Electronic Submission Protocol Based on Temporal Accountability

Michiharu Kudo
Tokyo Research Laboratory, IBM Japan Ltd.

Abstract

This paper describes various possible attacks on temporal properties such as temporal records of payment times and declarations of the closing times for electronic submissions, and explains defense measures that use a trusted third party to provide temporal accountability. We propose a secure electronic submission protocol as a typical time-sensitive application and a temporal accountability logic, which is an extension of Kailar's work. We analyze the proposed protocol by applying our temporal accountability logic, and describe some modifications of the protocol, which reduce the total number of flows while keeping the protocol as logically secure as the original one in terms of temporal accountability.

1 Introduction

Electronic commerce is becoming increasingly common on the Internet, and many security-related electronic payment systems have been built on the basis of various cryptographic methods and standards. These applications are designed to protect important properties of solutions from anticipated types of attack such as eavesdropping, forgery, and espionage. We think that one such important property is a temporal one. For example, an electronic payment server verifies whether the system clock of the end user's computer is within a permitted range of time-drift, because the shopping date measured in the server and the user's record created in the client must be in agreement. In practice, there are many time-critical applications other than payment applications. To use public facilities such as tennis courts in Japan, we usually apply for a lottery ticket by mailing an application form to the appropriate office. Since the application can be received only during a specified period, applicants must pay attention to such temporal constraints. In most cases, the postmark is used as official proof of the application. In a few cases, the arrival time of the application form is used. However, the former seems to be more acceptable to applicants, because there is no official proof in the latter case. We think that the importance of the postmark lies in the provability it provides for temporal activities such as time-critical submissions. In other words, this important temporal property should be protected from adversaries who may forge or alter it. This paper explains the danger of such attacks and a defense strategy. Then we propose a secure electronic submission protocol that uses a trusted third party to provide temporal accountability. We also propose a temporal accountability logic, which is an extension of Kailar's work [K96]. The proposed logic, unlike Kailar's, is capable of representing the provability of temporal activities. We analyze the proposed protocol by applying our temporal accountability logic, and describe some modifications of the protocol, which reduce the total number of flows while keeping the protocol as logically secure as the original one in terms of temporal accountability.

1.1 Related Work

Kailar proposed an accountability logic [K96] for electronic commerce protocols such as payment protocols and public key distribution protocols. He defines accountability as a property whereby the association of a unique originator with an object or action can be proved to a third party. Provability has an important role in the analysis of accountability. Since time-critical applications require proofs that guarantee the temporal activities of each principal, we think that Kailar's accountability logic can be extended for use in analyzing such applications. Although the original logic allows some temporal context such as a During and Until property to be added to represent the validation period of security-related information such as a time-critical delegation key, we extend the original logic so that the logic can represent the temporal accountability. Much work has been done on

belief logic in the analysis of key management protocols. Burrows, Abadi, and Needham proposed a belief logic for authentication, which is used for reasoning about authentication protocols and for detecting redundancies and security flaws [BAN90]. In belief logic, each party is said to have a belief when he/she is convinced of a statement after several communications. There is a difference between belief logic and accountability logic as regards provability to other principals. Stubblebine extended the previous belief logic and proposed the notion of recent-secure authentication [S95], which is based on the work of [LABW92], [ABLP93], and [SO94]. The statement has three time properties: at, notbefore, and notafter. These temporal properties were created to represent the secure channel of a principal and freshness constraints for entity authentication. Our method assumes the existence of a trusted third party and the absence of clock synchronization among principals. On the other hand, Stubblebine's work does not assume a trusted third party and requires clock synchronization. Stubblebine and Wright extend BAN logic in order to reach a more precise temporal notion [SW96]. Their logic has three temporal properties: at a certain time t, at a certain time between t1 and t2, and at all times between t1 and t2. They also assume that all principals have their own local times and need to synchronize them. While the primary purpose of their logic is to reason about the soundness of protocols that use certificates which include security-related information with a validation period, our method is to reason about the soundness of protocols that use trusted timestamps which can offer temporal proof. Haber and Stornetta proposed a hash-based timestamp method [HS90]. Since their method can prove the time at which data were received, it is possible to combine their protocol and ours; however, we omit the explanation.

In this paper, we analyze various types of attack, including conspiracy attacks, described in Section 2. Section 3 describes a secure electronic submission protocol that uses a trusted third party. We propose a temporal accountability logic in Section 4. In Section 5 we analyze the proposed protocol by applying the temporal accountability logic. In section 6, we describe further considerations related to protocol analysis.

2 Attacks and Defense Measures for Temporal Transactions

In this section, we describe various types of attack on secure temporal transactions, and propose defense measures. We also explain why a trusted third party that guarantees the accuracy of the current time is required.

2.1 Difficulties

In Internet credit-payment systems, a buyer can easily verify a purchase statement by comparing the price on the receipt and the amount paid later from his/her bank account. However, how can he/she verify that the merchant's system clock is always exact? Such verification becomes important if the merchant engages in temporal transactions such as discount selling during a limited period. If an order is sent by electronic mail, the mail server's system clock is trusted, and the buyer receives a distribution acknowledgement from the mail server. In this way, the buyer can verify that the discount rate on the purchased item is correct. However, we must handle the situation in a more general way. We list several situations that can be more general or less secure:

- The user directly connects to the application server (not via a trusted third-party agent).
- The application server's system clock is not trusted, and may therefore be slow or fast.
- The user's system clock is not trusted, and may therefore be slow or fast.
- A malicious party may alter or forge temporal records stored in the user's machine or server.

