. We stress this audit does not need to be performed by our auditing authority; it is sufficient the AA trusts the time-stamping auditors.

Time-stamping schemes have been used in business applications and are even included in international standards [ISO]. The major schemes in use are [sur,aut,dig].

The formal security conditions for time-stamping schemes are still a subject under discussion. In the following, we quote the syntax and security properties of a time-stamping scheme from [BLSW05]. Notice that the definitions below are abstracted away, so that the different concrete implementations found in the literature can be easily translated into our syntax. Additionally, an audit functionality can be added to time-stamping schemes with the syntax above. This auditing functionality is used when the TSA is not unconditionally trusted, and it allows to check that the TSA is behaving as expected.

Definition 1 *A time-stamping scheme* TS *is capable of: (1) assigning a time-value* $t \in \mathbb{N}$ *to each request* $x \in \{0,1\}^k$, *and (2) verifying whether* x *was time-stamped during the* t*-th (maybe logical) time unit). It consists of the following components:*

- REPOSITORY – *a write only database that receives* k*-bits digests and adds them to a list* $\mathcal{D}$. REPOSITORY *also receives queries* $t \in \mathbb{N}$ *and returns* $\mathcal{D}[t]$ *if* $\tau \leq |\mathcal{D}|$. *Otherwise,* REPOSITORY *returns* reject.
- STAMPER – *operates in discrete time variables called* rounds. *During a* t*-th round,* STAMPER *receives requests* x *and returns pairs* (x,t). *Let* L_t *be the list of all requests received during the* t*-th round. In the end of the round,* STAMPER *creates a certificate* $c = \mathsf{Stamp}(x; L_t, L_{t-1}, \ldots, L_1)$ *for each request* $x \in L_t$. *Besides,* STAMPER *computes a digest* $d_t = \mathsf{Publish}(L_t, \ldots, L_1)$ *and sends* d_t *to the repository.*
- VERIFIER – *a computing environment for verifying time stamps. It is assumed that* VERIFIER *has a tamper-proof access to* REPOSITORY. *On input* (x,t), VERIFIER *obtains a certificate* c *from* STAMPER, *and a digest* $d = \mathcal{D}[t]$ *from* REPOSITORY, *and returns* $\mathsf{Verify}(x,c,d) \in \{\mathtt{yes}, \mathtt{no}\}$. *Note that* x *can be verified only after the digest* d_t *is sent to* REPOSITORY.
- CLIENT – *any application-environment that uses* STAMPER *and* VERIFIER.

A time-stamping scheme $\mathsf{TS} = (\mathsf{Stamp}, \mathsf{Publish}, \mathsf{Verify})$ *must satisfy the following correctness property:* $\mathsf{Verify}(x, \mathsf{Stamp}(x, \mathcal{L}), \mathsf{Publish}(\mathcal{L})) = \mathsf{yes}$ *for every* $\mathcal{L} = (L_t, \ldots, L_1)$ *and* $x \in L_t$.

Security requirements

In the rest of the paper, an adversary is meant to be any probabilistic polynomial time algorithm. It is assumed that adversaries A is able to corrupt STAMPER, some instances of CLIENT and VERIFIER. The REPOSITORY is assumed to be non-corrupting. After publishing d_t it should be impossible to add a new request x to L_t and prove to a VERIFIER that $x \in L_t$ by building up a certificate c. The following security conditions must be then required:

Definition 2 (Consistency) *A time-stamping scheme is* consistent *if for every PPT adversary* A

$\Pr[\mathcal{L} = (L_t, \ldots, L_1, c, x) \leftarrow \mathsf{A}(1^k) \mid x \notin L_t,\ \mathsf{Verify}(x, c, \mathsf{Publish}(L_t, \ldots, L_1)) = \mathsf{yes}]$ *is negligible.*

With the security notion below, we want that an adversary A can not perform the following attack: A publishes a value d which is not computed by using the $\mathsf{Publish}$ function and then, after obtaining a new randomly generated string x, finds a certificate that $\mathsf{Verify}(x,c,d) = \mathsf{yes}$.

