Contract Signing Using PGP

Ole Kasper Olsen mail@olekasper.no
Ole Martin Dahl mail@olemartin.com
Torkjel Søndrol mail@torkjel.com
Fredrik Skarderud fredrik@skarderud.net

Gjøvik University College, NISlab

December 17, 2004

Abstract

Is it possible to achieve non-repudiation during electronic contract-signing? How can this be achieved in an e-mail based environment? These are the questions we will try to answer.

The process of getting these answers will involve a short introduction to some of the available contract-signing standards and an analysis of how well the given standard would work in a e-mail environment.

Additionaly, we will present a prototype we have implemented that provides secure contract signing using mail and PGP[1].

1 Introduction

— Contract[2]

1. *An agreement between two or more parties, especially one that is written and enforceable by law.*
2. *The writing or document containing such an agreement.*
3. *The branch of law dealing with formal agreements between parties.*

Contracts are vital for all kind of businesses, including electronic commerce. Every agreement between two or more parties involves a contract of some kind. E.g. on-line purchases imply a contract that promise to exchange money for some good or service. Electronic contracts make electronic commerce possible.

1.1 Contract Properties

Contract signing has some properties that are important to cover. Firstly the contract signing must be *atomic*. That is the contract must either be mutually agreed to or not agreed to at all, no intermediate state exist. A more detailed explanation of the importance of atomicity for e-commerce is given by Tygar in [1]. Ensuring full atomicity in unreliable network conditions is difficult.

[1] Pretty Good Privacy http://www.philzimmermann.com
[2] Source: http://www.dictionary.com

The contract signing must also be *durable*, meaning that when the contract is signed by both parties, there is no way to undo it. The only way to undo a signed contract, is to sign a new contract that specifically invalidates the previous contract, or an expiry date is reached.

The point of time when the agreement is formed is also an important property, called *agreement formation time*.

1.2 Contract Protocol Properties

To ensure support for the contract properties in a contract signing protocol, certain protocol properties are required.

One necessary property is *fair contract signing* [2, 3, 4]. A protocol is said to be fair if neither party could gain any advantage by terminating the protocol in the middle. The common solution for fair exchange is to use a TTP. The TTP approve the contract only after all signing is done. Traditionally this solution require that all the communication go through the TTP, i.e. an in-line TTP. Making the communication less dependent on processing from a TTP is often a goal.

Another important property in a contract scheme is *accountability*. Accountability means that if the TTP misbehaves in any way, then this can be proven. E.g. after the protocol execution with the parties Alice, Bob and the TTP, Alice does not obtain a valid contract. If the TTP does not misbehave, then Bob will not have a valid contract either, but if Bob does have a valid contract, and tries to enforce it in court, then the contract that Bob presents in court, together with information obtained by Alice during the execution of the protocol, can be used to prove that the TTP misbehaved. The contract is then ruled invalid. At the same time it must be infeasible for Alice and Bob to frame the TTP if it does not misbehave.

Non-repudiation is an important security service that give more trust to e-commerce.

> *Non-repudiation is a security service that creates, collects, validates, and maintains cryptographic evidence, such as digital signatures, in electronic transactions to support the settlement of possible disputes. [5]*

The research on non-repudiation is fairly new and the literature on the subject is much more scarce than in other information security areas like confidentiality, integrity and availability. This in spite of non-repudiation being one of the main areas in information security. The need for non-repudiation occur in many situations [5, 6, 4, 7], e.g. in electronic payment system, dispute resolution, protocols for network security, message-handling systems and is an absolute requirement in a contract signing protocol. If the Internet had more non-repudiation services, it would be more difficult to commit fraud and other malicious activities.

Completeness is also a desired property in contract signing protocols. A protocol should be robust against adversaries attempting to cause to abort without the consent of either party.

