Optimistic Fair Exchange with Transparent Signature Recovery

Olivier Markowitch and Shahrokh Saeednia

Université Libre de Bruxelles,
Département d'Informatique,
CP 212, Boulevard du Triomphe
1050 Bruxelles
Belgium

{omarkow,saeednia}@ulb.ac.be

Abstract. We propose a new protocol allowing the exchange of an item against a signature while assuring fairness. The proposed protocol, based on the Girault-Poupard-Stern signature scheme (a variation of the Schnorr scheme), assumes the existence of a trusted third party that, except in the setup phase, is involved in the protocol only when one of the parties does not follow the designated protocol or some technical problem occurs during the execution of the protocol. The interesting feature of the protocol is the low communication and computational charges required by the parties. Moreover, in case of problems during the main protocol, the trusted third party can derive the same digital signature as the one transmitted in a faultless case, rather than an affidavit or an official certificate.

Keywords: fair exchange, electronic commerce, digital signature.

1 Introduction

With the phenomenal growth of open networks in general and the Internet in particular, many security related problems have been identified and a lot of solutions have been proposed. Applications in which the fair exchange of items between users is required are becoming more frequent. Payment systems, electronic commerce, certified mail and contract signing are classical examples in which the fairness property is of crucial importance in the overall security of the related protocol. As defined originally, fairness must ensure that during the exchange of the items, no party involved in the protocol can gain a significant advantage over the other party, even if the protocol is halted for any reason. This paper addresses the problem of the fair exchange of an electronic item against a digital signature (which could be considered as an acknowledgement of receipt of the item).

The previous major works about fair exchange assume the existence of a trusted third party (TTP) in the protocol[1]. Independently of how the TTP is involved in the protocol, its role is mainly to resolve the problems that may occur between parties. Some proposals [22, 13, 11] use the TTP to store the details of the transaction in order to complete the exchange if one of the parties does not follow the predetermined protocol. As the TTP is actively involved in the protocol, this approach considerably reduces the efficiency of the exchange. To remedy this shortcoming, independently Micali and Asokan et al. [1, 4, 18] proposed a solution that avoids the presence of the TTP between the parties. They proposed not to use the TTP during the transaction when the parties behave correctly and the network works, but to invoke the TTP to complete the protocol in case of problems with one of the parties or the network. Such protocols are said to be *optimistic*.

The idea in that approach is as follows: one of the parties (that we call the *client*) sends a signature to the other party (that we call the *provider*) in exchange of a requested item. The provider should be convinced that the client's request and all other information he received from the client before sending him the item are sufficient to convince the TTP that the client actually asked that item. If so, in case of problem, the TTP can either make the client's signature available or give its own signature as an affidavit that has the same legal value than the client's signature. Methods based on this approach have firstly been proposed in [23, 22, 2, 3]. In these protocols, the TTP can complete the protocol by producing its own signature rather than the client's signature.

Recently Liqun Chen [10] proposed a protocol using discrete logarithm based signatures, in which the client commits his signature in a verifiable way for the provider. If the client does not send his final signature after having received the item, the TTP transmits information which has the same properties as a client's final signature when combined with the earlier committed signature. This recovered signature is not the same signature expected in a faultless case but is also a client's signature.

The use of an invisible TTP was first proposed by Micali [14] in the framework of certified mails. Asokan et al. [5] and Bao et al. [6], proposed fair exchange protocols allowing to recover, in case of problem, the original client's signature committed earlier in the protocol rather than affidavits produced and signed by the TTP. This kind of signatures is said to be *transparent*. Bao et al. proposed two protocols, from which the first one is inefficient, while the second one, though more efficient, has recently been broken by Boyd and Foo [8]. In the same paper, Boyd and Foo [8] proposed a fair exchange protocol for electronic payment. Their method allows to recover the original client's signature from the committed one, using designated convertible signatures [9]. They also proposed a concrete protocol based on the RSA signature scheme.