With these assumptions, we describe possible malicious attacks on temporal transactions.

2.2 Attacks

We divide possible attacks against temporal transactions into two categories:

A) *Adjustment of the system clock forward or backward by the user, the server, or both in conspiracy*

B) *Forgery, alteration, or removal of temporal records by the user, the server, or both in conspiracy*

One example of an attack on a server by putting forward the system clock is as follows. If the user sends an application form to a public facility office close to the deadline, and the server clock is set a little fast, the application will not be received owing to its late arrival. Another example of a conspiracy attack involving forgery of temporal records concerns paper submission. If an author cannot complete his/her paper by the due date, he/she may request one of the program editors to include the paper even if it is not received by the due date. If the postmark on the envelope is not used as a proof of the submission, the editor can easily add the requested paper to the officially received submissions. In this case, the

author and program editor mount a conspiracy attack that involves forging temporal records so that the paper appears to have arrived before the due date.

2.3 Solutions

Some countermeasures against the above attacks are as follows:

A1) Periodic adjustment of the system clock

A2) Distribution of temporal transactions on multiple machines

A3) Use of a trusted third party that offers a correct timestamp

B1) Distribution of temporal transaction records to all principals

B2) Registration of temporal transaction records with another trusted third party

B3) Timestamping of temporal transaction records by a trusted third party

For **A1**, the system clock can be automatically adjusted by running the Network Time Protocol (NTP) [M91] or by running a Global Positioning System driver that provides accurate information on the current time. However, we think that this solution is not sufficiently secure against attacks of type **A,** because the person with the highest administrative access rights can easily change the system clock at any time. For **A2**, Franklin and Reiter presented a distributed server group computation that ensures the integrity of temporal transactions for sealed-bid auction [FR95]. However, this solution still suffers from the above access right problem. For **A3**, a trusted timestamp server is assumed. For **B** categories, note that temporal transaction records must be published outside the server; otherwise, there is no way of preventing forgery, alteration, or removal of records inside the server through attacks of type **B**. For **B1**, all principals must monitor the server's temporal transactions synchronously. For **B2**, a trusted third party for registration must provide a huge amount of storage space for the expected transactions. For **B3**, a trusted timestamp server is assumed. We previously presented an auction protocol in which the temporal transaction records are timestamped by a trusted server [K98].

For reasons of feasibility and trustability, we adopt measures **A3** and **B3** in this paper. We assume the existence of a trusted timestamp server that adds a correct timestamp to the temporal transaction data, that cannot be altered or forged. In this way, we can protect the security of temporal transactions from attacks of types **A** and **B**. Transaction records that contain time-sensitive information are accompanied by temporal proofs generated by the timestamp server. The advantage of this method is that application servers do not have to provide a correct time service even though they provide time-sensitive services to the users. In other words, the timestamp and timestamp server have the same roles as a postmark on a physical envelope and a post office, respectively.

We define the accurate time in this paper as Coordinated Universal Time (UTC), because the latter is used in Stratum 1 of the NTP. The UTC is calculated from both the International Atomic Time (TAI) and corrections for small changes to the TAI determined by the U.S. Naval Observatory and other observatories [M91].

3 Secure Electronic Submission Protocol

We think that the most typical application involving temporal transactions is electronic submission of an application that is permitted within a certain period of time. Note that it is not as easy to design a submission protocol as might be imagined. If an applicant takes a timestamp from the timestamp server for his/her application before the closing time for submission and sends it to the submission server, can it be satisfactorily proved that the submission is correct? Some people may suspect the applicant of sending the application before the opening time for submission. Others may suspect the applicant of sending the application after the closing time for submission, even though his/her submission was generated before the closing time. Therefore we begin by stating the general requirements for the electronic submission protocol:

R1) The receiver must not receive any applications except during the permitted period.

R2) The applicants must submit application data during the permitted period.

R3) The receiver cannot forge, alter, or remove any application once it has been received

R4) The receiver must receive applications that arrive during the permitted period.

3.1 Notation

We use the following notation. Principals are denoted by capital letters {*A*, *B*, ...}. We use *A* to denote *Alice*, who is an applicant, *B* for *Bob*, who is the receiver of the application forms sent by multiple applicants, and *T* for *Trent*, who is the trusted timestamp service provider.

Messages are denoted by small letters $\{x, y\}$ except for t. We use t as a time parameter. The predefined time parameters are t_s, which denotes the opening time for submission, and t_e, which denotes the closing time for submission. K_A and K_A^{-1} denote the public key and private key, respectively, for the principal A. $\{x\}K_A^{-1}$ denotes the signature value for a message x calculated by A's private key. *desc* denotes a submission description that consists of t_s, t_e, and *id*. *id* denotes an index number specified for a call for submission. x_i denotes the i-th submission from A_i, which denotes the i-th A. y_i denotes $\{x_i, A, B, \{t_s,\{desc\}K^{-1}_B\}K^{-1}_T\}K^{-1}_{Ai}$. "," denotes an operation for concatenating parameters.

3.2 General Assumptions

The general assumptions for this protocol are as follows:

N1) The time stamp service provider is trusted to provide the current time accurately and to generate correct timestamps. We assume the existence of another function to send notification to other entities at previously specified times.

N2) The applicant's system clock and receiver's system clock are not trusted, and may be fast or slow. We do not assume any clock synchronization for them. We consider another situation in section 6, in which the receiver's clock is accurate.

