Two-phase Nonrepudiation Protocols

Muqheet Ali, Rezwana Reaz, and Mohamed Gouda

Department of Computer Science

University of Texas at Austin

Austin, TX, 78712

{muqheet,rezwana,gouda}@cs.utexas.edu

ABSTRACT

A nonrepudiation protocol from party S to party R performs two tasks. First, the protocol enables party S to send to party R some text x along with a proof (that can convince a judge) that x was indeed sent by S . Second, the protocol enables party R to receive text x from S and to send to S a proof (that can convince a judge) that x was indeed received by R . A nonrepudiation protocol from one party to another is called two-phase iff the two parties execute the protocol as specified until one of the two parties receives its complete proof. Then and only then does this party refrain from sending any message specified by the protocol because these messages only help the other party complete its proof. In this paper, we present methods for specifying and verifying two-phase nonrepudiation protocols.

CCS Concepts

•Networks → Protocol correctness; •Security and privacy → Formal methods and theory of security;

Keywords

Nonrepudiation; Trusted Party; Digital Signature

1. INTRODUCTION

A nonrepudiation protocol from party S to party R performs two tasks. First, the protocol enables party S to send to party R some text x along with a proof (that can convince a judge) that x was indeed sent by S . Second, the protocol enables party R to receive text x from S and to send to S a proof (that can convince a judge) that x was indeed received by R . Only one of two outcomes is possible when the execution of a nonrepudiation protocol from S to R terminates. The first outcome is that both S and R obtain their required proofs. The second outcome is that neither S nor R obtains its required proof.

Nonrepudiation protocols can support several important applications over the Internet. Examples of these applica-

tions are certified email [11], cloud storage [3], electronic billing [16], and meter reading in smart grids [13].

1.1 Related Work

The most cited nonrepudiation protocol was published by Zhou and Gollmann in 1996 [14, 8]. This protocol involves three parties S , R , and a trusted third party T . It turns out that each execution of this protocol requires the active participation of all three parties S , R , and T . A year later, Zhou and Gollmann published a second version [15] of their protocol, where each execution requires the active participation of parties S and R , but does not necessarily require the participation of party T . (Thus, some executions of this second version requires the participation of T and some don't.)

In 1999, Markowitch and Yves Roggeman published a nonrepudiation protocol that involves only the two parties S and R and does not involve a trusted third party T [10]. Correctness of this protocol is based on the assumption that party S knows an upper bound on the computing power of party R . Unfortunately, this assumption can't be guaranteed in practice and correctness of this protocol remains in question.

Later in 2000, Kremer and Markowitch generalized nonrepudiation protocols that are from a party S to a party R into protocols that are from a party S to several parties $R_1, \dots$, and R_n . They referred to the generalized protocols as multiparty protocols, and published the first two multiparty protocols [7, 9].

Most nonrepudiation protocols that have been published in the literature seem to suffer from the following three problems:

- Execution of the protocol is non-deterministic. This implies that execution of the protocol can follow any one of several alternative paths. For example, execution of the nonrepudiation protocol in [6] can follow anyone of six alternative paths.
- Some parties involved in the protocol need to utilize synchronized real-time clocks. For example each of the three parties S , R , and T involved in the nonrepudiation protocol in [6] needs to utilize real-time clocks. Moreover, these three clocks need to be synchronized.
- Exchanged messages during execution of the protocol are of many distinct types. For example, there are nine distinct message types in the multiparty protocol in [9].

These three problems seem to complicate the specification and verification of nonrepudiation protocols. It is no acci-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICCCNT July 6–8, 2016, Dallas, TX, USA

© 2016 ACM. ISBN 978-1-4503-4179-0/16/07...\$15.00

DOI: <http://dx.doi.org/10.1145/2967878.2967903>

dent that several nonrepudiation protocols have been shown to be incorrect after they have been published in the literature; see for example [5, 1, 4].

1.2 Our Contributions

In this paper, we identify a rich class of nonrepudiation protocols, called two-phase protocols, that do not suffer from the above three problems:

- Execution of a two-phase nonrepudiation protocol is deterministic and so it is guaranteed to follow exactly one path.
- None of the parties involved in a two-phase protocol needs to utilize a real-time clock. Therefore, proving correctness of two-phase protocols does not require reasoning about real-time.
- Exchanged messages during execution of a two-phase protocol are of two types only.

Also in this paper, we discuss how to specify and verify two-phase protocols and demonstrate that the task of specifying and verifying a two-phase protocol is simpler than the task of specifying and verifying a comparable protocol that is not two-phase.

