Nonrepudiation Protocols Without a Trusted Party

Muqeet Ali(✉), Rezwana Reaz, and Mohamed G. Gouda

Department of Computer Science, University of Texas at Austin,
Austin, TX 78712, USA
{muqeet,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 sufficient evidence (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 sufficient evidence (that can convince a judge) that x was indeed received by R. Almost every published nonrepudiation protocol from party S to party R involves three parties: the two original parties S and R, and a third party that is often called a trusted party. A well-known nonrepudiation protocol that does not involve a third party is based on an assumption that party S knows an upper bound on the computing power of party R. This assumption does not seem reasonable especially since by violating this assumption, party R can manipulate the nonrepudiation protocol so that R obtains all its needed evidence without supplying party S with all its needed evidence. In this paper, we show that nonrepudiation protocols that do not involve a third party can be designed under reasonable assumptions. Moreover, we identify necessary and sufficient (reasonable) assumptions under which these protocols can be designed. Finally, we present the first ever ℓ-nonrepudiation protocol that involves ℓ parties (none of which is trusted), where $\ell \geq 2$.

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 along with sufficient evidence (that can convince a judge) that the text was indeed sent by S to R. Second, the protocol enables party R to receive the sent text from S and to send to S sufficient evidence (that can convince a judge) that the text was indeed received by R from S.

Each nonrepudiation protocol is also required to fulfill the following opportunism requirement. During any execution of the nonrepudiation protocol from S to R, once a party (S or R, respectively) recognizes that it has already collected all its needed evidence, then this party concludes that it gains nothing by continuing to execute the protocol and so it terminates. The other party (R or S, respectively) continues to execute the protocol with the hope that it will eventually collect all its needed evidence.

P.A. Abdulla and C. Delporte-Gallet (Eds.): NETYS 2016, LNCS 9944, pp. 1–15, 2016.
DOI: 10.1007/978-3-319-46140-3_1

The opportunism requirement which is satisfied by each party in a nonrepudiation protocol can be thought of as a failure of that party. It is important to contrast this failure model with the failure model used in the celebrated paper of Cleve [6]. Recall that Cleve's paper states an impossibility result regarding agreement on random bits chosen by two processes provided that one of the processes is faulty. In Cleve's paper the faulty process can fail at any time during the execution of the protocol. Therefore, Cleve's impossibility result is not applicable in our case, as in our model, failures occur only as dictated by the opportunism requirement.

The intuitive reasoning behind our failure model is that parties do not wish to stop executing the protocol (and thus fail) before collecting their needed evidence. Once a party recognizes that it has already collected all its needed evidence, this party decides to stop executing the protocol (i.e., fail) because it gains nothing by continuing to execute the protocol.

The opportunism requirement, that needs to be fulfilled by each nonrepudiation protocol, makes the task of designing nonrepudiation protocols very hard. This is because once a party in a nonrepudiation protocol terminates (because it has recognized that it has already collected all its needed evidence), then from where will the other party continue to receive its needed evidence?

The standard answer to this question is to assume that a nonrepudiation protocol from party S to party R involves three parties: the two original parties S and R and a third party T, which is often referred to as a trusted party. Note that the objective of each original party is to collect its own evidence, whereas the objective of the third party is to help the two original parties collect their respective evidence. Therefore, the opportunism requirement for the third party T can be stated as follows. Once T recognizes that the two original parties have already collected their evidence (or are guaranteed to collect their evidence soon), T terminates.

An execution of a nonrepudiation protocol from party S to party R that involves the three parties S, R, and T can proceed in three steps as follows:

1. Party S sends some text to party T which forwards it to party R.
2. Party T computes sufficient evidence to establish that S has sent the text to R then T forwards this evidence to R.
3. Party T computes sufficient evidence to establish that R has received the text from S then T forwards this evidence to S.

Most nonrepudiation protocols that have been published in the literature involve three parties: the two original parties and a third party [10]. A nonrepudiation protocol that does not involve a third party was published in [12]. We refer to this protocol as the MR protocol in reference to its two authors Markowitch and Roggeman. Unfortunately, the correctness of this protocol is questionable as discussed next.