1.3 Trusted Third-Party Roles

TTP works as a third participant in scenarios where two parties need additional trust between them, for instance when it comes to contract signing. This trusted third party can in real life be compared to the post office receiving registered mail, and getting a receipt from the receiver before delivering the registered mail. In this case both parties trust the post office.

[5] defines three different kinds of TTPs:

- *In-line TTP* where the TTP acts as an intermediary between the two participants.
- *On-line TTP* where the TTP is actively involved in every instance of the non-repudiation service.
- *Off-line TTP* where the TTP provides non-repudiation without being involved in each instance of the service. It is only involved when needed.

The standardisation of the use of TTPs are further described in ISO 14516 [8].

2 Protocols

There are two main categories of contract signing protocols. *Two-party protocols* and *third-party protocols*. The literature often divide third-party protocols into *fully trusted third-party protocols* and *not so trusted third-party protocols*.

2.1 Two-Party Protocols

It is claimed that it is impossible to make a fully fair two party protocol. At some point during the contract signing protocol one of the participating parties always has an advantage or more information about the contract to be signed than the other party. Whichever party holding this advantage can at any point simply decide to stop participating in the protocol. Imagine for example that a user receive a signed contract from the other party, and never returns his signature to the other party. He now sits on a valid contract, and when the other party starts asking about his contribution to the signing scenario he simply says that he has sent it, it is probably lost in transit. Another scenario could be that a participant receive the contract and not approve of it, he simply can claim that he has never received the contract at all.

Several solutions and protocols are proposed to overcome this problem. One approach is using the *gradual exchange protocols*. Ralph Merkle's Puzzle Protocol [9] is an example of such an protocol. The basic idea in this kind of protocols is to give the receiving part of the contract signing a puzzle of the complete contract, in such a way that it will not make any sense before the entire protocol is completed. The receiver needs a complete list of hints to solve the puzzle, and to receive the hints, the receiver must request them. The puzzle can for instance be a cryptographic solution, where the hints are small pieces of the complete encryption key. The obvious problem of such a protocol is computational power. Holding the partial key, if one party has significantly more computational power than the other party, he could compute and decrypt the

contract before the complete key is revealed. Another major drawback lies in the fact that these protocols are gradual. Thus a large number of messages are needed to complete the protocol, and a large overhead is created.

A lot of work have been done to solve the computational power problem. The protocols in [10] and [2] propose ways to encounter this problem, however these protocols do not seem to solve the problem of one party stopping the exchange midway through, possibly holding an unfair advantage over the other party. In other words fairness is not achieved.

2.2 Third-Party Protocols

2.2.1 Fully Trusted Third-Party Protocols

TTP protocols (see section 1.3) are based on the idea that a third party who both participants trust are involved in some way during the contract signing protocol. The two main problems with this approach are that the TTP needs to be involved in every transaction, and if the TTP is evil, can it make false contracts and misuse its trust?

The first problem is solved in *optimistic* protocols. The idea behind optimistic protocols is that the TTP is not contacted unless something goes wrong or a conflict occurred during the protocol execution. Section 2.3 describes one of the optimistic protocols in more detail.

The problem of an evil TTP can be solved if the third party does not have the power to create contracts, or if the third party is not able to read or misuse the contracts or information which is to be signed by the two parties. This attribute is often referred to as *abuse-freeness* or resistant towards *release of message content* attacks[11]. If the contract signing protocol is designed in such a way that neither the third party nor the participants can show the contract to others, it is said that the protocol has *total abuse-freeness*. This concept was first introduced in 1999 [12]. The first version of the protocol however proved not to be abuse-free [13]. Following this analysis a new version of the protocol was developed. This version is not yet proven to lack abuse-freeness.

2.2.2 Not So Trusted Third-Party Protocols

Protocols which require a third-party, but does not have to trust the third party very much, are often referred to as not so trusted third-party protocols. In practise this means that the third party are unlikely or unable to forge contracts, because of the third party's nature.