2. TWO-PHASE PROTOCOLS

The objective that party S achieves by participating in the execution of a nonrepudiation protocol from S to R is different from the objective that party R achieves by participating in the execution of the same protocol. The objective of party S is to obtain a proof that party R has received some text x that was sent earlier from S to R . The objective of party R is to obtain a proof that party S has sent some text x that was later received by R from S .

A nonrepudiation protocol from party S to party R is called two-phase iff execution of the protocol by parties S and R proceeds in two phases. In the first phase, the party (S or R) recognizes that it has not yet received its complete proof and so it continues to execute the protocol as specified. In the second phase, the party (S or R) recognizes that it has already received its complete proof and so it refrains from sending any more messages specified by the protocol because these messages only help the other party complete its proof. In other words, the two parties S and R in any two-phase protocol will always act in their own self-interests during execution of the protocol.

Two-phase nonrepudiation protocols have the following advantages over nonrepudiation protocols that are not two-phase:

- (a) Execution of a two-phase protocol is inherently deterministic, whereas execution of a protocol that is not two-phase is usually non-deterministic.
- (b) The participating parties of a two-phase protocol do not need to have synchronized clocks, whereas parties of a protocol that is not two-phase need to have synchronized clocks.
- (c) It follows from (a) and (b) above that specifying and verifying the correctness of a two-phase protocol are easier than specifying and verifying the correctness of a comparable protocol that is not two-phase.

One more advantage of two-phase protocols is that the exchanged messages in these protocols are of two types only: send-proof messages and receive-proof messages.

The rest of this paper is organized as follows. In Section 3, we describe the (only) two types of messages that are exchanged during executions of two-phase protocols. Then in Section 4, we describe the syntax and semantics of a simple notation that can be used for specifying two-phase protocols. In Section 5, we present a procedure for verifying the correctness of two-phase protocols. In Section 6, we conclude that two-phase protocols that involve only two parties, namely S and R , are incorrect, and we present a (correct) two-phase protocol that involves three parties: S , R and a trusted third party T .

In Section 7, we discuss the security analysis of our two-phase protocols. In particular, we show that each one of these protocols can defend against four types of attacks: malicious parties, message loss, collusion attacks, and replay attacks. Concluding remarks of this paper are given in Section 8.

3. SEND AND RECEIVE PROOF MESSAGES

The messages exchanged during the executions of a two-phase protocol are of two types: snd-proof messages and rcv-proof messages. In this section, we describe these two types of messages.

A snd-proof message has five fields: message type, text, from, to, and message signature. Consider for example the following snd-proof message:

```
(type: snd-proof,
text:  $x$ ,
from:  $S$ ,
to:  $R$ ,
message signature: using the private key of  $S$ )
```

Because this message is signed using the private key of party S , only S could have constructed this message. In fact party S has constructed this message and sent it to party R in order to provide R with a proof that S is the one that sent text x to R . (This should explain why we call this message a snd-proof message.)

At any point in the future, party R can show this snd-proof message to an (impartial) judge and can get this judge to declare that party S has indeed sent text x to party R and that party R has indeed received text x from party S .

Like a snd-proof message, a rcv-proof message has five fields: message type, text, from, to, and message signature. Consider for example the following rcv-proof message:

```
(type: rcv-proof,
text:  $x$ ,
from:  $S$ ,
to:  $R$ ,
message signature: using the private key of  $R$ )
```

Because this message is signed using the private key of party R , only R could have constructed this message. In fact party R has constructed this message and sent it to party S in order to provide S with a proof that R is the one that received text x from S . (This should explain why we call this message a rcv-proof message.)

At any point in the future, party S can show this rcv-proof

message, that it has received from party R , to an (impartial) judge and can get this judge to declare that party S has indeed sent text x to party R and that party R has indeed received text x from party S .

From now on, each snd-proof message of the form: (type: snd-proof, text: x , from: S , to: R , message signature: using the private key of S) is written as:

snd-proof(x , S , R)

Also each rcv-proof message of the form: (type: rcv-proof, text: x , from: S , to: R , message signature: using the private key of R) is written as:

rcv-proof(x , S , R)

4. SPECIFICATION OF TWO-PHASE PROTOCOLS

In this section, we describe a simple notation for specifying two-phase protocols that involve three parties S , R , and T . (The role of party T , which is called a trusted third party, in two-phase protocols is discussed in Section 6 below.)

In our notation, each two-phase protocol is specified as a finite and non-empty sequence of (atomic) *steps*, each of which is of the following form:

$P \rightarrow Q: \text{msg.1}, \dots, \text{msg.k}$

P denotes a party out of the three parties S , R , and T .

Q denotes a party, different from party P , out of the three parties S , R , and T .