N3) The receiver is assumed to offer submission services to all applicants fairly.

N4) The computing resources for the receiver and the timestamp server are assumed to be unlimited. It takes only a short time to communicate or execute the protocol.

N5) The digital signature algorithm is based on public key cryptography, and it is impossible to forge or alter a signature owing to the computational complexity.

N6) It is assumed that no denial-of-service attack by a malicious party occurs.

N7) The channel for communication among the principals is assumed to be secured in order to preserve the confidentiality of each message.

3.3 General Flow

The submission protocol consists of four steps: a preliminary step, an opening step, a submission step, and a closing step. In the preliminary step, *Bob* asks *Trent* to notify *Bob* at the opening time and closing time for submission. In the opening step, which is taken when notification is received from *Trent*, an opening timestamp is used to guarantee that the applicant generates an application after the opening time for submission. In the submission step, one or more applicants, *Alice*, send applications to *Bob* and receive timestamps in acknowledgment of their submissions. In the closing step, *Bob* acquires a closing timestamp and sends it back to *Alice*.

We explain the meaning of each protocol flow step by step. In the Preliminary step, *Bob* requests *Trent* to notify *Bob* at the opening time and closing time for submission because *Bob* may not have the accurate time, from assumption **N2**. "NR" denotes a notification service request and "NA" denotes a notification acknowledgment, which is different from a timestamping service. We discuss another scenario, which assumes that *Bob* has the accurate time and does not require the notification service. *Bob* receives acknowledgment of the notification in message 2. *Trent* notifies *Bob* at the opening time for submission in message 3. *Bob* sends a description of a call for submission to *Trent*, which means that *Bob* declares the opening of the period for electronic submission at that time. *Trent* returns a timestamp, that is, a signature on *Bob*'s message with a correct current time value t_s without verifying the contents of the message, *desc*. Instead of using messages 4 and 5, we can also apply the Time Stamp Protocol [ACPZ98] with some modifications. The purpose of this timestamp is to provide a proof that meets requirement **R2** in message 8. *Alice* retrieves *Trent*'s timestamp in message 7 by making a query with *id*, which is the index number of the call for submission. *Alice* sends an electronic submission to *Bob* in message 8. The message consists of application data x_i, the origin, the destination, the timestamp, and the applicant's signature on them. Since the timestamp, which is *Trent*'s signature, can only be generated at the opening time for submission, *Alice* and *Bob* can prove that the applicant's submission was sent after time t_s. If it is before the closing time for submission, *Bob* makes *Trent* issue a timestamp on each application in messages 9 and 10, and returns the timestamp to each applicant in message 11. If *Bob* receives the application after the closing time for submission, *Bob* returns message 15 instead of 9, 10, and 11, which means that the submission period is closed. After receiving message 11, *Alice* verifies that *Bob* received message 8. When the closing time for submission comes, *Bob* is notified by *Trent* in message 12, and *Bob* sends *Trent* a hashed value calculated from all successfully received submissions, and receives a timestamp to declare that the period for electronic submission is closed, in messages 13 and 14, respectively.

