Some Remarks on a Fair Exchange Protocol

Jianying Zhou, Robert Deng, and Feng Bao

Kent Ridge Digital Labs
21 Heng Mui Keng Terrace
Singapore 119613
{jyzhou,deng,baofeng}@krdl.org.sg

Abstract. Fair exchange turns out to be an increasingly important topic due to the rapid growth of electronic commerce. An exchange is deemed to be fair if at the end of exchange, either each party receives the expected item or neither party receives any useful information about the other's item. Several protocols for fair exchange have been proposed in recent years. In this paper, we first examine a newly published fair exchange protocol and point out its flaws and weaknesses. We then put forward a more efficient and secure protocol and give an informal analysis.

Keywords: fair exchange, certified mail, secure electronic commerce

1 Introduction

Due to the rapid growth of electronic commerce nowadays, a related security issue on the fair exchange of electronic data between two parties over computer networks is of more and more importance. We can find various exchange instances in different types of commercial activity [2]:

- In contract signing, two parties exchange their non-repudiable commitment to the contract text.
- In purchasing, a payment is exchanged for a valuable item.
- In certified mail, a message is exchanged for an acknowledgement of receipt.

An exchange is *fair* if at the end of exchange, either each party receives the expected item or neither party receives any useful information about the other's item.

In electronic commerce scenarios, exchanges have to be carried over insecure networks and transacting parties may not trust each other. There could be subsequent disputes about what was exchanged during a transaction even if the exchange itself was completed fairly. In this case, evidence should be accumulated during the exchange to enable the settlement of any future disputes.

Solutions to the fair exchange problem reported in the literature fall into two categories:

H. Imai, Y. Zheng (Eds.): PKC 2000, LNCS 1751, pp. 46–57, 2000.

- *Gradual exchange protocols* [4,5,6,8,12,13] where two parties gradually disclose the expected items by many steps.
- *Third party protocols* [1,2,3,7,9,11,14,15] which make use of an on-line or off-line (trusted) third party.

The gradual exchange solutions may have theoretical value but seem to be too cumbersome for actual implementation because of the high communication overhead. Hence, recent research mainly focuses on the third party solutions.

As the use of (trusted) third party *TTP* in fair exchange may cause the bottleneck problem, it is desirable to minimize the *TTP*'s involvement when designing efficient fair exchange protocols. Such an attempt has been made in [14], where the *TTP* acts as a *notary* rather than a delivery authority. However, the *TTP* still needs to be involved in each protocol run, though this might be necessary in some applications [15].

The *TTP*'s involvement is further reduced in [1,3,15], where transacting parties are willing to resolve communications problems between themselves and turn to the *TTP* only as a last recourse. However, only the risk-taking party (originator) is allowed to invoke the *TTP*, the responder may not know the final state of exchange in time. If a short time limit is imposed on a protocol run, the originator may not be quick enough to invoke the *TTP* for recovery thus the fairness will be destroyed.

Asokan, Shoup and Waidner proposed a generic fair exchange protocol in [2] which uses the *TTP* only in the case of exceptions and tolerates temporary failures in the communication channels to the *TTP*. In addition, it allows either party to unilaterally bring a protocol run to completion without losing fairness.

In this paper, we examine an instantiation of their generic fair exchange protocol for certified mail and put forward proposals for improvement. The following general notation is used throughout the paper.

- X, Y: concatenation of two messages X and Y.
- $H(X)$: a one-way hash function applied to message X.
- $eK(X)$ and $dK(X)$: encryption and decryption of message X with key K.
- $sS_A(X)$: principal A's digital signature on message X with the private key S_A. The algorithm is assumed to be a 'signature with appendix', and the message is not recoverable from the signature.
- $A \rightarrow B : X$: principal A dispatches message X addressed to principal B.

2 ASW Protocol

A protocol for certified mail was proposed in [2] (see Figure 4 in the original paper). In this section, we give a brief description of the protocol, which is referred to as ASW protocol herein.

