

Master of Science Thesis¹

A Signing Protocol for Service Level Agreements in Virtual Organizations

By

Rabih Ghannoum

Olga Cerrato

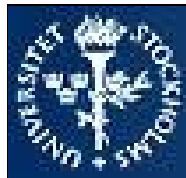
Supervisor

Gustav Bostrom(DSV,SICS)

Pablo Giambiagi(SICS)

Examiner

Louise Yngstrom



Department of Computer and Systems Sciences
Royal Institute of Technology/Stockholm's University
2006

¹The thesis corresponds to 20 weeks of full-time work for each of the authors.

Abstract

Electronic contract signing is one of the building blocks that makes formation and operation of virtual organizations possible. While contracts in the physical world are typically signed face-to-face, in the electronic world the signatories may be miles away and have probably never seen each other. A fair contract signing protocol allows two or more parties to exchange signatures on the pre-agreed contract in such a way that either each party gets each other's signature or none of them does. Fair protocols may involve a trusted third party (TTP) – an external party that adds to the trustworthiness of the exchange procedure. However, more the TTP is involved, more costly the protocol becomes.

TrustCoM project is currently developing a framework for trust, security and contract management for dynamic virtual organizations (VOs). One of the functions TrustCoM supports is the negotiation, creation and signing of service level agreements (SLAs) between VO partners. An SLA is a type of a contract.

In this thesis we are presenting various contract signing protocols and classify them according to the protocol properties. The objective is to choose a contract signing protocol from the pool of surveyed protocol that proves to be the most suitable one for the TrustCoM environment, and develop a prototype implementation of the chosen protocol.

Acknowledgements

We would like to thank our supervisors Pablo Giambiagi at the Swedish Institute of Computer Science (SICS) and Gustav Boström from SICS/KTH for giving us the opportunity to write a thesis for TrustCoM project, for their support and encouragement. We appreciate their patience and the time they have given to read and correct our work. It was a great experience to work in the environment of SICS!

We are very grateful to our professor Louise Yngström (KTH/DSV) who took her time to be our examiner and who helped us to come in contact with SICS.

Olga Cerrato and Rabih Ghannoum

I would like to thank my parents who have always encouraged me to pursue studies despite all obstacles encountered on the way. I would like to thank my thesis partner Rabih Ghannoum who was there during the whole process of writing, re-writing, accepting, rejecting, and discussing, discussing, discussing. I would like to thank my dear husband who, despite my complains, scientific talks and absent-mindedness in conversations, has been patiently supporting and encouraging me along the way.

Olga Cerrato

First and foremost, I thank God for everything.

My deepest gratitude to my parents for their non-stop support, love and care through out my education.

I am grateful to my thesis partner Olga Cerrato, my colleagues and friends, Maria and Saykibea; working together has been an inspiring, often exciting, sometimes challenging, but always interesting experience.

I would like to thank ALL my friends (without mentioning names) for their support and encouragement during my stay in Sweden – I will always remember you all!

Rabih Ghannoum

TABLE OF CONTENTS

ABSTRACT	I
ACKNOWLEDGEMENTS	II
LIST OF FIGURES	V
LIST OF TABLES	VI
1 INTRODUCTION	1
1.1 BACKGROUND.....	2
1.2 PROBLEM	4
1.3 OBJECTIVE/GOAL	4
1.4 AUDIENCE.....	5
1.5 METHOD	5
1.6 LIMITATIONS.....	8
2 FRAME OF REFERENCE	9
2.1 SERVICE-ORIENTED ARCHITECTURE.....	10
2.1.1 <i>Web Services</i>	12
2.2 SERVICE LEVEL AGREEMENTS (SLA)	13
2.3 VIRTUAL ORGANIZATIONS.....	15
2.3.1 <i>Characteristics of a VO</i>	17
2.3.2 <i>VO lifecycle</i>	18
2.3.3 <i>VO examples</i>	19
2.3.4 <i>Benefits and Challenges of VOs</i>	20
2.4 INTRODUCTION TO TRUSTCOM FRAMEWORK.....	23
2.4.1 <i>SLA Management Subsystem</i>	24
2.4.2 <i>SLA Signing – TrustCoM Model</i>	26
2.5 SECURITY REQUIREMENT ENGINEERING.....	27
2.5.1 <i>XP Extension for Security Requirements Engineering</i>	27
2.5.2 <i>CORAS Risk Assessment Methodology</i>	28
3 DERIVING SECURITY REQUIREMENTS	31
3.1 CONTEXT IDENTIFICATION.....	32
3.2 SYSTEM USER STORIES	33
3.3 ASSETS IDENTIFICATION	33
3.4 ABUSER STORIES IDENTIFICATION	34
3.5 ABUSER STORIES RISK ASSESSMENT – CORAS	36
3.6 SECURITY-RELATED USER STORIES.....	39
3.7 MAPPING ABUSER STORIES TO SECURITY-RELATED USER STORIES	40
3.8 CONCLUDED SIGNING PROTOCOL REQUIREMENTS	41
4 CONTRACT SIGNING PROTOCOLS	43
4.1 WHAT IS ELECTRONIC CONTRACT SIGNING?.....	44
4.2 CHALLENGES ASSOCIATED WITH ELECTRONIC SIGNING	45
4.3 PROPERTIES OF CONTRACT SIGNING PROTOCOLS	46
5 CONTRACT SIGNING PROTOCOL SURVEY	53
5.1 CONTRACT SIGNING PROTOCOLS WITH NO TRUSTED THIRD PARTY	54
5.1.1 <i>Even (1982)</i>	55
5.1.2 <i>Markowitch- Roggeman (1999)</i>	56
5.1.3 <i>Mitsianis (2001)</i>	59
5.2 CONTRACT SIGNING PROTOCOLS WITH A TRUSTED THIRD PARTY.....	62
5.3 PROTOCOLS WITH AN INLINE TTP.....	62
5.3.1 <i>Coffey and Saidha protocol (1996)</i>	62
5.3.2 <i>An Optimal Asynchronous Non-Optimistic Scheme (1998)</i>	63
5.4 PROTOCOLS WITH AN ONLINE TTP	65
5.4.1 <i>Rabin’s Random Beacons (1983)</i>	65
5.4.2 <i>Zhang and Shi protocol (1996)</i>	67
5.4.3 <i>Zhou and Gollmann protocol 1 (1996)</i>	68

5.4.4	<i>Zhou and Gollmann protocol 2 (1996)</i>	69
5.5	PROTOCOLS WITH AN OFFLINE/OPTIMISTIC TTP (INCLUDING TRANSPARENT TTP)	71
5.5.1	<i>Fair Non-Repudiation Protocol Respecting Timeliness</i>	72
5.5.2	<i>Non-Repudiation Protocol with Transparent TTP (based on Markowitch-Kremer protocol)</i>	75
5.5.3	<i>Optimistic Fair Contract Signing Protocol (Based on the “Optimistic Fair Exchange of Digital Signature” by N. Asokan, V. Shoup, M. Waidner)</i>	78
5.5.4	<i>A Time-Optimal Asynchronous Scheme</i>	81
5.6	PROTOCOL MATRIX	84
6	THE CHOICE	89
6.1	THE PROTOCOL MODIFICATIONS	93
7	CONCLUSION	99
	REFERENCES	101
	APPENDIX A - IMPLEMENTATION	104
	CLASSES:	104
	<i>Contract</i> :	104
	<i>alice</i>	104
	<i>bob</i> :	105
	<i>TTP</i> :	105
	<i>GenEnvelopedSig()</i> :	106
	<i>ValidateSig()</i> :	107
	<i>ComapreContracts()</i> :	108
	UML DIAGRAMS:	108
	SOURCE PSEUDO-CODE	109
	SOURCE CODE FOR THE SIGNING, SIGNATURE VALIDATION, AND CONTRACT COMPARISON CLASSES.....	111

