To the Graduate Council:

I am submitting herewith a thesis written by Harkeerat Singh Bedi entitled “Fair Electronic Exchange Using Biometrics”. I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

_____

Dr. Li Yang, Chairperson

We have read this thesis and recommend its acceptance:

_____

Dr. Joseph Kizza

_____

Dr. Mina Sartipi

Accepted for the Graduate Council:

_____

Dr. Stephanie Bellar

Interim Dean of the Graduate School

FAIR ELECTRONIC EXCHANGE USING BIOMETRICS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee at Chattanooga

Harkeerat Singh Bedi

August 2009

Acknowledgements

I would like thank all those who helped me complete my Master of Science degree in Computer Science. I would like to thank Dr. Yang for her constant support and guidance throughout my degree and research work. Her support in the form of graduate assistantship, publishing research papers, and participation in student research competitions and research conferences is highly appreciated. I would like to thank Dr. Kizza for his kind supervision during my research and assistance with writing my thesis. I would also like to thank Dr. Sartipi for serving on my committee.

Lastly, I would like to thank my family and friends, whose suggestions and encouragement made this research possible.

Dedication

I would like to dedicate my Thesis and my Master's degree to my parents, Gurmeet Singh Bedi and Narinder Kaur Bedi, and my teachers for believing in me and offering me with opportunities to help accomplish my goals.

Abstract

Fair exchange between two parties can be defined as an instance of exchange such that either both parties obtain what they expected or neither one does. Protocols that facilitate such transactions are known as “fair exchange protocols”. We analyze one such protocol by Micali that demonstrates fair contract signing, where two parties exchange their commitments over an already negotiated contract. In this research we show that Micali’s protocol is not completely fair and demonstrate the possibilities for one party cheating by obtaining the other party’s commitment and not offer theirs. A revised version of this protocol by Bao provides superior fairness by handling the above mentioned weakness but fails to handle the possibility of a replay attack. Our proposed protocol improves on Bao’s protocol by addressing the weakness that leads to a replay attack. We also demonstrate a software implementation of our system which provides fair contract signing along with properties like user authentication achieved through the use of a fingerprint based authentication system and features like confidentiality, data-integrity and non-repudiation achieved through implementation of hybrid cryptography and digital signatures algorithms based on Elliptic Curve Cryptography.

Table of Contents

List of Figures

Chapter 1

Introduction

Commerce has come a long way since the beginning of our civilization. The ability to exchange goods and services for items of equivalent value has been widely exercised. Based on the kind of items exchanged between two parties, it can either be classified as a barter system where goods and services are exchanged, or the act of selling and buying where goods and services are sold or bought between parties in exchange for money.

The notion of fair exchange can be expressed as the ability to exchange goods or services for other goods or services in a fair manner where both the parties obtain what they expected. Being a fundamental concept, this can be implemented in various scenarios which include exchanges based on barter system or buying and selling of goods.

With the advent of computers and the internet, new means of performing commerce have been invented. E-commerce is one such solution where goods and services are bought and sold between parties using computers over a network. With the rapid growth of the internet, the magnitude of commerce performed online has also increased significantly. This increase is primarily because commerce conducted online is convenient and fast when compared to traditional methods of trade. Even though commerce of this type offers qualities like speed and convenience, properties like fairness and security are equally essential. E-commerce cannot flourish or even sustain if it is not able to provide fairness and security. Therefore the concept of fair exchange plays a vital role in shaping this form of commerce. When carried out online using computers and the internet, such fair exchange is known as fair electronic exchange.

1.1 Fair Electronic Exchange

Fair electronic exchange can be demonstrated as e-commerce that takes place between two parties who are online where exchange of goods and services is performed such that both

parties either obtain what they expected or they obtain nothing at all. After an exchange is performed or aborted prematurely, none of the parties should have an unfair advantage over the other. If cheating takes place, where one party refuses to present their part of the exchange, resources for providing fairness should be available. This may include use of additional entities like a human judge or electronic ones that can comprehend the situation and act accordingly to provide fairness. Protocols that provide such facilities are known as *fair exchange protocols*. Such protocols can be used for the following purposes:

a. *Certified E-Mail* (CEM) where Alice sends a message to Bob and gets a receipt from him in return. Providing the quality of fairness would include Alice getting the receipt only when Bob gets the message or Bob getting the message only when Alice gets the receipt.

b. *Electronic Contract Signing* (ECS) where both Alice and Bob wish to sign a contract that has been already negotiated. This would involve Alice sending her commitment (digital signature on the contract) to Bob and him sending his commitment on the same in return. Providing fairness would involve Alice receiving Bob's commitment only when her commitment is received by Bob and vice versa. This example demonstrates contract signing between two parties. However, various multi-party contract signing protocols have also been proposed in [1, 2, 3].

c. *Online payment systems* (OPS) where Alice is the seller and Bob is the buyer and payment is given in return of the item of value [4].

In the ideal case, where both Alice and Bob are guaranteed to be honest and the communication channel is secure and provides resilience, fair exchange can be achieved trivially without the aid of any external fairness providers. The above described scenarios can thus be carried out as follows:

Fair Certified E-Mail:

Step 1: Alice sends her message to Bob.

Step 2: Bob sends his receipt for the message to Alice. Receipt may be the digital signature of Bob on the message. This will also provide non-repudiation.

Fair Electric Contract Signing:

It is assumed that both parties have negotiated the contract before hand.

Step 1: Alice sends her digital signature on the contract to Bob.

Step 2: Bob sends his digital signature on the contract to Alice.

Fair Online Payment System:

Step 1: Alice sells goods or services online by sending it to Bob.

Step 2: Bob buys these good or services by paying Alice online via an e-check or e-money.

However in practice, honesty of the parties like Alice and Bob participating in an exchange can never be guaranteed. The availability of secure communication channels is not always possible and unsecured channels are easily prone to attacks. Following are the types of outcomes that may take place rendering the above mentioned exchanges incomplete.

a. Cheating Bob can always refuse to send his signature on the message to Alice after receiving her signature. In such a case, there is not much that Alice can do to obtain fairness.

b. An intruder can always stop the messages from reaching the other party. In this case the messages signed by Bob may never reach Alice. Alice may think that she has been cheated where as Bob may be unaware of this taking place. This confusion may lead Alice to request for cancellation whereas Bob may not want the same.

c. An intruder who is listening to messages sent by Alice or Bob can resend them making the other party believe that they were sent from the original sender. In case of electronic contract signing, if the intruder replays Alice's messages, there is no way for Bob to learn that the messages are not from Alice but an intruder. This is because the signature on the intruder's messages will always be verifiable using Alice's corresponding public key. Honest Bob may sign

the message and think that he has signed a new contract with Alice, whereas Alice may never know of this taking place.

d. Bob can sign a fake contract in exchange to Alice's signature on the original contract, thus cheating her by providing his commitment on a contract that does not solve her purpose.

Therefore to prevent such unwanted outcomes, external fairness providers are used to comprehend the situation if cheating is suspected and provide resolution. Such fairness providers are entities known as *trusted third parties*. As the name suggests these entities are trusted by everyone. Considering the above scenarios, Alice can communicate with the external fairness provider and obtain fairness if Bob refuses to sign the contract after receiving Alice's signature. Alice can provide the provider with information relating to the contract along with the messages she sent to Bob. The provider, thus after verifying this information and being sure that Alice is not cheating can then provide fairness to her by regenerating Bob's part of exchange or issuing a certificate that can be used as a substitute for Bob's signature. Based on their role and method of providing fairness by preventing or handling such unwanted outcomes, these providers are classified into the following types:

One class of protocols depends on gathering evidence during the transaction that can later be used to provide fairness. Two parties during their transaction also send additional information along with their messages which later can be used for resolution if one believes that it has been cheated. The cheated party contacts a human judge which looks at the additional information exchanged (evidence) and provides fairness. Such protocols are classified as *weak fair-exchange protocols* due to their inability to provide fairness during the transaction. This becomes a drawback since resolution is provided only after the transaction has been completed. In case of e-commerce where the location and availability of parties taking part in a transaction is not always fixed or known, such methods of dispute resolution may always not be possible.

To handle drawbacks like these, a second class of protocols has been defined by various researchers that provide means of avoiding disputes and obtaining resolution all during the transaction. Such protocols are known as *strong-fair exchange protocols* [5]. These protocols too use a trusted third party which can provide fairness in case of a dispute. There are two major types of third parties.

Trusted third parties that have to involve in every transaction occurring between two parties are known as *online trusted third parties*. They suffer from the disadvantage that they are required to involve in every transaction occurring on the network and this increases the overhead. Trusted third parties that are not required to involve in every transaction but only when a dispute occurs are known as *invisible trusted third parties* or *offline trusted third parties* and the protocols implementing them are known as *optimistic protocols*.

Following are the advantages provided by protocols that use an invisible trusted third party:

- The invisible third party intervenes only when cheating is suspected. In such a case the invisible third party solves the conflict by providing the complaining party with what it truly deserves. Either party can initiate this procedure if it feels it has been cheated.
- An invisible third party generates no congestion or bottlenecks as it intervenes only when cheating occurs which is very rare. In normal execution, transactions between two parties are carried out directly, bypassing the third party altogether.
- An invisible third party generates minimal expense and minimal liabilities as it is liable only for the few messages that is sends, which is usually in case of a conflict. Even if a system adds a large number of clients that carry out numerous transactions, the expense generated by such a system is minimal, since the third party intervenes only with cheating occurs and stays dormant rest of the time.

1.2 Problem Statement

In 2003, during the ACM Symposium on Principles of Distributed Computing (PODC) Silvio Micali presented a fair exchange protocol that could perform electronic contract signing [23]. The protocol was also filed under US Patent No. 5666420 in 1997 [24]. In his protocol, contract signing was achieved in a fair way by using an invisible trusted third party. Fairness in this context means that either both parties are bound to the contract (i.e. obtain each others' signature on the contract) or neither one is.

However, the protocol is not as fair as claimed by the author, since either party is able to cheat the other under certain scenarios. The protocol also does not provide any means to handle or prevent replay attacks that may take place if an intruder resends messages sent earlier by one party to another. Some of these scenarios are also illustrated in the paper by Bao [22]. Following is the actual protocol for contract signing as proposed by Micali.

1.2.1 Micali's Electronic Contract Signing Protocol (ECS)

Contract signing can be implemented between two parties say, Alice and Bob through the following steps. Steps 4 and 5 are required only when a dispute occurs.

Pre-requisites:

It is assumed that both parties have negotiated the contract before hand. Alice begins by selecting the contract file "*C*" that she needs to sign with Bob. She also selects a random value M and creates a packet "*Z*".

Packet Z contains the following information:

1. Identity of sender: This is a string that represents the sender. Public key of the sender can also be used. In this case the sender is Alice "*A*".
2. Identity of the receiver: A string that represents the receiver. Public key of the receiver can also be used. In this case the receiver is Bob "*B*".

3. Random value M: This can also be a number. It is a value known only to Alice and will later be used for completion of the contract signing process.

This information is encrypted using the public key of the third party, which is known to everyone. Thus Z can be represented as:

$$Z = E^{R}_{TP}\ (A, B, M)$$

Where E^{R}_{TP} is performing encryption using the trusted third party's (*TP*) public key using the randomness *R*. The values *A* and *B* are the identifiers of Alice and Bob respectively. *M* is the random value known only to Alice.

Steps:

1. Once the packet *Z* is created, Alice initiates the protocol by sending her signature on the packet *Z* and contract *C* to Bob.

   A1: *A -> B:* $SIG_A\ (C, Z)$

2. On receiving Alice's message, Bob sends his signature of *(C, Z)* and *Z* to Alice.

   B1: *B->A:* $SIG_B\ (C, Z) + SIG_B\ (Z)$

3. After receiving Bob's message, Alice verifies his signatures on both *(C, Z)* and *Z* and if they are valid, she sends *M* to Bob.

   A2: *A->B:* *M*

Dispute Resolution Phase:

4. Bob receives the random *M* and uses it to reconstruct *Z*. If the newly created *Z* matches with the one he received in *Step 1*, he halts and the contact signing protocol is complete. Else, Bob sends his signature of *(C, Z)* and *Z* to the trusted third party.

   If values of Z do not match:

   B2: *B->TP:* $SIG_B\ (C, Z) + SIG_B\ (Z)$

5. Third Party verifies the signatures it received from Bob. If they are valid, it decrypts *Z* using its private key and sends *M* to Bob and $SIG_B\ (C, Z) + SIG_B\ (Z)$ to Alice.