In the MR protocol, party S first sends to party R the text encrypted using a symmetric key SK that only S knows. Then party S sends to party R an arbitrary number of random numbers, each of which looks like, but in fact is quite different from, the symmetric key SK. Finally, party S sends to party R

the symmetric key SK. After R receives each of these messages from S, R sends to S an ack message that acknowledges receiving the message from S.

The evidence that R needs to collect is the first message (containing the text encrypted using SK) and the last message (containing SK). The evidence that S needs to collect is the acknowledgements of R receiving the first and last messages.

In the MR protocol, when party R receives the i-th message from S, where i is at least 2, R recognizes that the message content is either a random number or the symmetric key SK. Thus, R can use the message content in an attempt to decrypt the encrypted text (which R has received in the first message). If this attempt fails, then R recognizes that the message content is a random number and proceeds to send an ack message to S. If this attempt succeeds, then R recognizes that the message content is the symmetric key SK and terminates right away (in order to fulfill the opportunism requirement) without sending the expected ack message to S. In this case, R succeeds in collecting all the evidence that it needs while S fails in collecting all the evidence that it needs.

To prevent this problematic scenario, the designers of the MR protocol adopted the following three assumptions: (1) there is a lower bound lb on the time needed by R to decrypt the encrypted text, (2) S knows an upper bound ub on the round trip delay from S to R and back to S, and (3) lb is larger than ub. Based on these assumptions, if party S sends a random number to party R and does not receive back the expected ack message for at least ub time units, then S recognizes that R has tried to cheat (but failed) by attempting to decrypt the encrypted text using the received random number. In this case, party S aborts executing the protocol and both S and R fail to collect all their needed evidence. Now, if party S sends a random number to party R and receives back the expected ack message within ub time units, then both parties continue to execute the protocol.

Unfortunately, party R can secretly decrease the value of lb, for example by employing a super computer to decrypt the encrypted text, such that assumption (3) above is violated. By violating assumption (3), the attempts of R to cheat can go undetected by party S and the MR protocol can end up in a compromised state where party R has terminated after collecting all its needed evidence but party S is still waiting to collect its needed evidence that will never arrive. This problematic scenario calls into question the correctness of the MR protocol.

In this paper, we discuss how to design nonrepudiation protocols that do not involve a trusted party. We make the following five contributions:

1. We first state the round-trip assumption as follows: Party R knows an upper bound on the round trip delay from R to S and back to R. Then we show that adopting this assumption is both necessary and sufficient for designing nonrepudiation protocols from S to R that do not involve a third trusted party and where no sent message is lost.
2. Our sufficiency proof in 1 consists of designing the first (provably correct) nonrepudiation protocol from S to R that does not involve a third party and where no sent message is lost.

3. We state the bounded-loss assumption as follows: Party R knows an upper bound on the number of messages that can be lost during any execution of the protocol. Then we show that adopting both the round-trip assumption and the bounded-loss assumption is both necessary and sufficient in designing nonrepudiation protocols from S to R that do not involve a third trusted party and where every sent message may be lost.
4. Our sufficiency proof in 3 consists of designing the first (provably correct) nonrepudiation protocol from S to R that does not involve a third party and where every sent message may be lost.
5. We extend the nonrepudiation protocol in 2 that involves only two parties, S and R, into an ℓ-nonrepudiation protocol that involves ℓ parties (none of which is trusted), where $\ell \geq 2$.

The proofs of all the theorems that appear in this paper are described in the technical report [1]

2 Related Work

The most cited nonrepudiation protocol was published by Zhou and Gollmann in 1996 [15]. 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 [16] 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.)

Later, 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 $R1, \cdots, R.n$. They referred to the generalized protocols as multiparty protocols, and published the first two multiparty protocols [9,11].

Most nonrepudiation protocols involve the services of a third party for successful completion of the protocol [8,9,11,15,16]. There are protocols which involve a third party in every execution of the protocol from party S to party R [9,15]. These protocols are said to have on-line third party. The involvement of third party in every execution can become a bottleneck therefore protocols were proposed to limit the involvement of third party [8,11,16]. In such protocols, the third party is not involved in every execution of the protocol. Such protocols are said to have an off-line third party, and are known as optimistic nonrepudiation protocols. Nonrepudiation has also found a number of applications [14,17].