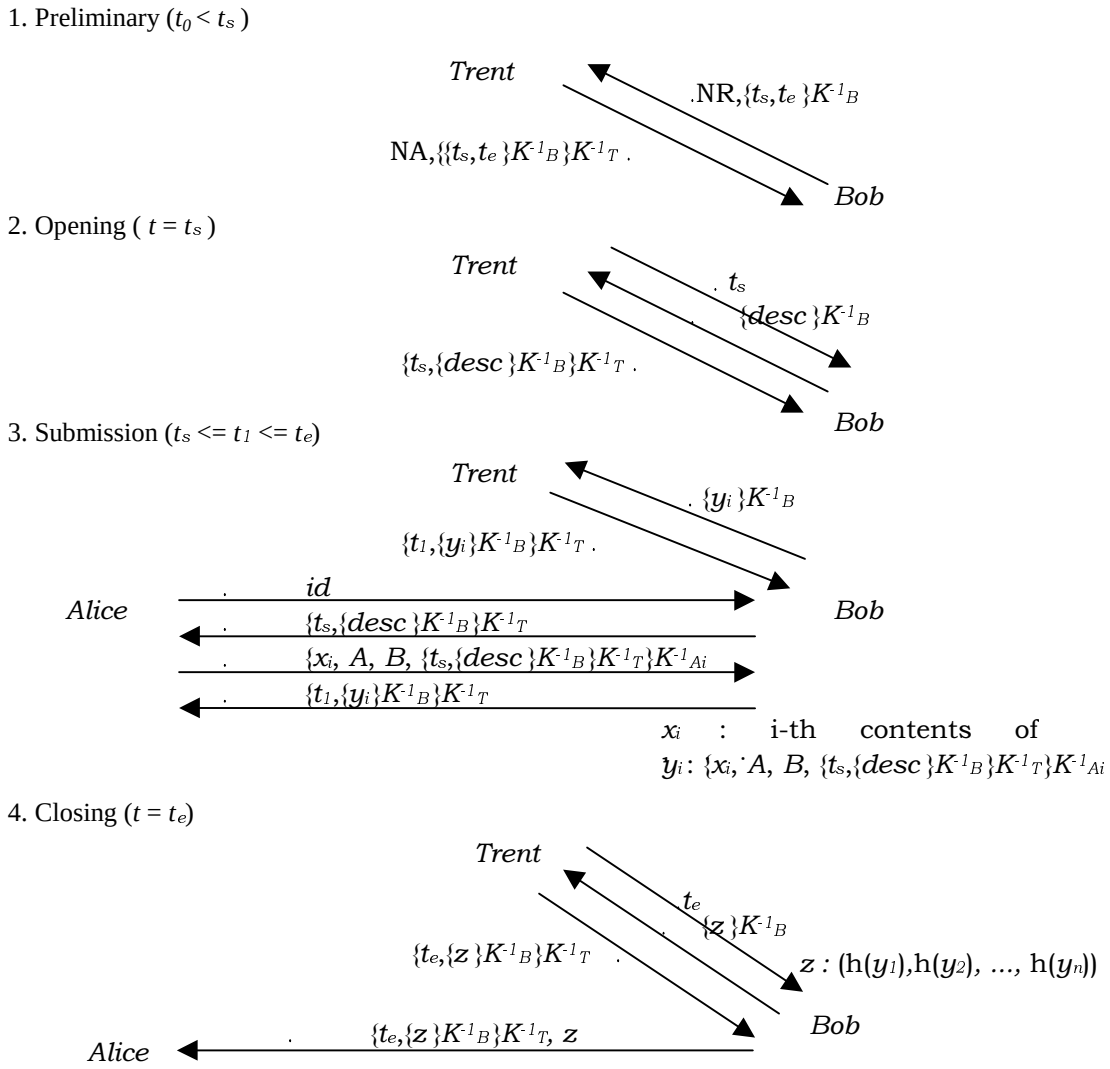

Figure 1. Protocol Flow

Alice receives message 15 and verifies that his/her submitted data are included in it by reconstructing the hashed value, z, from his/her submission data and other applicants' hashed values $h(y_i)$. This implies that each applicant does not have to know about other applicants' data when verifying submissions. The timestamp guarantees the validity of all submissions received by *Bob*, so forging operations on submissions after the closing time can easily be detected. In the case of a subsequent dispute, each applicant and the requester can prove that they followed requirements **R1** , **R2**, and **R3** correctly by showing message 15 and all successfully submitted data, y_i. As for **R4**, we assume that the requester is not malicious and able to receive all submissions, since we have assumptions **N4** and **N6**. Alternatively, some notary service must be used for this purpose.

4 Temporal Accountability Logic

In this section, we present a temporal accountability logic that is an extension of Kailar's framework. The original work is capable of analyzing payment protocols and key distribution protocols, and consists of logical constructs, analysis assumptions, and logical postulates. We add nine new logical constructs and ten new postulates to allow analysis of temporal accountability, which is not possible in the original framework. After briefly explaining the original framework, we describe the

proposed logic. We review most of the original constructs and postulates but not all of them.

4.1 Original Logical Constructs

The *Strong Proof* construct is denoted as *A* CanProve *x* , which means that a principal *A* can prove statement *x* to any principal *B*. After the sequence of operations, *B* is convinced of *x* without any secret *y* being revealed to *B*. The *Signature Authentication* construct is denoted as *K* Authenticates *A*, which means that key *K* can be used to authenticate the signature of principal *A* or to associate principal *A* unambiguously with any statement encrypted with K^{-1}. The *Message Interpretation* construct is denoted as x in *m* **,** which means that *x* is an interpretation of a field, or a combination of fields in message *m*. The *Signature Interpretation* construct is denoted as *x* => *m*, which articulates the interpretation of a signed message; this interpretation is defined by protocol designers and assumed to be evident to all principals involved. The *Statement* construct is denoted as *A* Says *x*, which means that principal *A* is accountable for making the statement *x*, and for anything that *x* implies. The *Message Receipt* construct is denoted as *A* Receives *m* SignedWith K^{-1}, which means that a principal *A* receives message *m* ,which is signed with K^{-1}. The construct *m* SignedWith K^{-1} means the same as $\{m\}K^{-1}$. The *Trust* construct is denoted as *A* IsTrustedOn *x*, which means that a principal *A* is trusted with regards to statement *x*. Thus, *A* has the authority to endorse statement *x*, and is liable for making statement *x*.

4.2 Original Postulates Used in the Analysis

The original work uses a notation for the postulates similar to that of [BAN90].

P; Q

R

means that if statements *P* and *Q* hold simultaneously, then statement *R* holds. ";" denotes conjunction and not temporal ordering. We use the following postulates in this paper. The *Inference* postulate

A CanProve *x* ; *x* => *y*

A CanProve *y*

means that, if a principal *A* can prove that *x* holds, and if *x* implies *y*, then it follows that *A* can prove that *y* holds.

The *Accountability of Digital Signatures* postulate

A Receives (*m* SignedWith K^{-1}) ; *x* in *m;*

A CanProve (*K* Authenticates *B*)

A CanProve (*B* Says *x*)

means that, if a principal *A* receives a message *m* signed with the key K^{-1}, the message *m* contains statement *x*, and *A* can prove that the key *K* authenticates principal *B*, then *A* can prove that *B* is accountable for *x*. The *Message Receipt* postulate

A Receives *(m* SignedWith K^{-1}*)* ; *x* in *m;*

A Receives *(x* SignedWith K^{-1}*)*

means that, if a principal *A* receives a message *m* signed with the key K^{-1}, and the message *m* contains statement *x*, then *A* can prove that *x* is also signed with K^{-1}. From the definition, *(m* SignedWith K^{-1}*)* includes all contents and signatures associated with the message. Then, we can also say *A* Receives *x* if the above conditional statement holds.