TP1: *TP->A:* $SIG_B (C, Z) + SIG_B (Z)$

TP2: *TP->B:* *M*

Micali defines the commitments of Alice and Bob on the contract C as following:

Alice's commitment to contract *C*:

$SIG_A (C, Z)$ *and M*

Bob's commitment to contract *C*:

$SIG_B (C, Z) + SIG_B (Z)$

To illustrate the fairness of the above mentioned contract signing protocol, Micali provided the following argument [23]: "Indeed, if Bob never performs Step B1, then he is not committed to C, but neither is Alice, because Bob only has received $SIG_A(C, Z)$, and has no way of learning M. However, if Bob performs Step B1, then he is committed to C, but Alice too will be so committed: either because she will honestly send M to Bob, or because Bob will get M from the invisible TP. Again, if Bob tries to cheat bypassing Step B1 and accessing directly the invisible TP to learn M, then Alice will get $SIG_B(C, Z)$ and $SIG_B (Z)$ from the invisible TP, because the invisible TP will not help Bob at all unless it first receives both signatures, and because once it decrypts Z to find M, it will also discover that Alice is the first signatory, and thus that she is entitled to $SIG_B(C, Z)$ and $SIG_B (Z)$."

1.2.2 Vulnerabilities in Micali's Protocol

This section discusses the vulnerabilities in Micali's contract signing protocol and how it is unfair in certain scenarios.

1.2.2.1 Insufficient requirements for dispute resolution:

The third party only requires Bob's signatures $(SIG_B (C, Z)$ and $SIG_B (Z))$ during the dispute resolution phase and nothing from Alice's side is required. This can cause the following attack: After receiving Alice's signature $(SIG_A (C, Z))$, dishonest Bob prepares a new contract C^l, creates the following signatures $SIG_B (C^l, Z)$ and $SIG_B (Z)$ and sends them both to third party requesting

for dispute resolution. Since these two signatures are valid and third party does not require any signatures from Alice, it forwards $SIG_B (C^1, Z)$ and $SIG_B (Z)$ to Alice and M to Bob.

This result in Bob having Alice's commitment on contract C and Alice only having Bob's commitment on a contract C^1 which is of no use to Alice.

Attack 1:

A1:	*A -> B:*	$SIG_A (C, Z)$
B1:	*B->TP:*	$SIG_B (C^1, Z) + SIG_B (Z)$

Since Z does not contain any information about the contract C and the signature of Bob on C^1 and Z are valid, the third party provides Bob with the value of M contained in Z and Alice with Bob's signature over the fake contract C^1.

TP1:	*TP->A:*	$SIG_B (C^1, Z) + SIG_B (Z)$
TP2:	*TP->B:*	M

1.2.2.2 Insufficient requirements for commitment of both parties:

As per Micali's definition, for Alice to show Bob's commitment on the contract C, she only requires Bob's signatures on (C, Z) and Z, which are $SIG_B (C, Z)$ and $SIG_B (Z)$. Alice is not required to provide the value M that creates Z. This flaw can be exploited such that Alice can always get Bob's commitment on a contract while Bob gets nothing as indicated below:

Dishonest Alice creates a random value of length Z and sends her signature of (C, Z) to Bob. Bob verifies Alice's signature and since it holds true, he sends his signatures of (C, Z) and Z to her. Alice now quits the protocol as she has received Bob's commitment. Bob on the other hand cannot get resolution form third party as Z is a random value and it cannot find M such that $Z = E^R_{TP} (A, B, M)$. This leads to Alice obtaining an advantage since she is not required to present a value M that can recreate the packet Z.

Attack 2:

Alice creates a random value of length Z.

A1: *A -> B:* $SIG_A (C, Z)$

B1: *B->A:* $SIG_B (C, Z) + SIG_B (Z)$

A2: *No response*

Bob contacts the third party for resolution:

B2: *B->TP:* $SIG_B (C, Z) + SIG_B (Z)$

The third party is not able to provide *M* to Bob as *Z* is a random value and it does not contain a value *M* such that $Z = E^R_{TP} (A, B, M)$.

TP: *Halts, as it is unable to provide the value of M.*

The above mentioned attacks are also explained in the paper by F. Bao et al [23]. They propose a revised protocol that handles these attacks by changing the requirements of the dispute resolution phase, the commitment parameters for both parties and the contents of *Z*.

1.2.2.3 Possibility of Replay Attacks:

Even though the above mentioned attacks are addressed in Bao's revised protocol, both protocols (Micali and Bao) still leave one possible attack. These protocols provide no means of identifying a replay attack that may take place during a contract signing instance. Since both the protocols do not handle this type of attack, one of them - Micali's original protocol - is used for this replay attack demonstration and illustrated below:

Attack 3:

Consider the normal execution of Micali's protocol:

A1: *A->B:* $SIG_A (C, Z)$

B1: *B->A:* $SIG_B (C, Z) + SIG_B (Z)$

A2: *A->B:* *M*

Let us assume an intruder has access to the transmissions between Alice and Bob and is able to record all the messages sent by Alice. An intruder can thus replay a message that was previously sent by Alice to Bob.

Consider the following execution of the protocol by an intruder:

I1: *I ->B:* SIG_A *(C, Z)*

B1: *B->A:* SIG_B *(C, Z) +* SIG_B *(Z)*

I2: *I->B:* *M*

If the intruder replays the message *A1* as a new request, there is no way for Bob to indentify the sender. Bob will assume the request as genuine since its signature will still be valid and respond back with his signature of the same. To this, the intruder can then send the message *A2*. This leads to commitment of a new contract between Alice and Bob. Bob assumes that he has signed a new contract with Alice and she on the other hand knows nothing about it. Consider a scenario where Alice periodically purchases selected items from Bob using the above protocol. The contracts then signed between them would also be the same, which is also the case in real world transactions and if it is agreed that all contracts expire immediately upon fulfillment (i.e. Bob gets the order, signs it, and then forgets about it), it would be hard to trace the intruder or identify the attack. It should also be noted that the intruder does not need to involve the third party to stage this attack thus making it even harder to identify.

1.3 Primary Features of Our System

We analyze Micali's protocol and uncover its shortcomings. We propose a revised protocol that handles certain weaknesses including replay attacks that can take place. We also integrate biometrics and cryptography to make a system that is complete, robust and more secure. We achieve these qualities by incorporating the following features:

1.3.1 Confidentiality

Confidentiality is the process of allowing access to information only to those who have authorization for it. It is one of the cornerstones of information security. Confidentiality is required in our system so that the information exchanged between the two parties participating in a fair exchange remains private and is not understood by intruders who try to hack their

communication. This feature can be achieved by use of cryptography, where a mathematical algorithm function is used to scramble information such that it cannot be understood by someone who is not authorized to access it and can later be de-scrambled by someone who has authorization for it.

Cryptography basically consists of the following four elements:

1. Plaintext: Information or data that can be understood by everyone, which is required to be encrypted (scrambled) using a secret key so that it cannot be understood by someone who does not possess that secret key.
2. Ciphertext: The resultant encrypted data that is achieved by use of a cryptographic mathematical algorithm function.
3. Cryptographic Function or Cipher: A mathematical function that is used to encrypt or decrypt information by using a secret key.
4. Key: Information that is usually kept as a secret, which is used along with the cryptographic function to encrypt or decrypt data.

Based on the type of key used for encryption and decryption, the process of cryptography can be classified into the following types:

1.3.1.1 Symmetric Cryptography:

This process of cryptography uses the same key for encryption and decryption. The secrecy of the information is dependent on how well this secret key can be kept private. Compared to its counterpart *asymmetric cryptography* that is explained later, symmetric cryptography is very fast and efficient. For this reason it is used widely for encrypting and decrypting large files. Even though this process is highly efficient, it has the following disadvantages:

- Key Sharing: Since an initial exchange of the secret key is required between the parties before they can begin encrypting and decryption data, safe transmission or sharing of this key becomes a problem.
- Key Management: A key is required to be shared between every two parties who are willing to exchange information securely. Therefore in a large network of users who want to exchange information with others, a unique key is required for every user pair. This storage and management of keys become difficult for each user who wants to participate is such transactions.
- Integrity: Since the receiver cannot verify whether the message has been altered or not before receipt, the integrity of data can be compromised.
- Repudiation: Since the same secret key has to be shared between users, the sender can always repudiate the messages because there is no mechanism for the receiver to make sure that the message has been sent by the claimed sender.

1.3.1.2 Asymmetric Cryptography:

This process of cryptography also known as *public key cryptography* uses different keys for encrypting and decrypting information. These keys together form a key-pair and are known as public and private keys. Public keys are shared with everyone and private keys on the other hand are kept secret and known only to the individual to whom it belongs. For a party to send information securely to another, they need to encrypt the information using the recipient's public key. The recipient can then decrypt this data and recover the message by using their corresponding private key. Since this private key is a secret known only to the recipient, the information can be communicated securely without the requirement of initial key exchange thus handling the key sharing problem. Key management also becomes convenient since there are no unique keys that are required for each user pair willing to communicate. The user can simply use the recipient's public key and begin secure communication. Non-repudiation and data integrity

can be achieved by making a party encrypt or sign the message using their private key. This can be verified by anybody using the signer's corresponding public key. Data can be securely communicated between two parties (sender and receiver) along with data integrity and non-repudiation through the following steps:

1. Sender first encrypts the message to be sent using their private key.
2. Sender then encrypts the resultant ciphertext using the receiver's public key.

The receiver on receiving the message first decrypts the ciphertext using its private key. This ensures secure communication since the receiver's private key is a secret known only to him. The resultant data is then decrypted again using the sender's public key. This provides non-repudiation and data integrity since the sender's public key is known to everybody and can be used to confirm his identity. Even though asymmetric cryptography offers features like non-repudiation and data integrity, its execution is still far slower than symmetric cryptography making it less favorable for encrypting and decrypting large files.

1.3.1.3 Hybrid Cryptography:

Hybrid cryptography handles the above mentioned disadvantages in symmetric and asymmetric cryptography. It does so by using both these cryptosystems together which provides the convenience of asymmetric cryptography along with the efficiency of symmetric cryptography.

Hybrid cryptography consists of the following two stages:

1. Data encapsulation: The process in which data to be communicated securely is encrypted using symmetric cryptography schemes which are highly efficient.
2. Key encapsulation: The symmetric secret key used to encrypt the data is then encrypted using any of the asymmetric cryptography schemes.

To encrypt a message for Bob, Alice performs the following steps:

1. Creates a random symmetric key and encrypts the message using the data encapsulation scheme.
2. Encrypts the symmetric key using Bob's public key under the key encapsulation scheme.
3. Sends both the encrypted message and the encrypted symmetric key to Bob.

To recover the message sent by Alice, Bob performs the following steps:

1. Use his private key to decrypt the encrypted symmetric key.
2. Use the recovered symmetric key to decrypt and recover the original message.

Since the major portion of the encryption that includes the actual message is encrypted using a symmetric cryptosystem, the efficiency of the system is improved. By encrypting the symmetric key using asymmetric cryptography, properties like key management, data integrity and non-repudiation are also achieved. Due to these advantages our system uses this form of cryptography.

1.3.2 User Authentication Using Biometrics

Authentication is the process identifying or confirming the authenticity of someone or something. In our case it is required so that the users can be authenticated and identified prior to their use of the fair exchange protocol. Weaknesses in user authentication can lead to one party misusing the fair exchange protocol by signing fake contracts or imposing as someone else. It is also required to prevent unauthorized access to the fair exchange system. There are various techniques that can be used for confirming a user's identity. These can be classified into three major categories.

- First category includes something that the user knows. User password or personal identification numbers are examples of such. These are used during user logins where the user enters its password to authenticate itself and use the services.

- Second category includes something that the user possesses, such as a smart card or an identification card. Examples include use for user authentication in buildings and offices to enter rooms or use services.
- Third category includes something that is part of the user. Examples can be implementation of fingerprints, iris, etc. In practice, a sample is provided to the biometric system during identification. The system then attempts to find the identity of the sample by comparing it with a database of records. During the database scanning a *matching value* is observed for every comparison made, if this values crosses a predefined threshold, then the sample is considered identified.

Our model implements an authentication system based on this third category. We use a fingerprint based authentication system, where the user scans its finger for fingerprint, which is later used for identification. Only if the user is identified, access to use the fair exchange system for contract signing is granted. We try to present a complete working system that provides mechanism for fair exchange between two parties. Our protocol improves on Micali's protocol by handling certain weaknesses including the possibilities of replay attack. We achieve confidentiality in an efficient way by using a hybrid cryptosystem and user authentication using fingerprint based authentication.

Chapter 2

Literature Review

In this chapter we present a comprehensive literature survey on recently proposed protocols that facilitate fair electronic exchange. We begin by briefly explaining the evolution of fair exchange over the recent years and how it has improved. We then analyze several fair exchange protocols and illustrate their weaknesses.