List of Figures

Figure 1: Method Overview	7
Figure 2: Service Model [6]	11
Figure 3: Service-Oriented Architecture Operations	12
Figure 4: The TeleManagement Forum SLA Life Cycle [1]	15
Figure 5: VO lifecycle [41, 48].....	18
Figure 6: SLA Management Subsystem Interactions [34].....	25
Figure 7: SLA Signing and Storage[34].....	26
Figure 8: CORAS Methodology[43]	29
Figure 9: System User Stories and Assets.....	33
Figure 10: Primitive Contract Signing Model	35
Figure 11: Mitisianis Protocol.....	61
Figure 12: Behavior of the Optimal Asynchronous Scheme with inline TTP	64
Figure 13: Rabin's Beacon.....	66
Figure 14: Overview of Optimistic Fair Exchange.....	79
Figure 15: Recovery Subprotocol.....	80
Figure 16: Abort Subprotocol.....	80
Figure 17: Optimistic Behavior of the Time-Optimal Asynchronous Protocol	82
Figure 18: Main sub protocol (initial).....	94
Figure 19: Recovery sub protocol (initial)	94
Figure 20: Abort sub protocol (initial).....	94
Figure 21: Recovery sub-protocol (improved)	95
Figure 22: Abort sub protocol (improved)	96
Figure 23: A's State Machine Diagram.....	97
Figure 24: B's State machine Diagram.....	98
Figure 25: TTP's machine Diagram.....	98
Figure 26: An Honest Execution Sequence Diagram for the chosen protocol.....	108
Figure 27: Four Dishonest Scenarios of Protocol Execution	109

List of Tables

Table 1: Assets Identified.....	34
Table 2: Abuser Stories.....	36
Table 3 : CORAS Impact Scale.....	37
Table 4: CORAS Probability Scale	37
Table 5: Risk Level Matrix	37
Table 6: Abuser Stories Risk Assessment.....	38
Table 7: Risk Actions Criteria.....	39
Table 8: Security-related User Stories	40
Table 9: User - Abuser Stories Mapping.....	40
Table 10: Abuser Stories-Security-Related User Stories Mapping	42
Table 11: Protocol Survey Matrix	88

1 Introduction

At a Glance:

This thesis work is about finding a suitable contract signing protocol to be implemented in a highly dynamic virtual organization environment. In this thesis contract signing protocols will be surveyed to choose a suitable one for TrustCoM¹. In the introduction chapter we present the background of the area of electronic contract signing, the electronic contract signing problem, the goal and purpose of this thesis and its limitations (scope). We describe the path we follow to arrive at our ultimate result in the method section.

1.1 Background

At present we are clearly heading into the era of e-business [22]. E-transactions have laid the ground for B2B (business-to-business) and B2C (business-to-consumer) electronic commerce and their scope in terms of application fields is growing constantly [21, 23, 26, 28]. As these opportunities have arisen the way of conducting business has changed too. Due to the general complexity of required products and services, rapidly changing customer demands and the competition challenges presented by their markets, smaller and middle-sized enterprises choose to unite and share their expertise, resources, experiences and capabilities to survive the battle [33]. Each enterprise in such alliances then concentrates on a specific part of the end-service/product production instead of the whole business process; e. g. core business processes are kept in-house, while marginal processes are outsourced for the sake of better competitiveness and higher product quality. With the aid of information technology such alliances can be constituted not only from parties that are close geographically but also from those who reside on different continents. Enterprises unite and cooperate *virtually* by building virtual organizations (VO). VOs are also called virtual corporations and virtual enterprises. Moreover, thanks to increasing specialization in industry as well as constantly changing customer needs, and thus offered services, participation in VOs has become more dynamic in nature. During the life-time of a VO new partners are accepted and some leave – all according to the business needs at the particular moment. A more detailed overview on VOs is given in chapter 2.3.

Different technological advances support formation of on-demand business environments as VOs. The major one is service-oriented architecture (SOA) that facilitates communication by offering loosely coupled highly interoperable application services. Such services are able to work together independently of the platform or programming language, and are solely dependent on the formal definition of an interface. Plenty of technologies and tools are involved in SOA including web-service related standards like XML, SOAP, UDDI, WSDL. (See chapter 2.1 for a more detailed introduction to SOA). Grid computing has also contributed to the development of VOs [44]. The main idea behind grid computing is to use many separate computers connected by a network to solve one large problem.

What such technologies as SOA and grid computing imply from a security perspective is that applications are “run on virtual, shared infrastructure, using physical resources that might be spread all over the world” and back-end servers are “directly exposed to the outside, as they offer services to many enterprises” [40]. This imposes security and privacy challenges from which *trust*

establishment, maintenance and justification are by far the most outstanding [34, 35, and 36]. Trust is about how to rely on a virtual partner's identity, reputation, technological infrastructure, liability etc. It is argued that the lack of trust will prevent the widespread adoption of ICT (Information and Communication Technology) [41]. That is why "the capability to represent, create, negotiate, monitor and evolve trust relationships in a secure way becomes mandatory" [41].

A way of providing trust and minimizing other security risks is through service-level agreements between partners. A service-level agreement is a contract that defines the level of services a service provider promises to a service consumer. Important elements in an SLA are the quality of the service (QoS) parameters, the obligations and the penalties imposed on either party in case of non-compliance [1]. SLAs are negotiated in a semi-automated manner and then signed. See chapter 2.2 for more information on SLAs.

SLA² signing constitutes the core of this thesis. By signing we mean the sequence of transactions two VO partners need to execute in order to exchange electronic signatures on the previously agreed SLA. These signatures as well as the action of signing cannot be repudiated later.

Electronic contract signing is a metaphor taken from the physical world. However in the paper-based scenario contract signing is simple due to the existence of "simultaneity", meaning that both signatories sign the contract simultaneously [25]. That is, contracting parties can sign the document at the same time and place, and exchange the signed copies simultaneously with or without the presence of a notary. If one party later denies the fact of signing and is not abiding to the terms of the contract, the signed copy is used as evidence. Forging a signature on a pre-agreed contract face-to-face in a physical world is a difficult matter, because it is obvious that a false person produces it.

Mimicking the same context in the electronic world is complicated. The biggest challenge is to reproduce "simultaneity". In its physical sense it is, in fact, impossible. Instead "simultaneity" is achieved through the notion of *fairness*. Fairness implies that at the end of the signing process either both parties have the counterpart's signature or none of them does [23]. Since information in computer networks is exchanged non-simultaneously, an unfair state during signing will inevitably occur. The mission of a fair contract-signing protocol is to make it impossible to cheat despite this fact.

²The terms 'SLA' and 'contract' are used interchangeably in this thesis.

There are various signing protocols that guarantee fairness, either with a trusted third party (TTP) or without it. All protocols have their advantages and disadvantages depending what contexts they are to be used in. The most important thing is that the protocol and the application requirements match [24].

1.2 Problem

The main idea behind this thesis is to investigate which contract signing protocol is the most suitable solution for signing SLA agreements in VOs. SLA agreements in VOs constitute a specific environment with its requirements. VO members should sign a general virtual organization agreement (GVOA) upon joining the VO. After signing the GVOA members are legitimate to engage in dual (two parties) sub-agreements, i.e. SLAs. This implies that some level of trust might have been established already though the GVOA and signing protocols that guarantee the most rigorous security levels might be unnecessary. Or, put it another way, a protocol that provides the optimal security might be unnecessary.