Definition 3 (Security against random back-dating) *A time-stamping scheme is* secure against random back-dating *if for every polynomially unpredictable distribution $\mathcal{D}$ on $\{0,1\}^k$ and $(\mathsf{A}_1, \mathsf{A}_2)$ probabilistic polynomial time algorithms*

$$\Pr[(d,a) \leftarrow \mathsf{A}_1(1^k),\ x \leftarrow \mathcal{D},\ c \leftarrow \mathsf{A}_2(x,a) \mid \mathsf{Verify}(x,c,d) = \mathtt{yes}] \quad \text{is negligible.}$$

3.2 Signature scheme

Definition 1. *A signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{VerSign})$ consists of three probabilistic polynomial time algorithms:*

- KeyGen *takes as input a security parameter 1^k, and outputs a pair (vk, sk), where sk is the secret key of the user, and vk is the matching verification key.*
- Sign *takes as input a message m and the secret key sk, and produces a signature σ.*
- $\mathsf{VerSign}$ *finally, the verification algorithm takes as input a message m, a signature σ and the verification key vk, and returns* `true` *if σ is a valid signature of m, and* `false` *otherwise.*

A signature scheme enjoys the correctness property if it satisfies the following condition: if $\mathsf{KeyGen}(1^k) = (sk, vk)$ and $\mathsf{Sign}(m, sk) = \sigma$, then $\mathsf{VerSign}(m, \sigma, vk) =$ `true`. *In this case, we say that (m, σ) is a valid message-signature pair.*

The standard security notion for signature schemes was introduced in [GMR88] and it is called *existential unforgeability against chosen-message attacks.* A signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{VerSign})$ is called secure in the latter sense if the success probability of any PPT adversary $\mathcal{A}$ in the following game is negligible in the security parameter 1^k:

1. $\mathsf{KeyGen}(1^k)$ outputs (vk, sk) and the adversary is given 1^k and vk.
2. $\mathcal{A}(1^k, vk)$ has access to a signing oracle $\mathsf{Sign}(sk, \cdot)$, which on input a message m outputs its signature $\sigma(m)$.
3. $\mathcal{A}$ succeeds if it outputs a valid signature on a message not submitted to the signing oracle.

4 A communication evidence protocol

Our goal is to design a decentralized protocol performing as less on-line operations as possible. Our system includes certification authorities CA trusted by the auditing authority, which will be used for keys authenticity and non-repudiation purposes. The AA can choose to trust/distrust the time-stamping servers. This election will determine which time-stamping schemes are accepted in the system (cf. the discussion in Section 3.1). The AA is an algorithm taking inputs from the agents, but it does not need itself to be in possession of any special input like a public key or similar.

Let us assume the existence of a set of time-stamping and certification authorities satisfying the trust requirements imposed by the AA, as well as the existence of a public board in which each user is inscribed in an unique TSA and CA. Let us denote by TSA_a, CA_a and TSA_b, CA_b the authorities in which a and b are respectively registered. A time-stamping scheme is used to provide temporal evidence to bit strings. In our setting consists each time stamping authority TSA_a runs a local time variable. We say that $\mathcal{T}_a$ is the local time managed by TSA_a. Despite the local time variables are run in a decentralized manner, we are still able to define an irreflexive partial ordering (denoted by $\prec$) between events in the different $\mathcal{T}_a$'s. Notice that every local time variable has a total order $<$, i.e. the natural ordering in the set $\mathbb{N}$.