2.1 Evolution of Fair Exchange Protocols

Earliest work on exchange protocols that provided fairness was based on a class of protocols known as *gradual exchange protocols* [6, 7]. These protocols provided fairness as a measure directly proportional to the number of rounds of messages exchanged between the two participating parties. Thus as the number of messages exchanged increases, so does the probability of fairness. These protocols were complex and made use of advanced cryptographic techniques. They also required the assumption that both parties involved in the exchange are to have equivalent computing power. This assumption was removed by the introduction of an improved set of fair exchange protocols called *probabilistic protocols* [8, 9]. These protocols did not require both parties to have equivalent computing power. However a fairly large number of transmissions between the parties were still required to increase the probability of fairness to acceptable levels. This downside was addressed by the introduction of a new class of protocols that used trusted third parties for providing fairness. This reduced the number of transactions required to be exchange to a smaller number. This began with the use of an *inline trusted third party* [10], where the third party was required to involve in every transaction between the parties participating in the fair exchange protocol. Even though the number of transactions required was reduced, the involvement of third party during the protocol was very high. This made the third party as the bottleneck, since every transaction was required to be passed through the third party.

Several improvements were made in this direction of third party implementation, beginning with the introduction of *online trusted third parties* [11, 12], where the third party was required to involve only once during the entire protocol. This was major improvement since the involvement of third party was reduced substantially. Another major breakthrough was the introduction of *offline trusted third parties* [13, 14], where the third party was required to involve only when cheating was suspected or had occurred. This was again a key improvement since now the third party was required to interfere only in case of a conflict, and therefore did not become a bottleneck during the normal execution of the protocol. Such protocols, as also stated previously are known as *optimistic* protocols.

This approach of implementing an offline trusted third party has also been used by researchers in execution of various non-repudiation protocols [15, 16, 17]. Non-repudiation with respect to exchange protocols can be defined as the concept of ensuring that a party having involved in an exchange or transmission cannot later deny their involvement. The advent of public key cryptography [18] and introduction of digital signatures served as the foundation for non-reputation. A property like non-repudiation is required by fair exchange protocols to provide fairness efficiently. However, the first set of non-repudiation protocols [19, 20, 21] proposed by the International Standards Organization did not support fairness exclusively.

We analyze several optimistic fair exchange protocols and demonstrate their weaknesses. We start by briefly recalling Micali's protocol and listing its design issues. Micali's protocol is based on the use of an offline trusted third party where it is used to demonstrate fair contract signing. We discuss the revisions made by Bao et al. [22] on Micali's protocol and how their protocol handled some of its weaknesses. We explain another optimistic protocol which was proposed by Asokan et al. [35] and analyze its weaknesses. We then discuss the revisions made by Shmatikov et al. [36] on Asokan's protocol.

2.2 Micali's Electronic Contract Signing Protocol

Micali's exchange protocol for fair contract signing as explained in detail in the previous chapter provides the ability for fair exchange of commitments between two parties willing to sign a contract together. The protocol consists of an *information packet Z*, which is used in case a party cheats during the signing process. The packet *Z* contains additional information about the transaction such as the identities of both the parties along with a secret value *M* that is later used for the completion of the protocol. Alice's unwillingness to share this value *M* with Bob during the execution of the last step of the protocol leads to him being cheated. In such a case cheated Bob contacts the third party to obtain this value *M* and obtain complete commitment of Alice over the contract. Being an optimistic protocol, the third party is required to intervene only when cheating is suspected or observed. Micali defines the commitments of Alice and Bob on the contract *C* as following:

Alice's commitment to contract *C*:

SIG_A *(C, Z) and M*

Bob's commitment to contract *C*:

SIG_B *(C, Z) +* SIG_B *(Z)*

2.2.1 Analysis of Micali's Protocol

This protocol suffers from several weaknesses due to the following design issues:

1. The packet *Z* does not contain any information about the contract the parties are about to sign. This leads so several possible attacks where Bob can sign a fake contract during the signing process which leads to him acquiring Alice's commitment over the actual contract and Alice acquiring Bob's commitment over a fake contract that provides no benefit to Alice.
2. The protocol provides no means or mechanisms for handling replay attacks that may take place if an intruder resends messages sent earlier by Alice to Bob. Bob will not be able to recognize if the message is a new request of a replay attack.

The first design issue was discussed by Bao [22] in their paper and several possible attacks were identified and demonstrated. Several changes were made to the commitments required by both parties to prove each other's obligation to the contract. One of the changes included the inclusion of the hash of the contract in the *information packet Z*. This made it possible to handle the attacks that may take place when one party tries to sign a fake contract because hash value of the fake contract differs from that of the original contract, thus identifying the change in contract. Following is Bao's protocol under normal execution where two parties sign a contract together by exchanging commitments.

2.3 Bao's Revised Electronic Contract Signing Protocol

In 2004, during the Australasian Conference on Information Security and Privacy (ACISP), Bao et al. proposed a contract signing protocol that improved on Micali's work. Attacks made possible due to the first design issue were identified and addressed. Bao's protocol handles the first two types of attacks (*Attack 1* and *2*) discussed in the previous sections by changing the requirements of the dispute resolution phase, the commitment parameters for both parties and the contents of *Z*. Contract signing between two parties, say Alice and Bob, can be achieved through the following prerequisites and steps.

2.3.1 Prerequisites:

It is assumed that both parties have negotiated the contract before hand. Alice begins by selecting the contract file "*C*" that she needs to sign with Bob and creates a hash *H(C)* of it. She also selects the random values *M* and *R* and creates a packet "*Z*". Packet *Z* contains the following information:

1. Identity of sender: This is a string that represents the sender. Public key of the sender can also be used. In this case the sender is Alice "*A*".
2. Identity of the receiver: A string that represents the receiver. Public key of the receiver can also be used. In this case the receiver is Bob "*B*".

3. Random value *M*: This can also be a number. It is a value known only to Alice and will later be used for completion of the contract signing process.
4. Hash of the contract *C*.

This information is encrypted using the public key of the third party, which is known to everyone. Thus packet *Z* can be represented as:

$$Z = E^{R}_{TP}\ (A, B, H(C), M)$$

Where E^{R}_{TP} is performing encryption using the trusted third party's (*TP*) public key using the randomness *R*. The values *A* and *B* are the identifiers of Alice and Bob respectively. *M* is the secret random value known only to Alice and *H(C)* stands for the hash of the contract.

2.3.2 Steps:

1. Once the packet *Z* is created by Alice, she initiates the protocol by creating her signature on the packet *Z*, contract *C*, and the identities of her, Bob and the third party, and sending them to Bob.

   A1: *A -> B:* $SIG_A\ (A, B, TP, C, Z)$

2. On receiving Alice's message, Bob verifies her signature and if valid, sends his signature of **(***A, B, TP, C, Z*) to Alice. Otherwise he halts.

   B1: *B->A:* $SIG_B\ (A, B, TP, C, Z)$

3. After receiving Bob's message. Alice verifies his signatures on **(***A, B, TP, C, Z*) and if valid, sends *M* and *R* to Bob.

   A2: *A->B:* *M, R*

2.3.3 Dispute Resolution Phase:

4. Bob receives the values *M* and *R* and uses them to reconstruct *Z*. If the newly created *Z* matches with the one he received in *Step 1*, he halts and the contact signing protocol is complete. Else, Bob sends his and Alice's signature of (*A, B, TP, C, Z*) to the trusted third party. If values of *Z* do not match:

B2: *B->TP:* $SIG_B\ (A, B, TP, C, Z) + SIG_A\ (A, B, TP, C, Z)$

5. Third Party verifies the signatures it received from Bob. If they are valid, decrypts *Z* using its private key and sends ($SIG_B\ (A, B, TP, C, Z), M, R$) to Alice and ($SIG_A\ (A, B, TP, C, Z), M, R$) to Bob.

   If contents of Z are legit and signatures are valid:

   TP1: *TP->A:* $SIG_B\ (A, B, TP, C, Z) + M + R$

   TP2: *TP->B:* $SIG_A\ (A, B, TP, C, Z) + M + R$

   Else:

   Halts or sends an error message

Following are the new commitments as revised by Bao:

Alice's commitment to the contract *C* can be defined as:

$SIG_A\ (A, B, TP, C, Z), M, R$ where $Z = E^R_{TP}\ (A, B, H(C), M)$

Bob's commitment to the contract *C* can be defined as:

$SIG_B\ (A, B, TP, C, Z), M, R$ where $Z = E^R_{TP}\ (A, B, H(C), M)$

The second design weakness of the unavailability of a mechanism to identify and prevent replay attacks was not addressed by both Micali and Bao. Our protocol provides a mechanism to address these types of attacks and is our unique contribution to the fair exchange protocol.

2.4 Asokan's Electronic Contract Signing Protocol

During the IEEE Symposium on Research in Security and Privacy in 1998, Asokan et al. presented asynchronous protocols for optimistic fair exchange. These protocols included one that facilitated electronic contract signing between two parties willing to sign a contract together. This contract signing protocol that is our topic of discussion consists of three independent sub-protocols namely *exchange*, *abort* and *resolve*. Both parties Alice and Bob start the contract signing process by executing the *exchange* sub-protocol. This sub-protocol includes the creation and exchange of commitments between the parties. If both parties are honest and their

commitments are exchange successfully, then the contract signing process is complete and both parties have successfully signed the contract together. The contracts thus created as known as *standard* contracts. However, if Bob tries to cheat or does not respond in a timely manner, Alice can then execute the *abort* sub-protocol. This involves Alice sending an *abort* token to the third party who then takes the steps required based on certain conditions explained below. If cheating is suspected or observed which may include either party not providing their commitment or providing false information, the other party can execute the *resolve* sub-protocol with the third party and obtain a *replacement* contract instead. As per the protocol definition in [10] the contracts, based on the sub-protocols executed can be of the following two forms:

a. Standard Contract: $\{me_1, N_A, me_2, N_B\}$

b. Replacement Contract: $\{me_1, me_2\}$

The abort token as provided by the third party is defined as: $SIG_T \{aborted, ma_1\}$

The variables contained in the above contract and token definitions are explained below.

2.4.1 Exchange sub-protocol

Contract signing between two parties, say Alice and Bob, can be achieved through following the below steps. It is assumed that both parties have negotiated the contract beforehand.

2.4.1.1 Steps:

1. Alice initiates the protocol by sending the following signed message to Bob.

   A1: *A -> B:* $me_1 = SIG_A (V_A, V_B, T, C, H (N_A))$

   The message me_1 contains the following information:

   - V_A: Public key of Alice
   - V_B: Public key of Bob
   - T: Identity of the trusted third party
   - C: Contract to be signed between the two parties

- H (N_A): Hash of the secret random number known only to Alice. This secret number, also known as *contact authenticator* is used later for contract complete and verification.

The above information altogether is signed by Alice using her private key which can later be verified by using her corresponding public key.

2. Once received, Bob replies by providing his own commitment in the following form:

   B1: B -> A: $me_2 = SIG_B\ (me_1, H\ (N_B))$

   This message contains the signature of Bob over the previously received message from Alice me_1 and has of N_B which is Bob's *contract authenticator*.

3. On receiving Bob's signature, Alice sends her contract authenticator to Bob.

   A2: A -> B: $me_3 = N_A$

4. On receiving Alice's contract authenticator, Bob sends his to Alice.

   B2: B -> A: $me_4 = N_B$

Thus, the final contract signed between both parties is of the standard form $\{me_1, N_A, me_2, N_B\}$.

2.4.2 Abort sub-protocol

Alice can execute this sub-protocol if she wants to abort the contract signing process. Reasons may include delay in receiving Bob's commitment or unwillingness in signing the contract with Bob. Alice executes the following steps to abort the process.

2.4.2.1 Steps:

1. Alice sends the following message to the third party:

   A1: A -> TP: $ma_1 = SIG_A\ \{aborted, me_1\}$

   The message contains her signature over her previously sent message me_1 to Bob and the token *aborted*. The definition of the token *aborted* is not explained clearly in [10] and therefore for the purpose of analysis, is assumed as a predefined string.

2. On receiving the abortion request from Alice, the third party checks its database for any resolution request received earlier regarding this contract. If this contract has been resolved

previously, then the abortion request is ignored and the replacement contract is sent instead. If no resolution request has been registered for this contract, then the third party aborts the contract and updates its database with the same.

If (resolved earlier):

T1: T -> A: $ma_2 = SIG_T \{me_1, me_2\}$

Else:

T1: T -> A: $ma_2 = SIG_T \{aborted, ma_1\}$ *and updates database entry with abort*.

Even though a party requests for an abortion, it cannot be certain of receiving it since the other party may have already requested to resolve the contract. There it becomes a matter of which party sends their request to the third party earlier, leading to uncertainty and an implicit race condition.

2.4.3 Resolve sub-protocol

Either party can request for resolution based on the information possessed by them. Resolution request requires the messages me_1 and me_2 signed by the party seeking resolution. Following are the steps executed by Bob seeking resolution from the third party:

2.4.3.1 Steps:

1. Bob can request resolution once it receives me_1 from Alice. Bob then creates me_2, signs both the messages together and sends them to the third party. Alice on the other hand can also request for resolution when she receives me_2 from Bob by performing similar steps.

   B1: B -> TP: $mr_1 = SIG_B \{me_1, me_2\}$

2. On receiving Bob's signature, third party verifies its database whether the contract has been aborted earlier. If aborted, third party ignores the request and sends the abort token instead. Else, third party signs the messages, sends the replacement contract to Bob and updates its database accordingly.