In certified mail, a sender O wants to send a mail message M to a receiver R. The sender O requires that the receiver R not be able to deny receiving the message M. To achieve this, O needs a non-repudiation of receipt token from R in exchange for the message M. Thus certified mail is a fair exchange of the message and its non-repudiation of receipt token.

ASW protocol has three sub-protocols: *exchange*, *abort*, and *resolve*. In the normal case, only the *exchange* sub-protocol is executed. The other two sub-protocols are used only if O or R presumes that something has gone wrong and decides to forcibly complete a protocol run. This is an indeterminate choice made *locally* by O or R without losing fairness. A (trusted) third party TTP will be invoked in the *abort* and *resolve* sub-protocols. It is assumed that communication channels between any two parties are *confidential*. It is further assumed that the communication channels between the TTP and each transacting party (O and R) are *resilient*, i.e. messages inserted into a resilient channel will eventually be delivered.

The notation below is used in the description of ASW protocol.

- P_{TTP}: the TTP's public encryption key.
- V_O and V_R: verification keys of O and R respectively.
- key_O and key_R: random numbers generated by O and R respectively.
- $C = eP_{TTP}(M, key_O, V_O, V_R)$: encrypted mail message.
- $H(M)$: receipt text of a mail message M.

The *exchange* sub-protocol is as follows.

1. $O \rightarrow R : me1 = V_O, V_R, TTP, C, H(M),\ sS_O(V_O, V_R, TTP, C, H(M))$
   IF R gives up **THEN** quit **ELSE**
2. $R \rightarrow O : me2 = H(key_R),\ sS_R(me1, H(key_R))$
   IF O gives up **THEN** *abort* **ELSE**
3. $O \rightarrow R : me3 = M, key_O$
   IF R gives up **THEN** *resolve_R* **ELSE**
4. $R \rightarrow O : me4 = key_R$
   IF O gives up **THEN** *resolve_O*

The *abort* sub-protocol is as follows.

1. $O \rightarrow TTP : ma1 = aborted, me1,\ sS_O(aborted, me1)$
   IF R has resolved **THEN** *resolve_O* **ELSE**
2. $TTP \rightarrow O : abort\_token = ma1,\ sS_{TTP}(ma1)$

The *resolve_R* sub-protocol is as follows.

1. $R \rightarrow TTP : mrr1 = V_R, me1, me2, key_R$
   IF aborted **THEN**
2. $TTP \rightarrow R : mrr2 = abort\_token$
   ELSE
3. $TTP \rightarrow R : mrr3 = M, key_O$

The *resolve_O* sub-protocol is as follows.

1. $O \rightarrow TTP : mro1 = V_O, me1, me2, M, key_O$
   IF aborted **THEN**
2. $TTP \rightarrow O : mro2 = abort\_token$
   ELSE
3. $TTP \rightarrow O : affidavit\_token = affidavit, mro1,\ sS_{TTP}(affidavit, mro1)$

In the *exchange* sub-protocol, if R decides to give up before sending $me2$, it can simply terminate the protocol run without losing fairness. If O decides to give up after sending $me1$ (usually because O does not receive $me2$ within a reasonable time), it invokes the TTP by running the *abort* sub-protocol. If R decides to give up after sending $me2$ (typically because R does not receive $me3$ in time), it invokes the TTP by running the *resolve_R* sub-protocol. If O decides to give up after sending $me3$ (typically because O does not receive $me4$ in time), it invokes the TTP by running the *resolve_O* sub-protocol.

The *abort* sub-protocol is used by O to abort the protocol so that the TTP will not resolve the protocol at a later time. The *resolve_O* and *resolve_R* sub-protocols are used by O and R respectively to force a successful termination. Clearly, only one of the *abort* or *resolve* sub-protocols can succeed for a given instance of exchange. On the TTP's system, each of the *abort* and *resolve* sub-protocol is guaranteed to be atomic.

In ASW protocol, either a tuple $(me1, me2, key_R)$ or an *affidavit_token* serves as a valid receipt for a mail message.