Riordan and Schneier have proposed a protocol that makes use of such a third party [14]. This protocol makes use of a public bulletin board as a third party. The general idea is as follows: A user A sends a signed e-mail encrypted with a random key using some kind of symmetric cryptographic algorithm to a user B. B then returns a signed e-mail, working as a receipt, back to A, saying that B would like A to publish the key for the encrypted message on the bulletin board by a time T. A then publishes the key before time T, and B can now download and decrypt the message.

Another interesting protocol in this category is [15]. In short this protocol makes use of a third-party whose only task is to generate random numbers, and tie these numbers to timestamps which increase by a given interval.

2.3 Optimistic Fair Exchange Protocol

Optimistic fair exchange [16] eliminates processing on the TTP in normal situations. The TTP is only needed when a party is unfaithful or when messages are lost.

The optimistic fair exchange protocol (OFEP) proposed in [16] is easily adapted for use in an Internet based contract signing environment [17].

The basics of OFEP [16] is as follows. Alice and Bob are exchanging a contract. m is the contract body. All parties are mutually authenticated, e.g. through the use of PKI[3]. First, Bob and Alice exchange *precontracts*.

- $Alice's\ precontract = Sig_{Alice}(m, Alice, Bob, TTP)$
- $Bob's\ precontract = Sig_{Bob}(m, Alice, Bob, TTP)$

Note that the precontracts themselves are not valid contracts. To prevent a dishonest player to use the same message several times, a nonce[4] must be inserted into the message. When Alice receive Bob's precontract in return for her own, she knows that Bob has committed to sign the contract so she can sign the contract without risk. If Bob refuses to sign the contract, Alice can go to the TTP with the two *precontracts* and have the TTP certify the contract. This implies two types of contracts; a standard and a notarised contract. If both parties behave as expected and no messages is lost during the protocol the contract is at the form: $contract = Sig_{Alice}(m, Alice, Bob), Sig_{Bob}(m, Alice, Bob)$. If something goes wrong during the exchange, the TTP can issue an alternative form of a contract by signing the two precontracts:
$contract = Sig_{TTP}(Sig_{Alice}(m, Alice, Bob, TTP), Sig_{Bob}(m, Alice, Bob, TTP)$. This is a notarised contract. These two types of contracts are equally valid.

3 Applied Theory

We aim to implement a prototype of a on-line service for signing arbitrary contracts between two parties. The goal of the implementation is fair exchange of non-refutable signatures of a contract, using no special technologies other than what is supported by current e-mail clients.

We have made two protocol drafts to solve this. The first draft is relying on a fully trusted on-line third-party where fairness is the main goal to achieve. We then improve the protocol with an aim to provide abuse-freeness.

3.1 Scenario

Our scenario is an e-mail environment, where the two parties register their public PGP keys on a contract-signing server in advance of a contract-signing process. The two contract-signing parties might then use their PGP key pairs when signing the contract they will negotiate.

[3] Public Key Infrastructure
[4] A number used once

3.1.1 Pretty Good Privacy

We base our protocol on currently available technology. PGP [18], created in 1991 by Philip Zimmermann, is a universally used method for secure communication via e-mail. PGP brings together many of the most used asymmetric and symmetric crypto algorithms and hash algorithms in such a way that the confidentiality and integrity of the message is ensured.

PGP is based on a *web of trust*, where each user has different degrees of trust towards the public PGP key of other users known to him. For further information about PGP, see [19].