If (aborted):

T1: T -> B: $mr_2 = SIG_T$ {aborted, ma_1}

Else:

T1: T -> B: $mr_2 = SIG_T$ {me_1, me_2} and updates the database with resolve.

2.4.4 Analysis of Asokan's Protocol

Asokan's contract signing protocol suffers from several weaknesses that occur due to a design issue in the exchange protocol. The messages exchanged between both parties during the protocol are not linked to the previous message. This leads to the following types of attacks take may take place and render the protocol unfair:

a. Replay attacks: Intruder replays messages sent by Alice earlier.
b. Malicious participant: Cheating Bob obtains a contract that differs from the one received by Alice.

Following is an illustration of both these types of attacks taking place:

2.4.4.1 Replay Attack:

Consider the normal compete execution of Asokan's exchange protocol:

A1: A -> B: $me_1 = SIG_A (V_A, V_B, T, C, H (N_A))$

B1: B -> A: $me_2 = SIG_B (me_1, H (N_B))$

A2: A -> B: $me_3 = N_A$

B2: B -> A: $me_4 = N_B$

Let us assume an intruder has access to the transmissions between Alice and Bob and is able to record all the messages sent by Alice. An intruder can thus always resend a message that was previously sent by Alice.

Now, consider the following execution of the protocol by an intruder:

I1: A -> B: $me_1 = SIG_A (V_A, V_B, T, C, H (N_A))$

B1: B -> A: $me_2 = SIG_B (me_1, H (N_B))$

I2: *A -> B:* $me_3 = N_A$

B2: *B -> A:* $me_4 = N_B$

If the intruder replays the message *A1* as a new request, there is no way for Bob to indentify the sender. Bob will assume the request as genuine as its signature will still be valid and respond back with his signature of the same. To this, the intruder can then send the message *A2* and Bob then responds back with *B2*, thus completing the contract signing protocol. This leads to signing of a new contract between Alice and Bob. Bob assumes that he has signed a new contract with Alice and she on the other hand knows nothing about it. Also, the intruder does not require the involvement of third party to execute this attack. Therefore no trace of the attack is left once it is executed.

2.4.4.2 Malicious Participant:

Consider the following protocol execution:

Alice and Bob perform the first three steps of the exchange protocol. Alice sends her signature me_1 to Bob. Bob responds with his signature me_2. On receiving, Alice sends her contract negotiator N_A to Bob.

A1: *A -> B:* $me_1 = SIG_A\ (V_A, V_B, T, C, H\ (N_A))$

B1: *B -> A:* $me_2 = SIG_B\ (me_1, H\ (N_B))$

A2: *A -> B:* $me_3 = N_A$

Once Bob receives Alice's contract negotiator N_A, he creates a new signature me_2^1 using a different contract negotiator N_B^1 and does not respond to Alice. Since Bob now has me_1, which is signed by Alice and also the corresponding contract negotiator N_A, he can cheat by creating a different valid contract as: $\{me_1, N_A, me_2^1, N_B^1\}$

Since Alice did not receive the contact negotiator from Bob, she contacts the third party for resolution by sending her the following request:

A2: *A -> TP:* $ma_1 = \{me_1, me_2\}$

The third party looks at the request and checks its database for the corresponding entry. Since this contract has not been aborted previously, it signs the messages and responds back with the replacement contract:

TP: *TP -> A* $ma_2 = SIG_T \{me_1, me_2\}$

We can notice that, two different valid contracts have been created. Bob holds a *standard contract* which includes the modified signature me_2^1. Alice on the other hand holds a *replacement contract* that includes the original me_2. As per the protocol, both parties hold a valid contract. However they are inconsistent due to differences in the hash values of the random numbers (N_B and N_B^1) and the commitments (me_2 and me_2^1). The original paper [10] does not provide any information regarding handling this situation.

2.5 Shmatikov's Revision of Asokan's Protocol

Asokan's exchange protocol suffers from the above mentioned attacks since the contract negotiators which are basically random numbers created by both parties, are not used appropriately. Messages are not linked to their previous message which breaks the continuation of the exchange protocol making room for attacks. Inclusion of information related to the random number (the *contract negotiator*) used by both parties in the messages exchanged can help handle these types of attacks. Following is the revised protocol by Shmatikov that handles the above mentioned attacks:

A1: *A -> B:* $me_1 = SIG_A (V_A, V_B, T, C, H (N_A))$

B1: *B -> A:* $me_2 = SIG_B (me_1, H (N_B))$

A2: *A -> B:* $me_3 = SIG_A (N_A, H (N_B)\}$

B2: *B -> A:* $me_4 = SIG_B (N_B, H (N_A)\}$

The steps *A2* and *B2* were modified such that the messages exchanged also include information related to the *contract negotiator* used by the other party. These steps were also signed so that the identity of the other party could be verified. The intruder now cannot stage a replay attack since

he would have to sign the message *A2*, which will not be possible since the private key of Alice is a secret known only to her. Linking Alice's contract negotiator N_A in step me_3 with Bob's contract negotiator addresses the attack that may takes place when Bob is malicious, since now Bob cannot use N_A with the different message me_2^1 to create a valid contract. The message me_4 signed by Bob is also linked to Alice's contract negotiator to prevent a symmetric replay attack.

The design structure in our protocol that addresses replay attacks is inspired by the above Shmatikov's revision of Asokan's protocol. The use of the random numbers generated by both parties appropriately by including their information in subsequent messages helps prevent replay attacks and linking of all messages to their previous message helps prevent symmetric replay attacks.

Chapter 3

Methodology

We present a complete working system that provides fair electronic exchange along with user authentication. We implement an invisible third party that is used to provide fairness if cheating is suspected. Our work comprises of three major parts. We begin with a revised protocol that is based on Micali's contract signing protocol. We improve on Micali's protocol by handling certain types of weaknesses that may lead to attacks where one party can obtain the other's commitment to contract without providing theirs. We also handle the possibility of replay attack that may take place if an intruder resends the messages sent earlier from one party to the other.

We implement a fingerprint based authentication technique to provide user authentication and enrollment. The user can scan their finger for fingerprint which is then used for authentication. When authenticated, the user can proceed and use the system to sign contracts with others. Feature of enrollment is also provided if the user is new and wants to use the system for the first time. To provide confidentiality, handle replay attacks and confirm the identity of the other party, cryptography and digital signatures are used. All messages communicated between users participating in the contract signing process are encrypted using a hybrid cryptosystem. This offers better performance by achieving the convenience of asymmetric cryptography with the efficiency of symmetric cryptography. These three parts are explained in detail in the upcoming sections.

3.1 Our Protocol

This section discusses our protocol and how it provides fair contract signing. Following is an adaptation of the protocol where the privacy of messages is not essential. It is assumed that both parties are not concerned about the privacy of their messages (or contract), provided that fairness is guaranteed. This approach is taken for a simpler illustration of the protocol. Privacy of

messages can be achieved using cryptography, which is explained in detail in the subsequent topics. Following is the protocol under normal execution when both Alice and Bob are honest and there is no intruder. Contract signing between two parties, say Alice and Bob, can be achieved through following the prerequisites and steps below.

3.1.1 Prerequisites

It is assumed that both parties have negotiated the contract before hand. Alice selects the contract file "*C*" that she needs to sign with Bob. She also selects a secret random value M and creates a packet "*Z*". Packet *Z* contains the following information:

1. Identity of sender: This is a string that represents the sender. Public key of the sender can also be used. In our case the sender is Alice "*A*".
2. Identity of the receiver: A string that represents the receiver. Public key of the receiver can also be used. In our case the receiver is Bob "*B*".
3. Random value *M*: This can also be a number. It is a value known only to Alice and will later be used for completion of the contract signing process.
4. Hash of the contract *C,* that is *H(C)*

This information is encrypted using the public key of the third party, which is known to everyone. Thus *Z* can be represented as: $Z = E^{R}_{TP}\ (A, B, H(C), M)$

Where E^{R}_{TP} is performing encryption using the trusted third party's (*TP*) public key using the randomness *R*. The values *A* and *B* are the identifiers of Alice and Bob respectively. *H(C)* is hash of the contract file *C*. The value *M* is a secret random known only to Alice.

3.1.2 Steps

1. Alice sends a nonce NA_1 to Bob. Nonce is a random number used only once for the prevention of replay attacks.

   A1: *A -> B:* NA_1

2. On receiving Alice's nonce, Bob signs it using his private key and sends it back to her along with his nonce. This step ensures Alice that it is indeed Bob who signed the message as Bob's private key is a secret known only to Bob.

   B1: *B->A:* $SIG_B (NA_1) + NB_1$

3. After receiving the above package Alice can verify the digital signature of Bob on NA_1 using his public key. If it matches, Alice is sure that it is indeed Bob and there is no replay attack. Alice now signs Bob's nonce using her private key so that he can be sure of the same. Alice also signs *Z* along with *H(C)* and sends all of it to Bob. This step makes Alice partially committed to the contract as she has signed the hash of contract using her private key.

   A2: *A->B* $SIG_A (NB_1) + SIG_A (H(C), Z) + Z + C$

4. On receiving Alice's signatures, Bob can now verify them using her public key. If they match, Bob is sure that it is indeed Alice and there is no replay attack. It is now Bob's turn to send his commitment to Alice. Bob does this by signing the hash of contract *H(C)* along with *Z* and sending it to Alice.

   B2: *B->A* $SIG_B (H(C), Z)$

5. After receiving the message, Alice can verify Bob's signature and if it holds true, she sends him the values *M* and *R* signed by her.

   A3: *A->B* $SIG_A (M, R) + M + R$

   Bob receives this package and learns the values *M* and *R* which is then used to reconstruct *Z*. If the newly created *Z* matches with the one he received in *Step 3*, the transaction is complete and both the parties have successfully signed the contract together.

Following is the protocol under abnormal execution where Alice does not provide Bob with the correct secret values of *M* and *R* in the final step.

3.1.3 Dispute Resolution Phase

On receiving Alice's signature of *Z* and *H(C)*, Bob sends his signature of the same to Alice. Alice is then required to send the values of *M* and *R* to Bob to provide her complete commitment on the contract. For the purpose of discussion let us assume that Alice cheats Bob by not providing him with the correct secret values of *M* and *R* after she receives his signature on the contract. Thus Bob is left with a partial commitment from Alice, where as Alice has Bob's complete commitment. Bob can thus execute the following steps to obtain resolution.

1. Bob sends the packet *Z* that he initially received from Alice, the contract *C* that was to be signed, Alice's signature of *Z* and *H(C)* and his own signature of *Z* and *H(C)* to the third party.

   B1: B->TP $Z + C + SIG_A (H(C), Z) + SIG_B (H(C), Z)$

2. Third party, on receiving Bob's request performs the following steps,
   a. Computes the value of *H(C)* from *C***.**
   b. Decrypts the packet *Z* since it knows its own private key and extracts the secret *M*.
   c. Verifies the contents of *Z* to include the identities *A* for Alice and *B* for Bob and *H(C)* for the hash of the contract.
   d. Verifies the signatures of Alice and Bob over *H(C)* and *Z***.** Third party can do this since it knows the public keys of both Alice and Bob. If the signatures are valid, third party provides Bob with *M* it recovered from *Z* and Alice with the signature of Bob over *H(C)* and *C***.**

   If contents of Z are legit and the signatures are valid:

   TP1: TP->B *M*

   TP2: TP->A $SIG_B (H(C), Z)$

   Else:

   No action

If secrecy of the contract is required, the cheating party can directly send the hash of the contract instead of the original contract to the third party during the dispute resolution phase. Thus the third party can skip the step where it is required to compute the hash of the contract, and can directly proceed to the next step where it decrypts the packet *Z* to extract the secret *M* as shown above in *Step 2*. The secrecy of the contract based on the above scenario is not possible in Micali and Bao's protocols since they use the actual contract as part of the signatures and not its hash. This completes the dispute resolution phase as the third party provides Bob with the correct value of *M* that gives him complete commitment from Alice on the contract.

3.2 Protocol Description

To sign a contract in theory would mean exchange of signatures from both the parties on the same contract. That is if Alice and Bob were to sign a contact with each other, Alice would need to have Bob's signature (commitment) on the contract in the form of $SIG_B(C)$ and Bob would need to have Alice's signature $SIG_A(C)$.

For this to be fair it would require that each party gets the other party's signature only when their signature is received by the other party. Implementation of this exchange may be straightforward under conventional transactions where both the parties are physically available or geographically locatable. However under e-commerce where the transactions take place over the internet, it becomes reasonably complicated. This is because these transactions transcend geographical boundaries and it is not always possible to contact the other party for resolution or queries once the transaction is complete.