[1] Though some fair exchange protocols without a TTP have already been proposed [7, 19–21] (implying often some communication and computation overheads).

In this paper, we propose a new protocol that allow the exchange of an item against a signature while assuring fairness. The protocol, based on the Girault-Poupard-Stern (GPS) signature scheme [12, 16] (a variation of the Schnorr signature scheme [17]), uses an offline TTP, acting only in case of problem, which produces the same digital signature that the client and the provider would produce in a normal case.

We assume that the communication channel between the provider and the client is unreliable (the transmitted data may be lost or modified), and the communication channels between the provider and the TTP, and also between the client and the TTP are resilient[2] (the transmitted data is delivered after a finite, but unknown amount of time; the data may be delayed, but will eventually arrive).

The last point we wish to make before describing our protocol is the following. In [8], Boyd and Foo denoted that the client's committed signature must be in such a way that only the provider can verify its correctness. For this purpose, they propose to use an interactive protocol between the client and the provider during which the latter is convinced that, in case of problem, the TTP can convert the committed signature into a normal one that anyone could verify. In our protocol we do not follow this point of view. In fact, we believe that the use of an interactive verification just increases uselessly the amount of communication and provides nothing useful in exchange. All we want is that the security from the client's and the provider's point of view be respected. If the final signature (that is accepted as a valid signature by anyone) is different from the committed signature and if the latter cannot be forged, nor converted into a valid final signature by someone else than the TTP and the client, this partial signature gives sufficient credence about the "non-transferability" of information exchanged during the protocol. We believe that this is sufficient for the purpose of fair exchange and this is what we implement in our protocol.

2 Generic fair exchange protocol

As it is also the case in recently proposed fair exchange protocols with offline TTP, in our protocol the provider and the client can verify the validity of a committed signature without being able to extract the final signature from it. More precisely, the provider and the client, after having received a committed signature, can make sure that it contains enough information for the TTP to open it and produce the final signature, if problems occur during the transaction.

Hereafter, we give an outline of our protocol, inspired by the Asokan et al. fair exchange protocol [4], that may be used to provide fair exchange with various signature schemes. We will see an instantiation in the following section.

[2] This kind of channel is also said asynchronous [15].

Main protocol.

1. The provider sends to the client the item ciphered with a session key together with the session key ciphered with the TTP's public key, those ciphered information signed by the provider and a committed signature on the item's description.

2. If the provider's committed signature may be opened by the TTP to provide the final signature and if the provider's signature on the ciphered information is valid, the client transmits to the provider his committed signature on the description of the requested item.

3. Upon receiving the committed signature from the client, the provider verifies its correctness and checks if it may be opened by the TTP to provide the client's final signature. If so, the provider answers by sending the item and his final signature to the client[3].

4. After having checked the validity of the received item, the client sends to the provider his final signature on the requested item.

Provider's recovery protocol. If, during the main protocol, the provider does not receive the client's final signature or if the one received is not valid, he initiates a recovery protocol with the TTP.

1. The provider sends to the TTP the item, his final signature on the item's description and the client's committed signature of the item's description.

2. If the protocol has already been recovered or aborted, the TTP stops the recovery protocol. Otherwise, it verifies if the item corresponds to the item's description, checks the validity of the provider's signature, the client's committed signature and whether the signature is actually addressed to the provider and is on the item's description. If all the checks are correct, the TTP extracts the client's final signature from the committed one and forwards it to the provider and transmits to the client the item and the final provider's signature. Otherwise, the TTP sends an abort token to the provider and the client.

Client's recovery protocol. If, during the main protocol, the client does not receive the session key from the provider or if the one received is not valid, he initiates a recovery protocol with the TTP.

1. The client sends to the TTP the ciphered item, the ciphered session key, the provider's signature on those ciphered information, his committed signature

[3] The provider may just send the session key, rather than the item, in order to decrease the amount of communications.

on the item's description[4] and the provider's committed signature on the item's description.