3 Some Remarks

The requirements for fair exchange were formulated in [2]:

- *Effectiveness.* If two parties behave correctly, they will receive the expected items without any involvement of the TTP.
- *Fairness.* After completion of a protocol run, either each party receives the expected item or neither party receives any useful information about the other's item.
- *Timeliness.* At any time during a protocol run, each party can unilaterally choose to terminate the protocol without losing fairness.
- *Non-repudiation.* If an item has been sent from party O to party R, O cannot deny origin of the item and R cannot deny receipt of the item.
- *Verifiability of Third Party.* If the third party misbehaves, resulting in the loss of fairness for a party, the victim can prove the fact in a dispute.

ASW protocol was designed to meet the above requirements. Nevertheless, some problems might exist.

Remark 1. *The abort sub-protocol is flawed.*

The *abort* sub-protocol is initiated by O, usually because O does not receive $me2$ in time. If R has already resolved the protocol, O is asked to initiate the *resolve_O* sub-protocol [1]. However, O is unable to initiate the *resolve_O* sub-protocol without $me2$. Thus O has neither an *abort_token* nor an *affidavit_token* at the end of a protocol run while R has received the mail message.

In addition, O may misbehave by initiating the *abort* sub-protocol after it has initiated the *resolve_O* sub-protocol and obtained an *affidavit_token*. If the TTP sends an *abort_token* to O in this case, the TTP will be in a dilemma when R initiates the *resolve_R* sub-protocol later.

These problems also exist in their generic protocol for fair exchange. A fixed *abort* sub-protocol is as follows.

1. $O \rightarrow TTP : ma1 = aborted, me1,\ sS_O(aborted, me1)$
   IF O has resolved **THEN**
2. $TTP \rightarrow O : ma2 = affidavit\_token$
   ELSE IF R has resolved **THEN**
3. $TTP \rightarrow O : ma3 = me2, key_R$
   ELSE
4. $TTP \rightarrow O : abort\_token = ma1,\ sS_{TTP}(ma1)$

Remark 2. *There is some redundancy in the resolve_O sub-protocol.*

O need not send M and key_O to the TTP in the *resolve_O* sub-protocol. With $me1$ and $me2$, it is sufficient for the TTP to issue the *affidavit_token*. If R resolves the protocol later, the TTP can obtain M by decrypting C contained in $me1$.

key_O was used in ASW protocol as a part of non-repudiation of origin token. In fact, $me1$ is already a complete non-repudiation of origin token. Hence, key_O can be omitted from all sub-protocols.

Remark 3. *The protocol performance may degrade when transmitting large mail messages.*

The *exchange* sub-protocol may become less efficient when the mail message is large since a mail message needs to be transmitted twice, that is cipher text C in $me1$ and plain text M in $me3$. The communication overheads will increase even more when the *abort* or *resolve* sub-protocols are invoked.

[1] Actually, R can initiate the *resolve_R* sub-protocol before sending $me2$ to O.

It is also a burden to the *TTP* to deal with the whole mail messages when the *abort* or *resolve* sub-protocols are invoked. The *TTP* needs a large space to store those mail messages and tokens safely until both parties have retrieved the expected items.

Remark 4. *The privacy of mail messages may not be well protected.*

As we just mentioned, if the *abort* or *resolve* sub-protocols are invoked, the content of a mail message has to be disclosed to the *TTP*. Such a situation may be undesirable to the parties who want to exchange mail messages secretly between themselves.

Although it is possible to encrypt the mail message either with a key shared between two parties or with the receiver's public encryption key before exchange, this will make the dispute resolution more complicated. If there is no evidence to prove what key is used and the times of encryption performed, the content of a mail message will be in dispute.