A party can always refuse to provide their signature once they receive the same from the other party. And since locating someone over the internet or to know when the other party will be available online again is not always possible, fairness cannot be guaranteed. Therefore to provide fairness, additional information (e.g. packet *Z*) is exchanged along with the contract (or messages) during these fair exchange transactions. This additional information is then examined

and used for resolution if a party does not respond appropriately. Thus the commitments for both parties are modified as follows:

For Bob to have Alice's commitment on the contract, he would need:

SIG_A *(H(C), Z), M and R* where $Z = E^R_{TP}$ *(A, B, H(C), M)*

For Alice to have Bob's commitment on the contract, she would need:

SIG_B *(H(C), Z), M and R* where $Z = E^R_{TP}$ *(A, B, H(C), M)*

To prove the commitments, parties will be required to present not only the signatures but also the contents *M, R, A, B* and *H(C)* that altogether satisfy $Z = E^R_{TP}$ *(A, B, H(C), M).* Failing to do so shall render the commitments as well as the signed contract as invalid.

The additional information used in our protocol is $Z = E^R_{TP}$ *(A, B, H(C), M)* and values *R* and ***M*.**

The packet *Z* is created using the following information:

1. Identities of both the parties: Those are *A* for Alice and *B* for Bob. We use text strings for this purpose. The public keys of parties or any other type of identification information can also be used.
2. Hash of the Contact: A Message digest created by using a cryptographic hash function. It is usually of fixed length and is unique to the data hashed. A change in hash generated would represent a change in the data hashed.
3. Random number *M*: Alice creates this random number that is used as a part of the contract signing process and for contract verification by Bob. Initially, Alice signs the packet **Z** which includes this random number *M* along with contract *C* and sends it to Bob. After receiving her signature, Bob still does not have her complete commitment over the contract as he does not know the value of *M* that was used to create the packet ***Z*.** Bob cannot find this value on his own since the packet *Z* is encrypted using the third party's public key and the third party's corresponding private key is a secret known only to the it. Bob also cannot obtain it from Alice unless he provides his commitment of the same to Alice. Thus the only option available

to Bob is to send his commitment to Alice. Alice then verifies his signature and if valid sends him the values *M* and *R*. On receiving *M* from Alice, Bob can reconstruct *Z* using *A*, *B* and *H(C).* Bob does so by encrypting these values using the third party's public key along with the random *R*. If the newly created *Z* matches with the one he received earlier, he has Alice's complete commitment on the contact.

4. The value R: It is this randomness that is used by the public-key cryptosystem during encryption to produce the same cipher text for a given data using the same public key. This is usually not the case since public-key cryptosystems produce different cipher texts for the same data over the same public key if this value is not explicitly specified. In our protocol, during the final step, comparison of cipher texts is performed for contract verification purposes. Thus the production of same cipher text over the same data and same public key becomes a requirement and can only be achieved if the randomness used for encryption is stored and reused.

3.3 Our Contribution

Our contribution in the revised protocol encompasses the following design changes:

3.3.1 Inclusion of Hash of the Contract in Z

Information about the hash of the contact is added in the packet *Z*. A Hash is basically a fixed length value returned by a cryptographic hash function that takes the contract file data as the input. These hash values are usually unique for a given message (contract) and changes if the message is altered. If one party modifies the contract during the contract signing protocol, the hash generated on the modified contract will not match with the one generated on the original contract. If the hashes on this contract do not match between the parties, it can be concluded that they have different contracts and cheating can be identified. It can be concluded so as it is extremely unlikely to find the same hash on two different contacts. We specifically assume that SIG_A *()* and SIG_B *()* are secure signing algorithms that exhibit the following four properties:

a. It is easy to compute hash for any given data.

b. It is extremely difficult to recreate the data from its hash.

c. It is extremely unlikely to find the same hash for two different data.

d. It is extremely difficult to modify a given data without changing its hash.

3.3.2 Exchange of Random Nonce Prior to Exchange of Commitments

Random nonce is exchanged between both parties prior to the exchange of the contract commitments. This step ensures that both parties are certain about the other's identity. It also helps to identify a replay attack that may occur if an intruder tries to replay messages previously sent by Alice. This can be achieved through the following three steps. Let the two parties be Alice and Bob,

1. Alice sends a random nonce to Bob.
2. Bob signs the nonce it received from Alice using his private key and sends his random nonce to her.
3. Alice verifies the signed value using Bob's public key. If valid, she signs his nonce using her private key and sends it to Bob. Bob can verify the same using her public key.

3.3.3 Inclusion of Hash of Contract in Signature

Hash of the contract instead of the actual contract is used during the digital signature generation process. This step ensures privacy of the contract between both the parties. On the contrary, if the contract is part of the digital signature, during dispute resolution the cheated party would also have to provide the contract along with the other information to the third party. This is necessary since all information that is part of the signature is required in order to verify it and this includes the contract. If the secrecy of the contract is essential to the cheated party, executing this step would make it lose the same. Protocols by Micali and Bao both use the actual contract instead of its hash during the digital signature generation process.

3.4 Handling of Attacks

This section discusses how the attacks discovered in Micali's protocol are being handled by our protocol.

3.4.1 Insufficient Requirements for Dispute Resolution by the Third Party:

In Micali's protocol, the third party only requires Bob's signatures *($SIG_B(C, Z)$* and *$SIG_B(Z)$)* during the dispute resolution phase. Nothing from Alice's side is required. This can lead to the following attack:

After receiving Alice's signature *($SIG_A(C, Z)$)* dishonest Bob can prepare a new contract C^1, create the following signatures $SIG_B(C^1, Z)$ and $SIG_B(Z)$ and send them to the third party requesting for resolution. Since these two signatures are valid and third party does not require any signatures from Alice or even the contract, it forwards $SIG_B(C^1, Z)$ and $SIG_B(Z)$ to Alice and *M* to Bob. This result in Bob having Alice's commitment on contract C and Alice only having Bob's commitment on another contract C^1.

This attack is handled by our protocol since now the third party requires signatures of both parties participating in the contract signing process. Also, since hash of the contract file is now included in the packet *Z*, if Bob signs a fake contract, the hashes will be different and the signature will not be verifiable.

3.4.2 Insufficient Requirements for Commitment of Both Parties:

As per Micali's definition, for Bob to be committed to the contract *C*, Alice only requires Bob's signatures on the contract *C* and packet *Z* that are $SIG_B(C, Z)$ and $SIG_B(Z)$. Alice is not required to provide the value *M* that creates *Z*. This flaw can be exploited such that Alice can always get Bob's commitment on a contract while Bob gets nothing. Following is the attack:

Dishonest Alice creates a random value of length Z and sends her signature of *(C, Z)* to Bob. Bob verifies Alice's signature and since it holds true, he sends his signatures of *(C, Z)* and *Z* to her. Alice now quits the protocol as she has received Bob's commitment. Bob on the other

hand cannot get resolution form third party as Z is a random value and it cannot find M such that $Z = E^{R}_{TP}(A, B, M)$. This leads to Alice obtaining an advantage since she is not required to present a value M that can recreate the packet Z.

This attack is handled by our protocol by changing the requirements of contract commitment such that Alice is also required to provide a value of M such that $Z = E^{R}_{TP}(A, B, M)$.

3.4.3 Replay Attacks:

Even though the above mentioned attacks are addressed by Bao's revised protocol, both protocols (Micali and Bao) still leave one possible attack. These protocols provide no means of identifying a replay attack that may take place during a contract signing instance as demonstrated in Chapter 1. Let us assume an intruder has access to the transmissions between Alice and Bob and is able to record all the messages sent by Alice. An intruder can thus always replay a message that was previously sent by Alice. Bob will assume the request as genuine as its signature will still be valid and respond back with his signature of the same. This leads to signature of a new contract between Alice and Bob. Bob assumes that he has signed a new contract with Alice and she on the other hand knows nothing about it.

This attack is handled by our protocol since both parties are required to exchange random nonce between each other before they exchange their commitments. Therefore if an intruder replays a contract signing request sent previously by Alice, Bob would respond with a nonce which has to be signed using Alice's private key. Since Alice's private key is a secret known only to Alice, the intruder will not be able to provide the signature and the contract signing process would halt, preventing the replay attack.

To recognize a replay attack Bob can also re-compute Z by using previously obtained values of M from Alice. If a match occurs Bob can conclude that it is a replay attack. However, only limited values of M (or contracts) can be accessible to a party in practice due to physical limitations on database sizes. Our use of nonce removes this limitation and does not require

storage of previous values of M or even the contract since the attack can be easily identified if nonce verification fails.

3.5 Role of Biometrics

Properties like user authentication increase the robustness of a system that implements such fair exchange protocols and creates a complete system that can be used for making end-to-end transactions. This section discusses the role biometrics in our system. Biometrics is used for user authentication where something the user has, in our case their finger, is used for their identification. The biometric data required by our system is captured by scanning a finger over the fingerprint scanning device. This model is known as a fingerprint-based authentication system. It is one of the cheapest, fastest and most reliable ways to identify someone. Being one of the oldest is it also the most widely used authentication system. Based on their technique of processing fingerprints and identifying users, fingerprint based authentication systems are classified as graph based or minutiae based. Our system implements the minutiae based one since it is more efficient [26].

Once a finger is scanned, it is temporarily stored in raw image format. This capture also known as the *fingerprint* is then used for creating a *template*. Templates are small files created from the unique characteristics extracted from these fingerprints. These unique characteristics are features in the ridges and furrows of the friction skin observed on the fingerprint image and are known as *minutiae* [25]. Points on the fingerprint where ridges end and split are known as *ridge endings* and *bifurcations* respectively. These are illustrated in the following figure:

Figure 1: Ridge ending (circle marker) and Bifurcation (square marker)

The circle marker in the above figure shows a *ridge ending* where a ridge on the fingerprint ends abruptly. The square marker shows a *bifurcation*, where a single ridge divides into two ridges forming a "Y" shape.

Templates in general only take a fraction of the size of a fingerprint image and usually range from a couple of bytes to a couple of kilobytes at most thus making them very efficient for comparisons and storage. A template usually has around 100 to 200 minutiae. It should be noted that it is these templates that are used for verification and identification purposes and not the actual fingerprint image. Biometric template matching algorithms are used to compare these templates with one another which basically look for similarities in their minutiae. During a comparison of two templates, every time similarities in minutiae are discovered, a scalar count variable is incremented. Once this count crosses a pre-defined threshold value, the templates are considered to be of the same person.

Following is a screenshot of the template file generated by a utility called “MINDTCT” when it was provided a fingerprint image. This is added to provide a better understanding of this subject. MINDTCT is part of the NIST Biometric Image Software package. It is a minutiae detector that automatically locates and records ridge ending and bifurcations (minutiae) in a fingerprint image. This template file contains a list of minutiae that were discovered from the fingerprint image. These values represent their location based on X and Y axis and their proximity to other minutiae in the fingerprint. For detailed information on this utility and the description of the contents of the template file, readers are requested to refer the manual (man) pages of MINDTCT software.

```
280 Minutiae Detected

  0 :   20,  168 : 17 :  0.079 :RIG : APP :  0 :  199, 203;  5 :
  1 :  111,  261 :  4 :  0.061 :BIF : DIS :  2 :  199, 203;  3 :
  2 :  133,  250 :  5 :  0.070 :RIG : APP :  0 :  203, 196;  2 :
  3 :  138,  274 : 20 :  0.069 :RIG : APP :  0 :  199, 203;  1 :
  4 :  158,  390 : 18 :  0.162 :RIG : APP :  0 :  217, 332;  3 :
  5 :  161,  260 : 20 :  0.332 :RIG : APP :  0 :  203, 196;  1 :
  6 :  169,  402 :  2 :  0.373 :RIG : DIS :  1 :  217, 332;  1 :
  7 :  178,  495 :  4 :  0.348 :RIG : DIS :  1 :  199, 450;  1 :
  8 :  181,  588 :  5 :  0.149 :RIG : APP :  0 :  236, 575;  2 :
  9 :  186,  687 : 21 :  0.417 :RIG : DIS :  1 :  203, 620;  3 :
 10 :  190,  607 :  5 :  0.070 :RIG : APP :  0 :  202, 591;  0 :
 11 :  199,  203 : 19 :  0.374 :RIG : APP :  0 :  203, 196;  0 :
 12 :  199,  450 :  4 :  0.752 :RIG : DIS :  1 :  214, 384;  2 :
 13 :  202,  591 :  6 :  0.350 :RIG : APP :  0 :  264, 555;  2 :
```

Figure 2: Template of a fingerprint

Our software is written in Java and implements a minutia based fingerprint authentication system and the API’s used were provided by Griaule Biometrics. Following is a screenshot of the minutiae that were extracted from the fingerprint image captured using our software by scanning a finger over the fingerprint reader.