3.2 Our Protocol

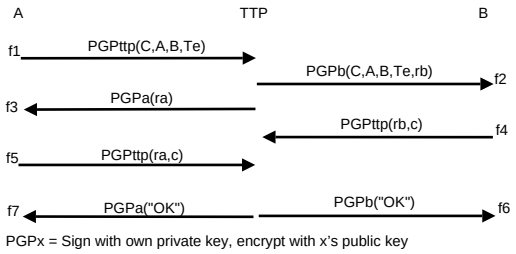

Figure 1: Our in-line TTP protocol

We have created a prototype of the contract signing server (the TTP), which handle the process of negotiating the contracts between the two signing parties. We assume that all the contract signing parties are already registered at the TTP with their public PGP keys. Figure 1 shows how the protocol works:

f_1 – Alice (the initiator) sends a contract proposal C to the TTP along with her own and Bob's (the responder) identities (A and B). She also specifies how long she and Bob can wait until signing the contract before it is invalidated (T_e).

f_2 – The TTP verifies the signature. He then generates two random numbers; r_A and r_B which will later be used to verify the ownership of Alice's and Bob's private PGP keys. The random numbers is also used to bind the contract to each party and to prevent replay. The TTP sends the contract, Alice and Bob's identities, the expiration time and r_B to Bob.

f_3 – The TTP then sends r_A to Alice. Then there is a time window as long as the time specified in T_e, where Alice and Bob must decide whether they want to sign the contract. If they don't sign it within the limit of T_e, the contract is declared invalid by the TTP. The time window implies that message f_4 and f_5 might come in any order as long as they come within the time window.

f_4 and f_5 – If Alice and Bob decides to sign the contract, they return the random number they received from the TTP along with a signature of a concatenation of the contract and the random number.

f_6 and f_7 – When the TTP has received both f_4 and f_5, he will send a confirmation to Alice and Bob that the contract was signed if r_A and r_B received in f_4 and f_5 matches those sent out in f_2 and f_3. The content of f_6 and f_7 are

the signed contract of both parties. These messages are also available via the Web interface.

All messages during the protocol are signed and encrypted with PGP.

We acknowledge that this is not a protocol with general applicability, but a proprietary application of non repudiation principles using PGP.

3.2.1 SDL Diagrams

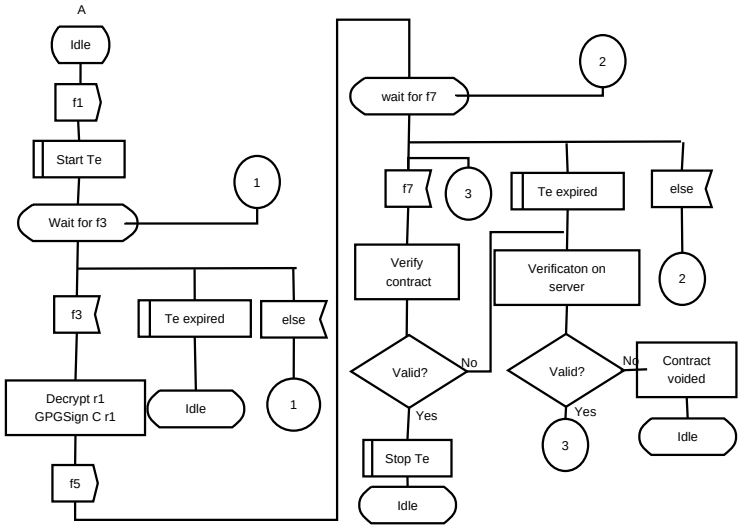

Figure 2: SDL diagram for initiator (Alice)

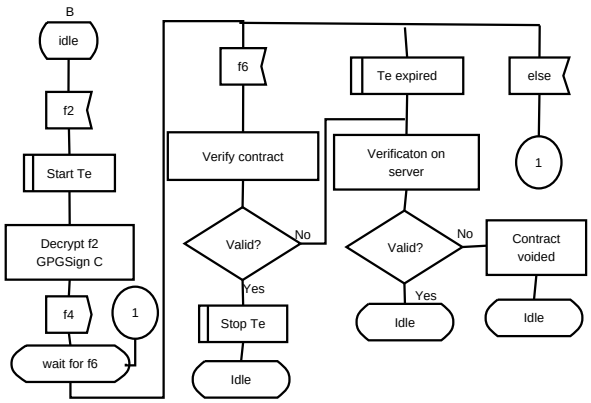

Figure 3: SDL diagram for responder (Bob)

