J. A. Onieva, J. Zhou, and J. Lopez, "Attacking an asynchronous multi-party contract signing protocol", *Proceedings of 6th International Conference on Cryptology in India*, LNCS vol. 3797, pp. 311321, 2005.
NICS Lab. Publications: https://www.nics.uma.es/publications

Analysis of an Asynchronous Multi-Party Contract Signing Protocol

Jose A. Onieva[1], Jianying Zhou[2], and Javier Lopez[1]

[1] Computer Science Department, University of Malaga
29071 - Malaga, Spain
{onieva,jlm}@lcc.uma.es

[2] Institute for Infocomm Research
21 Heng Mui Keng Terrace
Singapore 119613
jyzhou@i2r.a-star.edu.sg

Abstract. Contract signing is a fundamental service in doing business. The Internet has facilitated the electronic commerce, and it is necessary to find appropriate mechanisms for contract signing in the digital world. From a designing point of view, digital contract signing is a particular form of electronic fair exchange. Protocols for generic exchange of digital signatures exist. There are also specific protocols for two-party contract signing. Nevertheless, in some applications, a contract may need to be signed by multiple parties. Less research has been done on multi-party contract signing. In this paper, we analyze an optimistic N-party contract signing protocol, and point out its security problem, thus demonstrating further work needs to be done on the design and analysis of secure and optimistic multi-party contract signing protocols.

Keywords: secure electronic commerce, multi-party contract signing, security protocol analysis.

1 Introduction

The Internet has facilitated the electronic commerce. Many business transactions have been shifted to the Internet. The motivation for such a trend is the efficiency and cost-saving. However, as new risks may arise in the digital world, sufficient security measures should be taken. This will help users to establish the confidence for doing business on the Internet.

Contract signing is a fundamental service for business transactions, and has been well practiced in the traditional paper-based business model. Now, it is necessary to find appropriate mechanisms for contract signing in the digital world. Consider several parties on a computer network who wish to exchange some digital items but do not trust each other to behave honestly. *Fair exchange* is a problem of exchanging data in a way that guarantees either all participants obtain what they want, or none does. From a designing point of view, contract

signing is a particular form of fair exchange, in which the parties exchange commitments to a contract (typically, a text string spelling out the terms of a deal). That is, a contract is a non-repudiable agreement on a given text such that after a contract signing protocol instance, either *each* signer can prove the agreement to any verifier *or none* of them can. If several signers are involved, then it is a *multi-party contract signing* (MPCS) protocol.

There are some two-party contract signing protocols in the literature. Nevertheless, less research has been done on multi-party contract signing. In this paper, we analyze an optimistic multi-party contract signing protocol, and point out its security problem, thus demonstrating further work needs to be done on the design and analysis of secure and optimistic multi-party contract signing protocols.

The rest of this paper is organized as follows. In Section 2, we review the previous work related to contract signing, outline the properties to be satisfied when designing an optimistic contract signing protocol and give explicit definitions for some terms used along the descriptions of these protocols. In Section 3, we analyze an optimistic N-party contract signing protocol presented in [11] and demonstrate that the protocol cannot achieve fairness. We conclude the paper in Section 4.

2 Related Work

As contract signing is a particular case of fair exchange, any fair exchange protocol found in the literature in which digital signatures are exchanged can be considered as the related work. In all practical schemes, contract signing involves an additional player, called *Trusted Third Party* (TTP). This party is (at least to some extent) trusted to behave correctly, thus playing the role of a notary in paper-based contract signing and somehow sharing the legal duties the former ones have. In fact, designing and implementing a contract signing protocol using an on-line TTP should not be a complicated task. In this case, if Alice and Bob wish to enter into a contract, they each sign a copy of the contract and send it to the TTP through a secure channel. The TTP will forward the signed contracts only when it has received valid signatures from both Alice and Bob.

Nevertheless, in our continuous search for speeding up our daily life activities, it is desirable not using a TTP in a contract signing protocol. Additionally, if the TTP is not involved, the notary fee could be avoided. Some protocols appear in the literature trying to eliminate the TTP's involvement using *gradual exchange* of signatures [9, 10]. But these solutions are not deterministic, thus may not be accepted by signatories. Our objective is to focus on contract signing protocols that necessarily use a TTP only in those cases in which an exception occurs (i.e., a network communication failure or a dishonest party's misbehavior). Otherwise (all-honest-case), the TTP will not be contacted, and parties will bring the protocol to its end by themselves. In the literature, these protocols are called *optimistic contract signing* protocols [2–4, 14–17].