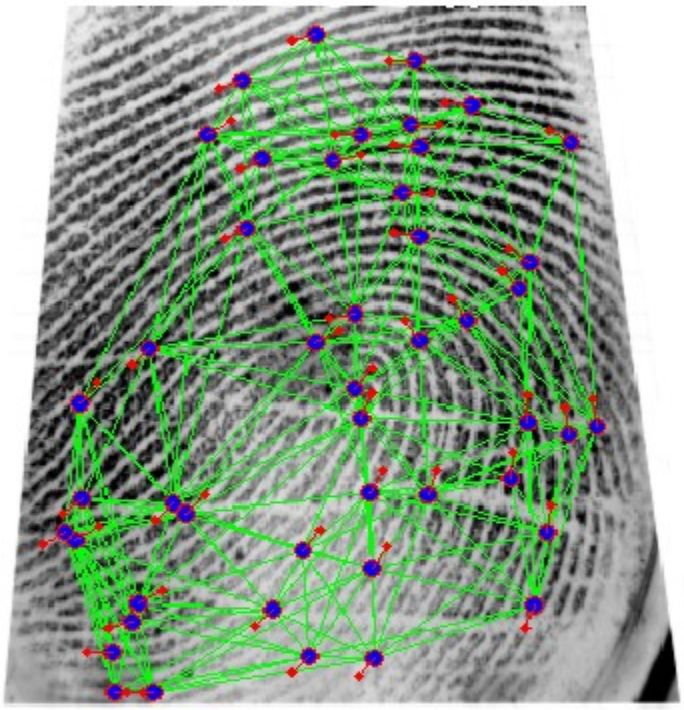

Figure 3: Minutiae extracted from a fingerprint using Griaule software API

The large blue colored circles represent the minutiae that were extracted from the fingerprint. The smaller red colored circles are used to represent the angles each minutia make from the horizon. The green lines connecting the blue colored circles show proximity between two discovered minutiae. A combination of the above information is contained in each template which is then stored and used later for user authentication.

3.5.1 Advantages of Minutiae Based Fingerprint Authentication System

Minutiae are widely believed to be the most selective and dependable features of a fingerprint. Following are some advantages of using minutiae based fingerprint authentication system:

- Compared to its competitive graph-based technique, the amount of information required to be stored in the database for fingerprint matching and the amount of time required for processing are both less [26]. This is beneficial as fewer resources are used for storage and processing.

- Fingerprints have a property of being unique and immutable, allowing for a rather small False Acceptance Rate (1%) and False Rejection Rate (0.1%).
- The extracted minutiae cannot be used to recreate the original fingerprint making it a one way procedure that increases security.

3.5.2 Use of Biometrics for Cryptographic Key Generation

Apart from the use for authentication and identification in our system, biometrics is also being used for the generation of cryptographic keys. These keys are used for signing contracts and verifying digital signatures during the contract signing protocol. Our system stores each user's template and their corresponding cryptographic key pair in a database. During user authentication once a user is identified, their corresponding keys are retrieved from the database and loaded in the software for contract signing purposes. These keys are generated by a cryptographic key generator that uses the user's template as a *seed*. A seed is a random value used by a pseudorandom number generator or a key generator for creating unique and unpredictable outputs or keys. Selection of a good random seed is very critical for the robustness of any security model. Templates solve this purpose efficiently since each template generated is unique.

3.6 Role of Cryptography

In our system, cryptography is being used for two main purposes that include creation and verification of digital signatures and implementation of hybrid cryptography. Following is a brief discussion on both these purposes and their implementation in our system.

3.6.1 Digital signatures

Digital signatures are derived from asymmetric cryptography where messages signed by a party using its private key can later by verified by anybody using the party's corresponding public key. This provides the property of non-repudiation where the signer cannot refuse to have signed the message since the private key used to sign the message is a secret known only to the

signer. Digital signatures can be considered equivalent to traditional handwritten signatures in many aspects and when implemented properly are extremely effective. During a contract signing instance, the initiating party can sign messages using any of the various digital signature algorithms. This way during the transaction it can always be proved that the messages signed and sent by the initiating party indeed belong to them.

We implement a digital signature algorithm based elliptic curves known as Elliptic Curve Digital Signature Algorithm or ECDSA [27]. The ECDSA API's used by our software were provided by a cryptography provider called *FlexiProvider*. FlexiProvider provides a powerful toolkit available for both Java Cryptography Architecture and Java Cryptography Extension. It provides various cryptographic modules that can be plugged into any application that is built on top of the Java Cryptography Architecture such as ours. Our implementation of ECDSA produces a digital signature of 448 bits [28] on the provided data in byte array format. Algorithm of ECDSA is available in Appendix A for further reading. Following are snippets of code that were used for creation and verification of digital signatures in our software.

Creation of Digital Signature:

```
public byte[] createDigitalSignature(byte[] byteArray) {
...
Signature signature = Signature.getInstance("ECDSA");
signature.initSign(userPrivKey);
signature.update(byteArray);
sigBytes = signature.sign();
...
return sigBytes;
}
```

The above method takes data as input in the form of byte array and produces a digital signature in the same format. Variable *userPrivKey* provides the private key of the signer that is used to sign the data provided by variable *byteArray*. Method call *sign()* performs the signature creation

operation and the newly created digital signature is stored in the variable *sigBytes*, and returned to the calling method in the final statement.

Verification of Digital Signature:

```
public boolean verifyDigitalSignature(byte[] toBeVerifiedBytes, byte[] sigBytes) {
    ...
        Signature signature = Signature.getInstance("ECDSA");
        signature.initVerify(receivedPublicKey);
        signature.update(toBeVerifiedBytes);
        if (signature.verify(sigBytes)) {
            aliceFingerPrintObject.writeLog("Signature Verification Succeeded.");
            return true;
        } else {
            aliceFingerPrintObject.writeLog("Signature Verification Failed.");
            return false;
        }
    ...
    return false;
}
```

This method verifies the digital signature signed by a party. It takes the byte array data that is to be verified and the digital signature bytes, performs the verification, and returns a Boolean value providing the verification results.

3.6.2 Hybrid cryptography

Our system uses hybrid cryptography for secure communication since it offers better efficiency and benefits like data integrity and non-repudiation when compared individually to techniques like asymmetric and symmetric cryptography.

We use Elliptic Curve Integrated Encryption Scheme, also known as ECIES for performing asymmetric cryptography operations. The ECIES API's used by our software are provided by the same Java cryptography provider called *FlexiProvider*. Algorithm of ECIES is available in Appendix A for further reading. We use Advanced Encryption Standard (AES) for symmetric cryptography. It is the current cryptography standard for symmetric cryptosystems and was announced by the National Institute of Standards and Technology (NIST) under the Federal

Information Processing Standards (FIPS) 197 [29] in November 26, 2001. It is also one of the most popular algorithms used for symmetric cryptography at present.

3.6.3 Elliptic Curve Cryptography Overview

Both Elliptic Curve Integrated Encryption Scheme (ECIES) and Elliptic Curve Digital Signature Algorithm (ECDSA) are part of cryptography that is based on elliptic curves. Also known as Elliptic Curve Cryptography or ECC, it is an approach to public key cryptography that is primarily based on elliptic curves which are defined over a finite field. A field is basically a mathematical group that offers operations for addition, subtraction, multiplication, and division that always construct results within the field. A finite field can be defined as a field that contains only finitely many elements. It is this property of being finite that makes it possible to perform cryptography with these elliptic curves that exists over the fields. The use of elliptic curves for cryptography was proposed independently by Neal Koblitz [30] at the University of Washington and Victor Miller [31] at IBM in 1985. Being grown into a mature public key cryptosystem system, it is also endorsed by the United States government [32].

The security of any cryptographic system is based on a hard mathematical problem that is computationally infeasible to solve. For example, RSA gets its security from the difficulty of factoring large numbers. The public and private keys used in RSA cryptography is a function of a pair of large prime numbers, and recovering the plain text from the cipher text that was created using the public key is believed to be computationally equivalent to finding the factors of the primes used to create the pair of public and private keys. Elliptic Curve Cryptography along with many other cryptographic systems achieves their security from the difficulty of solving the discrete logarithmic problem (DLP). ECC to be specific is based on the difficulty of solving Elliptic Curve Discrete Logarithm Problem (ECDLP) which offers a better implementation when compared to previous generation techniques as used by RSA. ECDLP can be demonstrated with the help of the equation $Ax = B$**.** For very large values of x, it gets computationally infeasible to

derive its value as no efficient algorithm is available for solving it. The primitive approach of solving this would be to keep adding and/or multiplying the value of A to itself until the result matches B. This approach is used on elliptic curve groups where, a point on the group is selected and multiplied by a scalar value. When the scalar value is very large, it becomes computationally infeasible to solve the problem. The primitive approach then becomes using the addition and doubling operations together until the matching value is observed. For example, *7P* can be expressed as *2* ((2 * P) + P)) + P*. This calculation of a point *nP* is referred to as Scalar Multiplication of a point. ECDLP is based on the intractability of scalar multiplication products.

Not all curves can provide strong security and that ECDLP for some curves can be resolved efficiently. Therefore NIST offers a set of recommended curves [32] whose security properties are well understood and can be safely used for cryptography. Standardization of elliptic curves also makes it convenient for interoperability and use by external third party cryptographic providers to offer cryptographic solutions that comply with the security standards.

Chapter 4

Analysis

4.1 Our Software

This section provides a detailed explanation of the software implementation for our research work. Our software is written in Java using NetBeans as the Integrated Development Environment (IDE). Biometrics part of our software was achieved using API's from *Griaule Biometrics*. This provided our software the ability to communicate with external USB fingerprint scanners and other features described later in this section. We use a Microsoft Fingerprint Reader to scan user fingers for fingerprints. Cryptography based on elliptic curves and AES was achieved using the external cryptography provider called *FlexiProvider*. Following is a screenshot that shows the main window of our application followed by the protocol execution and explanation:

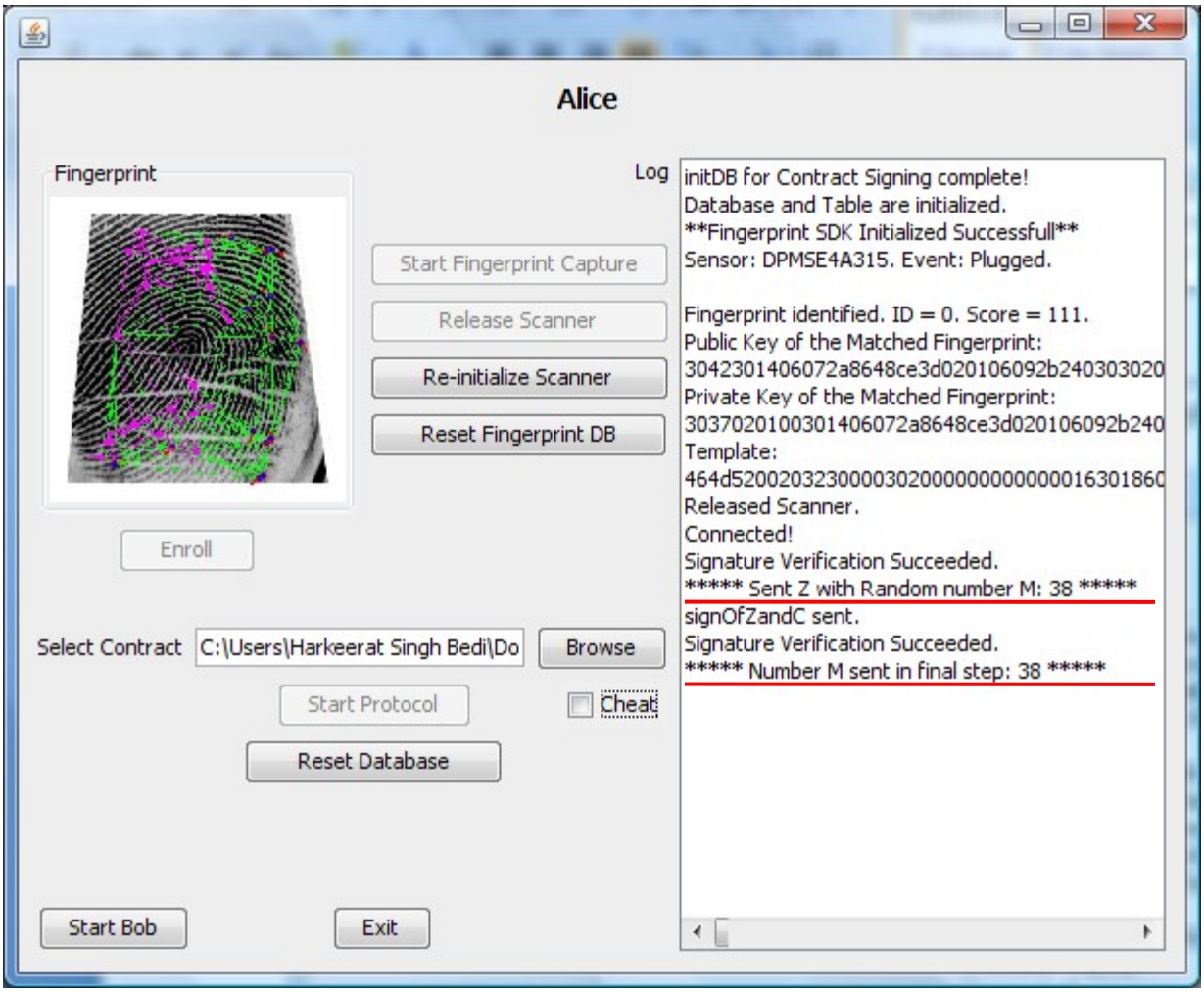

Figure 4: Contract Signing - Fair Execution

4.1.1 Fair Execution

This window is used by each party who wants to sign a contract electronically using our improved fair exchange protocol. The above window is of the client Alice who wants to sign a contract with Bob. When the application is first run, the Microsoft Fingerprint Reader along with the databases which store the templates and cryptographic keys are initialized. The user, which is Alice in this case, is then required to scan her finger over the fingerprint reader. Doing so, an image of her fingerprint scanned is displayed in the fingerprint section of our application window. Once the image is scanned, a new template is created based on the fingerprint and is compared with all the templates in the database to look for matches. The template matching algorithm is run

and if a match is found, the user, which in our case is Alice, is successfully identified. In the log window, we can see that the fingerprint was identified with the *ID* of *0* (zero). This *ID* value of zero is an integer that represents the first user enrolled in the database; the second user is represented by 1 and the third by 2 and so on. It also displays a score that shows the number of minutiae similarities identified between the template in question and the template stored in the database. Once identified, the cryptographic keys (both public key and private key) stored along with the matching template are retrieved. These retrieved keys are then used during the contract signing protocol. The log window shows the public and private keys retrieved from the database for quick reference along with the matching template in hexadecimal format.