Our task is to outline what properties a contract signing protocol should possess in a VO environment to carry out SLA signing. A signed SLA hinders the possibility that any party breaks its promises. At the same time the chosen signing protocol should be implementation-feasible and cost-effective. To narrow down the scope as well as to be more specific about context and requirements gathering procedure, we will use TrustCoM framework to illustrate this process³. See chapter 2.4 for more information about TrustCoM.

1.3 Objective/goal

The objective of this thesis work is to choose and implement a prototype of the contract signing protocol that is most suitable for SLAs in dynamic VOs operating in an open network, i.e. Internet. The derivation of the desired protocol properties is done on the basis of requirements gathered from the TrustCoM framework and on the basis of the contract signing protocol properties obtained from the surveyed protocols.

³<http://www.eu-trustcom.com>

1.4 Audience

As the result of the work is a signing protocol for SLAs used in the context of VOs the beneficiaries of this thesis are the developers of similar projects to TrustCoM, those who use the TrustCoM framework or similar frameworks for VO development and those who research on relevant technologies. In addition readers who are interested in contract signing for other reasons might appreciate the general overview of contract signing protocols whose properties are summarized in a protocol-property matrix.

1.5 Method

Since the goal of this thesis is the choice and implementation of an SLA signing protocol, it is an example of an *artifact development*⁴ work. At the logical level the research method is *inductive*⁵, because we are essentially digging into the realm of the unknown without any predefined idea of what the solution might be. In other words, we have no hypothesis of what protocol is the best one in our specific setting. Our approach is to explore the situation by gathering and analyzing relevant data. Data about protocols, protocol properties, application requirements, and implementation guidelines is gathered through literature surveys firstly and unstructured interviews with the developers involved in TrustCoM project secondly. These data are analyzed qualitatively and presented through descriptions or matrices.

A more detailed description of our method is presented below. We also refer to Figure 1 for a schematic overview.

The first step is to understand the context (VOs) for which we have to choose a protocol. This should result in the list of security requirements that are relevant to signing SLAs. As mentioned earlier, we will use TrustCoM documentation to facilitate context comprehension and exemplify it [42 and 43 in particular]. Therefore, our work is also a case study of TrustCoM framework in which the SLA signing is included. Since signing is a matter relevant to security, risk analysis will be used to come up with a requirements list. We will proceed by identifying threats that may influence a fair signing exchange and propose actions to counter those threats. Threat identification will be done

⁴Artifact development is a piece of work in which a new artifact is developed/proposed as a result of this work. An artifact may be a system, program, algorithm, model, method, etc.

⁵Inductive scientific research method (as opposed to the deductive one) requires reasoning without a predefined hypothesis. The premises of an argument support the conclusion but do not ensure it. It is used in situations where it is required to find out something previously unknown and when a researcher is unable to make any suppositions about the solution.

using the Extended XP Practices for Security Requirements Engineering. On the basis of such counter measuring actions, SLA signing security requirements will be arrived at. These requirements will be expressed in terms that can be mapped to the generic properties of contract signing protocols. Risk analysis activity will follow the method described in [44]. Some SLA signing requirements may be arrived at through the general literature review done on virtual organizations.

The second step is to get an overview of existing fair contract signing protocols in order to identify their generic properties. This is achieved by conducting a literature survey. After a considerable number of scientific articles and other material (like relevant chapters from books) have been read, the partial result of it is presented in a “protocol name-property” matrix. This demonstrates what properties each protocol provides. The result is partial because the end number of protocols is too large to be covered in the scope of this thesis.

In step three the requirement list for SLA signing is mapped to the list of properties produced after step two. Such mapping may either result in a choice of an existing contract signing protocol, some requirements-protocol trade-off, or an improvement of one of the presented protocols.

In step four the protocol prototype is implemented. The prototype was implemented using the Java language with Apache XML security package.

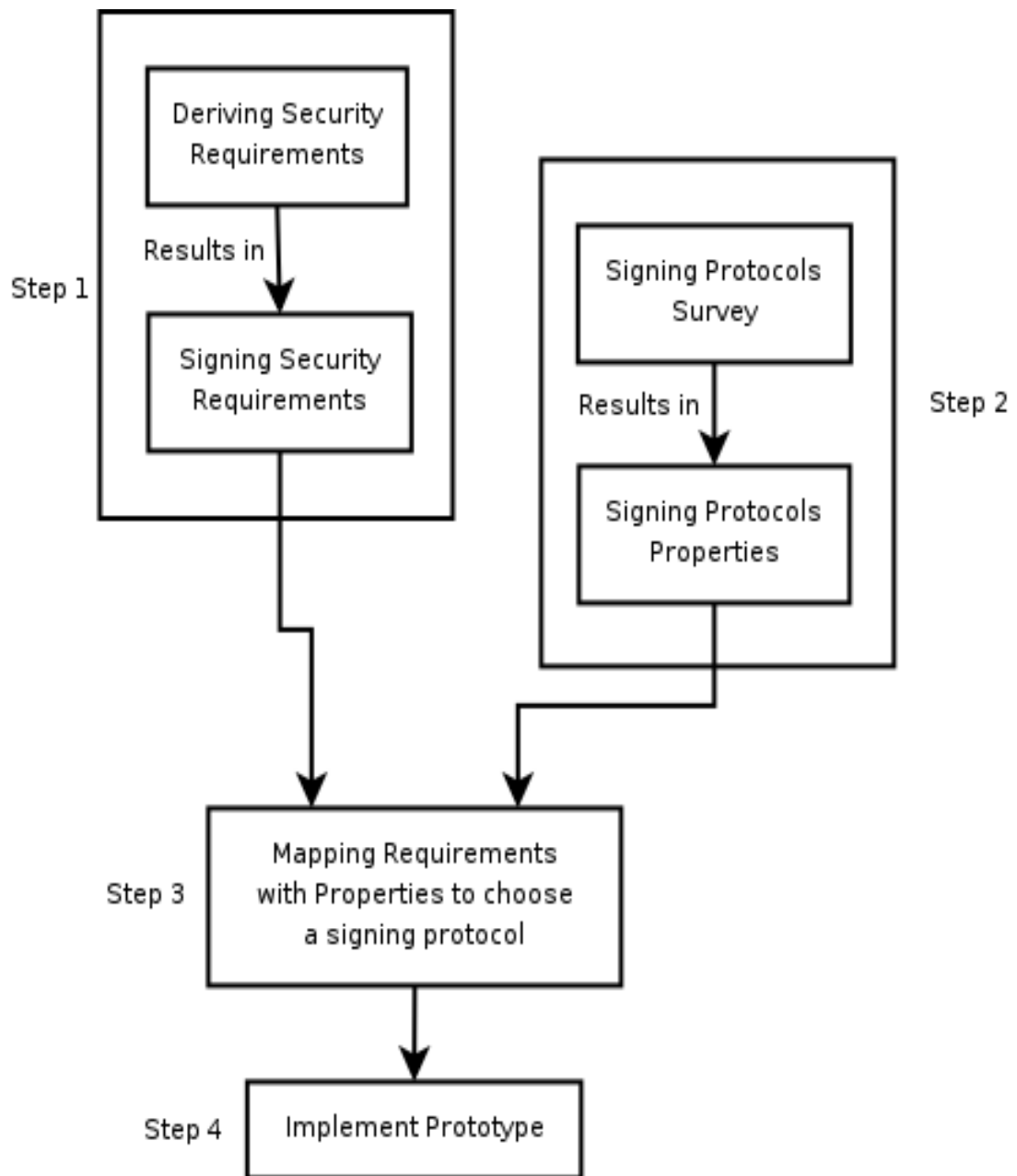


Figure 1: Method Overview

1.6 Limitations

This thesis solely covers contract signing protocols and not other types of signing like CEM (certified electronic mail) or electronic purchase. Moreover not all contract signing protocols that exist are presented but only representative ones for each outlined category. All contract signing protocols described here are fair. We have not taken into account any contract signing protocol or electronic signature directives. Readers interested in the EU directive on electronic signatures are referred to [32].