4.3 Temporal Logical Constructs

We propose the following temporal logical constructs:

- The *Generation Before* construct is denoted as *x* GeneratedBefore *t*, which means that the message *x* was generated before or at time *t*, and therefore *x* exists at time *t* and also continues to exist after *t* according to the usual temporal causal relation. Note that we use the words "after" and "before" in the construct names a little differently from the usual meaning, which does not include a terminal point of time literally.
- The *Generation At* construct is denoted as *x* GeneratedAt *t*, which means that the message *x* is generated at time *t*. GeneratedAt differs from GeneratedBefore in that the former represents the generated time of *x* explicitly whereas the latter cannot.
- A *Generation After* construct is denoted as *x* GeneratedAfter *t*, which means that the message *x* is generated at or after time *t*. This construct is weaker than GeneratedAt, because it conveys no information about when *x* is generated.
- A *Timestamp* construct is denoted as *x* TimestampedWith *t* With K^{-1} , which means that the message *x* is timestamped with *t* with the key K^{-1} at time *t*. This construct has the similar meaning as (*t, x*) SignedWith K^{-1}. *x* TimestampedWith *t* With K^{-1} includes all contents and signatures associated with the message.
- A *Timestamp Signature* construct is denoted as *x* TimesignWith *t* With K^{-1}, and is used for representing only a signature value of *x* TimestampedWith *t* With K^{-1}. It does not include any plain text contents such as *t* and *x*.
- A *Says At* construct is denoted as *A* Says *x* At *t*, which means that a principal *A* is accountable for

making the statement *x* at time *t*.

- A *Reception Before* construct is denoted as *A* Receives *x* Before *t* From *B*, which means that a principal *A* receives a message *x* from a principal *B* before or at time *t*.
- A *Reception After* construct is denoted as *A* Receives *x* After *t* From *B*, which means that a principal *A* receives a message *x* from a principal *B* at or after time *t*.
- A *Reception During* construct is denoted as *A* Receives *x* During $[t_s,t_e]$ From *B*, which means that a principal *A* receives a message *x* from a principal *B* during the closed interval $[t_s, t_e]$. We assume $t_s < t_e$.

4.4 Temporal Postulates

We propose the following temporal postulates:

- *Generation Before*

The *Generation Before* postulate relates the provability of a timestamp and the *Generation Before* construct. If a principal *A* receives a message *x* that is timestamped with *t* with a private key K^{-1}, and the timestamp is issued by a principal *T*, and *T* is trusted as regards the accuracy of his/her time, then *A* can prove that message *x* is generated before or at time *t*.

A Receives (*x* TimestampedWith *t* With K^{-1});
A CanProve (*K* Authenticates *T*);
A CanProve (*T* IsTrustedOn *t*)
A CanProve (*x* GeneratedBefore *t*)

- *Generation At*

The *Generation At* postulate relates the provability of a timestamp and the *Generation At* construct. If a principal *A* receives a message *x* that is timestamped with *t* with a private key K^{-1} at time t, and the timestamp is issued by a principal *T*, and *T* is trusted as regards the accuracy of his/her time, then *A* can prove that *T*'s signature itself, which is denoted by (*x* TimesignWith *t* With K^{-1}), is generated at *t*.

A Receives (*x* TimestampedWith *t* With K^{-1});
A CanProve (*K* Authenticates *T*);
A CanProve (*T* IsTrustedOn *t*)
A CanProve ((*x* TimesignWith *t* With K^{-1}) GeneratedAt *t*)

- *Generation Relation1*

Generation Relation1 postulate describes a relation between the *Generation At* construct and the *Generation Before* construct. If a principal *A* can prove that *x* is generated at *t*, and that *x* is a component of *y*, then *A* can prove that *y* is generated at or after *t*.

A CanProve (*x* GeneratedAt *t*);
x in *y*
A CanProve (*y* GeneratedAfter *t*)

For example, suppose there are two messages, *x1* and *x2*, and *x1* is generated at *t1*, and *x2* at *t2*. If we think of a combination of the above messages, *x1* and *x2*, the combined message cannot be generated before *t2*, because *x2* is generated at *t2*. This implies that combined messages must be generated at the latest time for the generation of each components. Therefore *Generation Relation1* holds whatever *x* and combinations we take.

- *Generation Relation2*

The *Generation Relation2* postulate describes a feature of the *Generation After* construct. If a principal *A* can prove that *x* is generated at or after *t*, and *x* is a component of *y*, then *A* can prove that *y* is generated at or after *t*.

A CanProve (*x* GeneratedAfter *t*);
x in *y*
A CanProve (*y* GeneratedAfter *t*)

- *Generation Relation3*

The *Generation Relation3* postulate describes a relation between the *Generation At* construct and the *Generation Before* construct. If a principal *A* can prove that *x* is generated at *t*, and *y* is a component of *x*, then *A* can prove that *y* is generated before or at *t*.

A CanProve (*x* GeneratedAt *t*);
y in *x*
A CanProve (*y* GeneratedBefore *t*)

- *Generation Relation4*

The *Generation Relation4* postulate describes a feature of the *Generation Before* construct. If a principal *A* can prove that *x* is generated before or at time *t*, and *y* is a component of *x*, then *A* can prove that *y* is also generated before or at time *t*.

A CanProve (*x* GeneratedBefore *t*);
y in *x*
A CanProve (*y* GeneratedBefore *t*)

- *Says At*

The *Says At* postulate defines a principal's statement at a certain time. If a principal *A* can prove that *x* is generated before or at *t*, where *x* includes *B*'s signature on *y*, then *A* can prove that *B* says *y* at *t*.