2. If the protocol has already been recovered or aborted, the TTP stops the recovery protocol. Otherwise, it verifies if the item (obtained by deciphering) corresponds to the item's description, checks the validity of the client's final signature and the provider's committed signature and whether the signature is actually addressed to the client. If the received signatures are invalid the TTP stops the protocol. If the other checks are incorrect, it sends an abort token to the provider and to the client. Otherwise, if all the checks are correct, the TTP extracts the provider's final signature from the committed one, transmits it to the client along with the item. The TTP extracts the client's final signature from the committed one and forwards it to the provider.

Abort protocol. If, during the main protocol, the provider does not receive the committed signature from the client or if the one received is not valid, he initiates an abort protocol with the TTP.

1. The provider sends to the TTP an abort request.

2. If the protocol has not already been recovered or aborted, the TTP sends an abort confirmation to the client and to the provider.

As mentioned in the introduction, in [8] the sending of a committed signature is followed by an interactive proof of correctness of this committed signature. In our protocol, described in the next section, when receiving a committed signature, its correctness may be checked non-interactively and efficiently, without being able to take any advantage from it.

Remarks:

– At the beginning of the main protocol, the provider sends the item ciphered with a session key. The client cannot check if the ciphered item corresponds to the item that he asked, however if the provider does not give the expected item at the third step of the main protocol, when, thanks to the ciphered item, the client runs a recovery protocol, the TTP will detect that the provider has cheated, and so sends an abort token to both parties, ending this protocol run.
– The provider must send his signature on the ciphered information in step 1 of the main protocol, in order to prevent the client to take advantage from the

[4] If the client sends directly his final signature when realizing a recovery, just after the provider has aborted the protocol (see the abort protocol hereafter), then the provider can obtain the client's final signature by observing the communication channel between the client and the TTP, while the client does not obtain the requested item, as the protocol has been aborted.

protocol. In fact, the client can stop the main protocol after having received the requested item and the provider's final signature (in the step 3) and launch a recovery protocol with the TTP. Without the provider's signature on the ciphered information, the client can hand an incorrect ciphered session key, which leads to an incorrect item deciphered by the TTP, forcing the latter to send an abort token to the provider. This prevents the provider to obtain the client's final signature afterward by launching a recovery protocol.
- The provider's committed signature cannot relate to the ciphered item and to the ciphered session key because it must be converted into a final signature that should only be related to the item's description.

Fairness. After a successful execution of the main protocol, the final signature is exchanged against the item. If the client stops the main protocol after receiving the first message of the main protocol, the provider can realize the abort protocol. If the provider does not send the session key or if the client stops the main protocol after receiving the item, they can initiate independently their recovery protocol and the TTP will either send the final client's signature to the provider and the requested item to the client or an abort token if the information are inconsistent; the protocol is remaining fair, due to the resilient channels between the TTP and respectively the provider and the client. If the client does not receive the first message of the main protocol, he stops the main protocol and the protocol remains fair as no target information (neither the item nor the final client's signature) has been transmitted.

3 A fair exchange protocol based on the GPS signature scheme

Before describing our protocol, let us introduce some definitions and notations:

- *item* is an item to be transmitted to the client.
- *descr* is a string containing the client's request, the description of the requested item and some other information allowing the provider, the TTP and any other external party to recognize the item.
- C, P, TTP identify respectively the following entities: the client, the provider and the trusted third party.
- $A \rightarrow B : X$ denotes that the entity A sends a message X to the entity B.
- $h(X)$ is the output of a one-way hash function h applied to the message X.
- $\mathrm{S}_P(X)$ is a "classical" provider's digital signature (and not recoverable) of the message X.
- $\mathrm{E}_k(X)$ is a symmetric encryption of the message X with the session key k.
- $\mathrm{E}_{TTP}(X)$ is an asymmetric encryption of the message X with the TTP's public key.
- $f_{com,msg,ack,rec,abort,aborted}$ are flags indicating the purpose of a message sent (respectively "committed signature", "expected message", "final signature", "ask for recovery", "ask for abort" and "confirmation of abort").