The solution offered here might not be applicable to all types of VOs since we are using only one VO framework. VOs may have different requirements and settings. However the process of arrival at the final choice and information on protocols may be useful for developers dealing with contract signing.

2 Frame of Reference

At a Glance:

In this chapter we present an extended background for our thesis work. The major concepts of our thesis are described here. The following is covered:

- 1- Service-Oriented Architecture (SOA)
- 2- Service Level Agreements (SLAs)
- 3- Virtual Organizations (VOs)
- 4- Introduction to TrustCoM Environment and its SLA Signing Subsystem
- 5- Security Requirement Engineering (Extended XP Practices)

2.1 Service-Oriented Architecture

Service-Oriented Architecture is a component model whose goal is to achieve loose coupling⁶ between interacting software agents. This architecture consists of different functional units called “services”. A loosely coupled system has the benefit of being agile and able to survive noticeable changes in the service's internal structure and its implementation. This feature makes SOA very convenient for business dynamic environment and on-demand businesses, and thus for TrustCoM where virtual organizations (discussed later) are to be dynamic, created on-the-fly, and can live for one or many transactions.

Achieving loose coupling in SOA is done through two architectural constraints:

1. A small set of simple interfaces to all participating software agents (services); these interfaces are defined in a generic way to provide universality.
2. Extensible and descriptive messages allow no or minimal system-specific behavioral prescribing, limiting the structure of the messages and defining new versions by using its extensibility property.

SOA also provides discovery services that help to discover available services and their description in a system. Other constraints are also applied to SOA to provide better scalability, performance and reliability [6]. For the constraints previously mentioned two types of services exist, stateless and state-keeping services. In stateless services all necessary information is included in the message, so the service provider does not have to store state information. Stateless services improve scalability and visibility since monitoring is easier. Only one request and no intermediate results to track are required which makes recovery simpler.

State-keeping services are used when a session is to be created for efficiency reasons. State-keeping services necessitate storing some consumer-specific information, which could affect the scalability property.

SOA is enabled and defined by WSA (Web Services Architecture). According to W3C [12], the Web services architecture is “*interoperability architecture: it identifies those global elements of the global Web services network that are required in order to ensure interoperability between Web services*”. In WSA services interact by using XML (Extensible Markup Language) messages. WSA is mainly characterized by open standards support [10], XML-technologies, e.g. XML Information

⁶Loose coupling is the concept by which the dependency of interacting software components is minimized as much as possible to achieve interoperability and flexibility of implementation.

Model [7], XML Base, and XML schema [8]. WSA also provides independence of web service implementation and the underlying transport protocols (e.g. HTTP and SMTP). The usage of XML technologies enables WSA to use SOAP message (Service-Oriented Architecture Protocol messages) [9], very extensible and interoperable XML-messages. In addition WSA enables publishing (using WSDL - Web Services Description Language) and discovery functionalities (discussed later) to enhance the architecture with transaction and workflow management.

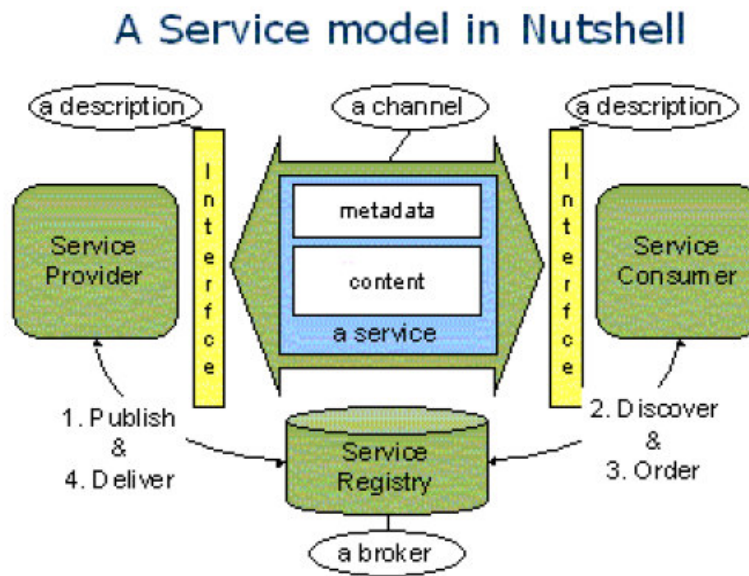


Figure 2: Service Model [6]

Figure 2 shows a service model and consists of:

Service - a software entity designed to support the characteristics of WSA mentioned above. It is a software entity that has an interface(s)⁷. It uses SOAP messages (maybe asynchronously also) in its interactions. A service is an abstraction made concrete by developing a software agent.

Service Provider – is a software entity that implements a service specification or a service description.

Service Consumer – also called a requestor, is the software entity client that calls a service provider to use a service. This can be a user application or another service.

Service Registry – is a software entity that acts as a service locator. It implements discovery and order functions for the requestor for a specific service. Here new services are published and delivered.

Service Broker – is a special service that can pass service requests to other service providers.

⁷An interface can be common to many services and one service can have many interfaces (many-to-many relation).

The service provider, consumer (requestor), and registry are the main roles in an SOA.

In addition SOA implements three operations that define the contracts between roles [11].

1. *Publish*: Is an operation that acts as service registration or advertisement. It is operated between the service registry and service provider.
2. *Find (Discover)*: Is the complementary operation of *Bind* because services are published so that they can be found. It is the contract between a service requestor and a service registry. Find operation is executed on the registry according to the search criteria specified by the requestor. Search criteria might be the type of service, QoS (Quality of Service), etc.
3. *Bind*: This operation binds both the service provider and the requestor in a client-server like relationship. This relationship can be dynamic or static. Dynamic such as dynamic generation of client-side proxy. If static, the developer hand-codes the way of invoking the service to the client.

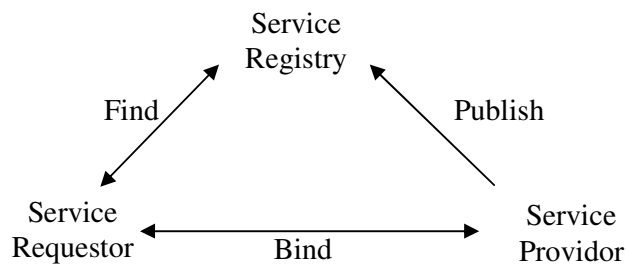


Figure 3: Service-Oriented Architecture Operations

2.1.1 Web Services

Although web services and SOA are related, in concept they are distinct. SOA and web services are commonly viewed together. SOA is an architectural concept that focuses on building loosely coupled sets (pools) of dynamic components. Web services are only a way to build or implement SOA.

In W3C Working Group Note on Web Services Architecture a web service is defined as the following: [12]

“[Definition: A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner

prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.]”

A web service is an abstraction that is to be implemented through a concrete hardware or software code that sends and receives messages, called an agent. A web service is a set of abstract functionalities that characterizes a resource. A web service can be implemented with one or more agents (e.g. different programming languages); these are usually described in WSDL. Web service's *semantics* (the “WHAT”) describes what the expected behavior of a certain web service is. Web service's *mechanics* (the “HOW”) is a description of how the web service is to achieve what is expected. Semantics and mechanics are important concepts of a service. For example, assume a service that delivers weather news about some city. The semantics of this service will be like the required input parameters (e.g. city name), the output (e.g. temperature), the service server address (e.g. www.weather.com), etc. The mechanics of the service is the implementation of the service at the service server using any programming language.