Each msg.i denotes either a snd-proof message or a rcv-proof message, as defined in the previous section.

Informally, execution of this step consists of party P sending to party Q the messages $\text{msg.1}, \dots, \text{msg.k}$, and of party Q receiving these messages. (There are exceptions to this description; these are described below.)

Execution of a two-phase protocol consists of executing the steps of the protocol, one by one, starting from its first step. Execution of the protocol terminates when the last step in the protocol is executed or when any step in the protocol is executed and its execution causes execution of the protocol to terminate, as described below.

Associated with each two-phase protocol are three sets, called the S -proof set, the R -proof set, and the T -proof set. The S -proof set contains the snd-proof messages and rcv-proof messages that are received by party S during execution of the protocol. Similarly, the R -proof set (or the T -proof set, respectively) contains the snd-proof and rcv-proof messages that are received by party R (or party T , respectively) during execution of the protocol.

Before execution of a two-phase protocol starts, the three proof sets of the protocol are all empty. Then, as described below, executing each step adds some messages to at most one of these three proof sets.

The S -proof set of a two-phase protocol is said to be *complete* at some point during execution of the protocol iff the messages in the S -proof set at this point are sufficient to prove (to a judge) that text x has been received by party R or by party T .

The R -proof set of a two-phase protocol is said to be *complete* at some point during execution of the protocol iff the messages in the R -proof set at this point are sufficient to prove (to a judge) that text x has been sent from S to R .

Note that the concept of the T -proof set of a two-phase

protocol being complete is not defined.

Formally, the *execution* of a step

$P \rightarrow Q: \text{msg.1}, \dots, \text{msg.k}$

in a two-phase protocol proceeds as follows

if P denotes party S

and-if the S -proof set of this protocol is complete before executing this step

then executing this step does not change any of the three proof sets but causes execution of the protocol to terminate

else-if P denotes party R

and-if the R -proof set of this protocol is complete before executing this step

then executing this step does not change any of the three proof sets but causes execution of the protocol to terminate

else executing this step adds the messages $\text{msg.1}, \dots, \text{msg.k}$ to the Q -proof set of the protocol but does not change the other two proof sets

This formal definition of the execution of a step

$P \rightarrow Q: \text{msg.1}, \dots, \text{msg.k}$

merits the following explanation.

Most of the time, execution of this step consists of party P sending the messages $\text{msg.1}, \dots, \text{msg.k}$ to party Q and of Q receiving these messages and adding them to the Q -proof set of the protocol. Later, party Q can use these messages in constructing its proof. There are two exceptions to this rule.

The first exception is when party P is in fact party S and the S -proof set of the protocol is already complete. In this case, party S recognizes that participating in the execution of this step does not help party S in constructing its proof (since its proof is already complete). In this case, execution of this step does not cause any of the messages $\text{msg.1}, \dots, \text{msg.k}$ to be sent or received, does not change any of the three proof sets of the protocol, but causes execution of the protocol to terminate.

The second exception is when party P is in fact party R and the R -proof set of the protocol is already complete. In this case, Party R recognizes that participating in the execution of this step does not help party R in constructing its proof (since its proof is already complete). In this case, execution of this step does not cause any of the messages $\text{msg.1}, \dots, \text{msg.k}$ to be sent or received, does not change any of the three proof sets of the protocol, but causes execution of the protocol to terminate.

Note that there are two possible outcomes for the execution of a protocol step: either one or more messages are added to one of the three proof sets of the protocol, or the three proof sets remain unchanged and execution of the protocol terminates.

A two-phase protocol is *correct* iff the S -proof set and R -proof set of the protocol are both complete when execution of the protocol terminates.

5. VERIFICATION OF TWO-PHASE PROTOCOLS

The procedure for verifying the correctness of a two-phase protocol proceeds as follows:

1. Start with the three proof sets of the protocol (namely

the S -proof set, the R -proof set, and the T -proof set) being empty.

2. Trace the three proof sets as the protocol steps are executed one by one starting with the first step in the protocol and until execution of the protocol terminates. (Recall that execution of the protocol terminates when the last step in the protocol is executed or when any protocol step, whose execution causes execution of the protocol to terminate, is executed.)
3. If both the S -proof set and R -proof set are complete when execution of the protocol terminates, then the protocol is correct. Otherwise, the protocol is incorrect.

In the remainder of this section, we present a two-phase protocol and use this verification procedure to show that this protocol is incorrect. Then, in the rest of the paper, we present three other two-phase protocols and use this verification procedure to show that these protocols are all correct.