Some properties extracted from the different previous work on optimistic contract signing are summarized as follows.

- *Effectiveness* - if each party behaves correctly, the TTP will not be involved in the protocol.
- *Fairness* - no party will be in an advantageous situation at the end of the protocol.
- *Timeliness* - any party can decide when to finish a protocol run without loosing fairness.
- *Non-repudiation* - no party can deny its action.
- *Verifiability of TTP* - if the TTP misbehaves, all harmed parties will be able to prove it.
- *Transparency of TTP* - if the TTP is contacted to resolve the protocol, the resulting contract will be similar to the one obtained in case the TTP is not involved.
- *Abuse-Freeness* - it is not possible for an attacker (either a legitimate participant or an outsider) to show a third party that the contract final state is under its control.

In [8], Ben-Or *et al.* presented an optimistic contract signing protocol based on a *probabilistic approach*. Such a contract signing protocol is said to be (ν, ϵ)-fair if for any contract C, when signer A follows the protocol properly, if the probability that signer B is privileged to validate the contract with the TTP's help is greater than ν, the conditional probability that "A is not privileged", given that "B is privileged", is at most ϵ.

Previous work in which several signatories are involved in a contract can be found in [1, 6, 11, 12]. Only Asokan *et al.* addressed the MPCS problem in synchronous networks [1]. As Asokan states, this solution clearly improves the efficiency of those asynchronous protocols previously presented with respect to the number of messages; $4(n-1)$ messages in the all-honest-case and $6n-4$ messages in the worst case. This is possible due to a better reliability of the underlaying network as we can see in Definition 1 below.

Some authors considered the abuse-freeness property in [13, 7]. Baum-Waidner proposed new protocols in [6] that improve the solutions presented for asynchronous networks in [7] such that the number of rounds is significantly reduced in the case that the number of dishonest participants t is considerably less than the total number of participants n - the smaller t is, the better results the new protocols achieve.

Definition 1 *A "synchronous" contract signing protocol is used in synchronous networks in which there is a limited time for a message to reach its destination (otherwise it has been lost and the appropriate transport layer manages these events) even if an attack occurs. Thus a party can determine that a message has not been sent by other party if it did not arrive within the limited time. Users' clocks are assumed to be synchronized.*

Definition 2 *An "asynchronous" contract signing protocol is used in asynchronous networks in which there is no limited time for a message to reach its destination. Loss and unsorted arrival of messages are possible and have to be managed by the contract signing protocol itself. Clocks are not assumed to be synchronized among users.*

A number of protocols exist in the literature which use an asynchronous model of network (i.e., messages can be reordered and lost) with deadline parameters. But when a deadline is introduced, and thus, synchronized clocks among users are assumed (at least at the moment the deadline is approaching), these protocols are converted into synchronous protocols.

In the literature, MPCS protocols make use of either a ring or a matrix topology. Throughout these solutions, authors use the terms *round* and *step* without clearly defining them, which often brings on confusion with respect to the metric to be used for its efficiency evaluation. For this reason we explicitly define these terms as follows:

- *Round* is understood as the existing time slot in which messages are distributed in synchronous networks. In asynchronous networks, the entities need to wait a local time before going to the next round (in case the round is not completed).
- *Step* refers to the action of sending or receiving a message. It is the operation performed by a participating entity. Each round means one step (when all the messages from all entities are distributed or broadcasted in the same time slot, usually in matrix topologies) or several steps (when messages from the same round are distributed from one entity to another, usually in ring topologies).

It has to be noted that there is some confusion in the literature with respect to the term 'round'. Some authors explain that when the next message to be sent depends on the previous one, that is a different round. But we claim two different cases can be found (1) message to be sent depends on the previous one because the entity needs to compute/verify it before sending the next one or (2) message to be sent depends on the previous one because there is a distribution order to be respected (as in ring topologies). We consider a round occurs in Case (1).