As a framework for providing trust, security, and contract management in dynamic virtual organizations TrustCoM is assumed to operate in an open service environment. TrustCoM architecture takes on some constraints [35]. These constraints include SOA in addition to the adherence to the ‘loose coupling’ concept, EDA (Event-driven Architecture), compliance with open standards, interoperability. SOA delivers much to such constraints (compliance, interoperability, and asynchronous transactions – EDA).

2.2 Service Level Agreements (SLA)

As mentioned in the introduction chapter, the emergence of grid computing and Service-Oriented Architectures have led to evolution in terms of how applications are built and managed. Services provide d in some department in Europe can be accessed from different places in the world. This new perspective of how things are done needs a new way of management. Such management involves constraints and parameters the service, service provider, and service consumer (a customer) are to adhere to. When a customer finds a service that he wants to use he engages in a negotiation phase with various available service providers. An agreement is then achieved and the customer and the service provider sign it. This agreement is called Service Level Agreement (SLA). The TeleManagement Forum [1] defines an SLA as “*a formal negotiated agreement between two parties, sometimes called a service level guarantee. It is a contract (or part of one) that exists*

between the service provider and the customer, designed to create a common understanding about services, priorities, responsibilities, etc...” So an SLA promises what is possible to deliver and delivers what is had promised.

SLAs define a service provider vs. service customer relation in a formal way [2]. The issues mentioned in an SLA define levels that a service can meet. The levels of a service are expressed in terms of parameters and metrics that specify which penalties and liabilities are to be enforced by an independent authority on a violator. The parameters and metrics defined in an SLA specify QoS (Quality of Service) of the service. QoS parameters include response time, availability, throughput, latency. According to the International Telecommunication Union (ITU), Quality of Service is defined as *“the collective effect of service performances, which determine the degree of satisfaction of a user of the service. The quality of service is characterized by the combined aspects of service support performance, service operability performance, service integrity, and other factors specific to each service”* [3]. A well-defined SLA typically consists of a number of components [2].

- *A Description of the provided service nature:* includes the type of the service and the description of the technical issues associated with the service such as network connectivity, operation and maintenance and in addition the server's and client's configurations.
- *The level of responsiveness and reliability of the service:* includes availability requirements and how soon the service performs in a normal state.
- *Service problems reporting:* includes who is to be contacted in case of a certain problem, and steps and formalities that must be followed to guarantee quick problem resolution.
- *Problem response time and resolution:* defines the time after which a reported problem will be solved.
- *Monitoring and service reporting:* defines how quality levels are monitored and reported, who is responsible of that, how and how often reporting is done.
- *Liabilities on the service provider if promises were not met:* in such cases a service customer may be given extra credits, for instance, the customer may gain the power to terminate the contract or ask for refunding.
- *Extra Conditions:* these are conditions upon which the SLA is not valid anymore. For example, it can occur in case of natural disasters (e.g. flooding, fire,). It may also involve the customer trying to breach the security of the network or the provided service.
- According to W. Sun et al.[4], an SLA life cycle consists of five phases, which are shown in figure 4.



Figure 4: The TeleManagement Forum SLA Life Cycle [1]

1. *SLA Development*: In this phase the SLA templates are developed.
2. *Negotiation and Sales*: In this phase the SLA is negotiated and the contracts are concluded.
3. *Implementation*: The SLA is generated.
4. *Execution*: The SLA is executed, monitored, and maintained.
5. *Assessment*: Evaluation of the SLA's performance. In this phase a re-evaluation of the initial SLA template might be performed.

The on-line learning community *nextslm.org* [5, p.2] provides clear explanations about SLA's and service level management issues and practices. It also has some templates for service level agreement (www.nextslm.org). An SLA template should include the previously mentioned components of an SLA in addition to penalties, problem resolution procedures and responsibilities of both parties. In this thesis the SLA document is referred as the contract document that two parties wish to sign (after negotiating and reaching an agreement) using an appropriately chosen contract signing protocol.

2.3 Virtual Organizations

The aim of this chapter is to explain the concept of 'virtual organization' (VO) in a more detailed manner. Below we present definitions of VOs, give some examples, and describe the main characteristics of a VO. We will also list the associated benefits and security challenges.

A VO is a relatively new concept that emerged in the beginning of 1990s. There is no single clear-cut definition of what a VO is. Different sources define it somewhat differently. For example, in [44] a VO is defined as

- 1) "A dynamic collection of individuals and institutions which are required to share resources to achieve certain goals".

[33] defines a VO as

- 2) “a temporary or permanent coalition of geographically dispersed individuals, groups, organizational units or entire organizations that pool resources, capabilities and information to achieve common objectives”.

Yet in [36] a concept IT infrastructure, namely network, is introduced:

- 3) “VO refers to both the members of a switchable interorganizational electronic network and to the network itself that delivers non-standard products”.

The first two definitions speak about “certain goals” or “common objectives” while the third opens up what these goals/objectives can be, i.e. non-standard products. Indeed, the incentive behind the creation of VOs is a collaborative delivery or/and creation of customer-specified products or services. The reasons why organizations cannot provide such products/services on their own and why they need virtual amalgamation are technical complexity of demanded products and constantly changing customer and market requirements [44]. In addition the development, production and support of modern products are highly complex [44]. Often the provision of such products would involve great risks for organizations unless they unite virtually, share resources, core competencies, risk, capital, capabilities and experience – all to meet a specific customer demand.

It is not a novelty in the history of organizations to unite in order to grab a good market opportunity. In the case of virtual organizations the “virtual” part of the concept cannot be ignored. As pointed out in [37] the related terms “virtual”, “virtually” and “virtuality” imply that something exists having a potential effect but this something is not tangible. In classical organizations the boundaries are clearly defined, while VOs are characterized by fuzzy boundaries, flexible structure and the ability to include new partners as the need arises. All this became possible with the rise of the Internet, networking, distributed computing and sophisticated web services. The central concept of VOs can be seen as vertical or horizontal⁸ relationships between several independent companies in which many of the commands and controls of traditional hierarchical organizations are performed by an advanced information and decision support system [39]. So, according to our point of view, in the definition of a VO the IT infrastructure can not be ignored. Therefore in this thesis we define a VO as follows:

VO is a temporary or permanent coalition of geographically dispersed individuals, groups, organizational units or entire organizations that pool resources, capabilities and information to achieve common objectives, while decisively relying on information technology (IT).

⁸Vertical relationships are relationships among the members of the value adding system, i.e. the partners; horizontal – are relationships among competitors. Both types of relationships are present in today’s VOs and involve interactions between the network partners [49].