Remark 5. *The encrypted data in the non-repudiation of receipt token may not be publicly verifiable, which makes the dispute resolution inefficient.*

Suppose O sends the following $me1$ to R where $M' \neq M$.

$$me1 = V_O, V_R, TTP, C, H(M'),\ sS_O(V_O, V_R, TTP, C, H(M'))$$

If R does not receive $me3$ in time after sending $me2$, R may execute the *resolve_R* sub-protocol by sending $mrr1 = (V_R, me1, me2, key_R)$ to the *TTP*. If O has not aborted the protocol, the *TTP* will decrypt the mail message in $me1$ and send $mrr3 = (M, key_O)$ to R. If the *TTP* discloses key_R to O, O will have a complete receipt $(me1, me2, key_R)$ which can make an arbitrator to believe that R received M' instead of M by only checking the receipt text $H(M')$. However, the *TTP*'s misbehavior cannot be verified since key_R could be sent to O by R itself after receiving M' from O.

This problem could be tackled if the arbitrator further checks whether the decrypted mail message M and the receipt text $H(M')$ in $me1$ match. If true, $me1$ is regarded as a valid part of receipt. However, this may require the *TTP*'s involvement because the mail message is encrypted with the *TTP*'s public key [2]. That means the *TTP* may need to be *on-line* for dispute resolution. If the *TTP* is temporary unavailable, arbitration has to be postponed. If the *TTP*'s private key has lost when a dispute arises, the arbitrator cannot make a proper conclusion.

[2] If the ElGamal public-key cryptosystem [10] is used, the encrypted message cannot be verified even with the plain message and the public key unless the random seed for ElGamal encryption is also provided. But the random seed is usually not saved after encryption.

The above problems may not be fatal to ASW protocol, but could affect its efficiency and security.

4 A Variant Protocol

Here we present a more efficient and secure protocol for certified mail, mainly based on the ideas from [2,14]. We split the definition of a mail message M into two parts, a commitment C and a key K. In the normal case, the originator O sends (C, K) (plus evidence of origin) to the recipient R in exchange for evidence of receipt without any involvement of the *TTP*. If there is something wrong in the middle of exchange, either O or R can unilaterally bring a protocol run to completion with the help from the *TTP*. The *TTP* only needs to notarise and/or deliver the message key K by request, which is usually much shorter than the whole mail message M.

The notation below is used in the description of our protocol.

- M: mail message being sent from O to R.
- K: message key defined by O.
- $C = eK(M)$: commitment (cipher text) for message M.
- $L = H(M, K)$: a unique label linking C and K.
- f_i $(1 \leq i \leq 8)$: flags indicating the intended purpose of a signed message.
- $EOO\_C = sS_O(f_1, R, L, C)$: evidence of origin of C.
- $EOR\_C = sS_R(f_2, O, L, EOO\_C)$: evidence of receipt of C.
- $EOO\_K = sS_O(f_3, R, L, K)$: evidence of origin of K.
- $EOR\_K = sS_R(f_4, O, L, EOO\_K)$: evidence of receipt of K.
- $sub\_K = sS_O(f_5, R, L, K, TTP, EOO\_C)$: evidence of submission of K to the *TTP*.
- $con\_K = sS_{TTP}(f_6, O, R, L, K)$: evidence of confirmation of K issued by the *TTP*.
- $abort = sS_{TTP}(f_8, O, R, L)$: evidence of abortion.
- P_{TTP}: the *TTP*'s public encryption key.

Like ASW protocol, our protocol has three sub-protocols: *exchange*, *abort*, and *resolve*. We also assume that the communication channels between the *TTP* and each transacting party (O and R) are *resilient*. In addition, we assume that the communication channel between O and R is *confidential* if the two parties want to exchange mail messages secretly. The *exchange* sub-protocol is as follows.

1. $O \rightarrow R : f_1, f_5, R, L, C, TTP, eP_{TTP}(K), EOO\_C, sub\_K$
   IF R gives up **THEN** quit **ELSE**
2. $R \rightarrow O : f_2, O, L, EOR\_C$
   IF O gives up **THEN** *abort* **ELSE**
3. $O \rightarrow R : f_3, R, L, K, EOO\_K$
   IF R gives up **THEN** *resolve* **ELSE**
4. $R \rightarrow O : f_4, O, L, EOR\_K$
   IF O gives up **THEN** *resolve*

The *abort* sub-protocol is as follows.