All of previous solutions to the asynchronous multi-party contract signing problem reach the lower bound on the number of rounds described in Theorem 3 given in [13]:

> *Any complete and optimistic* asynchronous *contract-signing protocol with n participants requires at least n rounds in an optimistic run.*

Describing the theorem, Garay *et al.* stated that for each party P_i, when it sends a message that can be used (together with other information) by other entities to obtain a valid contract, as the protocol is fair, it must have received in a previous round, a message from the rest of participants in order to be able to

get a valid contract too (probably with the TTP's help), no matter how others behave. By an inductive argument, they showed the number of rounds is at least n. This stands for $t = n - 1$ and the lower bound decreases with t number of dishonest parties.

Ferrer's asynchronous protocol presented in [11] with only three rounds is an exception. It claimed some years ago to use a number of rounds independent from the number of participants. As this is supposed a huge improvement with respect to efficiency, we analyze the protocol and demonstrate in the next section, it is flawed.

3 Analysis of a MPCS Protocol

To the best of our knowledge, only the asynchronous protocol in [11] can finish multi-party contract signing in 3 rounds. Here we demonstrate that this protocol does not fulfill the property of fairness.

It is not clear whether the informal argument given in the theorem above apply to this protocol, since at the end of the first round (that is, after the first n+1 steps needed for all entities to distribute all messages in a ring topology), every party has received at least the initial commitment from every other party [3]. So we give another argument for the theorem above to make clearer that this protocol is not compliant with the theorem, and demonstrate with an attack, the validity of our argument.

We base our argument on the number of rounds a TTP needs to determine whether a party is misbehaving when requesting resolution (i.e., it requests the TTP to cancel but continues the main protocol): The TTP cannot determine whether a party is misbehaving until $round = round_{current} + 1$, since the TTP needs to wait until the next round to see whether this entity cheated and continued the protocol. That means if $n - 1$ dishonest parties exist in the worst case, and each of them requests the *cancel* sub-protocol in a different round, n rounds are the minimum required to satisfy fairness in an asynchronous optimistic contract signing protocol.

3.1 Protocol Description

The original notation used in the protocol description is as follows:

- M : message containing the contract to be signed. The contract specifies the order of players in a ring for exchanging signature of principal i on the contract M.
- $h_i = S_i(H(M))$: signature of principal i on contract M, where $H(.)$ is a collusion-resistant one-way hash function.
- $ACK_i = S_i(H(h\text{-}ACK))$: signature of principal i on $h\text{-}ACK$, all the signatures and acknowledgments given by other parties in the ring.

[3] Note that only one step would have been needed if a matrix topology is used and all distribute their commitments at the same time.

- $ACK2_B = S_B(H(ACK_C))$: last acknowledgment sent to the last player in the protocol.
- $cancel, cancel_A, cancel_C, finish$: variables used by the TTP to maintain the state of a protocol run.

Suppose A, B, and C are 3 parties going to sign a contract. A is the first principal in the ring architecture, C is the last one, and all the rest ($P_2 \cdots P_{n-1}$ for n parties) behave as B. This has been previously agreed upon a setup phase by all entities. The *exchange* sub-protocol is as follows:

1. $A \rightarrow B : M, h_A$
2. $B \rightarrow C : M, h_A, h_B$
3. $C \rightarrow A : h_B, h_C$
4. $A \rightarrow B : h_C, ACK_A$
5. $B \rightarrow C : ACK_A, ACK_B$
6. $C \rightarrow A : ACK_B, ACK_C$
7. $A \rightarrow B : ACK_C$
8. $B \rightarrow C : ACK2_B$

If the protocol run is completed, everybody will hold non-repudiation (NR) evidence. In order to demonstrate to an external party the existence of a contract signed by all parties, following NR evidence has to be provided:

- A holds h_B, h_C, ACK_B, ACK_C
- B holds h_A, h_C, ACK_A, ACK_C
- C holds $h_A, h_B, ACK_A, ACK_B, ACK2_B$

If there is an exception in the main exchange protocol, then the parties get involved in either *cancel* or *finish* sub-protocols with the TTP. First of all, the TTP's intervention is to verify the correctness of the information given by parties. If this information is incorrect, the TTP will send an error message to that party. Some state variables (*cancel*, $finish$, $cancel_A$ and $cancel_C$) are used, all of which with a value false at the beginning for a particular exchange. ACK_{TTP} is the TTP's signature on $H(M)$; this is equivalent to an acknowledgement from a party that should have sent.