The problem of designing nonrepudiation protocols is similar to the problem of designing contract signing protocols. However, in contract signing protocols the contract C to be signed is known to all parties before the execution of contract signing protocol begins, while in nonrepudiation protocols the text x is only known to party S before the execution of the nonrepudiation protocol begins. Most contract signing protocols do make use of a trusted third party. See for example [2,3]. However, the trusted third party in some of the published

contract signing protocols is rather weak [4,13]. For example, the trusted third party in Rabin's protocol [13] does not receive any messages from either of the two parties S or R. Rather, Rabin's third party periodically generates random numbers and sends them to the original parties S and R.

Some published contract signing protocols do not employ a trusted third party. See for example [5,7]. However, the opportunism requirement that is fulfilled by these protocols is weaker than our opportunism requirement. Thus, in each of these protocols, a party (S or R) may continue to execute the contract signing protocol even after this party recognizes that it has collected all its needed evidence with the hope that the cost of processing its collected evidence can be reduced dramatically.

3 Nonrepudiation Protocols

In this section, we present our specification of a nonrepudiation protocol that does not involve a third party. A nonrepudiation protocol from party S to party R that does not involve a third party is a communication protocol between parties S and R that fulfills the following requirements.

(a) **Message Loss:** During each execution of the protocol, each message that is sent by either party (S or R, respectively) is eventually received by the other party (R or S, respectively).

(b) **Message Alternation:** During any execution of the protocol, the two parties S and R exchange a sequence of messages. First, party S sends msg.1 to party R. Then when party R receives msg.1, it sends back msg.2 to party S, and so on. At the end when R receives msg.$(r-1)$, where r is an even integer whose value is at least 2, R sends back msg.r to S. This exchange of messages can be represented as follows:

$$
\begin{aligned}
&S \rightarrow R\text{: msg.}1\\
&S \leftarrow R\text{: msg.}2\\
&\cdots\\
&S \rightarrow R\text{: msg.}(r-1)\\
&S \leftarrow R\text{: msg.}r
\end{aligned}
$$

(c) **Message Signatures:** Party S has a private key that only S knows and the corresponding public key that all parties know. Party S uses its private key to sign every message before sending this message to R so that S can't later repudiate that it has generated this message. Similarly, Party R has a private key that only R knows and the corresponding public key that all parties know. Party R uses its private key to sign every message before sending this message to S so that R can't later repudiate that it has generated this message.

(d) **Collected Evidence:** Both parties S and R collect evidence during execution of the protocol. The evidence collected by party S is a subset of those

messages received by party S (from party R). Similarly, the evidence collected by party R is a subset of those messages received by party R (from party S).

(e) **Guaranteed Termination:** Every execution of the protocol is guaranteed to terminate in a finite time.

(f) **Termination Requirement:** Party S terminates only after S sends some text to R and only after S receives from R sufficient evidence to establish that R has indeed received the sent text from S. Similarly, party R terminates only after R receives from S both the sent text and sufficient evidence to establish that S has indeed sent this text to R.

(g) **Opportunism Requirement:** If during any execution of the protocol, party S recognizes that it has already sent some text to R and has later received from R sufficient evidence to establish that R has indeed received this text, then S terminates. Similarly, if during any execution of the protocol, party R recognizes that it has received from S some text and sufficient evidence to establish that S has indeed sent this text, then R terminates.

(h) **Judge:** Anytime after execution of the nonrepudiation protocol terminates, each of the two parties, S or R, can submit its collected evidence to a judge that can decide whether the submitted evidence is valid and should be accepted or it is invalid and should be rejected. The decision of the judge is final and legally binding on both S and R. To help the judge make the right decision, we assume that the judge knows the public keys of S and R. We also assume that the judge has a public key (that both S and R know) and a corresponding private key (that only the judge knows). (Note that the role of the judge is different than that of a trusted third party. The trusted third party is used to generate and distribute the needed evidence to the two parties, S and R. Therefore, the trusted third party is directly involved during the execution of the nonrepudiation protocol. The judge, however, is not directly involved during the execution of the protocol, and it never generates nor distributes any part of the needed evidence to either party. The judge only verifies the submitted evidence after it has already been collected during execution of the protocol. In fact every nonrepudiation protocol that has been published in the past has both a trusted third party and a judge)

The opportunism requirement needs some explanation. Once party S recognizes that it has already collected sufficient evidence to establish that party R has indeed received the text from S, S concludes that it gains nothing by continuing to participate in executing the protocol and so S terminates. (In this case, only R may gain by continuing to participate in executing the protocol.)