The partial order is defined in terms of the relation 'D_1 *existed before* D_2', where D_1, D_2 are strings. The relation is determined via time-stamps: a valid time-stamp certificate c^a issued on round t^a by TSA_a on a string m, implies m existed before round t^a was closed. As a consequence, a time-stamp c^b issued on round t^b by TSA_b on a string m' such that c^b is a substring of m', implies that c^b existed before round t^b was closed. Summing up, the t^a-th round in TSA_a ended before the t^b-th round in TSA_b did so. This enables to establish the partial order $t^a \prec t^b$.

Any agent a in the system has a unique pair of matching verification/signature keys (vk_a, sk_a) corresponding to a secure signature scheme Σ_a, and it is registered in a *single* TSA. We will refer to this as a is registered in TSA_a. We assume that these keys are revocable. Therefore, agents have access to certification authorities CA which ensure the verification key vk_a belongs to a and provide revocation mechanisms.

A communication evidence $\mathsf{comm}(t_1^a, t_2^b, a \Rightarrow b, \phi)$ must satisfy the requirements outlined in Section 2: meaningfulness, unforgeability, liability and comparability. The temporal tag includes two temporal values; a first value t_1^a will prevent non-repudiation by a (and therefore makes a liable for having permission to communicate ϕ at 'time' t_1^a), and a second value t_2^b will refer to the moment in which b is allowed to use the policy ϕ. The value t_2^b will prevent forging evidences by b even if a and b collude together.

Definition 4 (Syntaxis of our communication evidence protocol) *A* communication evidence protocol *is a pair of functions* $\mathsf{CE} = (\mathsf{Create}, \mathsf{Validate})$.

- Create *The parties initiating a run of the protocol* Create *are two agents* $a, b \in \mathcal{A}$, *where* a *is willing to send a policy* ϕ *to* b *and* b *is willing to receive and therefore use this policy. The protocol additionally involves their respective time-stamping authorities* $\mathrm{TSA}_a, \mathrm{TSA}_b$*; and the certification authorities for* a, b *which are denoted by* $\mathrm{CA}_a, \mathrm{CA}_b$. *The output is a communication evidence* $\mathsf{comm}(t_1^a, t_2^b, a \Rightarrow b, \phi)$.
- $\mathsf{Validate}$ – *can be run by any agent in the system, and requires interaction with* TSA_i *and* TSA_j *and the certification authorities agents* $\mathrm{CA}_a, \mathrm{CA}_b$. *It takes as input a communication evidence* $\mathsf{comm}(t_1^a, t_2^b, a \Rightarrow b, \phi)$, *and it returns* `true` *if* comm *is valid, and* `false` *otherwise.*

4.1 Protocol specification

In order to be able to prevent non-repudiation of communication evidences, we need to slightly modify the revocation mechanism used by the certification authority. In particular, the certification authorities must ask the TSA in which the user is registered to time-stamp the revocation information for that user's public key. In this way, it is possible to check if a a communication evidence was created before either the sender's verification key vk_a or the receiver's verification key vk_b were revoked.

- $\mathsf{Create}(a, b, \phi)$:
  1. a signs the concatenated string (ϕ, b) using the scheme Σ_a and the signing key sk_a. Let σ_1 denote the signature thus obtained.
  2. a sends σ_1 to $\mathsf{Stamper}_a$, and gets back a valid stamp (t_1^a, c_1^a) when the t_1^a-th round is closed.
  3. a sends $ev1 := (a, vk_a, b, \phi, \sigma_1, t_1^a, c_1^a)$ to b.
  4. b verifies that:
     (a) vk_a is a's verification key.
     (b) vk_a was not revoked before TSA$_a$'s local time t_1^a (this is done by interacting with CA$_a$ and TSA$_a$).
     (c) $\mathsf{VerSign}\big((\phi, b), \sigma_1, vk_a\big) = \texttt{true}$.
     (d) $\mathsf{Verify}(\sigma_1, c_1^a, d_{t_1^a}) = \texttt{yes}$, where $d_{t_1^a}$ is the corresponding entry in REPOSITORY$_a$.