If A says that she has not received the first message sent by C, A may initiate the following *cancel* sub-protocol:

1. $A \rightarrow TTP : h(M), h_A$
2. $TTP \rightarrow A :$ IF $finish = true$ THEN ACK_{TTP}
   ELSE $S_{TTP}(H(cancelled, h_A))$;
   TTP stores $cancel = true$

If the variable $finish$ is true, it means B or/and C had previously finished the protocol with the TTP (see paragraphs below). The TTP had given the NR token ACK_{TTP} to B or/and C, and now it has to give the same NR token to A. If B and C had not contacted the TTP previously, the TTP will send a message to A to cancel the transaction, and it will store this information ($cancel$ = true) in order to satisfy future requests from B or C.

If A says that she has not received the last message from C, A may initiate the following *finish* sub-protocol:

1. $A \rightarrow TTP : h(M), h_A, h_B, h_C$
2. $TTP \rightarrow A :$ IF $cancel = true$ THEN $S_{TTP}(H(cancelled, h_A))$;
   TTP stores $cancel_A = true$
   ELSE ACK_{TTP};
   TTP stores $finish = true$

If the variable *cancel* is true, it means B had previously contacted the TTP (see paragraphs below). The TTP had given a message to B to cancel the transaction, and now it has to send a similar message to A. Additionally, the TTP will store the variable $cancel_A$ with value true to satisfy potential petitions from C. If B had not contacted the TTP previously, the TTP will send the NR token ACK_{TTP} to A. In this case the TTP will assign the value true to the variable $finish$, in order to satisfy future petitions from B or/and C.

If B says that he has not received the second message from A, B may initiate the following *cancel* sub-protocol:

1. $B \rightarrow TTP : h(M), h_A, h_B$
2. $TTP \rightarrow B :$ IF $finish = true$ THEN ACK_{TTP}
   ELSE $S_{TTP}(H(cancelled, h_B))$;
   TTP stores $cancel = true$

If the variable $finish$ is true, it means A or/and C had previously finished the protocol with the TTP. The TTP had given the NR token ACK_{TTP} to A or/and C, and now it has to give the same NR token to B. If A and C had not contacted the TTP previously, the TTP will send a message to B to cancel the transaction, and it will store this information ($cancel$ = true) in order to satisfy future petitions from A or C.

If B says that he has not received the last message from A, B may initiate the following *finish* sub-protocol:

1. $B \rightarrow TTP : h(M), h_A, h_B, h_C, ACK_A$
2. $TTP \rightarrow B :$ IF $cancel_C = true$ THEN $S_{TTP}(H(cancelled, h_B))$
   ELSE ACK_{TTP};
   TTP stores $finish = true$

If the variable $cancel_C$ is true, it means A and C (in this order) had previously contacted the TTP (see paragraphs above). The TTP had given a message to A and C to cancel the transaction, and now it has to send a similar message to B. If A had not cancelled the exchange, the TTP will send the NR token ACK_{TTP} to B. In this case the TTP will assign the value true to the variable $finish$, in order to satisfy future petitions from A or/and C.

If C says that he has not received the second message from B, C may initiate the following *finish* sub-protocol:

1. $C \rightarrow TTP : h(M), h_A, h_B, h_C$
2. $TTP \rightarrow C$: IF $cancel = true$ THEN $S_{TTP}(H(cancelled, h_C))$;
   TTP stores $cancel_C = true$
   ELSE ACK_{TTP};
   TTP stores $finish = true$

If the variable *cancel* is true, it means A or/and B had previously contacted the TTP (see paragraphs above). The TTP had given a message to A or/and B to cancel the transaction, and now it has to send a similar message to C. Additionally, the TTP will store the variable $cancel_C$ with value true to satisfy potential future petitions from B. If A and B had not cancelled the exchange with the TTP previously, the TTP will send the NR token ACK_{TTP} to C. In this case the TTP will assign the value true to the variable $finish$, in order to satisfy future petitions from A or/and B.