2.3.1 Characteristics of a VO

VO partners achieve their common goal by pooling their assets and processes. The interest of this cooperation is that the VO will end up with more capabilities and power than each partner initially possesses. VOs can be large or small, long- or short-lived. Other characteristics of VOs are (based on [33], [36], [37] and [39]):

- A VO exists for a specific purpose, e.g. to implement a long-term marketing strategy, to launch a new groundbreaking product or to achieve some scientific goal. Non-standard product is the end-goal of a VO. VOs quickly deliver products/services that are innovative and customized.
- A VO should appear as a single organization to the customer.
- The membership and structure of a VO evolve over time.
- Switching: VO members can switch from one project to another depending on the current needs within the VO. Switching is the process of assignment and reassignment to the VO tasks/roles.
- Dynamic VOs have a capability to unite *quickly*.
- Usually members of a VO have shared responsibilities, shared control, shared leadership, shared access to computing resources and services, and shared loyalty. Members of a VO are also connected by common values and norms.
- The resources, services and people that comprise a VO can be single- or multi-institutional, homogenous or heterogeneous.
- Principle of synergy (many-to-one): A VO exhibits a unifying property because it is constituted from different organizational entities that create the effect of a single organization.
- Principle of divergence (one-to-many): a single organization can exhibit a multiplication property by participating in many VOs at the same time.
- Virtuality is a matter of degree rather than a categorical property of an organization. An organization can choose to virtualize one or more of its parts, like production core, front end or back end.
- The presence of an IT infrastructure is a necessary but not sufficient condition for VO formation. Examples of information technologies (IT) that can be used are e-mail, electronic file transfer, telephone, fax, screen sharing applications, videoconferencing, groupware tools, project management applications, web services, grid computing, etc.
- A VO is dependent on electronic linking, lies in the electronic space and is characterized by loose coupling (see SOA section in 2.1 for a definition).

- Spatial dispersion: VOs can be formed across country borders throughout the world.

2.3.2 VO lifecycle

Any VO lives through four stages/phases mentioned in [33] and [38]: *identification, formation operation and dissolution* as depicted in figure 5.

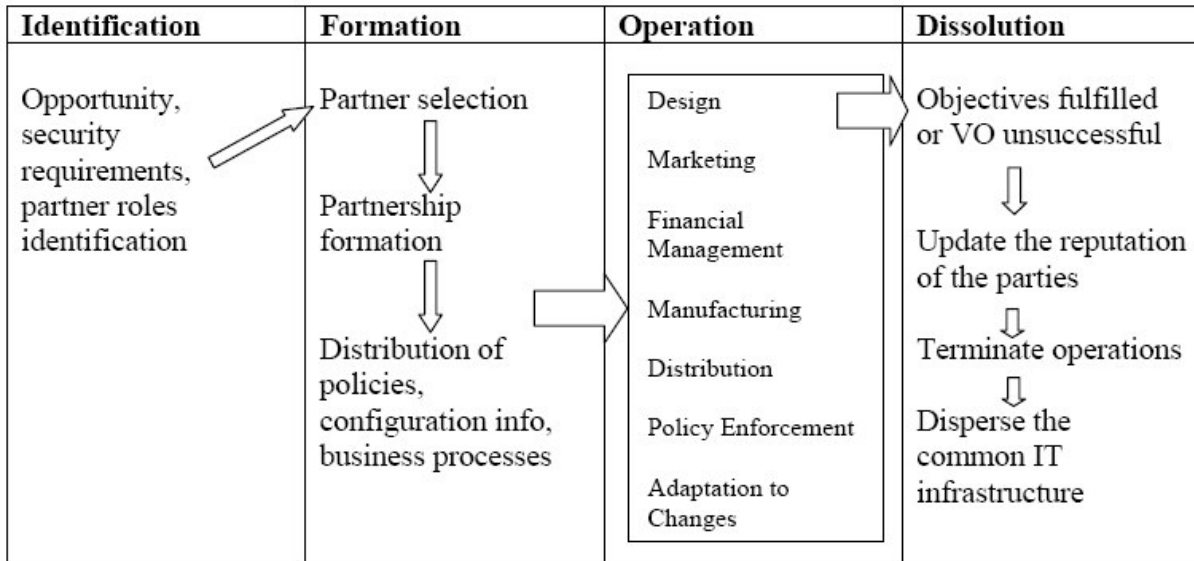


Figure 5: VO lifecycle [41, 48]

During *identification* a VO initiator, typically some organization, selects potential business partners (VO members) by using search engines and registries. Here it is important that potential VO members are trustworthy and are capable of providing the required services, which means they need to possess an appropriate level of expertise. Trustworthiness is typically synonymous to the reputation of a potential partner. The VO initiator deduces security requirements for the VO, identifies needed roles and makes sure that the partners can collectively fulfill them.

In the *formation* phase a VO initiator distributes configuration information, i.e. security and adaptation policies to the parties. Security policies include obligations, permissions and prohibitions. Adaptation policies describe what is to be done in case of security breaches, SLA violations and other organizational changes, e.g. a new member joining the VO.

Operation is the main phase in the VO lifecycle. It constitutes the execution of the VO tasks according to the pre-defined business processes. At this stage the overall performance monitoring and enforcement of policies are important. The performance of each partner is monitored and

gathered to be used as evidence when constructing or updating the partner's reputation. Any violation is reported to all other parties and may result in contract cancellations/adaptations. All parties enforce security mechanisms at their local sites and adapt to changes and violations. Dynamic changes of the VO structure occur during this phase as well. Some partners may be replaced due to their completed mission, contract term re-negotiation or contract violations. The initiator may also find new partners. In such cases the structure of the VO is re-configured and a new partner is integrated.

Dissolution takes place when the objectives of the VO have been fulfilled or the VO was not successful and partners were not able to operate (as pointed out in [37]). At this phase the reputation information of the participants is updated, all shared resources are invalidated and the common infrastructure is dissolved.

2.3.3 VO examples

Let's look briefly at some examples of what a VO may look like. This will illustrate the usefulness of VOs.

Example 1 (taken from [36]). A small software company wants to bid for a new contract, which initially is beyond the scope of the company's resources. This company forms a VO with other similar small companies and by doing that it is able to compete with larger corporations to gain the contract.

Example 2 ([36]). A business traveler comes to a foreign country and wants to do something in the evening. He/she goes to a WLAN-equipped café and requests a multi-media city guide to be shown using his personal computer and the café's facilities. This service can only be provided if the café, content providers, the user's home service provider, and other network operators have formed a VO previously.

Example 3 ([37]). EC (Electronics Company) has been producing equipment for broadband networks. EC develops 1) software control modules for broadband networks and 2) new strategic products. Control modules for broadband networks are the most critical products of EC. They are very complex too. EC found out that it is more efficient and cost-effective to entrust the programming of the machinery to external parties. So, EC has engaged itself in a VO relationship with multiple software vendors. EC runs several projects at a time and each project involves more

than one vendor. External software vendors are small software shops situated either in the neighborhood or far away. The business process goes typically in the following manner. A customer orders a network control unit from EC. EC assembles a project team responsible for this request. Project engineers design the product. A part of this product – the computer-driver machinery for producing the boards - needs to be reprogrammed, and new code needs to be developed. Engineers determine the software specification. Software vendors are selected and the contracts with them signed electronically. The software specification is then sent to the vendors and after a while (typically several days) the vendor delivers the code electronically. Engineers download the code and continue developing their prototype for the customer. To develop new strategic products EC collaborates with similar companies located at other continents. These are long-term projects that aim to move broadband network technology forward.

Example 4 [39]. GANT is a successful and worldwide known brand of clothing. With a small core company in Sweden, Pyramid Sweden, of eight people they are running a global network capable of coping with the intensified competition in the branch. Pyramid in Sweden is concerned with forming the strategy and the customer relationships of GANT and, thus, is holding the VO together. All other functions of the value chain are carried out by a variety of legally independent companies. All partners adhere to the common set of norms and values and have a common objective of creating collaboratively value to customers. The interrelations among the partners are characterized by a mutual exchange of trust. Creation of common trust leads to symbiotic relationships within the VO. The interactions are facilitated by the shared IT infrastructure.

2.3.4 Benefits and Challenges of VOs

To summarize our discussion on VOs we would like to list the benefits of engaging into a VO and the security risks associated with this process.