– l is a label identifying, with the identities P and C, the protocol run.

The protocol is based on the GPS signature [12, 16]. The signature of a message m is realized on one hand by choosing a random value r and computing $t = \alpha^r \bmod n$ where n is a composite modulus and α is a basis of maximum order, $\lambda(n)$, and on the other hand by computing $z = r + xh(t, m)$ where x is a secret value associated to $y \equiv \alpha^{-x} \bmod n$ the corresponding public value. The verification is achieved by comparing t and $\alpha^z y^{h(t,m)} \bmod n$.

Initialization. Our protocol assumes the existence of a TTP that knows some secret information. In this phase of the protocol, the TTP chooses an integer $n = pq$, where p and q are large random primes of almost the same size such that $p = 2p' + 1$ and $q = 2q' + 1$ for some primes p' and q'. The TTP chooses also a base α of order $s = p'q'$ and a very small integer c such that $\gcd(s, c) = 1$. We recommend $c = 3$ for some reasons that we discuss further in this section.

The TTP now computes d such that $cd \equiv 1 \pmod{s}$ and $\beta = \alpha^c \bmod n$. Finally, the TTP makes n, β, c, h and α public, keeps d secret and discards p and q.

Key generation. In order to prepare a pair of public and secret keys, each user u chooses respectively a random integer x_u as secret key and computes the relative public key $y_u = \alpha^{x_u} \bmod n$.

Note that, here, for the purpose of simplicity we do not consider the authenticity of the public keys. However, it is easy to see that public keys may be converted to self-certified keys as explained in [12] and be used in a straightforward way in the following protocol.

Main protocol. When a client wishes to receive an item from a provider against a valid signature, they follow this protocol:

1. The provider chooses a random r_P and computes:

$$t_P = \beta^{r_P} \bmod n \qquad \text{and} \qquad z_P = c \cdot r_P + h(t_P, m_P) \cdot x_P$$

   where $m_P = (f_{msg}, P, C, l, \mathit{descr})$. He also selects a random session key k and forms $\mathrm{E}_k\,(\mathit{item})$ and $\mathrm{E}_{TTP}\,(k)$. The pair (t_P, z_P) (being the provider's committed signature) is sent to the client together with $\mathrm{E}_k\,(\mathit{item})$, $\mathrm{E}_{TTP}\,(k)$ and the provider's signature on those ciphered information.

   $P \rightarrow C : f_{com_1}, P, C, l, \mathit{descr}, \mathrm{E}_k\,(\mathit{item})\,, \mathrm{E}_{TTP}\,(k)\,, \mathit{sigP}, t_P, z_P$

   where $\mathit{sigP} = \mathrm{S}_P\,(f_{com_1}, P, C, l, \mathrm{E}_k\,(\mathit{item})\,, \mathrm{E}_{TTP}\,(k))$.

2. The client forms m_P and checks whether sigP is valid and whether

$$\alpha^{z_P} \equiv t_P \cdot {y_P}^{h(t_P, m_P)} \pmod{n}$$

If so, the client chooses a random r_C and computes

$$t_C = \beta^{r_C} \mod n \qquad \text{and} \qquad z_C = c \cdot r_C + h(t_C, m_C) \cdot x_C$$

where $m_C = (f_{ack}, C, P, l, \mathit{descr})$. The pair (t_C, z_C), being the client's committed signature, is sent to the provider.

$C \rightarrow P : f_{com_2}, C, P, l, t_C, z_C$

3. The provider forms m_C and checks whether

$$\alpha^{z_C} \equiv t_C \cdot {y_C}^{h(t_C, m_C)} \pmod n$$

If so, the provider computes

$$t'_P = \alpha^{r_P} \mod n$$

and sends the item and t'_P to the client. The pair (t'_P, z_P) being the final signature.