1. $O \rightarrow TTP : f_7, R, L, sS_O(f_7, R, L)$
   IF resolved **THEN**
2. $TTP \rightarrow O : f_2, f_6, O, R, L, K, con\_K, EOR\_C$
   ELSE
3. $TTP \rightarrow O : f_8, O, R, L, abort$

The *resolve* sub-protocol is as follows, where the initiator U is either O or R.

1. $U \rightarrow TTP : f_2, f_5, O, R, L, TTP, eP_{TTP}(K), sub\_K, EOO\_C, EOR\_C$
   IF aborted **THEN**
2. $TTP \rightarrow U : f_8, O, R, L, abort$
   ELSE
3. $TTP \rightarrow U : f_2, f_6, O, R, L, K, con\_K, EOR\_C$

In our protocol, evidence (EOR_C, EOR_K) or (EOR_C, con_K) can be used to prove that R received the message M, evidence (EOO_C, EOO_K) or (EOO_C, con_K) can be used to prove that O sent the message M.

If the *exchange* sub-protocol is executed successfully, R will receive C and K and thus $M = dK(C)$ together with non-repudiation of origin tokens (EOO_C, EOO_K). Meanwhile, O will receive non-repudiation of receipt tokens (EOR_C, EOR_K).

R can simply quit the transaction without losing fairness before sending EOR_C to O. Otherwise, R has to run the *resolve* sub-protocol to force a successful termination. Similarly, O can run the *abort* sub-protocol to quit the transaction without losing fairness before sending EOO_K to R. Otherwise, O has to run the *resolve* sub-protocol to force a successful termination.

The *resolve* sub-protocol can be initiated either by O or by R. When the TTP receives such a request, the TTP will first check the status of a transaction identified by (O, R, L) uniquely. If the transaction has been aborted by O, the TTP will return the *abort* token. If the transaction has already been resolved, the TTP will deliver the tuple $(f_2, f_6, O, R, L, K, con\_K, EOR\_C)$ to the current initiator of the *resolve* sub-protocol. Otherwise, the TTP will

- decrypt $eP_{TTP}(K)$ and verify with sub_K that K is submitted by O;
- check that EOR_C is consistent with sub_K in terms of L and EOO_C;
- generate evidence con_K;
- deliver the tuple $(f_2, f_6, O, R, L, K, con\_K, EOR\_C)$ to the current initiator;
- set the status of the transaction *resolved*.

The third component in the tuple indicates the key supplier, which is authenticated by sub_K and notarised in con_K. Hence, intruders cannot mount a denial-of-service attack by sending bogus keys to the TTP for confirmation. Evidence con_K can be used to prove that

- a transaction identified by (O, R, L) has been resolved successfully,
- the message key K originated from O, and
- the message key K is available from the TTP by request.

In comparison with ASW protocol, our protocol has the following merits.

- The TTP's overhead will not increase when transmitting large mail messages.
- The content of a mail message need not be disclosed to any outsiders including the TTP.
- The evidence is publicly verifiable without any restrictions on the types of signature and encryption algorithms.

Therefore, our protocol is more efficient and secure than ASW protocol both at the stage of exchange and at the stage of dispute resolution.

5 Security Analysis

We analyse our protocol with respect to the requirements listed in Section 3.

Claim 1. *If the communication channel between O and R is resilient, the protocol satisfies the effectiveness requirement.*

Proof: If both O and R are honest, they will send their messages according to the protocol description. If the communication channel between them is *resilient*, a message sent by either party will eventually be received by the other party. Thus the *exchange* sub-protocol can be executed successfully without invoking the TTP. R will receive C and K and thus $M = dK(C)$ together with non-repudiation of origin tokens (EOO_C, EOO_K). Meanwhile, O will receive non-repudiation of receipt tokens (EOR_C, EOR_K).