Two-phase Protocol 1:

Consider a two-phase protocol that consists of the following four steps.

$$\begin{aligned} S &\rightarrow R: \text{snd-proof}(\text{encrypt}(x, \text{key}), S, R) \\ S &\leftarrow R: \text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R) \\ S &\rightarrow R: \text{snd-proof}(\text{key}, S, R) \\ S &\leftarrow R: \text{rcv-proof}(\text{key}, S, R) \end{aligned}$$

The text in the message of the first step (and of the second step) consists of the “encryption” of text x using a “key” that only party S knows. The text in the message of the third step (and of the fourth step) consists only of the encryption “key”.

The S -proof set of this protocol is complete when this set contains the two messages: $\text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R)$ and $\text{rcv-proof}(\text{key}, S, R)$. Also the R -proof set of this protocol is complete when this set contains the two messages: $\text{snd-proof}(\text{encrypt}(x, \text{key}), S, R)$ and $\text{snd-proof}(\text{key}, S, R)$.

To check whether this protocol is correct, we trace the three proof sets of this protocol as the protocol steps are executed. This tracing leads to the following proof sets after the four protocol steps are executed:

$$\begin{aligned} S\text{-proof} &= \text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R) \\ R\text{-proof} &= \text{snd-proof}(\text{encrypt}(x, \text{key}), S, R), \\ &\quad \text{snd-proof}(\text{key}, S, R) \\ T\text{-proof} &= \text{empty set} \end{aligned}$$

At this point, execution of the protocol terminates and S -proof is not complete but R -proof is complete. Because S -proof is not complete when execution of the protocol terminates, the protocol is not correct.

6. A TWO-PHASE PROTOCOL WITH A THIRD PARTY T

The two-phase protocol 1 is incorrect as follows. In the last step of this protocol, party R is supposed to send a message to party S . But the R -proof set of the protocol becomes complete before executing this last step and the message, that was supposed to be sent from R to S , is not sent. Thus, the S -proof set of the protocol remains incomplete when execution of the protocol terminates. This makes protocol 1 incorrect.

THEOREM 1. *There is no correct two-phase nonrepudiation protocol from party S to party R that involves only parties S and R .*

PROOF. Assume that we can design a correct two-phase nonrepudiation protocol that involves only the two parties S and R . Also, assume that the last step of this protocol is of the form:

$$R \rightarrow S: \text{msg}.1, \dots, \text{msg}.k$$

Now, there are two cases to consider:

Case 1: The R -proof set is not complete before executing this last step, then the R -proof set will not be complete when execution of this last step terminates, as execution of this last step will not change the R -proof set.

Case 2: The R -proof set is complete, then according to the specification of two-phase nonrepudiation protocol, the execution of this step does not change any proof set and causes execution of the protocol to terminate. We know that both the S -proof set and the R -proof set cannot be complete before the execution of this step. This is because if the S -proof set was complete before execution of this last step, then it was also complete before the execution of the previous step which was from S to R , and execution of that step would have resulted in termination of the protocol. It follows that S -proof set of the protocol was incomplete before execution of the last step, and will remain incomplete after execution of the last step.

In the above two cases, either the S -proof set or R -proof set is not complete when execution of the protocol terminates. This observation contradicts the assumption that this protocol was a correct two-phase nonrepudiation protocol.

A similar argument applies if the last step of the protocol is of the form:

$$S \rightarrow R: \text{msg}.1, \dots, \text{msg}.k$$

This proves that there is no correct two-phase nonrepudiation protocol from party S to party R that involves only parties S and R . $\square$

In the remainder of this paper, we only consider two-phase protocols from S to R that involve three parties S , R and T . We assume that the third party T is not biased to either of the two parties S or R . This means that unlike the two parties S and R , which refrain from sending any message specified by the protocol when their respective proof sets are complete, party T never refrains from sending any message specified by the protocol. (See the definition of step execution in section 4 above.) Therefore, party T is often referred to as a trusted third party [14]. A trusted third party T is involved in each of the correct two-phase protocols discussed below.

Two-phase Protocol 2:

Consider a two-phase protocol that consists of the following three steps.

$$\begin{aligned} S &\rightarrow T: \text{snd-proof}(x, S, R) \\ S &\leftarrow T: \text{rcv-proof}(x, S, T), \text{snd-proof}(x, T, R) \\ R &\leftarrow T: \text{snd-proof}(x, S, R) \end{aligned}$$

In the first step, S sends a $\text{snd-proof}(x, S, R)$ message to T which turns around and forwards the message to R in the third step. This message makes the R -proof set of the protocol complete. In the second step, T sends two messages to S . The first message acknowledges that T has received text x that was sent earlier by S and the second

message promises that T will forward text x to R . These two messages acknowledge that text x that was sent earlier by S has been received or will shortly be received by R . These two messages make the S -proof set of the protocol complete.