Similarly, once party R recognizes that it has already collected sufficient evidence to establish that party S has indeed sent the text to R, R concludes that it gains nothing by continuing to participate in executing the protocol and so R terminates.

Next, we state a condition, named the round-trip assumption and show in the next section that adopting this assumption is both necessary and sufficient to design nonrepudiation protocols from party S to party R that do not involve a third party.

Round-Trip Assumption: Party R knows an upper bound t (in time units) on the round trip delay from R to S and back to R.

4 Necessary and Sufficient Conditions for Nonrepudiation Protocols

We prove that it is necessary to adopt the round-trip assumption when designing nonrepudiation protocols from party S to party R that do not involve a third party.

Theorem 1. *In designing a nonrepudiation protocol from party S to party R, that does not involve a third party, it is necessary to adopt the round-trip assumption. (The proof of this theorem is omitted due to lack of space.)*

Next, we prove that it is sufficient to adopt the round-trip assumption in order to design a nonrepudiation protocol from party S to party R that does not involve a third party.

Theorem 2. *It is sufficient to adopt the round-trip assumption in order to design a nonrepudiation protocol from party S to party R that does not involve a third party.*

Proof. We present a design of a nonrepudiation protocol from party S to party R that does not involve a third party and show that correctness of this protocol is based on adopting the round-trip assumption. Our presentation of this protocol consists of four steps. In each step, we start with a version of the protocol then show that this version is incorrect (by showing that it violates one of the requirements in Sect. 3). We then proceed to modify this protocol version in an attempt to make it correct. After four steps, we end up with a correct nonrepudiation protocol (that satisfies all eight requirements in Sect. 3).

First Protocol Version: In this protocol version, party S sends a txt message to party R which replies by sending back an ack message. The exchange of messages in this protocol version can be represented as follows.

$$S \rightarrow R\text{: txt}$$
$$S \leftarrow R\text{: ack}$$

The txt message contains: (1) the message sender S and receiver R, (2) the text that S needs to send to R, and (3) signature of the message using the private key of the message sender S. Similarly, the ack message contains: (1) the message

sender R and receiver S, (2) the text that S needs to send to R, and (3) signature of the message using the private key of the message sender R.

The txt message is the evidence that R needs to collect and can later present to the judge to get the judge to declare that the text in the message was indeed sent by S to R. Similarly, the ack message is the evidence that S needs to collect and can later present to the judge to get the judge to declare that the text in the message was indeed received by R from S.

This protocol version is incorrect for the following reason. When R receives the txt message, it recognizes that it has already collected sufficient evidence to establish that S has indeed sent the text to R and so R terminates, by the opportunism requirement, before it sends the ack message to S. Party S ends up waiting indefinitely for the ack message that will never arrive violating the guaranteed termination requirement.

To make this protocol version correct, we need to devise a technique by which R does not recognize that it has collected sufficient evidence to establish that S has indeed sent the text to R, even after R has collected such evidence.

Second Protocol Version: In this protocol version, party S sends n txt messages to party R, where n is a positive integer selected at random by S and is kept as a secret from R. The exchange of messages in this protocol version can be represented as follows.

$$
\begin{aligned}
&S \rightarrow R\text{: txt.}1\\
&S \leftarrow R\text{: ack.}1\\
&\cdots\\
&S \rightarrow R\text{: txt.}n\\
&S \leftarrow R\text{: ack.}n
\end{aligned}
$$

Each txt.i message contains: (1) the message sender S and receiver R, (2) the text that S needs to send to R, (3) the sequence number of the message i, and (4) signature of the message using the private key of the message sender S. Similarly, each ack.i message contains: (1) the message sender R and receiver S, (2) the text that S needs to send to R, (3) the sequence number of the message i, and (4) signature of the message using the private key of the message sender R.

The txt.n message is the evidence that R needs to collect and can later present to the judge to get the judge to declare that the text in the message was indeed sent by S to R. Similarly, the ack.n message is the evidence that S needs to collect and can later present to the judge to get the judge to declare that the text in the message was indeed received by R from S.