Figure 2, 3 and 4 are lightweight SDL[5] descriptions of our first protocol. SDL diagrams is that it makes it easy to see the potential weaknesses one must

[5]Specification and Description Language

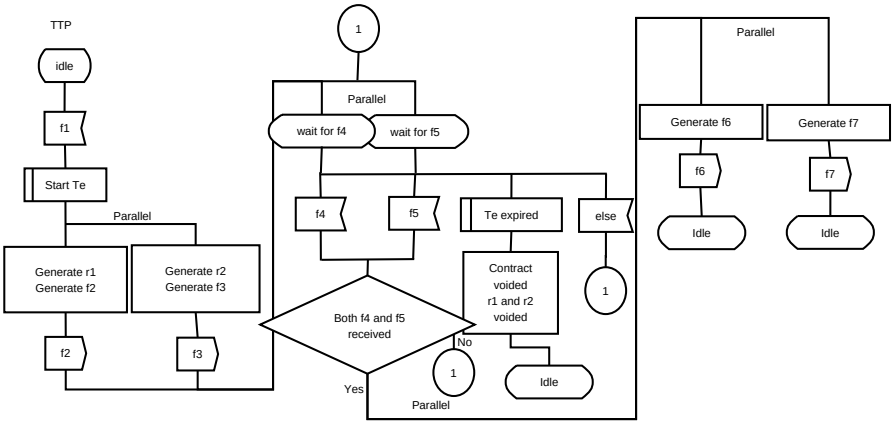

Figure 4: SDL digram of the TTP

consider when implementing the protocol. Deadlocks and endless loops are easy to discover using such diagrams.

As we can see from Alice's SDL diagram (figure 2), she will wait for the f_3 message to arrive from the TTP within the limit of T_e after she has sent the message f_1. Note that T_e might be very long in a computer-perspective—maybe several days. T_e is therefore not an ordinary software timer which requires her computer system to be idle while waiting, but rather the waiting period at the TTP for the signatures from Alice and Bob. If both of them do not sign the contract within the limit of T_e, the TTP will invalidate the contract.

If T_e expires before Alice receives the f_3 message, she will assume the contract has been voided, and go into idle state. She may confirm that the contract has been voided by logging onto the TTP's web site. If she receive any other unexpected messages, she will just continue waiting. When the f_3 message is received, and she has sent her f_5 message, she will start waiting for the confirmation message (f_7) from the TTP. If this does not arrive before T_e expires, she will contact the TTP and ask whether the contract is valid. If it is, everything is OK, and the contract has been signed. If the contract isn't valid, the contract is voided and the contract signing procedure is aborted. If she receives the f_7 message, she should verify Bob's signature, before stopping T_e, and everything is OK.

Bob's SDL (figure 3) is in many ways identical to Alice's (figure 2). It is simpler, since the contract signing procedure will start when he receives the f_2 message from the TTP, hence the only message he have to wait for is the confirmation message from the TTP. The procedure of controlling this message is identical to Alice's.

When the TTP receives the f_1 message from Alice (figure 4), he will start the T_e timer, generate Alice and Bob's random numbers and send both f_2 and f_3. Then he will start waiting for f_4 and f_5. When both of them are received, he will generate the confirmation messages (f_7 and f_8) and send them to Alice and Bob. If T_e expires before both f_4 and f_5 are received, the contract will be voided.

3.2.2 Implementation

The TTP was implemented using Perl, a flexible script language capable of performing the PGP and e-mail related operations needed by the prototype. The Perl script interfaces with the free Linux-based PGP equivalent GNU Privacy Guard (GnuPG), which is capable of properly signing and encrypting e-mails.