It follows that the S -proof set of this protocol is complete when this set contains the two messages: $\text{rcv-proof}(x, S, T)$ and $\text{snd-proof}(x, T, R)$. The R -proof set of this protocol is complete when this set contains the message $\text{snd-proof}(x, S, R)$.

To check that this protocol is correct, we trace the three proof sets of this protocol as the protocol steps are executed. The tracing leads to the following proof sets after the three protocol steps are executed:

$$\begin{aligned} S\text{-proof} &= \text{rcv-proof}(x, S, T), \text{snd-proof}(x, T, R) \\ R\text{-proof} &= \text{snd-proof}(x, S, R) \\ T\text{-proof} &= \text{snd-proof}(x, S, R) \end{aligned}$$

At this point, both S -proof and R -proof are complete. Because both S -proof and R -proof are complete when execution of the protocol terminates, the protocol is correct.

The two-phase protocol 2, though technically correct, has a problem. In this protocol, party S never gives party R a chance to express its agreement to participate in the protocol. Rather, party S starts by communicating directly with party T in order to complete the S -proof set (without soliciting any agreement from party R to participate in the protocol). Then party T turns around to communicate with party R in order to complete the R -proof set. In essence, party T “forces” party R to participate in the protocol.

To solve this problem, party S needs to start by communicating with party R (to ensure that R agrees to participate in the protocol) before S communicates with party T . This protocol is discussed next.

Two-phase Protocol 3:

Consider a two-phase protocol that consists of the following five steps.

$$\begin{aligned} S &\rightarrow R: \text{snd-proof}(\text{encrypt}(x, \text{key}), S, R) \\ S &\leftarrow R: \text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R) \\ S &\rightarrow T: \text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R), \\ &\quad \text{snd-proof}(\text{key}, S, R) \\ S &\leftarrow T: \text{rcv-proof}(\text{key}, S, T) \\ R &\leftarrow T: \text{snd-proof}(\text{key}, S, R) \end{aligned}$$

In the first step, party S sends a $\text{snd-proof}(\text{encrypt}(x, \text{key}), S, R)$ message to party R to allow R to agree to participate in the protocol. In the second step, party R agrees to participate in the protocol by sending back to party S a $\text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R)$ message. This message acknowledges that R has received an encryption of text x using a key that only S knows. In the third step, S sends two messages to T . The first message proves that party R has agreed to participate in the protocol, and the second message carries the encryption key, which is to be forwarded from T to R in the fifth step. In the fourth step, the message $\text{rcv-proof}(\text{key}, S, T)$ is sent from T to S in order to acknowledge that T has received the key that was sent by S in the third step.

It follows that the S -proof set of this protocol is complete when this set contains the two messages: $\text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R)$ and $\text{rcv-proof}(\text{key}, S, T)$. The R -proof set of this protocol is complete when this set contains the two messages: $\text{snd-proof}(\text{encrypt}(x, \text{key}), S, R)$ and $\text{snd-proof}(\text{key},$

$S, R)$.

To check that this protocol is correct, we trace the three proof sets of this protocol as the protocol steps are executed. This tracing leads to the following proof sets after the five protocol steps are executed:

$$\begin{aligned} S\text{-proof} &= \text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R), \\ &\quad \text{rcv-proof}(\text{key}, S, T) \\ R\text{-proof} &= \text{snd-proof}(\text{encrypt}(x, \text{key}), S, R), \\ &\quad \text{snd-proof}(\text{key}, S, R) \\ T\text{-proof} &= \text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R), \\ &\quad \text{snd-proof}(\text{key}, S, R) \end{aligned}$$

Because both S -proof and R -proof are complete when execution of the protocol terminates (after executing the five protocol steps), the protocol is correct.

In verifying the correctness of the two-phase protocol 3, we assumed that the three proof sets of this protocol are empty when execution of the protocol starts. However, this assumption cannot be enforced (and in fact can be violated) in practice.

For example, consider the case where this protocol is executed twice: In the first execution, S sends $\text{encrypt}(x_1, \text{key})$ and key to R , and in the second execution, S sends $\text{encrypt}(x_2, \text{key})$ and key to R . Note that S sends different texts but sends the same key in these two executions.