Benefits:

- VOs make it possible to satisfy constantly changing customer and market requirements in a competitive manner [36]. The access to market increases [37].
- It becomes possible to provide services precisely tailored to a specific customer need [46, 47].
- A company participating in a VO can concentrate on its core competencies and by that higher its strategic potential while getting the rest of the critical skills from the partners.

- The ability to participate in VOs increases the total service range a company can offer to its customers [36].
- Participation in VOs increases the total number of end-customers a company can reach indirectly via its partners [36].
- A particular organization can both “multiply itself” virtually by participating in several VOs and initiate a VO that will be constituted from different parties.
- By participating in a VO the concept-to-cash time is reduced [37].
- VOs are generally more flexible and adaptable to changes than traditional hierarchic organizations. For example, as stated in [39], VOs may be able to handle external influences like economic or financial crisis better than traditional companies.

Security Risks:

The biggest challenge for VOs is the establishment and maintenance of *trust* ([45 - 49]). As pointed out in [44], trust needs to be established at several levels: authentication, policy based management, and business rules. Partners who only relate to each other through IT (e-mail, phone, and fax) may find it difficult to trust each other. Trust can be seen as the organizational glue that keeps it from coming apart. It is particularly difficult to foster trust if the VO partners are not committed to the same vision and to the same ethical principles. Besides, if the company changes its partners very often and without a clear justification the partners may find it problematic to trust the company [39].

The main challenges related to trust establishment are:

- Each party has its own policies on access control and conditions of use.
- The allocation of resources is often dynamic since the structure of VOs may change dynamically. This implies that the VO initiator may not know until part way through the job that additional resource X is required.
- VO parties need to establish trust between them on a peer-to-peer basis.
- Trust needs to be established “on the fly”, which means that some mechanism is needed to negotiate conditions of use through the delegation of trust from one party to another.
- Parties may be located in different countries under different jurisdictions and, as a consequence, adhere to different legal and business requirements.
- Since VOs rely on IT, exposure to fraud or misuse of technology is a big concern.
- The security systems of VO partners must be mutually trusted. This brings up the challenge to come up with an effective and flexible security system.
- Contract management needs to be effective in order to be able to quickly reconfigure a VO. Services for management of electronic contracts must be trusted.

- SLA monitoring is important to ensure that parties perform according to contracts.
- Confidentiality, privacy, integrity, availability and accountability at a VO level have to be assured. At the same time parties have to provide access to their services and resources, as specified in agreements.
- It is a challenging task to choose between different potential vendors/parties ([47, 49]). The partners should contribute with their core competencies to the value adding system of the overall VO. Sometimes options are closed and only a few players are available on the market.
- If the VO is too decentralized there is a risk of lack of coordination and leadership within the business strategy. In such cases a clear leading partner may be identified which would control that other partners follow the chosen strategy.
- It can be difficult to anchor an IT infrastructure in a VO. People may experience as being forced to use it rather than as having it for assistance. Example: decision support system.
- It may happen that dispersed project members are incapable of communicating in a professional domain. For example, one may prefer open standard technologies, while the other proprietary ones.

Trust is established by the means of digital identities, certification, access control mechanisms, authentication, secure connection, reputation and inspection of the parties.

SLA signing is a building block of the extensive trust requirement within a VO. The security level of an SLA signing protocol should be connected to the overall security level provided by other mechanisms within a VO and the demands on the rigidity of security from the VO partners. Since VOs are characterized by instability, meaning that partners may change quickly and often, our SLA signing protocol should be efficient. It should be simple and should not create communication bottlenecks. Simplicity and efficiency are beneficiary from several points: the protocol should be simple to understand and, thus, to implement, it should be quick to execute, the simpler the protocol the less likely it is that it would create problems with the policies and jurisdictions of different countries, simple programs/algorithms/protocols tend to have less security holes, simpler protocols are easier to implement on different platforms. In addition the security of the protocol should be formally proven.

2.4 Introduction to TrustCoM Framework

As defined in [35], TrustCoM project aims at “developing a framework for trust, security and contract management for secure, collaborative business processes and resource sharing in dynamically-evolving virtual organizations”. TrustCoM environment is supposed to operate in open service architecture.

TrustCoM project is motivated by the problem of interoperation of IT infrastructures of various organizations, so the solution lies in an open service-oriented architecture. TrustCoM can be implemented with three possible architectures like CORBA, GRID, or Web services. CORBA was not chosen for development for consistency and superficial specification [34]. So TrustCoM builds on both Grid and Web services for they provide advantage in maintenance and support through their standards (see SOA part in chapter 2.1). As specified in [34] the service-oriented architecture (SOA) is adopted as the framework architecture.

As a part of any business practice signing contracts is an essential one. TrustCoM framework aims at ensuring secure and trustworthy contract signing where all VOs can enact with confidence. In TrustCoM framework the SLA management subsystem (discussed later) is devoted for managing negotiation, signing, and storing of SLAs. A general overview is given on all subsystems that TrustCoM framework includes. Subsystems are architectural components; each specialized in a certain task(s). They are related by certain dependencies and interactions. The subsystems are the following:

1. *VO Management subsystem*: This subsystem is responsible for coordinating VO's functionalities through its life cycle. This involves storing specific information about each VO existing.
2. *Business Process (BP⁹) Enactment and Orchestration*: This subsystem defines models and stores collaboration definition templates¹⁰ to support re-usage and fast definition of new collaborations. It defines the roles participating in collaboration, their tasks, interactions, and other specific TSC (trust, security, and contract management) requirement. Also this subsystem defines how the VO will meet its business objective. After deployment, BP management subsystem offers runtime management for the BP engine.

⁹Business Process (BP) is an abstract workflow that describes the action and tasks a unit has to enact.

¹⁰A Collaboration Definition Template captures the recurring best practices for a specific well known business collaboration in the format of a collaboration definition (CD). It is usually stored in a repository [TC2, Glossary].

3. *SLA¹¹ Management Services*: offer autonomous monitoring of service providers and check their adherence to the agreed upon quality of service parameters. They also provide the following functionalities:
 - i. SLA negotiation and signing between service provider and consumer.
 - ii. Performance monitoring of the service.
 - iii. Auditing service performance with respect to the negotiated and agreed terms.
4. *Trust and Security Services*: Services for the establishment and maintenance of trust relationships. Issue and validate security tokens¹² in addition to logging and auditing functions (e.g. for the use in the reputation system).
5. *Policy Control*: This subsystem defines, deploys, and enforces authorization policies for services access in addition to triggering adaptation actions in response to failures.
6. *Infrastructure Support*: This subsystem provides the basic functionalities for the framework, which are establishing a communication layer, maintaining services locations, discovery, and instantiation.

2.4.1 SLA Management Subsystem

In this thesis our concern is focused on the SLA Management subsystem because it is responsible for the signing process. The aim of this system, in addition to negotiating and signing SLAs, is to publish QoS information, monitor services run and their performance with respect to the defined QoS parameters, raise alerts and notifications in case of any violation. This is done by sending messages to other subsystems (e.g. performance data are sent to the trust and security subsystem), or by maintaining a local (to the subsystem) performance log. Below is a figure taken from [34], where all interactions between the SLA Management subsystem and other subsystems are shown.

¹¹A contract is an equal counterpart to an SLAs.

¹²A security token contains authentication information and maybe also access rights.