If C has not received the last message from B, C may initiate the following second *finish* sub-protocol:

1. $C \rightarrow TTP : h(M), h_A, h_B, h_C, ACK_A, ACK_B$
2. $TTP \rightarrow C$: IF $cancel_A = true$ THEN $S_{TTP}(H(cancelled, h_C))$
   ELSE ACK_{TTP};
   TTP stores $finish = true$

If the variable $cancel_A$ is true, it means B and A (in this order) had previously contacted the TTP. The TTP had given a message to A and B to cancel the transaction, and now it has to send a similar message to C. If A and B had not cancelled the exchange with the TTP previously, the TTP will send the NR token ACK_{TTP} to C. In this case the TTP will assign the value true to the variable $finish$, in order to satisfy future petitions from A or/and B.

3.2 Security Analysis

Once we have described the protocol, the first step of our analysis is to check whether this protocol fulfills all the defined properties.

- It is an asynchronous protocol, i.e., messages can be lost or reach their destination in an unsorted way.
- It is effective, since no TTP participation is needed if all parties behave correctly.
- It is *not* fair, since some parties can be in an advantageous position at the end of the protocol. Furthermore, this flaw cancels the rest of properties. It does *not* fulfill timeliness, non-repudiation (parties could deny their actions), verifiability of TTP, transparency of TTP (the TTP signs affidavits), and abuse-freeness (it allows an entity to decide the final outcome of the protocol).

Using the TTP's resolution protocol just described, let us have a look at the following scenario. Suppose there are 4 participants (A, B_1, B_2, C), where

B_1 and B_2 behave as B in the resolution protocol as they are the intermediate participants in the ring architecture.

1. $A \rightarrow B_1$: M, h_A
2. $B_1 \rightarrow B_2$: M, h_A, h_{B_1}
3. $B_2 \rightarrow C$: M, h_A, h_{B_1}, h_{B_2}
4. $C \rightarrow A$: h_{B_1}, h_{B_2}, h_C
5. $A \rightarrow B_1$: h_{B_2}, h_C, ACK_A
   : B_1 resolves:: $cancel = true$; and continues the protocol
6. $B_1 \rightarrow B_2$: h_C, ACK_A, ACK_{B_1}
7. $B_2 \rightarrow C$: $ACK_A, ACK_{B_1}, ACK_{B_2}$
8. $C \rightarrow A$: $ACK_{B_1}, ACK_{B_2}, ACK_C$
   : A resolves:: $cancel_A = true$; and continues the protocol
9. $A \rightarrow B_1$: ACK_{B_2}, ACK_C
10. $B_1 \rightarrow B_2$: ACK_C
    : B_2 resolves:: $finish = true$; gets the contract
    : and continues the protocol
11. $B_2 \rightarrow C$: $ACK2_{B_2}$
    : C resolves:: *cancelled*

In this arbitrary protocol execution, all entities contact the TTP for resolving the protocol even though they continue its execution, getting different results. B_1 invokes its cancel sub-protocol in step 5, A invokes its finish sub-protocol in step 8, B_2 invokes its finish sub-protocol in step 10, and after finishing the protocol C invokes its second finish sub-protocol.

At the end of the protocol, and with the TTP's help, B_2 and C obtain different results. This is due to lack of state variables to maintain the actual status of the protocol. $cancel_A$ and $cancel_C$ are not enough for controlling the status if $n-1$ ($n > 3$) parties are dishonest when they cancel the protocol. In other words, there is no sub-protocol when B_i says it has not received the last message from B_j $(i > j)$ $\in [P_2 \cdots P_{n-1}]$ and the wrong sub-protocol is used.

Furthermore, an inconsistency can be found in its explanation for 3 parties. In the 3-party version, B and A cancel the protocol (in this order) but continue with its execution. If the protocol stops before the last step, A and B have the signed contract, but C can obtain only a cancel token instead. If priority to the cancel token is assigned by the arbitrator, then the protocol is fair, since C presents its cancel token to the arbitrator in case of dispute and the arbitrator settles that the contract has been cancelled.

But if the last step occurs, then now, all get the signed contract but A and B have cancel tokens. In this case priority to the finished state should be assigned, as the honest party acting properly (C) got the contract. Nevertheless, the arbitrator does not know who is the honest party, and cannot assign priority to the tokens. This is a serious contradiction. And this happens with any sequence (e.g., also if A and C cancel in this order while B is honest). As shown above, parties cheat about their protocol state when contacting the TTP. Obviously, the TTP can detect it, but can do nothing to repair the situation.