Claim 2. *If the communication channels between the TTP and each transacting party (O and R) are resilient, the protocol satisfies the fairness requirement.*

Proof: We first consider the possible unfair situations that O may face.

- O did not receive any message from R after sending message 1 in the *exchange* sub-protocol. In this case, O can initiate the *abort* sub-protocol, which is guaranteed to be completed within a finite period under the assumption. If R has not resolved the protocol, the TTP will not resolve the protocol at a later time, thus R cannot obtain K, which means R cannot receive M. If R has already resolved the protocol, O will obtain EOR_C and con_K from the TTP, which can be used to prove that R received M.
- O did not receive EOR_K after sending message 3 in the *exchange* sub-protocol. In this case, O can initiate the *resolve* sub-protocol to obtain con_K from the TTP, which can be used in place of EOR_K to prove that R received (or is able to receive) K provided the assumption holds.

The only possible unfair situation that R may face is that R did not receive K and EOO_K after sending message 2 in the *exchange* sub-protocol. In this case, R can initiate the *resolve* sub-protocol to obtain K and con_K under the same assumption.

Thus, the protocol satisfies the fairness requirement from both points of view of O and R.

Claim 3. *If the communication channels between the TTP and each transacting party (O and R) are resilient, the protocol satisfies the timeliness requirement.*

Proof: We first look at the possible ways that O can conclude a protocol run.

- terminating normally after sending message 4 in the *exchange* sub-protocol;
- invoking the *abort* sub-protocol at any time before sending message 3 in the *exchange* sub-protocol;
- invoking the *resolve* sub-protocol at any time after receiving message 2 in the *exchange* sub-protocol.

As we assume that the communication channel between the TTP and O is resilient, the *abort* and *resolve* sub-protocols initiated by O are guaranteed to be completed within a finite period. Thus at any time, there is always a way for O to conclude the protocol run.

On the other hand, the possible ways that R can conclude a protocol run are

- terminating normally after receiving message 4 in the *exchange* sub-protocol;
- simply quitting at any time before sending message 2 in the *exchange* sub-protocol;
- invoking the *resolve* sub-protocol at any time after receiving message 1 in the *exchange* sub-protocol.

Again, each of these will result in the timely conclusion of the protocol run for R.

Claim 4. *If the communication channels between the TTP and each transacting party (O and R) are resilient, the protocol meets the non-repudiation requirement.*

Proof: By the protocol description, R will hold the following non-repudiation of origin tokens

- (EOO_C, EOO_K) if the protocol terminates normally;
- (EOO_C, con_K) otherwise.

Meanwhile, O will hold the following non-repudiation of receipt tokens

- (EOR_C, EOR_K) if the protocol terminates normally;

– (EOR_C, con_K) otherwise.

EOO_C proves that O sent C with label L to R while EOO_K proves that O sent K with label L to R. Thus (EOO_C, EOO_K) proves that $M = dK(C)$ is from O. The link between C and K is computationally unique which should satisfy $L = H(dK(C), K)$.

In the same way, EOR_C proves that R received C with label L from O while EOR_K proves that R received K with label L from O. Thus (EOR_C, EOR_K) proves that R received $M = dK(C)$.

Alternatively, con_K proves that the TTP notarised K with label L at O's request, and that R received (or is able to receive) K with label L from the TTP under the assumption of a resilient channel with the TTP. Thus con_K can be used with EOO_C and EOR_C to prove the origin and receipt of M.

Claim 5. *If the communication channels between the TTP and each transacting party (O and R) are resilient, and that the TTP can be forced to eventually send a valid response to any request sent to it, the TTP is verifiable.*

Proof: Under the assumptions, the TTP's possible misbehavior could be

– R receives the *abort* token while O receives con_K;
– R receives K and con_K while O receives the *abort* token.

In the first case, if O uses EOR_C and con_K to prove that R received M, R can use the *abort* token to prove the TTP's misbehavior.