During the initial user enrollment, users are required to scan their finger over the fingerprint reader. Once scanned, a template is created from the fingerprint image and is used as a *seed* for the cryptographic key generation algorithm. Since templates generated are always unique in nature, they provide as a good random seed value for such algorithms. The cryptographic keys generated, along with the template are stored in the database under a unique user identification number represented by the variable *ID* as discussed above.

Once the user is identified, the *Start Protocol* button is highlighted. Users can now select the file that they what to sign electronically using the *Browse* button. Once the file is selected, the user can press the *Start Protocol* button and perform contract signing with other users. Note: Both users who want to sign a contract together are required to be authenticated in their own application windows prior to execution of the protocol. Once the *Start Protocol* button is pressed, our fair exchange protocol is executed and messages are exchanged between both the participating parties. All messages exchanged are encrypted using a hybrid cryptosystem which provides efficient encryption and decryption along with properties like non-repudiation and data integrity. In the screenshot above, the two values underlined in red above show the secret value M sent by Alice to Bob during the Steps 3 and 5 of our protocol. For the contract signing to be

fair, Alice is required to send the same value of M to Bob in the final Step 5 so that Bob can recreate the information packet Z and obtain complete resolution of Alice over the contract.

4.1.2 Unfair Execution

To demonstrate the functionality of the third party which provides resolution if Alice cheats, a *Cheat* button is provided in our application that forces Alice to cheat by sending the wrong value of *M* in the final step 5. This way the other party, which is Bob in our case, is not able to recreate the packet *Z* with the received *M* and therefore does not have complete commitment of Alice over the contract. Following is a screenshot that demonstrates Alice cheating Bob and its explanation:

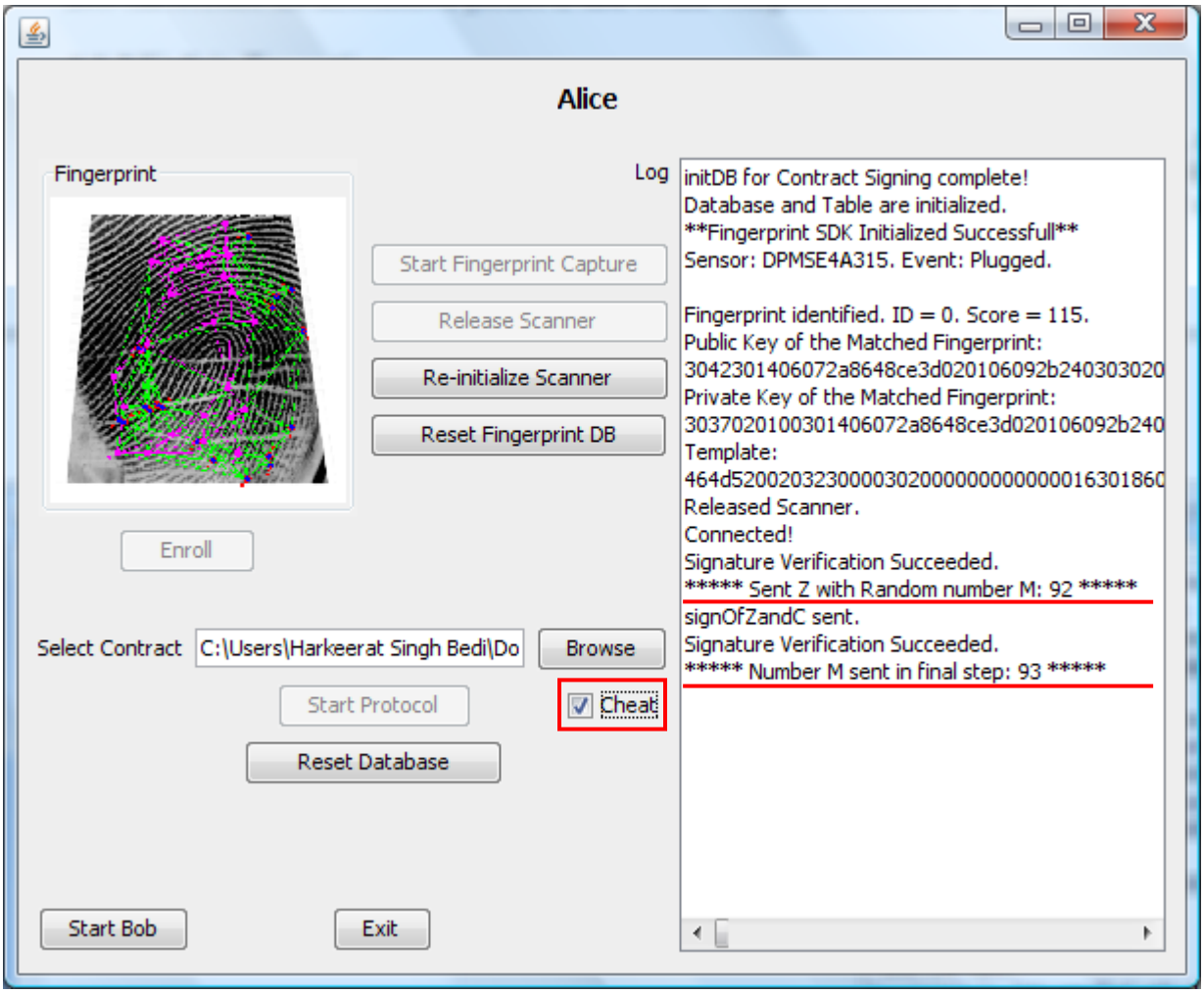

Figure 5: Contract Signing - Unfair Execution

The above screenshot shows the *Cheat* button as enabled. Doing so, forces Alice to cheat by sending the wrong value of *M*. In our software, Alice is forced to send a value that is one integer higher than the original value. In the example above the original value of *M* used for creating the packet *Z* was 92. In the final step Alice sends 93 to Bob. Therefore, Bob is not able to recreate the packet *Z* as he gets 93 and not 92.

4.1.3 Dispute Resolution

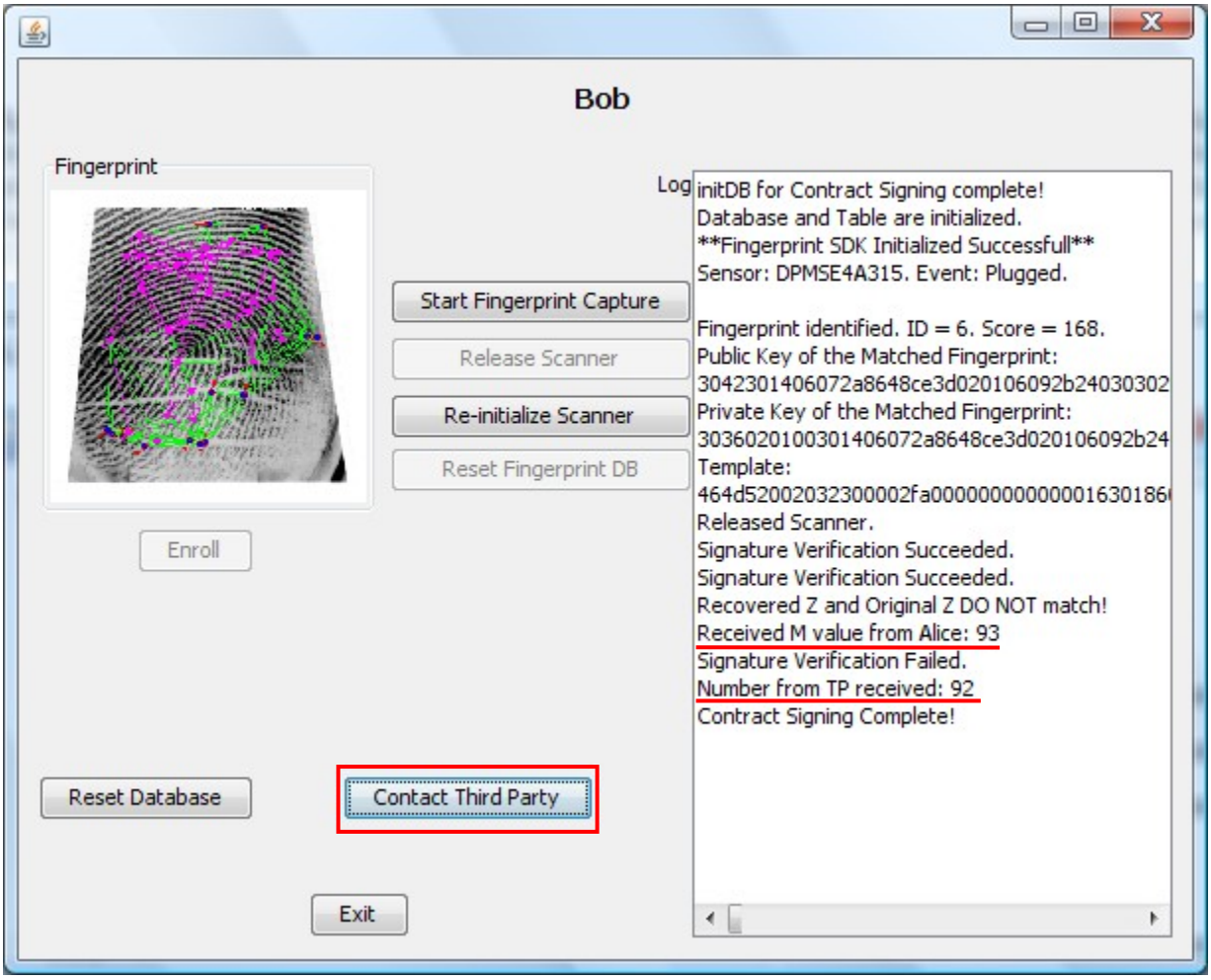

Figure 6: Contract Signing - Dispute Resolution

In the screenshot above, the first red underline shows the wrong value of *M* (93) received by Bob from Alice. Our software provides a provision for contacting the third party using the *Contact Third Party* button to obtain resolution. Once the *Contact Third Party* button is pressed,

Bob sends his and Alice's signatures on the contract along with the packet *Z* and the hash of the contract to the third party. On receiving, the third party verifies the signatures of both parties and if valid, decrypts the packet *Z* using its private key and extracts the original value *M* (92). This value of *M* is then sent to Bob, who on receiving the same tries to recreate the packet *Z*. Since it is the same value used by Alice, Bob is able recreate the packet *Z*. The second underline shows the value received from the third party which is 92. Thus, cheated Bob was able to obtain resolution from the third party that provided him with complete commitment from Alice.

4.2 Performance Improvements Using Elliptic Curve Cryptography

Majority of security systems still use first generation public key cryptographic algorithms like RSA and Diffie-Hellman (DH) which were developed in mid-1970. For these systems the current NIST recommended public key parameter size is 1024 bits. NIST states that these systems can be used securely till 2010 after which it is recommended to shift to other systems which provide better security. One alternative can be to keep increasing the bit size to higher values so that these systems can be used for some more time. Another option can be to shift to next generation cryptographic systems like elliptic curve cryptography which provide equivalent security for smaller key sizes and are also more efficient. Following is a table that compares ECC with schemes like RSA and Diffie-Hellman in terms of key sizes required for securing symmetric keys for varying length by NSA [33].