All participant data, as the contract, the signatures and their public PGP keys are stored in a MySQL database. Every customer of the contract signing server may log onto the PHP-implemented[6] Web interface and access the status of all of their signed contracts. The presentation of contract status does not jeopardise the fairness of the contract signing, i.e., the contract will only have status as "signed" if both parties have signed it.

3.2.3 Analysis

As shown in the description of our protocol (section 3.2), we propose a contract signing protocol with an in-line TTP making use of PGP technology. This protocol has several advantages. It uses PGP, meaning the contract signing parties might use any e-mail client they like as long as it support PGP.

The combination of the time-window and the use of the TTP, implies that none of the contract signers know whether or not the other part has signed the contract in advance of his own signing. He can therefore not take advantage of such a knowledge, i.e. fairness is achieved.

The contract signing process is also atomic, since the contract is either valid or not valid at all after the time-window T_e has expired.

The PGP signatures yield that the contract is durable when a participant has signed the contract. If one of the participants should deny the validity of the contract, the use of the TTP can prove the participant wrong since the TTP has both signatures and the contract.

With the valid signatures of each party accountability is achieved. The TTP is not able to generate keys for Alice and Bob to claim that the participants have signed a fake contract. However, this accountability relies on PGP's web of trust. The participants can easily verify each others PGP keys without involvement of the TTP.

There are also some weaknesses in this protocol. As shown in the SDL diagrams (figures 2, 3 and 4), there are some points in the communication in which Alice and Bob will wait for messages from the TTP. If T_e expires before the parties receive all their messages, they must contact the TTP to verify whether or not the contract has been mutually agreed to. If they do not care to do this, they risk believing that a signed contract not is signed nor valid.

Ensuring completeness has not been a high priority. However the use of PGP ensures some level of completeness to the protocol, especially since every message is both signed and encrypted using PGP. An adversary can therefor not cause a protocol abort by intercepting and modifying messages. However interception and withholding of a message may cause a protocol abort.

The most severe misuse the TTP can commit is withholding information regarding the status of the contract. This implies that the TTP does not send verification to the participants and that contract status is not published on the Web. This can lead to confusion and faults about the contract validity and

[6] `http://www.php.net`

signing status. See section 2.3 for a protocol that solves this. In our protocol we must trust that the TTP does this correctly.

The fact that the protocol requires an in-line TTP which communicates with both parties during the entire contract-signing phase is a weakness. This is done in such a way that the contract is readable in plain text for the TTP. Hence, the protocol is not abuse-free. Improving the protocol and making it abuse-free is fairly simple by introducing hashes and encrypting the protocol before transferring it to the TTP.

3.2.4 Improvements

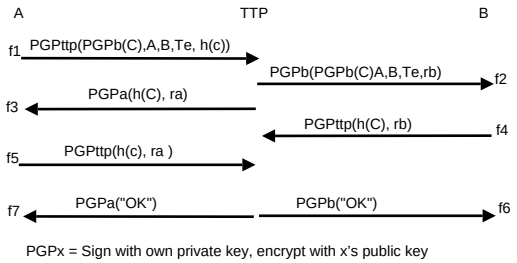

Figure 5: Our in-line, abuse-free TTP protocol

Based on our analysis of the contract signing protocol we proposed in the beginning of section 3.2, we have redesigned it to take the discovered weaknesses into account.

In an abuse-free protocol the TTP is not fully trusted by the participants. The participants will therefor not trust the TTP to reliably obtain true public key of the registered users. In the previous protocol the responsibility of obtaining the true keys form the users falls on the TTP. In this protocol the responsibility falls on the contract signing parties. When using this protocol, Alice will encrypt the contract using Bob's public key. She will also include a hash of the contract. Other than the contract being sent encrypted, the messages f_2 and f_3 are equal to those in our first protocol (figure 1). Bob will read the contract by decrypting it with is own private PGP key and then create a hash value of the contract. He will then sign a concatenation of the hash and the random number. Alice will sign her random number and the hash of the contract. When both Alice and Bob has signed, the TTP will send a notification to both of them, and they can check the signatures in the same way as they could in the previous version.