A CanProve (*x* GeneratedBefore *t*);
(*y* SignedWith K^{-1}) in *x*;
A CanProve (*K* Authenticates *B*)
A CanProve (*B* Says *y* At *t*)

- *Reception Before, After, During*

Reception postulates define reception accountability that represents when the message is received. These postulates are domain-specific provability properties.

- *Reception Before*

The *Reception Before* postulate means that if a principal *A* receives *y*, which includes a message with the origin and the destination, C's signature on them, and B's signature on all, and if *A* can prove that *y* is generated before or at *t*, and that K_C and K_B authenticate *C* and *B*, respectively, then *A* can prove that *B* receives *x* before or at *t* from *C*.

A Receives *y;*
(((*x*,*C*,*B*) SignedWith K^{-1}_C) SignedWith K^{-1}_B) in *y*;
A CanProve (*y* GeneratedBefore *t*);
A CanProve (K_C Authenticates *C*);
A CanProve (K_B Authenticates *B*);
A CanProve *B* Receives *x* Before *t* From *C*

From the second, fourth, and fifth statements of the *Reception Before* postulate, the *Message Receipt* postulate, and the *Accountability* postulate, we obtain *A* CanProve *B* Says (*x* SignedWith K^{-1}_C) and *A* CanProve *C* Says *x*. We think that it is intuitively reasonable that if someone says something - for example, *x* - that has been already said by a different person, then the former person must have received the contents from the latter beforehand. Although we cannot reason whether the sender sent the message directly to the receiver or not, the contents with the delivery information, such as (*x*,*C*,*B*), can show the original sender and the final destination. We use the word From as the origin of the message.

- *Reception After*

The *Reception After* postulate means that if a principal *A* receives the message *x* with the origin and the destination, which is signed by *C* and *B* sequentially, and if *A* can prove that *y* is generated at or after *t*, and that K_C and K_B authenticate *C* and *B*, respectively, then *A* can prove that *B* receives *x* at or after *t* from *C*.

A Receives (((*x*,*C*,*B*) SignedWith K^{-1}_C)
SignedWith K^{-1}_B);
A CanProve (*x* GeneratedAfter *t*);
A CanProve (K_C Authenticates *C*);
A CanProve (K_B Authenticates *B*);
A CanProve *B* Receives *x* After *t* From *C*

This postulate is intuitively reasonable, because if a component of the original message can be proved to be generated at or after time *t*, the receiver cannot receive the whole message before time *t*. Therefore the receiver must receive the message at or after time *t*.

- *Reception During*

The *Reception During* postulate is intuitively introduced on the basis of the *Reception After* and *Reception Before* postulates.

A CanProve *B* Receives *x* After t_1 From *C*;
A CanProve *B* Receives *x* Before t_2 From *C*;
$t_1 \le t_2$
A CanProve *B* Receives *x* During [t_1, t_2] From *C*

5 Protocol Analysis

In this section, we analyze the electronic submission protocol by applying the proposed temporal accountability logic. We apply the same analysis framework as that used in Kailar's work. It consists of stating the protocol's accountability goals, interpreting protocol messages, articulating the initial state assumptions, and analyzing each message. After analyzing the protocol, we can verify that all accountability goals can be obtained from the initial state assumptions and protocol message interpretations.

5.1 Accountability Goals

We present accountability goals for submission transactions according to the requirements stated in section 3. Although the following goals are neither necessary nor sufficient, they seem reasonable as general goals.

G1) *receiver, applicants* CanProve *(opening time and closing time are accurate)*
G2) *applicants* CanProve *(receiver will receive applications sent during the permitted period)*
G3) *receiver* CanProve *(applicants send their applications during the permitted period)*
G4) *applicants* CanProve *(receiver received application during the permitted period)*
G5) *applicants* CanProve *(no application is forged or removed once received)*

5.2 Protocol Interpretation

Since an unsigned message has no effect on the achievement of goals in accountability logic, the following flows can be interpreted:

1) *T* Receives ((t_s,t_e) SignedWith K^{-1}_B)
2) *B* Receives (((t_s,t_e) SignedWith K^{-1}_B)
SignedWith K^{-1}_T)
4) *T* Receives (*desc* SignedWith K^{-1}_B)
5) *B* Receives ((*desc* SignedWith K^{-1}_B)
TimestampedWith t_s With K^{-1}_T)
7) *A* Receives ((*desc* SignedWith K^{-1}_B)
TimestampedWith t_s With K^{-1}_T)
8) *B* Receives ((x_i, *A*, *B*, (*desc* SignedWith K^{-1}_B)
TimestampedWith t_s With K^{-1}_T)SignedWith K^{-1}_{Ai})

9) T Receives (y_i SignedWith K^{-1}_B)
10) B Receives ((y_i SignedWith K^{-1}_B) TimestampedWith t_1 With K^{-1}_T)
11) A Receives ((y_i SignedWith K^{-1}_B) TimestampedWith t_1 With K^{-1}_T)
13) T Receives (z SignedWith K^{-1}_B)
14) B Receives ((z SignedWith K^{-1}_B) TimestampedWith t_e With K^{-1}_T)
15) A Receives ((z SignedWith K^{-1}_B) TimestampedWith t_e With K^{-1}_T)

5.3 Initial State Assumptions

The initial state assumptions required in the analysis are as follows:

A1) A,T CanProve (K_B Authenticates B)
A2) B,T CanProve (K_A Authenticates A)
A3) A,B CanProve (K_T Authenticates T)
A4) A,B CanProve (T IsTrustedOn *time*)
A5) (B Says (t_s,t_e)) =>
(B *requests notification at time* t_s *and* t_e.)
A6) (T Says (t_s,t_e)) =>
(T *guarantees that T will notify B at times* t_s *and* t_e.)
A7) (B Says *desc* At t_s) =>
(*the opening time is accurate and B guarantees that it will receive all applications sent during the permitted period.*)
A8) (B Says z At t_e) =>
(*the closing time is accurate and B guarantees that no submissions can be forged or removed after closing time.*)
A9) (B Receive x During [t_s, t_1] From A) =>
(B *received A's submission sent during the permitted period if* $t_s < t_1 < t_e$ *holds.*)

The assumption **A1** denotes that the principal A and T can prove that B is accountable for any statement signed with K^{-1}_B. This assumption is made when the principal can acquire a certificate for the receiver that includes K_B. **A2** and **A3** have similar meanings to **A1**. Note that A denotes not one but multiple applicants. **A4** is from the general assumption **N1**. The last five assumptions are implications of accountability goals.

5.4 Protocol Analysis

- *Message 1*:

This preliminary flow is required when the general assumption **N2** is made. When T receives message 1, T sees a notification service request tag (NR) and can prove the following statement by applying the *Accountability* postulate and the assumption **A1**.

T CanProve (B Says (t_s,t_e))

This statement can be transformed by applying **A5** and the *Inference* postulate.

T CanProve (B *requests notification at time* t_s *and* t_e.)

- *Message 2*:

T sends a notification service acknowledgment in response to Message 1. When B receives message 2, B can prove the following statement by applying the *Accountability* postulate, **A3**, and **A6.**

B CanProve (T *guarantees that T will notify B at times* t_s *and* t_e.)

- *Message 4*:

When notified by T at t_s, B instantly sends a description of the call for submission with his/her signature to T to be timestamped. Although we need a nonce value in the timestamp request message, we omit it for the readability purpose in this paper.

- *Message 5*:

When B receives message 5, B can prove the following statement by applying the *Generation Before* and *Says At* postulate:

B CanProve (B Says *desc* At t_s)

First, variables x and t in the *Generation Before* postulate are instantiated as {*desc*}K^{-1}_B and t_s, respectively. Next, variables x, y, and t in the *Says At* postulate are instantiated as {*desc*}K^{-1}_B, *desc*, and t_s, respectively. Finally, the following statement can be obtained by applying the assumption **A7**:

B CanProve (*the opening time is accurate and B guarantees that it will receive all applications sent during the permitted period.)*

- *Message 7*:

When A receives message 7, A can obtain the following statement in a similar manner to message 5.

A CanProve (B Says *desc* At t_s)

This can be further transformed by applying **A7**.

A CanProve (*the opening time is accurate and B guarantees that it will receive all applications sent during the permitted period.)*

Then the goal **G2** can be achieved.

- *Message 8*:

When B receives message 8, B can prove the statement that A generated his/her submission after the opening time for submission t_s. Message 8 can be transformed as follows:

B Receives v
where
y : (*desc* SignedWith K^{-1}_B)
w : (y TimesignWith t_s With K^{-1}_T)

u : (y TimestampedWith t_s With K^{-1}_T)

v : (x_i, u)

Using the above variables, **A3**, and **A4**, the statements

B Receives *w*

B CanProve (K_T Authenticates *T*)

B CanProve (*T* IsTrustedOn t_s)

hold, and we can obtain the following from the *Generation At* postulate.

B CanProve (*w* GeneratedAt t_s)

We can obtain the following statements from the *Generation Relation1* and *2* postulates since *w* in *u*, and *u* in *v*.

B CanProve (*u* GeneratedAfter t_s)

B CanProve (*v* GeneratedAfter t_s)

The final statement is used in reasoning in message 10.

- *Messages 9* and *10*:

B requests a timestamp with the value of message 8 in message 9. When *B* receives message 10, the following statement can be proved by applying the *Generation Before* postulate and the *Reception Before* postulate.

B CanProve *B* Receives *v* Before t_1 From *A*

We can also obtain the following result from the result obtained in message 8 and the *Reception After* postulate:

B CanProve *B* Receives *v* After t_s From *A*

We obtain the following result from the *Reception During* postulate if $t_s < t_1 < t_e$:

B CanProve *B* Receives *v* During [t_s,t_1] From *A*

This means that *v*, the combination of the application x_i and the timestamp *u*, is received by *B*. Finally, we obtain the accountability goal **G3**.

- *Message 11*:

When *A* receives message 11, from the *Reception During* postulate,

A CanProve *B* Receives *v* During [t_s,t_1] From *A*

can be obtained. Finally, we obtain the following from **A9**:

A CanProve (*B received A's submission sent during the permitted period if* $t_s < t_1 < t_e$ *holds)*

This is the accountability goal **G4**.

- *Messages 13* and *14*:

When notified by *T* at the closing time for submissions, *B* instantly sends a hashed value calculated from all applications which are successfully received. When *B* receives message 14, The following statement;

B CanProve (*the closing time is accurate and B guarantees that no submissions can be forged or removed after the closing time)*

can be obtained from **A8**, the *Accountability* postulate, and the *Says At* postulate.

- *Message 15*:

When *A* receives message 15, *A* can prove the following statement from the *Says At* postulate and **A8**.