After the first execution of this protocol terminates, the S -proof set and the R -proof set are as follows:

$$\begin{aligned} S\text{-proof} &= \text{rcv-proof}(\text{encrypt}(x_1, \text{key}), S, R), \\ &\quad \text{rcv-proof}(\text{key}, S, T) \\ R\text{-proof} &= \text{snd-proof}(\text{encrypt}(x_1, \text{key}), S, R), \\ &\quad \text{snd-proof}(\text{key}, S, R) \end{aligned}$$

Assume that before the second execution of this protocol starts, the S -proof set is allowed to keep the message $\text{rcv-proof}(\text{key}, S, T)$ and the R -proof set becomes empty, as usual. Now, the second execution of the protocol starts by S sending the message $\text{snd-proof}(\text{encrypt}(x_2, \text{key}), S, R)$ to R and receiving back the message $\text{rcv-proof}(\text{encrypt}(x_2, \text{key}), S, R)$ from R . At this point, the S -proof set and the R -proof set of the protocol become as follows:

$$\begin{aligned} S\text{-proof} &= \text{rcv-proof}(\text{key}, S, T), \\ &\quad \text{rcv-proof}(\text{encrypt}(x_2, \text{key}), S, R) \\ R\text{-proof} &= \text{snd-proof}(\text{encrypt}(x_2, \text{key}), S, R) \end{aligned}$$

Note that at this point, the S -proof set is complete but the R -proof set is not. Thus, executing the third step of the protocol causes the second execution of the protocol to terminate yielding a complete S -proof set and an incomplete R -proof set. Therefore this protocol is incorrect.

The fact that this protocol is incorrect is caused by two factors. First, when the second execution of the protocol starts, the S -proof set is allowed to keep a message that was sent during the first execution of the protocol. Second, S is allowed to use the same key in two distinct executions of the protocol. Both these factors need to be allowed in order to make the protocol incorrect. Therefore, to make the protocol correct, we need to disallow either one of these two factors. Next we describe how to design keys such that S is compelled to use distinct keys in distinct executions of the protocol.

Consider any execution of two-phase protocol 3, where some text x is to be sent from party S to party R via party T . In this execution of the protocol, we require S to use a key that is the result of applying a secure hash function H to the concatenation of four items: text x , identifier of party S , identifier of party R , and identifier of party T . Thus,

key = $H(\text{text } x \mid \text{id of } S \mid \text{id of } R \mid \text{id of } T)$
 where “ $\mid$ ” denotes the concatenation operator. Note that this design of the key ensures that S will generate distinct keys to be used in distinct execution of the protocol.

In the third step of this protocol execution, S sends two messages $\text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R)$ and $\text{snd-proof}(\text{key}, S, R)$ to T . Then T performs two checks to ensure that these messages are correct. First, T checks that the key in the second message can indeed decrypt the encrypted text x in the first message. Second, T checks that the key in the second message has the right structure described above.

Next, we describe two-phase protocol 4, which enables party S to send the same text x to n parties, denoted $R.1, \dots, R.n$, via trusted third party T .

Two-phase Protocol 4:

Consider the following two-phase multiparty protocol that consists of $(3n + 2)$ steps.

```

 $S \rightarrow R.1$ :  $\text{snd-proof}(\text{encrypt}(x, \text{key}), S, R.1)$ 
...
 $S \rightarrow R.n$ :  $\text{snd-proof}(\text{encrypt}(x, \text{key}), S, R.n)$ 
 $S \leftarrow R.1$ :  $\text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R.1)$ 
...
 $S \leftarrow R.n$ :  $\text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R.n)$ 
 $S \rightarrow T$  :  $\text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R.1),$ 
            $\text{snd-proof}(\text{key}, S, R.1),$ 
           ...
            $\text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R.n),$ 
            $\text{snd-proof}(\text{key}, S, R.n)$ 
 $S \leftarrow T$  :  $\text{rcv-proof}(\text{key}, S, T)$ 
 $R.1 \leftarrow T$ :  $\text{snd-proof}(\text{key}, S, R.1)$ 
...
 $R.n \leftarrow T$ :  $\text{snd-proof}(\text{key}, S, R.n)$ 

```

Note that this protocol has $(n+2)$ proof sets, namely S -proof, $R.1$ -proof, $\dots$, $R.n$ -proof, and T -proof.

The S -proof set in this protocol is complete iff this set contains the following $(n + 1)$ messages:

```

 $\text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R.1),$ 
...
 $\text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R.n),$ 
 $\text{rcv-proof}(\text{key}, S, T)$ 

```

Also each $R.i$ -proof set in this protocol is complete iff this set contains the two messages: $\text{snd-proof}(\text{encrypt}(x, \text{key}), S, R.i)$ and $\text{snd-proof}(\text{key}, S, R.i)$.