     If everything is fine, then b proceeds to the next step. Otherwise, b does not use policy ϕ.
  5. b signs $ev1$ using sk_b. Let σ_2 denote the signature thus obtained.
  6. a sends σ_2 to $\mathsf{Stamper}_b$, and gets back a valid stamp (t_2^b, c_2^b) when the t_2^b-th round is closed (and therefore $t_1^a \prec t_2^b$). Let $ev2 := (b, vk_b, \sigma_2, t_2^b, c_2^b)$.
  7. Finally
$$\mathsf{comm}(t_1^a, t_2^b, a \Rightarrow b, \phi) := (ev1, ev2).$$
- $\mathsf{Validate}\big(a, b, \phi, \mathsf{comm}(t_1^a, t_2^b, a \Rightarrow b, \phi)\big)$:
  1. Contact TSA$_b$ and get the value T_b of the current (non-closed) round.
  2. Verify that:
     (a) $\mathsf{VerSign}\big((\phi, b), \sigma_1, vk_a\big) = \texttt{true}$.
     (b) vk_a is the a's verification key.
     (c) vk_a was not revoked before TSA$_a$'s local time t_1^a.
     (d) $\mathsf{Verify}(\sigma_1, c_1^a, d_{t_1^a}) = \texttt{yes}$, where $d_{t_1^a}$ is the corresponding entry in REPOSITORY$_a$.
     (e) vk_b is the b's verification key.
     (f) vk_a was not revoked before TSA$_b$'s local time t_2^b.
     (g) $\mathsf{VerSign}\big(ev_2, \sigma_2, vk_b\big) = \texttt{true}$.
     (h) $\mathsf{Verify}(\sigma_2, c_2^b, d_{t_2^b}) = \texttt{yes}$, where $d_{t_2^b}$ is the corresponding entry in REPOSITORY$_b$.
     (i) $t_2^b < T_b$.

     If every checking is correct, then return `true`. Otherwise, return `false`.

5 Security analysis

Meaningfulness. "A string ϕ has been communicated from a to b"
Firstly, if the communication evidence is verified in the positive, neither a's or b's verification keys were revoked beforehand. That a is the origin of the communication and b is the receiver, is guaranteed by two facts: on the one hand, a valid signature on the message (ϕ, b) can only be produced by a as long as we are using an unforgeable signature scheme Σ_a; on the other hand, b and only b is able to compute the signature σ_2 for a similar reason, and he can do that only after a sends $ev1$ to him.

Unforgeability. "It is not feasible to create an evidence $\mathsf{comm}(t_1^a, t_2^b, a \Rightarrow b, \phi)$ when the local logical clock for b is set to $t_2'^b$ with $t_2'^b > t_2^b$"
For creating such an evidence at time $t_2'^b > t_2^b$, the adversary must break the security against random back-dating of the time-stamping scheme.

Liability. "a commits itself to send ϕ to b at logical time t_1"
a commits to message (ϕ, b) as soon as he signs it; a valid time-stamp (t_1^a, c_1^a) on σ_1 implies (ϕ, b) was signed before the local time counter at TSA_a was set to t_1^a. Therefore, a expresses at time t_1^a his willingness to transfer ϕ to b if he follows the protocol. Finally, if the communication evidence is verified in the positive, a's verification key was not revoked beforehand.

Comparability. "communication evidences with origin or destination a are totally ordered with respect to a's local logical time"
This is guaranteed by the fact that a's logical time is $\mathcal{T}_a$ and that $\mathcal{T}_a$ has a total order by definition.

6 Extensions

Privacy against curious time-stamping servers. In the likely case that agents want to keep the policies exchanged confidential against an eavesdropping time-stamping server, the protocol proposed in Section 4.1 is not guaranteed to be secure. In particular, it might be very well the case that the signature on a message m could leak some information the agents wish to remain private.