$P \rightarrow C : f_{msg}, P, C, l, \mathit{item}, t'_P$

4. The client verifies that

$$\alpha^{z_P} \equiv {t'_P}^{c} \cdot {y_P}^{h({t'_P}^{c} \mod n, m_P)} \pmod n$$

If so, after having checked the validity of the received item, the client computes

$$t'_C = \alpha^{r_C} \mod n$$

and sends t'_C to the provider. The pair (t'_C, z_C) being the final signature.

$C \rightarrow P : f_{ack}, C, P, l, t'_C$

5. The provider verifies that

$$\alpha^{z_C} \equiv {t'_C}^{c} \cdot {y_C}^{h({t'_C}^{c} \mod n, m_C)} \pmod n$$

If so, the provider accepts the signature, since it will also be accepted by any external party.

Remarks:

– The occurrence of f_{ack} and f_{msg} (rather than f_{com_2} and f_{com_1}) in m_P and m_C are just because m_P and m_C are used in z_P and z_C and they constitute a part of the final signature.

– The use of the identities (P and C) in m_P and m_C is of particular importance, because this guarantees that a committed signature is addressed to a given recipient. In fact, without P in m' (for example), any provider (providing the requested item) can capture the committed signature and use it afterward to obtain a client's final signature by launching a recovery protocol with the TTP. So, it is essential that the TTP verifies the provider's identity in the recovery protocol and checks the correctness of a committed signature with respect to it (see below).
– In steps 4 and 5, it is actually sufficient to check whether ${t'_P}^c \equiv t_P \bmod n$ and ${t'_C}^c \equiv t_C \bmod n$. The full equations are described above in order to highlight how to check the the final signature.

Provider's recovery protocol. If the client does not send his final signature or if the last signature is not valid, the provider runs the following protocol with the TTP, in order to recover the client's final signature.

1. The provider sends the item, *descr*, the pair (t_C, z_C) and his final signature to the TTP.

   $P \rightarrow TTP : f_{rec_P}, P, C, l, descr, item, t_C, z_C, t_P, t'_P, z_P$

2. If the protocol was not already recovered or aborted, the TTP makes sure that the item corresponds actually to *descr* and if so it forms m_C and m_P and verifies the validity of (t_C, z_C) and (t_P, t'_P, z_P). If all the checks are successful, the TTP sends

   $$t'_C = {t_C}^d \mod n$$

   to the provider and the item to the client. Otherwise, it sends an abort token to both parties.

   $TTP \rightarrow P : f_{ack}, C, P, l, t'_C$

   $TTP \rightarrow C : f_{msg}, P, C, l, item, t'_P$

Client's recovery protocol. If the provider does not realize the third sending of the main protocol or if this message is not valid, the client runs the following protocol with the TTP.

1. The client sends the received ciphered information, *descr*, the provider's signature on them, the pair (t_p, z_p) (the provider's committed signature) and his final signature to the TTP.

   $C \rightarrow TTP : f_{rec_C}, C, P, l, descr, \mathrm{E}_k\,(item)\,, \mathrm{E}_{TTP}\,(k)\,, sigP, t_P, z_P, t_C, z_C$

where $sigP = \mathrm{S}_P\left(f_{com_1}, P, C, l, \mathrm{E}_k\left(item\right), \mathrm{E}_{TTP}\left(k\right)\right)$.

2. If the protocol was not already recovered or aborted, the TTP first makes sure that the received item (obtained after deciphering) corresponds actually to *descr* and that the provider's signature *sigP* is valid, if so it forms m_C and m_P and verifies the validity of (t_P, z_P) and (t_C, z_C). If the signatures are invalid the TTP stops the recovery protocol. If the other checks are not successful the TTP sends an abort token to the provider and to the client, as in the abort protocol. Otherwise, if the checks are successful, the TTP sends $t'_P = {t_P}^d \bmod n$ and the item to the client and $t'_C = {t_C}^d \bmod n$ to the provider.