Also note that the key in this protocol is structured as follows:

key = $H(\text{text } x \mid \text{id of } S \mid \text{id of } R.1 \mid \dots \mid \text{id of } R.n \mid \text{id of } T)$
 where “ $\mid$ ” is the concatenation operator.

7. SECURITY ANALYSIS:

In this section, we present a security analysis of two-phase protocol 3 in this paper. (A similar security analysis can also be presented for two-phase protocol 4 in the paper.) Protocol 3 is designed to defend against the four security attacks: malicious parties, message loss, collusion attacks, and replay attacks. Next, we describe these four attacks and show that protocol 3 can succeed in defending against all four attacks.

7.1 Malicious Parties:

By definition, the trusted third party T in protocol 3 is not malicious. This means that party T sends every message that is supposed to send when it is supposed to send it, as specified by the protocol.

On the other hand, either one of the other two parties S and R can be malicious. A malicious party can refrain from sending any message that is supposed to send, as specified by protocol 3, in order to prevent this other party from obtaining its complete proof. However, because the protocol is two-phase, the malicious party cannot refrain from sending any message specified by the protocol until this party has already obtained its own complete proof.

Referring to protocol 3, party S cannot refrain from sending the snd-proof message in the first step and cannot refrain from sending the rcv-proof and snd-proof messages in the third step because party S does not obtain its complete proof until after the fourth step of the protocol. Similarly, party R cannot refrain from sending the rcv-proof message in the second step of the protocol because party R does not obtain its complete proof until after the fifth step of the protocol.

It follows from this discussion that no party in protocol 3 can refrain from sending any message that is supposed to send as specified by the protocol.

7.2 Message Loss:

Any message that is supposed to be sent by one party and to be received by another party during the execution of protocol 3 can be lost and not received by the other party. There are two possible sources of message loss: an unreliable communication medium between the two parties and maliciousness of the sending party.

We assume that protocol 3 is to be executed on top of a reliable transport service (such as TCP in the internet). This transport service takes any message that is sent by any party during the execution of protocol 3 and transmits this message any number of times until the message is ultimately received by its intended party. Therefore, message loss due to an unreliable communication medium is not possible.

In Section 7.1 above, we argued that no party in protocol 3, whether malicious or not, can refrain from sending any message that is supposed to send as specified by the protocol. Therefore, message loss due to the maliciousness of the sending party is also not possible.

7.3 Collusion Attacks:

Consider an attack scenario where a party S executes protocol 3 to send text x to a party R via a trusted party T . After executing protocol 3, party S obtains its proof which consists of the following two messages:

```

 $\text{rcv-proof}(\text{encrypt}(x, \text{key}), S, R)$ 
 $\text{rcv-proof}(\text{key}, S, T)$ 

```

Later party S forwards these two messages to a malicious party S' which attempts to modify these messages in order to obtain a proof that party S' has sent text x to party R via a trusted party T .

However, any attempt of party S' to modify these two messages will fail because the first message is signed by the private key of party R and the second message is signed by the private key of the trusted party T and neither S nor S' has access to these private keys.

7.4 Replay Attacks:

If party S is malicious, then it will use same key in two distinct executions of protocol 3. Note that if the same key is used in two distinct executions of protocol 3, then some messages that have been generated and used as part of the proofs of one execution can also be used as part of the proofs of a later execution. Thus, these messages need not be generated in the second execution of the protocol.

Assume for example that party S uses the same key in two distinct executions of protocol 3. In the first execution of protocol 3, S uses key to send text x to party $R.1$ via the trusted party T . In the second execution of protocol 3, S uses the same key to send the same text x to another party $R.2$ via the trusted party T .

The proof that S needs to obtain in the first execution of protocol 3 consists of the two messages:

rcv-proof(encrypt(x , key), S , $R.1$)
rcv-proof(key, S , T)

Also the proof that S needs to obtain in the second execution of protocol 3 consists of the two messages:

rcv-proof(encrypt(x , key), S , $R.2$)
rcv-proof(key, S , T)

Therefore, party S obtains the message rcv-proof(key, S , T) in the first execution of protocol 3 and does not need to obtain it in the second execution of protocol 3. Thus, party S needs to execute only the first step of the protocol during its second execution of protocol 3.