In the second case, if R uses EOO_C and con_K to prove that O sent M to R, O can use the *abort* token to prove the TTP's misbehavior. It should be noted that if R uses EOO_C and EOO_K to prove that O sent M to R, O cannot use the *abort* token to prove the TTP's misbehavior since the TTP did not issue conflicting evidence.

6 Conclusion

We investigated ASW protocol and found out the following weaknesses besides some minor flaws being fixed.

– The performance may degrade when transmitting large mail messages.
– The privacy of mail messages may not be well protected.
– The TTP may need to be *on-line* for dispute resolution.

We proposed a variant protocol which overcomes the above weaknesses. The security analysis shows that our protocol meets the requirements for fair exchange.

References

1. N. Asokan, M. Schunter and M. Waidner. *Optimistic protocols for fair exchange.* Proceedings of 4th ACM Conference on Computer and Communications Security, pages 7–17, Zurich, Switzerland, April 1997. 47
2. N. Asokan, V. Shoup and M. Waidner. *Asynchronous protocols for optimistic fair exchange.* Proceedings of 1998 IEEE Symposium on Security and Privacy, pages 86–99, Oakland, California, May 1998. 46, 47, 49, 52
3. F. Bao, R. H. Deng and W. Mao. *Efficient and practical fair exchange protocols with off-line TTP.* Proceedings of 1998 IEEE Symposium on Security and Privacy, pages 77–85, Oakland, California, May 1998. 47
4. M. Ben-Or, O. Goldreich, S. Micali and R. Rivest. *A fair protocol for signing contracts.* IEEE Transactions on Information Theory, IT-36(1):40–46, January 1990. 47
5. E. F. Brickell, D. Chaum, I. B. Damgard and J. van de Graaf. *Gradual and verifiable release of a secret.* Lecture Notes in Computer Science 293, Advances in Cryptology: Proceedings of Crypto'87, pages 156–166, Santa Barbara, California, August 1987. 47
6. R. Cleve. *Controlled gradual disclosure schemes for random bits and their applications.* Lecture Notes in Computer Science 435, Advances in Cryptology: Proceedings of Crypto'89, pages 573–588, Santa Barbara, California, August 1989. 47
7. B. Cox, J. D. Tygar and M. Sirbu. *NetBill security and transaction protocol.* Proceedings of the First USENIX Workshop on Electronic Commerce, pages 77–88, July 1995. 47
8. I. B. Damgard. *Practical and provably secure release of a secret and exchange of signatures.* Lecture Notes in Computer Science 765, Advances in Cryptology: Proceedings of Eurocrypt'93, pages 200–217, Lofthus, Norway, May 1993. 47
9. R. H. Deng, L. Gong, A. A. Lazar and W. Wang. *Practical protocols for certified electronic mail.* Journal of Network and Systems Management, 4(3):279–297, 1996. 47
10. T. ElGamal. *A public-key cryptosystem and a signature scheme based on discrete logarithms.* IEEE Transactions on Information Theory, IT-31(4):469–472, July 1985. 51
11. M. Franklin and M. Reiter. *Fair exchange with a semi-trusted third party.* Proceedings of 4th ACM Conference on Computer and Communications Security, pages 1–6, Zurich, Switzerland, April 1997. 47
12. T. Okamoto and K. Ohta. *How to simultaneously exchange secrets by general assumptions.* Proceedings of 2nd ACM Conference on Computer and Communications Security, pages 184–192, Fairfax, Virginia, November 1994. 47
13. P. Syverson. *Weakly secret bit commitment: Applications to lotteries and fair exchange.* Proceedings of 11th IEEE Computer Security Foundations Workshop, Rockport, Massachusetts, June 1998. 47
14. J. Zhou and D. Gollmann. *A fair non-repudiation protocol.* Proceedings of 1996 IEEE Symposium on Security and Privacy, pages 55–61, Oakland, California, May 1996. 47, 52
15. J. Zhou and D. Gollmann. *An efficient non-repudiation protocol.* Proceedings of 10th IEEE Computer Security Foundations Workshop, pages 126–132, Rockport, Massachusetts, June 1997. 47