When R receives the txt.n message from S, then (because R does not know the value of n) R does not recognize that it has just received sufficient evidence to establish that S has indeed sent the text to R. Thus, R does not terminate and instead proceeds to send the ack.n message to S.

When S receives the ack.n message from R, then (because S knows the value of n) S recognizes that it has just received sufficient evidence to establish that R

has received the text sent from S to R. Thus, by the opportunism requirement, S terminates and R ends up waiting indefinitely for the txt.$(n+1)$ message that will never arrive violating the guaranteed termination requirement.

This protocol version is incorrect. To make it correct, we need to devise a technique by which party R recognizes, after S terminates, that R has already collected sufficient evidence to establish that S has indeed sent the text to R.

Third Protocol Version: This protocol version is designed by modifying the second protocol version (discussed above) taking into account the adopted round-trip assumption, namely that R knows an upper bound t (in time units) on the round trip delay from R to S and back to R.

The exchange of messages in the third protocol version is the same as that in the second protocol version. However, in the third protocol version, every time R sends an ack.i message to S, R activates a time-out to expire after t time units.

If R receives the next txt.$(i+1)$ before the activated time-out expires, then R cancels the timeout. If the activated time-out expires before R receives the next txt.$(i+1)$ message, then R recognizes that the last received txt.i message is in fact the txt.n message and so R recognizes that it has already collected sufficient evidence to establish that S has already sent the text to R and so R terminates (by the opportunism requirement). Execution of this protocol version can be represented as follows:

$S \rightarrow R$: txt.1

$S \leftarrow R$: ack.1; R activates time-out

$S \rightarrow R$: txt.2; R cancels time-out

...

$S \rightarrow R$: txt.n; R cancels time-out

$S \leftarrow R$: ack.n; R activates time-out; S terminates

time-out expires; R terminates

This protocol version still has a problem. After execution of the protocol terminates, party R may decide to submit its collected evidence, namely the txt.n message, to the judge so that the judge can certify that S has indeed sent the text in the txt.n message. The judge can make this certification if it observes that the sequence number of the txt.n message equals the random integer n that S selected when execution of the protocol started. Unfortunately, the value of n is not included in the txt.n message.

Similarly, after execution of the protocol terminates, party S may decide to submit its collected evidence, namely the ack.n message, to the judge so that the judge can certify that R has indeed received the text in the ack.n message. The judge can make this certification if it observes that the sequence number of the ack.n message equals the random integer n that S selected when execution of the protocol started. Unfortunately, the value of n is not included in the ack.n message.

To solve these two problems, we need to devise a technique by which the value of n is included in every txt.i message and every ack.i message such that the following two conditions hold. First, the judge can extract the value of n from any txt.i or ack.i message. Second, party R can't extract the value of n from any txt.i message nor from any ack.i message.

Fourth Protocol Version: In this protocol version, two more fields are added to each txt.i message and each ack.i message. The first field stores the encryption of n using a symmetric key KE that is generated by party S and is kept as a secret from party R. The second field stores the encryption of the symmetric key KE using the public key of the judge.

These two fields are computed by party S when execution of the protocol starts and are included in every txt.i message before this message is sent from S to R. When party R receives a txt.i message from party S, party R copies these two fields from the received txt.i message into the next ack.i message before this message is sent from R to S.

After execution of this protocol version terminates, party S can submit its collected evidence, namely the last ack.i message that S has received from R, to the judge so that the judge can examine the ack.i message and certify that R has received the text in this message from S. The judge makes this certification by checking, among other things, that the sequence number i of the ack.i message is greater than or equal to n. The judge then forwards its certification to party S.

Similarly, after execution of this protocol version terminates, party R can submit its collected evidence, namely the last txt.i message that R has received from S, to the judge so that the judge can examine the txt.i message and certify that S has sent the text in this message to R. The judge makes this certification by checking, among other things, that the sequence number i of the txt.i message is greater than or equal to n. The judge then forwards its certification to party R.