   $TTP \rightarrow C : f_{msg}, P, C, l, item, t'_P$

   $TTP \rightarrow P : f_{ack}, C, P, l, t'_C$

Abort protocol. If the client does not send the second message of the main protocol, the provider runs the following protocol with the TTP, in order to abort the protocol.

1. The provider sends an abort request to the TTP.

   $P \rightarrow TTP : f_{abort}, C, P, l, \mathrm{S}_P\left(f_{abort}, C, P, l\right)$

2. If the protocol was not already recovered or aborted, the TTP sends an abort confirmation to the provider and to the client.

   $TTP \rightarrow P : f_{aborted}, P, C, l, \mathrm{S}_{TTP}\left(f_{aborted}, P, C, l\right)$

   $TTP \rightarrow C : f_{aborted}, P, C, l, \mathrm{S}_{TTP}\left(f_{aborted}, P, C, l\right)$

Security. The security of the protocol may be discussed around two questions:

1. Is it possible to create false signatures linked to a given client?
2. Is it possible to convert a committed signature to a final one without knowing r or d?

First, let us see why we recommend to choose $c = 3$. When a committed signature is given, we have $z = cr + h(t, m)x$, where c and h are known. Since there is no modular reduction (unlike the Schnorr scheme), we can immediately compute $x \bmod c$. Hence, with $c = 3$ we minimize the amount of information that anybody can learn about the secret key.

To answer the first question, let us notice that the committed signature is essentially the same as the Schnorr scheme with composite modulus. Since r is random, cr in z may be seen as a random r' (even though the knowledge of c gives a "bit" of information about x), while t may be considered as $\alpha^{r'} \bmod n$. So, the pair (t, z) actually constitutes a GPS signature on m'. The security of this scheme is already discussed in [16] by Poupard and Stern.

On the other hand, creating a final signature just from the public key and known signatures (but without having the corresponding committed signature) is at least as hard as forging a committed signature. In fact, if it is possible to create such a signature, i.e., producing a pair (t', z) for a message m', then it is also possible to create $(t = t'^c \bmod n, z)$, as a committed signature, or more generally a GPS signature on m'.

To answer the second question, it is straightforward to see that, in order to compute t', one should either know r (to do as the real client does) or d (to do like the TTP). Otherwise, it would be possible for a cheater to use a committed signature (t, z) to create a correct final signature $(\hat{t'}, \hat{z})$ such that $\hat{z} \neq z$ (that implies that $\hat{t'}^c \neq t$) or $\hat{z} = z$ but $\hat{t'}^c \neq t$. We believe that in either cases, establishing a final signature using t and z is equivalent to forging a GPS signature. In fact, this precisely means that it is possible to create a GPS signature $(\hat{t} = \hat{t'}^c, \hat{z})$ on m' from a signature (t, z) on the same message. However, creating a new signature from an existing signature on the same message is equivalent to forging a signature for a message m from known signatures of messages $m_1, m_2, ...$ This can clearly be shown by the same techniques used in theorem 10 in [16].

4 Conclusion

We have considered a new protocol, based on the GPS signature scheme (a variation of the Schnorr signature schemes), allowing the exchange of an item against a signature while assuring fairness. Our protocol assumes the existence of a trusted third party whose role is to guarantee fairness and that, expect in the setup phase, is involved in the protocol only when one of the parties does not follow the protocol correctly. We proposed to use a committed signatures that gives sufficient assurance about the TTP's ability of recovering the final signature from the committed signature, in case of problem. The interesting feature of the protocol is the low communication and computational charges required by the parties during the transactions.

It seems to us that our protocol is not an isolated instance based on the framework introduced in section 2. We are currently working on protocols based on Guillou-Quisquater and Fiat-Shamir signature schemes.

The possibility of using DSA and ElGamal schemes for designing fair exchange protocols based on our model is not clear to us and remains as an open problem.

5 Acknowledgment

We are grateful to Michael Waidner for his criticism of the earlier version of our protocol and for many helpful discussions. We also wish to thank Guillaume Poupard and David Pointcheval for their invaluable remarks. We would like to express our gratitude to anonymous referees for their interesting suggestions.