A CanProve (*the closing time is accurate and B guarantees that no submissions can be forged or removed after closing time)*

Therefore, goals **G1** and **G5** can be finally achieved in messages 5, 7, 14, and 15.

6 Discussion

We further discuss on the protocol refinement by using temporal accountability logic and the accuracy of the receiver's clock.

6.1 Protocol Refinement

In the proposed protocol, the receiver has to acquire timestamps in proportion to the number of applicants. If the general assumption **N4** is not made and the number of applicants increases greatly, the possibilities of a performance bottleneck problem at the timestamp server should be considered. By making some modifications to the original protocol, we can avoid this problem while protecting the security of the temporal transactions. We remove messages 9 and 10, and modify message 11 to $\{y_i\}K^{-1}_B$. Then, we can reduce the required number of timestamps per call for submission from n+2 to 2 (where n denotes the number of applicants), while maintaining the original security level in terms of temporal accountability. The difference lies in the time at which **G3** is achieved for the receiver and **G4** for applicants. After refinement, the receiver can acquire the proof of **G3** in message 14, and the applicant can acquire the proof of **G4** in message 15**,** at the closing time for submissions, because the *Reception During* postulate holds at that time. We think that this is a tradeoff problem between efficiency and usability. By adding one more modification to the previous refinement, applicants can acquire the proof of the goal **G3** immediately after receiving the modified message 11 by executing an additional timestamping protocol with the timestamp server, which is similar to messages 9 and 10. We think that the last protocol is more optimal in terms of efficiency than the previously proposed one, because the applicants can decide when they should obtain the goal **G3**. As we have stated, the temporal accountability logic is effective for analyzing protocol and modifying messages, so that we can reduce the number of flows and improve protocol efficiency.

6.2 If the Receiver's Clock Is Accurate

We assume here that the general assumption **N2** is not made and that the receiver is capable of synchronizing his/her local clock with another accurate clock. We can then omit the preliminary step, message 3 in the opening step, and message 12 in the closing step, because the receiver can begin the opening and closing steps without being notified by the timestamp server. Note that a verifier can still verify the temporal accuracy of the submission process by checking the values of the timestamps. If the requester's time is accurate, the verifier can reason that the receiver's clock is accurate and the whole process has been carried out correctly. Otherwise, the verifier can detect that the receiver's clock is not accurate or that the submission-receiving process has been carried out incorrectly. In the latter case, the accountability goal **G1** cannot be obtained.

7 Conclusion

We have explained that electronic commerce applications are closely related to the security of temporal properties. We described possible attacks on temporal properties and proposed a defense measurement that assumes the existence of a trusted third-party of a timestamp server. We presented a secure electronic submission protocol as the most typical electronic commerce application that has a feature for temporal transactions. We also proposed the temporal accountability logic, which is capable of representing the provability of temporal activities, and analyzed the proposed protocol. We explained that the protocol can be refined by making some modifications to the original protocol while maintaining the original security level in terms of temporal accountability. Although the proposed logic does not attempt to solve all problems through temporal accountability, the logic is effective in analyzing protocol, and modifying messages, to improve the number of flows and improve protocol performance.

Acknowledgments

I would like to thank Anish Mathuria for the numerous useful comments on earlier versions of this paper.

References

[ABLP93] M. Abadi, M. Burrows, B. Lampson, and G. A. Plotkin, "A Calculus for Access Control in Distributed Systems," ACM Trans. Program. Lang. Sys., 15(4), Sep. 1993, pp. 706-734.

[ACPZ98] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato, "Time Stamp Protocols," Internet Draft, ftp.ietf.org.

[BAN90] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," ACM Trans. on Computer Systems, Vol. 8, No. 1, Feb. 1990, pp. 18-36.

[FR95] M. K. Franklin and M. K. Reiter, "The Design and Implementation of a Secure Auction Service," IEEE Trans. on Software Engineering, Vol. 22, No. 5, 1995, pp. 302-312.

[HS90] S. Haber and W. S. Stornetta, "How to Time-Stamp a Digital Document," Proceedings of Crypto '90, Lecture Notes in Computer Science, 537, Springer-Verlag, 1991, pp. 437-455.

[K96] R. Kailar, "Accountability in Electronic Commerce Protocols," IEEE Trans. on Software Engineering, Vol. 22, No. 5, May, 1996, pp. 313-328.

[K98] M. Kudo, "Secure Electronic Sealed-Bid Auction Protocol with Public Key Cryptography," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E81-A, No. 1, Jan. 1998, pp. 20-27.

[LABW92] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in Distributed Systems: Theory and Practice," ACM Trans. Computer Systems, 10(4), Nov. 1992, pp. 265-310.

[M91] D. L. Mills, "Internet Time Synchronization: The Network Time Protocol," IEEE Transactions on Communication, Vol. 39, 1991, pp. 1482-1493.

[S95] S. G. Stubblebine, "Recent-Secure Authentication: Enforcing Revocation in Distributed Systems," 19th IEEE Symposium on Research in Security and Privacy, 1995, pp. 224-235.

[SO94] P. F. Syverson, and P. C. van Oorschot, "On Unifying Some Cryptographic Protocol Logics," 18th IEEE Symposium on Research in Security and Privacy, 1994, pp. 14-28.

[SW96] S. G. Stubblebine, and R. N. Wright, "An Authentication Logic Supporting Synchronization, Revocation, and Recency," 3rd ACM Conference on Computer and Communications Security, Mar. 1996, pp. 95-105.