Note that the judge can certify at most one ack.i message from party S, and at most one txt.i message from party R. This restriction forces S to send to the judge only the last ack.i message that S has received from R. This restriction also forces R to send to the judge only the last txt.i message that R has received from S. □

5 Nonrepudiation Protocols with Message Loss

In the remainder of this paper, we consider a richer class of nonrepudiation protocols where sent messages may be lost before they are received. Each protocol in this class is required to fulfill the following requirements.

(a) **Message Loss:** During each execution of the protocol, each message that is sent by either party (S or R, respectively) can be lost before it is received by the other party (R or S, respectively).
(b) **Message Alternation:** During any execution of the protocol where no sent message is lost, the two parties S and R exchange a sequence of messages.

First, party S sends msg.1 to party R. Then when party R receives msg.1, it sends back msg.2 to party S, and so on. At the end when R receives msg.$(r-1)$, where r is an even integer whose value is at least 2, R sends back msg.r to S. This exchange of messages can be represented as follows:

$$
\begin{aligned}
&S \rightarrow R\text{: msg.}1\\
&S \leftarrow R\text{: msg.}2\\
&\cdots\\
&S \rightarrow R\text{: msg.}(r-1)\\
&S \leftarrow R\text{: msg.}r
\end{aligned}
$$

(c) **Message Signatures:** This requirement is the same as the message signature requirement in Sect. 3.
(d) **Collected Evidence:** This requirement is the same as the collected evidence requirement in Sect. 3.
(e) **Guaranteed Termination:** This requirement is the same as the guaranteed termination requirement in Sect. 3.
(f) **Termination Requirement:** This requirement is the same as the termination requirement in Sect. 3.
(g) **Opportunism Requirement:** This requirement is the same as the opportunism requirement in Sect. 3.
(h) **Judge:** This requirement is the same as the requirement of the judge in Sect. 3.

Next, we state a condition, named the bounded-loss assumption. We then show in the next section that adopting this assumption along with the round-trip assumption (stated in Sect. 3) is both necessary and sufficient to design nonrepudiation protocols from party S to Party R that do not involve a third party and where sent messages can be lost.

Bounded-Loss Assumption: Party R knows an upper bound K on the number of messages that can be lost during any execution of the nonrepudiation protocols.

6 Necessary and Sufficient Conditions for Nonrepudiation Protocols with Message Loss

We prove that it is necessary to adopt both the round-trip assumption and the bounded-loss assumption when designing nonrepudiation protocols from party S to party R that do not involve a third party and where sent messages may be lost.

Theorem 3. *In designing a nonrepudiation protocol from party S to party R that does not involve a third party and where sent messages may be lost, it is necessary to adopt the round-trip assumption. (The proof of this theorem is omitted due to lack of space.)*

Theorem 4. *In designing a nonrepudiation protocol from party S to party R that does not involve a third party and where sent messages may be lost, it is necessary to adopt the bounded-loss assumption. (The proof of this theorem is omitted due to lack of space)*

Next, we prove that it is sufficient to adopt both the round-trip assumption and the bounded-loss assumption in order to design a nonrepudiation protocol from party S to party R that does not involve a third party and where sent messages may be lost.

Theorem 5. *It is sufficient to adopt both the round-trip assumption and the bounded-loss assumption in order to design a nonrepudiation protocol from party S to party R that does not involve a third party and where sent messages may be lost.*

Proof. In our proof of Theorem 2, we adopted the round-trip assumption in order to design a nonrepudiation protocol, which we refer to in this proof as protocol P, from party S to party R that does not involve a third party and where no sent message is ever lost. In the current proof, we adopt the bounded-loss assumption in order to modify protocol P into protocol Q, which is a nonrepudiation protocol from party S to party R that does not involve a third party and where sent messages may be lost.

In protocol P, every time party R sends an ack.i message to party S, R activates a time-out to expire after t time units, where (by the round-trip assumption) t is an upper bound on the round trip delay from R to S and back to R. Because no sent message is ever lost in protocol P, then if the activated time-out expires before R receives the next txt.$(i+1)$ message from S, R concludes that S has already terminated, the next txt.$(i+1)$ message will never arrive, and R has already collected sufficient evidence to establish that S has sent the text to R. In this case, R also terminates fulfilling the opportunism requirement.