Here only the participants know the plain-text of the contract and it can not be read and abused by any other party, i.e. abuse-freeness.

4 Further Work and Conclusion

4.1 Further Work

What we have implemented is merely a prototype of the first protocol we proposed. User friendliness is vital to a technology's survival and while we believe

that our implementation of the first proposed protocol to be relatively easy to use, a prototype of the second protocol will be difficult to implement in such a way that the client only needs currently available e-mail clients. To improve the user friendliness of the solution, creating special client-side software might be necessary.

A further evolution of the protocol would be to implement ideas from OFEP protocols (see section 2.3) to limit the TTP's involvement in the protocol, which is often a goal in protocols like these.

We have not conducted a thorough security analysis of the protocols proposed. Even though weaknesses in the protocols might be found, we hope that they still can be used to further build a solid contract signing scheme using PGP.

Another aspect which is not covered here, is the implementation of the contract negotiation phase, where both parties negotiate the content of the contract.

4.2 Conclusion

We have discussed different aspects around contract signing, where non-repudiation is of paramount importance. We believe that a scheme which makes it possible for individual users to securely sign contracts among each other will increase the use of the Internet as a medium for contract signing. By using PGP, we utilise an already proven secure method of e-mail messaging to facilitate easy contract signing.

We have shown that it is indeed possible to achieve non-repudiation during electronic contract signing in an e-mail based environment, without using any specially crafted client side software specifically for this scheme.

References

[1] D. Tygar. Atomicity in electronic commerce. *Internet Besieged*, pages 389–406, October 1997.

[2] S. Micali M. Ben-Or, O. Goldrich and R. Riverst. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[3] David Molnar. Signing electronic contracts. *Crossroads*, 7(1):6–ff., 2000.

[4] Indrajit Ray and Indrakshi Ray. Fair exchange in e-commerce. *SIGecom Exch.*, 3(2):9–17, 2002.

[5] Jianying Zhou. *Non-repudiation in Electronic Commerce*. Artech House, Inc., 2001.

[6] Donal O'Mahony, Michael Peirce, and Hitesh Tewari. *Electronic Payment Systems for E-Commerce*. Artech House, Inc., 2001.

[7] Rong Du, Ernest Foo, Colin Boyd, and Brian Fitzgerald. Defining security services for electronic tendering. In *Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, pages 43–52. Australian Computer Society, Inc., 2004.

[8] ISO 14516. Information Technology - Guidelines on the Use and Management of Thursted Third Party (TTP) Services, May 1999.

[9] Ralph Merkle. Secure communications over insecure channels. *CACM*, 1978.

[10] S. Micali S. Even, O. Goldrich and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[11] William Stallings. *Network Security Essentials: Applications and Standards*. Prentice Hall, second edition, 2003.

[12] Juan A. Garay, Markus Jakobsson, and Philip MacKenzie. Abuse-free optimistic contract signing. *Lecture Notes in Computer Science*, 1666:449–466, 1999.

[13] Vitaly Shmatikov and John C. Mitchell. Analysis of abuse-free contract signing. *Lecture Notes in Computer Science*, 1962:174+, 2001.

[14] B. Schneier J. Riordan. A certified e-mail protocol with no trusted third party. In *13th Annual Computer Security Applications Conference*. ACM Press, 1998.

[15] M. O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27:256–267, 1983.

[16] 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.

[17] Hiroshi Maruyama, Taiga Nakamura, and Tony Hsieh. Optimistic fair contract signing for web services. In *Proceedings of the 2003 ACM workshop on XML security*, pages 79–85. ACM Press, 2003.

[18] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. RFC 2440 OpenPGP Message Format, November 1998.

[19] Simon Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly, 1994.