References

1. N. Asokan. *Fairness in Electronic Commerce.* PhD thesis, University of Waterloo, May 1998.
2. N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. Research Report RZ 2858 (#90806), IBM Research, Sept. 1996.
3. N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In T. Matsumoto, editor, *Proceedings of the fourh ACM Conference on Computer and Communications Security*, pages 6, 8–17, Zurich, Switzerland, Apr. 1997. ACM Press.
4. N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, May 1998.
5. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *Advances in Cryptology: Proceedings of Eurocrypt'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 591–606. Springer-Verlag, 1998.
6. F. Bao, R. H. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line TTP. In *IEEE Symposium on Security and Privacy*, May 1998.
7. D. Boneh and M. Naor. Timed commitments. In *Advances in Cryptology: Proceedings of Crypto 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer-Verlag, 2000.
8. C. Boyd and E. Foo. Off-line fair payment protocols using convertible signatures. *Lecture Notes in Computer Science*, 1514:271–285, 1998.
9. D. Chaum. Designated confirmer signatures. In A. D. Santis, editor, *Advances in Cryptology: Proceedings of Eurocrypt'94*, volume 950 of *Lecture Notes in Computer Science*, pages 86–91. Springer-Verlag, 1995, 9–12 May 1994.
10. L. Chen. Efficient fair exchange with verifiable confirmation of signatures. *Lecture Notes in Computer Science*, 1514:286–299, 1998.
11. T. Coffey and P. Saidha. Non-repudiation with mandatory proof of receipt. *ACM-CCR: Computer Communication Review*, 26, 1996.
12. M. Girault. Self-certified public keys. In *Advances in Cryptology: Proceedings of EuroCrypt'91*, volume 547 of *Lecture Notes in Computer Science*, pages 490–497. Springer-Verlag, 1991.
13. Y. Han. Investigation of non-repudiation protocols. In *ACISP: Information Security and Privacy: Australasian Conference*, volume 1172 of *Lecture Notes in Computer Science*, pages 38–47. Springer-Verlag, 1996.
14. S. Micali. Certified E-mail with invisible post offices. Available from author; an invited presentation at the RSA '97 conference, 1997.
15. B. Pfitzmann, M. Schunter, and M. Waidner. Optimal efficiency of optimistic contract signing. In *PODC: 17th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 113–122, 1998.

16. G. Poupard and J. Stern. Security analysis of a practical "on the fly" authentication and signature generation. In *Advances in Cryptology: Proceedings of Eurocrypt'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 422–436. Springer-Verlag, 1998.
17. C. P. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology: Proceedings of Crypto 89*, pages 239–252, Berlin, Aug. 1990. Springer.
18. M. Schunter. *Optimistic Fair Exchange*. PhD thesis, Technische Fakultät der Universität des Saarlandes, Saarbrücken, Oct. 2000.
19. P. Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *Proceedings of the 1998 IEEE Computer Security Foundations Workshop (CSFW11)*, pages 2–13, june 1998.
20. T. Tedrick. How to exchange half a bit. In D. Chaum, editor, *Advances in Cryptology: Proceedings of Crypto'83*, pages 147–151, New York, 1984. Plenum Press.
21. T. Tedrick. Fair exchange of secrets. In G. R. Blakley and D. C. Chaum, editors, *Advances in Cryptology: Proceedings of Crypto'84*, volume 196 of *Lecture Notes in Computer Science*, pages 434–438. Springer-Verlag, 1985.
22. J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *IEEE Symposium on Security and Privacy*, Research in Security and Privacy, pages 55–61, Oakland, CA, May 1996. IEEE Computer Society,Technical Committee on Security and Privacy, IEEE Computer Security Press.
23. J. Zhou and D. Gollmann. An efficient non-repudiation protocol. In *Proceedings of The 10th Computer Security Foundations Workshop*, pages 126–132. IEEE Computer Society Press, June 1997.