In protocol Q, every time party R sends an ack.i message to party S, R activates a time-out to expire after t time units. However, because every sent message in protocol Q may be lost, if the activated time-out expires before R receives the next txt.$(i+1)$ message from S, then R concludes that either the ack.i message or the txt.$(i+1)$ message is lost, and in this case R sends the ack.i message once more to S and activates a new time-out to expire after t time units, and the cycle repeats.

By the bounded-loss assumption, R knows an upper bound K on the number of messages that can be lost during any execution of protocol Q. Therefore, the cycle of R sending an ack.i message then the activated time-out expiring after t time units can be repeated at most K times.

If the activated time-out expires for the $(K+1)$-th time, then R concludes that S has already terminated, the next txt.$(i+1)$ message will never arrive, and R has already collected sufficient evidence to establish that S has sent the text to R. In this case, R terminates fulfilling the opportunism requirement. □

7 An ℓ-Nonrepudiation Protocol

In the Proof of Theorem 2, we presented a nonrepudiation protocol from party S to party R that does not involve a trusted party and where no sent message is lost. In this section, we discuss how to extend this protocol to a nonrepudiation protocol that involves ℓ parties, namely $P1, \cdots, P\ell$, and satisfies three conditions: (1) $\ell \geq 2$, (2) none of the involved parties in the protocol is a trusted party, and (3) no sent message during any execution of the protocol is lost. We refer to this extended protocol as an ℓ-nonrepudiation protocol.

The objectives of an ℓ-nonrepudiation protocol are as follows. For each two parties in the protocol, say Pi and Pj, the protocol achieves two objectives:

1. One of the two parties, say Pi, is enabled to send the text to the other party Pj and to receive from Pj sufficient evidence that can convince a judge that Pj has indeed received the text from Pi.
2. The other party Pj is enabled to receive the text from Pi and to receive sufficient evidence from Pi that can convince a judge that Pi has indeed sent the text to Pj.

Therefore, each party Pi ends up collecting sufficient evidence from every other party Pj indicating that either Pj has indeed received the text from Pi or Pj has indeed sent the text to Pi.

Before we describe our ℓ-nonrepudiation protocol, as an extension of the 2-nonrepudiation protocol in the proof of Theorem 2, we need to introduce two useful concepts: parent and child of a party Pi. Each of the parties $P1, \cdots, P(i-1)$ is called a parent of party Pi. Also each of the parties $P(i+1), \cdots, P\ell$ is called a child of party Pi. Note that party $P1$ has no parents and parent $P\ell$ has no children. Note also that the number of parents plus the number of children for each party is $(\ell - 1)$.

Execution of our ℓ-nonrepudiation protocol proceeds as follows:

1. Party $P1$ starts by sending a txt.1 message to each one of its children.
2. When a party Pi, where $i \neq 1$ and $i \neq \ell$, receives a txt.1 message from each one of its parents, party Pi sends a txt.1 message to each one of its children.
3. When party $P\ell$ receives a txt.1 message from each one of its parents, party $P\ell$ sends back an ack.1 message to each one of its parents.
4. When a party Pi, where $i \neq 1$ and $i \neq \ell$, receives an ack.1 message from each one of its children, party Pi sends an ack.1 message to each one of its parents.
5. When party $P1$ receives an ack.1 message from each one of its children, party $P1$ sends a txt.2 message to each one of its children, and the cycle consisting of Steps 2, 3, 4, and 5 is repeated n times until $P1$ receives an ack.n message from each one of its children. In this case, party $P1$ collects as evidence the ack.n messages that $P1$ has received from all its children, then $P1$ terminates.

6. Each party Pi, where $i \neq 1$, waits to receive a txt.$(n+1)$ message (that will never arrive) from each one of its parents, then times out after $(i-1)$*T time units, collects as evidence the txt.n messages that Pi has received from all its parents and the ack.n messages that Pi has received from all its children, then Pi terminates.

Note that T is an upper bound on the round trip delay from any party Pi to any other party Pj and back to Pi. It is assumed that each party, other than $P1$, knows this upper bound T.

8 Concluding Remarks

In this paper, we address several problems concerning the design of nonrepudiation protocols from a party S to a party R that do not involve a trusted third party. In such a protocol, S sends to R some text x along with sufficient evidence to establish that S is the party that sent x to R, and R sends to S sufficient evidence that R is the party that received x from S.