However, it turns out that the protocol can be easily extended to provide confidentiality against time-stamping servers by using commitment schemes. In order to maintain a low number of on-line communications we choose to use non-interactive commitments. A *non-interactive commitment scheme* $\mathcal{C}$ consists of two probabilistic algorithms $\mathcal{C} = (\mathsf{Send}, \mathsf{Receive})$. The algorithm Send is run by the *sender* party, and on input a pair security parameter/message $(1^k, m)$ it outputs a pair commitment/de-commitment (ct, dt). The algorithm $\mathsf{Receive}$ is run by the *receiver,* and on input $(1^k, ct, dt)$ outputs a string m or the reject symbol $\perp$. Executing Send is called the commitment phase while executing $\mathsf{Receive}$ is called the de-commitment phase. Typically, when using a commitment scheme in a protocol, the parties execute the commitment phase first and a later stage the de-commitment phase; there will be some some other parts of the protocol between the two phases.

Roughly speaking, a commitment scheme should ensure that after the commitment phase, the receiver does not know learn about the message yet, but the sender should not be able to change it anymore. After the de-commitment phase the receiver obtains the message. For formal definitions and efficient construction of non-interactive commitment schemes, we refer the reader to [HM96].

The main idea is that agents send to the time-stamping server a commitment on the signatures instead of sending signatures on the clear. Remember that these signatures contain the policies and their destination, so that they could eventually reveal information the agents want to keep confidential.

The extended protocol is defined in Appendix A.

7 Conclusions

We define a cryptographic protocol to provide valid communication evidences, that can be used later by auditing authorities. Even though our protocol is aimed towards guaranteeing data accountability in the settings of [CEdH+04,CCD+05], we believe that our protocol can be easily adapted to provide secure transport of arbitrary payloads in decentralized settings, where exchanges need to be logged with communication evidences recording the relative domain times in which the exchanges took place.

References

[ABSW01] A. Ansper, A. Buldas, M. Saarepera, and J. Willemson. Improving the availability of time-stamping services. In *ACISP 2001*, volume 2119 of *Lecture Notes in Computer Science*, pages 360–375, 2001.

[aut] http://www.authentidate.com/.

[BCFP03] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security (TISSEC)*, pages 71–127, 2003.

[BLSW05] A. Buldas, Peeter Laud, M. Saarepera, and J. Willemson. Universally composable time-stamping schemes with audit. In *ISC 2005*, volume 3650 of *Lecture Notes in Computer Science*, pages 359–373, 2005.

[CCD+05] J. G. Cederquist, R. J. Corin, M. A. C. Dekker, S. Etalle, and J. I. den Hartog. An audit logic for accountability. In A. Sahai and W. H. Winsborough, editors, *6th Int. Workshop on Policies for Distributed Systems & Networks (POLICY), Stockholm, Sweden*, pages 34–43, Los Alamitos, California, June 2005. IEEE Computer Society Press. Imported from DIES.

[CEdH+04] R. Corin, S. Etalle, J. I. den Hartog, G. Lenzini, and I. Staicu. A logic for auditing accountability in decentralized systems. In T. Dimitrakos and F. Martinelli, editors, *Proc. of the 2nd IFIP Workshop on Formal Aspects in Security and Trust (FAST)*, volume 173, pages 187–202. Springer, 2004.

[CES06] R. Corin, S. Etalle, and A. Saptawijaya. A logic for constraint-based security protocol analysis. In *IEEE Symposium on Security and Privacy*, 2006.

[dig] http://www.digistamp.com/.

[DY83] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[GMR88] S. Goldwasser, S. Micali, and R.L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[HM96] S. Halevi and S. Micali. Practical and provably-secure commitment schemes from collision-free hashing. In *CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215, 1996.

[ISO] ISO IEC 18014-3,time-stamping services part 3: Mechanisms producing linked tokens.

[JSSB97] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In J. Peckham, editor, *SIGMOD 1997, Proc. International Conference on Management of Data*, pages 474–485. ACM Press, 1997.