To defend against this attack by a malicious party S , we prevent S from choosing any key that it likes. Instead, we require that any key that party S chooses to send text x to party R via a trusted party T should be the result of applying a secure hash function H to the concatenation of four items: text x , identifier of party S , identifier of party R , and identifier of party T :

key = $H(\text{text } x \mid \text{id of } S \mid \text{id of } R \mid \text{id of } T)$

In this case, the above attack cannot succeed because the key of the first execution of protocol 3, say key.1, will be different from the key of the second execution of protocol 3, say key.2:

key.1 = $H(\text{text } x \mid \text{id of } S \mid \text{id of } R.1 \mid \text{id of } T)$
key.2 = $H(\text{text } x \mid \text{id of } S \mid \text{id of } R.2 \mid \text{id of } T)$

8. CONCLUDING REMARKS

In this paper, we identified a rich class of nonrepudiation protocols called two-phase protocols. We discussed how to specify and verify protocols in this class and argued that the specification and verification of a two-phase protocol are simpler than those of a comparable non-two-phase protocol. Our argument is based on three observations. First, executions of two-phase protocols are deterministic. Second, the parties involved in a two-phase protocol do not need to utilize real-time clocks. Third, the messages exchanged in a two-phase protocol need to be of only two types: snd-proof messages and rcv-proof messages.

The two two-phase protocols 3 and 4 are new. In particular, neither of these protocols utilizes session labels that are distinct for distinct execution sessions of the protocol. Instead, each of these two protocols requires the encryption keys to have a special structure which allows these keys to be used as distinct session labels for distinct execution sessions.

(With the notable exception of the non-two-phase protocol in [2] and [12], most non-two-phase protocols utilize ses-

sion labels that are distinct for distinct execution sessions. Unfortunately, the non-two-phase protocol in [2] and [12] is quite involved. For example, this protocol consists of three phases: main protocol phase, abort phase, and recovery phase. Moreover, the exchanged messages in this protocol are of eight types. And execution of this protocol is highly non-deterministic.)

Acknowledgement

Research of Mohamed Gouda is supported in part by NSF award #1440035

9. REFERENCES

- [1] C. Boyd and P. Kearney. Exploring fair exchange protocols using specification animation. In *Information Security*, pages 209–223. Springer, 2000.
- [2] J. Cederquist, R. Corin, and M. T. Dashti. On the quest for impartiality: Design and analysis of a fair non-repudiation protocol. In *Proceedings of the 7th International Conference on Information and Communications Security, ICICS'05*, pages 27–39. Springer-Verlag, 2005.
- [3] J. Feng and Y. Chen. A fair non-repudiation framework for data integrity in cloud storage services. volume 2, pages 20–47. Inderscience, 2013.
- [4] S. Gürgens and C. Rudolph. Security analysis of (un-) fair non-repudiation protocols. In *Formal aspects of security*, pages 97–114. Springer, 2003.
- [5] S. Gürgens, C. Rudolph, and H. Vogt. On the security of fair non-repudiation protocols. In *Information Security*, pages 193–207. Springer, 2003.
- [6] J. L. Hernandez-Ardieta, A. I. Gonzalez-Tablas, and B. R. Alvarez. An optimistic fair exchange protocol based on signature policies. In *computers and security*, volume 27, pages 309–322. Elsevier, 2008.
- [7] S. Kremer and O. Markowitch. A multi-party non-repudiation protocol. In *Information Security for Global Information Infrastructures*, pages 271–280. Springer, 2000.
- [8] S. Kremer, O. Markowitch, and J. Zhou. An intensive survey of fair non-repudiation protocols. *Comput. Commun.*, 25(17):1606–1621, Nov. 2002.
- [9] O. Markowitch and S. Kremer. A multi-party optimistic non-repudiation protocol. In *ICISC 2000*, pages 109–122. Springer, 2001.
- [10] O. Markowitch and Y. Roggeman. Probabilistic non-repudiation without trusted third party. In *Second Conference on Security in Communication Networks, Amalfi, Italy*, 1999.
- [11] R. Oppliger. Providing certified mail services on the internet. In *Security and Privacy, IEEE*, volume 5, pages 16–22. IEEE, 2007.
- [12] J. Santiago and L. Vigneron. Optimistic non-repudiation protocol analysis. In *Information security Theory and practices. Smart cards, Mobile and ubiquitous computing systems*, pages 90–101. Springer, 2007.
- [13] Z. Xiao, Y. Xiao, and D.-C. Du. Non-repudiation in neighborhood area networks for smart grid. In *Communications Magazine, IEEE*, volume 51, pages 18–26. IEEE, 2013.

- [14] J. Zhou and D. Gollman. A fair non-repudiation protocol. In *1996 IEEE Symposium on Security and Privacy*, pages 55–61. IEEE Computer Society, 1996.
- [15] J. Zhou and D. Gollmann. An efficient non-repudiation protocol. In *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pages 126–132. IEEE, 1997.
- [16] J. Zhou and K.-Y. Lam. Undeniable billing in mobile communication. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 284–290. ACM, 1998.