Table 1: NIST Recommended Key Sizes

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384

256	15360	521

We can see that NIST recommends 1024 bit key sizes for securing 80 bits symmetric keys. The same security can be provided by ECC by using 160 bit key size which makes ECC a better solution. Securing a 256 bit symmetric key would require a RSA key with the bit parameters of size 15,360 which is fifteen times the current size used in internet today. Comparing to ECC, one would only require keys of size 521 bits which is far smaller.

ECC is also more computationally efficient when compared to RSA and Diffie-Hellman. Even though ECC has more complex arithmetic than RSA and DH, the security added per bit increase in key size does make for the extra time used to handle such complexity. Following is a table that compares the computation required by ECC and schemes like Diffie-Hellman for varying key sizes by NSA [33].

Table 2: Relative Computation Costs of Diffie-Hellman and Elliptic Curves

Security Level (bits)	Ratio of DH Cost: EC Cost
80	3:1
112	6:1
128	10:1
192	32:1
256	64:1

We can see that as the security level based on key sizes is increased, the difference in the computation required increases at a high rate which makes ECC much more efficient than the first generation cryptographic algorithms. To further emphasize the benefits of using ECC, following is a snippet from a Microsoft Research paper that discusses the results obtained by them during their comparison between ECC and RSA [34]: “At the 163-bit ECC/1024-bit RSA security level, an elliptic curve exponentiation for general curves over arbitrary prime fields is roughly 5 to 15 times as fast as an RSA private key operation, depending on the platform and

optimizations. At the 256-bit ECC/3072-bit RSA security level the ratio has already increased to between 20 and 60, depending on optimizations. To secure a 256-bit AES key, ECC-521 can be expected to be on average 400 times faster than 15,360-bit RSA."

Due to the above mentioned reasons which include smaller key sizes, better computational efficiency and greater security; Elliptic Curve Cryptography can be considered as a better solution when compared to first generation techniques like RSA and DH. National Security Agency has also decided to move to Elliptic curve cryptography for protecting both classified and unclassified national security information [33].

Conclusion

In this research we discuss the concept of fair electronic exchange and its implementations. Fair exchange between two parties can be defined as an instance of exchange such that either both parties obtain what they expected or neither one does. Protocols that facilitate such transactions are known as “fair exchange protocols”. We analyze one such protocol by Micali that demonstrates fair contract signing, where two parties exchange their commitments over an already negotiated contract. The protocol claims fairness by providing resolution if a party refuses to provide their commitment. In this research we demonstrate that Micali’s protocol is not completely fair and show the possibilities for one party cheating and getting away with it. We discuss Bao’s protocol which is a revised version of Micali’s protocol that provides superior fairness by handling certain types of weakness. However both these protocols fail to handle the possibility of a replay attack. Our proposed protocol improves on Bao’s protocol by addressing the weakness that leads to a replay attack. We also demonstrate a software implementation of our system which provides fair contract signing along with properties like user authentication achieved through the use of a fingerprint based authentication system and features like confidentiality, data-integrity and non-repudiation achieved through the implementation of hybrid cryptography and digital signatures algorithms based on Elliptic Curve Cryptography.

Appendix A

ECIES Algorithm

Integrated Encryption Scheme (IES) is a public key encryption which provides semantic security against an adversary. For a cryptosystem to be semantically secure, it should be infeasible for a computationally-bounded adversary to derive relevant information about the plaintext when given only its ciphertext and the corresponding public encryption key. ECIES is one incarnation of IES that is standardized.

To send an encrypted message to Bob using ECIES, Alice needs the following information:

1. EC domain parameters: Includes variables (p, a, b, G, n, h) for a curve over prime field. The variables in brief are [24]:
    i. p: definition of prime field
    ii. a and b: constants used to define the elliptic curve
    iii. G: generator (base point) used to define the cyclic subgroup
    iv. n: order of G. It is the smallest non-negative number such that nG = O. Must be prime.
    v. h: called cofactor, this number at least must be small and , preferably h = 1.
2. Symmetric encryption scheme E, e.g., AES, Triple DES.
3. Key Derivation Function, e.g., ANSI-X9.63-KDF with SHA-1 option.
4. Message Authentication Code, e.g., HMAC-SHA-1
5. Optional shared information: s_1 and s_2

To encrypt a message m, Alice takes the following steps [25]:

1. generates a random number $r \in [1, n-1]$ and calculates R = rG;
2. derives a shared secret: S = P_x, where P = (P_x,P_y) = rK_B (and P ≠O);
3. uses KDF to derive a symmetric encryption and a MAC keys: $k_E \| k_M = \mathrm{KDF}(S \| S_1)$

4. encrypts the message: $c = E(k_E; m)$;
5. computes the tag of encrypted message and S2: $d = \mathrm{MAC}(k_M; c \| S_2)$;
6. Outputs R || c || d.

To decrypt the ciphertext R || c || d Bob does the following:

1. derives the shared secret: $S = P_x$, where $P = (P_x, P_y) = k_B R$, or outputs failed if P = O;
2. derives keys the same way as Alice did: $k_E \| k_M = \mathrm{KDF}(S \| S_1)$;
3. uses MAC to check the tag and outputs failed if $d \neq \mathrm{MAC}(k_M; c \| S_2)$;
4. Uses symmetric encryption scheme to decrypt the message $m = E^{-1}(k_E; c)$.

ECDSA Algorithm

Elliptic Curve Digital Signature Algorithm (ECDSA) is a variant of the Digital Signature Algorithm that operates on elliptic curve groups. Following are the algorithms for signature generation and verification:

Assuming Alice wants to sign a message for Bob. Initially, the curve parameters are agreed upon. Also, Alice must have a key pair suitable for elliptic curve cryptography.

Signature Generation:

For Alice to sign a message m, the following steps are followed:

1. Calculate e = HASH(m), where HASH is a cryptographic hash function, such as SHA-1.
2. Select a random integer k from [1, n − 1].
3. Calculate $r = x_1 \pmod n$, where $(x_1, y1) = kG$. If r = 0, go back to step 2.
4. Calculate $s = k^{-1}(e + rd_A) \pmod n$. If s = 0, go back to step 2.
5. The signature is the pair (r,s).

Signature Verification:

For Bob to verify Alice's signature, following steps are followed:

1. Verify that r and s are integers in [1, n − 1]. If not, the signature is invalid.
2. Calculate e = HASH (m), the HASH has to be the same function used in the signature generation.
3. Calculate $w = s^{-1} \pmod n$.
4. Calculate $u1 = ew \pmod n$ and $u2 = rw \pmod n$.
5. Calculate $(x_1, y_1) = u_1G + u_2Q_A$.
6. The signature is valid if $r = x_1 \pmod n$, invalid otherwise.

References

[1] B. Baum-Waidner. Optimistic asynchronous multi-party contract signing with reduced number of rounds. ICALP'01, LNCS 2076, pages 898-911, Springer, 2001.

[2] J. L. Ferrer-Gomila, M. Payeras-Capella, and L. Huguet-Rotger. Efficient optimistic n-party contract signing protocol. 2001 Information Security Conference, LNCS 2200, pages 394-407, Springer, 2001.

[3] J. Garay and P. MacKenzie. Abuse-free multi-party contract signing. 1999 International Symposium on Distributed Computing, LNCS 1693, pages 151-165, Springer, 1999.

[4] B. Cox, J. D. Tygar, and M. Sirbu. NetBill security and transaction protocol. In Proc. 1st USENIX Workshop on Electronic Commerce, pages 77–88, 1995.

[5] I. Ray and I. Ray. Fair Exchange in E-commerce. ACM SIGecom Exchange, Vol. 3, No. 2, May 2002, Pages 9-17.

[6] T. Tedrick, How to exchange half a bit, in: D. Chaum (Ed.), Advances in Cryptology: Proceedings of Crypto 83, Plenum Press, New York and London, 1984, 1983, pp. 147–151.

[7] T. Tedrick, Fair exchange of secrets, in: G. R. Blakley, D. C. Chaum (Eds.), Advances in Cryptology: Proceedings of Crypto 84, Vol. 196 of Lecture Notes in Computer Science, Springer-Verlag, 1985, pp. 434–438.

[8] M. Ben-Or, O. Goldreich, S. Micali, R. Rivest, A fair protocol for signing contracts, IEEE Transaction on Information Theory 36 (1) (1990) 40–46.

[9] O. Markowitch, Y. Roggeman, Probabilistic non-repudiation without trusted third party, in: Second Conference on Security in Communication Networks'99, Amalfi, Italy, 1999.

[10] T. Coffey, P. Saidha, Non-repudiation with mandatory proof of receipt, ACMCCR: Computer Communication Review 26.

[11] N. Zhang, Q. Shi, Achieving non-repudiation of receipt, The Computer Journal 39 (10) (1996) 844–853.

[12] J. Zhou, D. Gollmann, A fair non-repudiation protocol, in: IEEE Symposium on Security and Privacy, Research in Security and Privacy, IEEE Computer Society,Technical Committee on Security and Privacy, IEEE Computer Security Press, Oakland, CA, 1996, pp. 55–61.

[13] N. Asokan, M. Schunter, M. Waidner, Optimistic protocols for fair exchange, in: T. Matsumoto (Ed.), 4th ACM Conference on Computer and Communications Security, ACM Press, Zurich, Switzerland, 1997, pp. 6, 8–17.

[14] S. Micali, Certified E-mail with invisible post offices, Available from author; an invited presentation at the RSA '97 conference (1997).

[15] S. Kremer, O. Markowitch, Optimistic non-repudiable information exchange, in: J. Biemond (Ed.), 21st Symp. on Information Theory in the Benelux, Werkgemeenschap Informatie- en Communicatietheorie, Enschede (NL), Wassenaar (NL), 2000, pp. 139–146.

[16] J. Zhou, D. Gollmann, An efficient non-repudiation protocol, in: Proceedings of The 10th Computer Security Foundations Workshop, IEEE Computer Society Press, 1997, pp. 126–132.

[17] J. Zhou, R. Deng, F. Bao, Evolution of fair non-repudiation with TTP, in: ACISP: Information Security and Privacy: Australasian Conference, Vol. 1587 of Lecture Notes in Computer Science, Springer-Verlag, 1999, pp. 258–269.

[18] W.Diffie, M.E.Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (6) (1976) 644–654.

[19] ISO/IEC 13888-1, Information technology - Security techniques - Non-repudiation - Part 1: General (1997).

[20] ISO/IEC 13888-2, Information technology - Security techniques - Non-repudiation - Part 2: Mechanisms using symmetric techniques (1998).

[21] ISO/IEC 13888-3, Information technology - Security techniques - Non-repudiation - Part 3: Mechanisms using asymmetric techniques (1997).

[22] F Bao, G Wang, J Zhou, H Zhu. Analysis and Improvement of Micali's Fair Contract Signing Protocol, in: Book: Information Security and Privacy, Volume 3108/2004, Pages:176-187, 2004.

[23] S. Micali. Simple and Fast Optimistic Protocols for Fair Electronic Exchange. 2003 ACM Symposium on Principles of Distributed Computing, pages 12-19, 2003.

[24] S. Micali. Simultaneous Electronic Transactions. US Patent No. 5666420, September 1997.

[25] C. Watson, M. Garris, E. Tabassi, C. Wilson, R. McCabe, S. Janet and K. Ko. User's Guide to NIST Biometric Image Software. NIST. 2007.

[26] Pallav Gupta, Srivaths Ravi, Anand Raghunathan, Niraj K. Jha, "Efficient Fingerprint-based User Authentication for Embedded Systems". Annual ACM IEEE Design Automation Conference: Proceedings of the 42nd annual conference on Design automation; 13-17 June 2005. 2005

[27] ANSI X9.63, "Elliptic Curve Key Agreement and Key Transport Protocols", American Bankers Association, 1999.

[28] K. Piotrowski, P. Langendoerfer, S. Peter, How public key cryptography influences wireless sensor node lifetime, in: Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks, Pages: 169 – 176. 2006.

[29] Federal Information Processing Standards (FIPS) 197, November 2001, see: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[30] N. Koblitz, "Elliptic curve cryptosystems", Mathematics of Computation, 48:203-209, 1987.

[31] V. Miller, "Uses of elliptic curves in cryptography", Lecture Notes in Computer Science 218: Advances in Cryptology - CRYPTO '85, pages 417-426, Springer-Verlag, Berlin, 1986.

[32] NIST, "Recommended Elliptic Curves for Federal Government Use", July 1999, see http://csrc.nist.gov/csrc/fedstandards.html

[33] NSA: The Case for Elliptic Curve Cryptography. 2009, see:

http://www.nsa.gov/business/programs/elliptic_curve.shtml

[34] K. Lauter. Microsoft Research. IEEE Wireless Communications. Topics in Wireless Security. February 2004.

[35] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In Proc. IEEE Symposium on Research in Security and Privacy, pages 86–99, 1998.

[36] V. Shmatikov, J.C. Mitchell. Analysis of a Fair Exchange Protocol. In Proc. of 7th Annual Symposium on Network and Distributed System Security (NDSS 2000), pages 119—128, 2000.