[PS02] J. Park and R. Sandhu. Towards usage control models: Beyond traditional access control. In E. Bertino, editor, *Proc. of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 57–64. ACM Press, 2002.

[SS94] R. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.

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

A A communication evidence protocol providing confidentiality

– $\mathsf{Create}(a, b, \phi)$:
  1. a signs the concatenated string (ϕ, b) using the scheme Σ_a and the signing key sk_a. Let σ_1 denote the signature thus obtained.
  2. a runs $\mathsf{Send}(1^\ell, \sigma_1)$ for an appropriate security parameter 1^ℓ. Let ct_1 and dt_1 denote respectively the commitment and de-commitment thereby obtained.
  3. a sends ct_1 to $\mathsf{Stamper}_a$, and gets back a valid stamp (t_1^a, c_1^a) when the t_1^a-th round is closed.
  4. a sends $ev1 := (a, vk_a, b, \phi, \sigma_1, t_1^a, c_1^a, ct_1, dt_1)$ to b.

  5. b verifies that:
     (a) $\mathsf{Receive}(1^\ell, ct_1, dt_1) = \sigma_1$.
     (b) vk_a is the a's verification key.
     (c) $\mathsf{VerSign}\big((\phi, b), \sigma_1, vk_a\big) = \mathtt{true}$.
     (d) vk_a was not revoked before TSA$_a$'s local time t_1^a.
     (e) $\mathsf{Verify}(ct_1, c_1^a, d_{t_1^a}) = \mathtt{yes}$, where $d_{t_1^a}$ is the corresponding entry in Repository$_a$.

     If everything is fine, then b proceeds to the next step. Otherwise, b does not use policy ϕ.
  6. b signs $ev1$ using sk_b. Let σ_2 denote the signature thus obtained.

7. b runs $\mathsf{Send}(1^\ell, \sigma_2)$ and let ct_2 and dt_2 denote respectively the commitment and de-commitment thereby obtained.
8. a sends ct_2 to $\mathsf{Stamper}_b$, and gets back a valid stamp (t_2^b, c_2^b) when the t_2^b-th round is closed (and therefore $t_1^a \prec t_2^b$). Let $ev2 := (b, vk_b, \sigma_2, t_2^b, c_2^b, ct_2, dt_2)$.
9. Finally
$$\mathsf{comm}(t_1^a, t_2^b, a \Rightarrow b, \phi) := (ev1, ev2).$$

– $\mathsf{Validate}\big(a, b, \phi, \mathsf{comm}(t_1^a, t_2^b, a \Rightarrow b, \phi)\big)$:
1. Contact TSA$_b$ and get the value T^b of the current (non-closed) round.
2. Verify that:
   (a) $\mathsf{Receive}(1^\ell, ct_1, dt_1) = \sigma_1$.
   (b) vk_a is the a's verification key.
   (c) $\mathsf{VerSign}\big((m, \phi), \sigma_1, vk_a\big) = \mathtt{true}$.
   (d) vk_a was not revoked before TSA$_a$'s local time t_1^a.
   (e) $\mathsf{Verify}(ct_1, c_1^a, d_{t_1^a}) = \mathtt{yes}$, where $d_{t_1^a}$ is the corresponding entry in REPOSITORY$_a$.
   (f) $\mathsf{Receive}(1^\ell, ct_2, dt_2) = \sigma_2$.
   (g) vk_b is the b's verification key.
   (h) $\mathsf{VerSign}\big(ev_2, \sigma_2, vk_b\big) = \mathtt{true}$.
   (i) vk_b was not revoked before TSA$_b$'s local time t_2^b.
   (j) $\mathsf{Verify}(ct_2, c_2^b, d_{t_2^b}) = \mathtt{yes}$, where $d_{t_2^b}$ is the corresponding entry in REPOSITORY$_b$.
   (k) $t_2^b < T^b$.

   If every checking is correct, then return `true`. Otherwise, return `false`.