Moreover, in the original work there is no dispute resolution process defined for the multi-party version, which makes it more difficult to explicitly resolve possible conflicts. We claim that in any design of a contract signing protocol, a well-defined dispute resolution process has to be provided.

4 Conclusions

Contract signing is a particular form of fair exchange, in which the parties exchange commitments to a contract. Previous work mainly focused on two-party contract signing. In some applications, however, a contract may need to be signed by multiple parties. In this paper, we analyzed an optimistic N-party contract signing protocol, and pointed out its security problem. This clearly demonstrates that if we want a bank deposit to be made with several beneficiaries, further research is needed on the multi-party fair exchange protocols and more concretely on multi-party contract-signing protocols. Part of this research contemplates formal verification and analysis of existing solutions.

This finding encourages us to study and improve existing multi-party contract signing solutions. Definitely further work is needed in synchronous networks, and an improvement of efficiency in asynchronous networks needs to be achieved to bring these applications into reality with the user's confidence.

5 Acknowledgements

The work described here is partially funded by the FP6-2002-IST-1 project UBISEC, contract number 506926 and the first author has been funded by the Consejeria de Innovacion, Ciencia y Empresa (Junta de Andalucia) under the III Andalusian Research Plan.

References

1. N. Asokan, Birgit Baum-Waidner, Matthias Schunter, and Michael Waidner. Optimistic synchronous multi-party contract signing. Technical Report RZ 3089, IBM Zurich Research Lab, 1998.
2. N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for multi-party fair exchange. Technical Report RZ 2892 (# 90840), IBM, Zurich Research Laboratory, 1996.
3. N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *Proceedings of 4th ACM conference on Computer and communications security*, pages 7–17. ACM Press, 1997.
4. N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000.
5. Feng Bao, Robert Deng, and Wenbo Mao. Efficient and practical fair exchange protocols with off-line ttp. In *Proceedings of 1998 IEEE Symposium on Security and Privacy*, pages 77–85. IEEE, May 1998.

6. Birgit Baum-Waidner. Optimistic asynchronous multi-party contract signing with reduced number of rounds. In *Proceedings of 28th International Colloquium on Automata, Languages and Programming*, pages 898–911. Springer, 2001.
7. Birgit Baum-Waidner and Michael Waidner. Round-optimal and abuse-free multi-party contract signing. In *Proceedings of 27th International Colloquium on Automata, Languages and Programming*, LNCS 1853, pages 524–535. Springer, 2000.
8. M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, volume 36, pages 40–46, 1990.
9. M. Blum. Three applications of the oblivious transfer, 1981.
10. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, volume 28, pages 637–647, 1985.
11. Josep Lluís Ferrer-Gomila, Magdalena Payeras-Capellà, and Llorenç Huguet-Rotger. Efficient optimistic n-party contract signing protocol. In *Proceedings of 4th International Conference on Information Security*, pages 394–407. Springer, 2001.
12. Josep Lluís Ferrer-Gomila, Magdalena Payeras-Capellà, and Llorenç Huguet-Rotger. Optimality in asynchronous contract signing protocols. In *Proceedings of 1st International Conference on Trust and Privacy in Digital Business*, LNCS 3184. Springer, August 2004.
13. Juan A. Garay and Philip D. MacKenzie. Abuse-free multi-party contract signing. In *Proceedings of 13th International Symposium on Distributed Computing*, pages 151–165. Springer, 1999.
14. N. González-Deleito and O. Markowitch. An optimistic multi-party fair exchange protocol with reduced trust requirements. In *Proceedings of 4th International Conference on Information Security and Cryptology*, LNCS 2288, pages 258–267. Springer, December 2001.
15. O. Markowitch and S. Saeednia. Optimistic fair-exchange with transparent signature recovery. In *Proceedings of Financial Cryptography 2001*. Springer, February 2001.
16. Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *Proceedings of 22nd Annual Symposium on Principles of Distributed Computing*, pages 12–19. ACM Press, 2003.
17. Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Optimal efficiency of optimistic contract signing. In *Proceedings of 17th Annual ACM Symposium on Principles of Distributed Computing*, pages 113–122. ACM Press, 1998.
18. ITU-T. Information technology - Open Systems Interconnection - Security frameworks for open systems: Non-repudiation framework. October 1996.