Designing such a protocol is not an easy task because the protocol is required to fulfill the following opportunism requirement. During any execution of the protocol, once a party recognizes that it has received its sufficient evidence from the other party, this party terminates right away without sending any message to the other party. In this case, the other party needs to obtain its evidence without the help of the first party. (To fulfill this opportunism requirement, most published nonrepudiation protocols involve a trusted third party T so that when one of the two original parties recognizes that it has already received its sufficient evidence and terminates, the other party can still receive its evidence from T.)

Our main result in this paper is the identification of two simple conditions that are both necessary and sufficient for designing nonrepudiation protocols that do not involve a trusted third party.

In proving that these two conditions are sufficient for designing nonrepudiation protocols, we presented an elegant nonrepudiation protocol that is based on the following novel idea. By the time party S recognizes that it has received its evidence, party S has already sent to R its evidence, but R has not yet recognized that it has received all its evidence. In this case, S terminates as dictated by the opportunism requirement but R continues to wait for the rest of its evidence from S. Eventually R times-out and recognizes that S will not send any more evidence. This can only mean that party S has terminated after it has already sent all the evidence to R. Thus, R terminates as dictated by the opportunism requirement.

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

References

1. Ali, M., Reaz, R., Gouda, M.: Nonrepudiation protocols without a trusted party. University of Texas at Austin, Department of Computer Science. TR-16-02 (regular tech. report) (2016)
2. Asokan, N., Schunter, M., Waidner, M.: Optimistic protocols for fair exchange. In: Proceedings of the 4th ACM Conference on Computer and Communications Security, CCS 1997, pp. 7–17. ACM, New York (1997)
3. Baum-Waidner, B.: Optimistic asynchronous multi-party contract signing with reduced number of rounds. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 898–911. Springer, Heidelberg (2001)
4. Ben-Or, M., Goldreich, O., Micali, S., Rivest, R.L.: A fair protocol for signing contracts (extended abstract). In: Brauer, W. (ed.) ICALP 1985. LNCS, vol. 194, pp. 43–52. Springer, Heidelberg (1985)
5. Boneh, D., Naor, M.: Timed commitments. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 236–254. Springer, Heidelberg (2000)
6. Cleve, R.: Limits on the security of coin flips when half the processors are faulty. In: Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, STOC 1986, pp. 364–369. ACM, New York (1986)
7. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM **28**(6), 637–647 (1985)
8. Hernandez-Ardieta, J.L., Gonzalez-Tablas, A.I., Alvarez, B.R.: An optimistic fair exchange protocol based on signature policies. Comput. Secur. **27**(7), 309–322 (2008)
9. Kremer, S., Markowitch, O.: A multi-party non-repudiation protocol. In: Qing, S., Eloff, J.H.P. (eds.) Information Security for Global Information Infrastructures. IFIP, vol. 47, pp. 271–280. Springer, New York (2000)
10. Kremer, S., Markowitch, O., Zhou, J.: An intensive survey of fair non-repudiation protocols. Comput. Commun. **25**(17), 1606–1621 (2002)
11. Markowitch, O., Kremer, S.: A multi-party optimistic non-repudiation protocol. In: Won, D. (ed.) ICISC 2000. LNCS, vol. 2015, p. 109. Springer, Heidelberg (2001)
12. Markowitch, O., Roggeman, Y.: Probabilistic non-repudiation without trusted third party. In: Second Conference on Security in Communication Networks, Amalfi, Italy (1999)
13. Rabin, M.O.: Transaction protection by beacons. J. Comput. Syst. Sci. **27**(2), 256–267 (1983)
14. Xiao, Z., Xiao, Y., Du, D.C.: Non-repudiation in neighborhood area networks for smart grid. IEEE Commun. Mag. **51**(1), 18–26 (2013)
15. Zhou, J., Gollman, D.: A fair non-repudiation protocol. In: 1996 IEEE Symposium on Security and Privacy, pp. 55–61. IEEE Computer Society (1996)
16. Zhou, J., Gollmann, D.: An efficient non-repudiation protocol. In: 10th Proceedings of Computer Security Foundations Workshopp, pp. 126–132. IEEE (1997)
17. Zhou, J., Lam, K.Y.: Undeniable billing in mobile communication. In: Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking, pp. 284–290. ACM (1998)