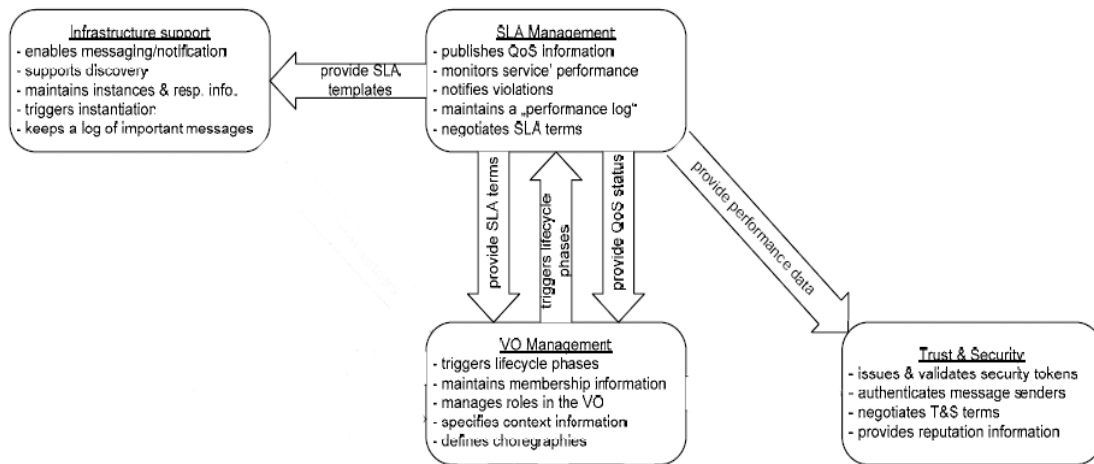


Figure 6: SLA Management Subsystem Interactions [34]

The subsystem has the following components:

SLA Manager: It is the coordinator of this subsystem responsible for the configuration of SLA monitors and evaluators.

SLA Template Repository: This is the place where a new service template is published. The template includes QoS parameters. This repository can be queried according to the given criteria.

Notary: This is the trusted third party; it witnesses every SLA signing process.

SLA Negotiator: supports negotiation processes. It offers negotiation function and implementation of negotiation protocols.

SLA Signer: implements the SLA signing protocol.

SLA Repository: A secure storage where all signed SLAs are stored. It is maintained by the Notary.

SLA Monitor: These are two types - internal and external monitors. Internal monitors have access to applications and resources, e.g. ASP¹³ and Host Domain Monitors. External monitors only monitor services' interfaces, e.g. trusted third party monitors.

¹³ASP: Application Service Provider: is an enterprise, company or individual that provides application services and offer them via searchable registers to customers [TC2, Glossary].

SLA Evaluators: This component is responsible for sending notifications on any SLA violation or success. Recipients of such alerts are the VO management and reputation services.

SLA Performance Log: collects SLA performance data for later evaluation and use.

2.4.2 SLA Signing – TrustCoM Model

Since signing process is the core of our thesis we will explain here what is thought by TrustCoM experts about signing. As previously mentioned, signing is hosted by the SLA Management subsystem. In [34] it was assumed that no party wants to sign the SLA contract first, so the solution was to involve a trusted third party that collects both parties' signatures first, signs the SLA and sends copies back to the two parties. This seems to be a natural solution but it is difficult to accept it in the TrustCoM environment because, firstly, this solution necessitates the involvement of the TTP in *every* execution of the contract signing and, secondly, the VO environment, where this protocol is to operate, is too dynamic to accept all this overhead traffic to and from the TTP. However, it is stated in the version2 of TrustCoM framework that it supports signing schemes only with a notary.

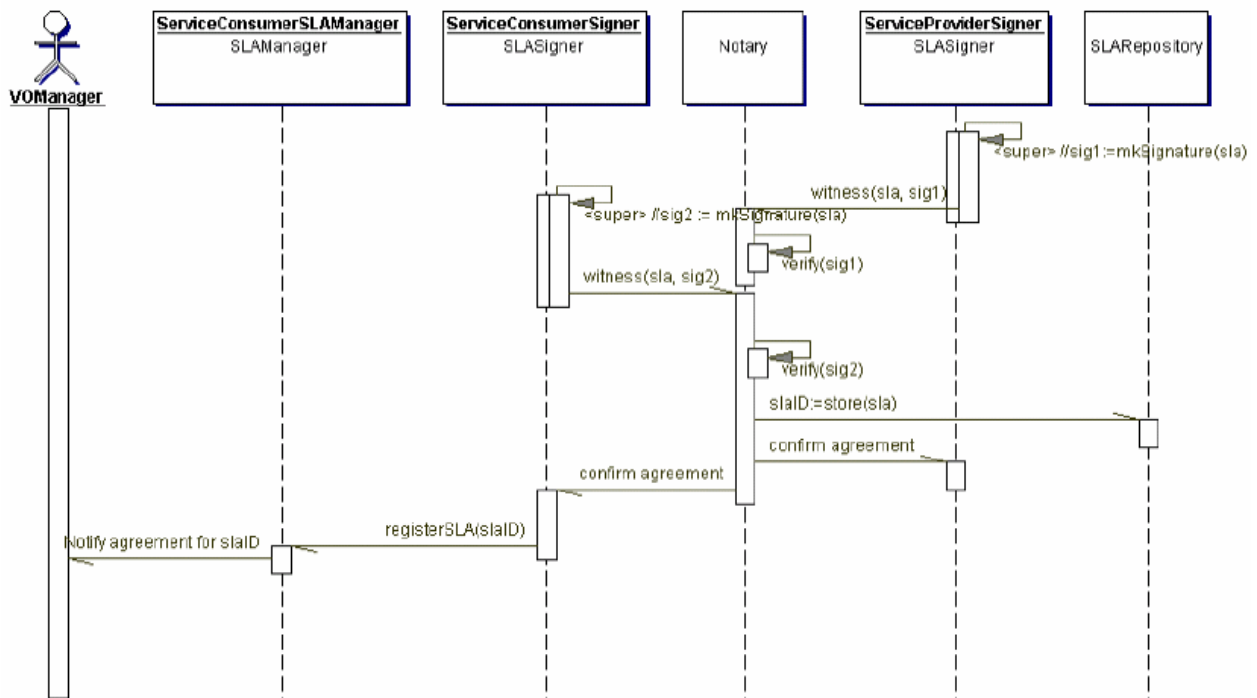


Figure 7: SLA Signing and Storage[34]

Below we describe a signing scenario of TrustCoM, which is sketched in the SLA signing and storage diagram [42, p. 55].

A company A is a consumer and a company B is a provider. Company B provides memory storage services. Company A runs an online hotel reservation system. Due to the complexity of the memory and storage management of their records they have decided that it is more suitable to use B's services. After negotiating the SLA terms (QoS parameters, maintenance, and obligations to each party) through the SLA Negotiator component both companies need to sign the agreed-upon SLA. Assuming that the company A (Consumer SLA Signer in figure 7) is the initiator, company A generates its own signature over the SLA and sends SLA together with the SLA's signature to the Notary. Company B (Provider SLA Signer) does the same. After receiving both SLA copies with the signatures the Notary checks and verifies the parties' signatures and their SLA copies. If the verification is successful the Notary stores the signed SLA in the SLA Repository and sends a confirmation (through the SLA Evaluator component) to both companies. The confirmation states that the SLA is signed and effective. After that the signed SLA is registered in the Consumer SLA manager and the VO manager is notified of the signed SLA. During the signing process the SLA Monitor component is listening to the execution, log violations and results.

2.5 Security Requirement Engineering

Deriving (Engineering) security requirements is an important and critical phase in any security engineering process. Deriving security requirements is the first step in designing a system with security in mind from the kick start.

2.5.1 XP Extension for Security Requirements Engineering

XP (eXtreme Programming) maintains an iterative and rapid feedback-driven nature for developing software systems. Extended XP practices addresses planning, designing, coding, and testing in a software development process. In [44], Böstrom et *al* proposes extended these practices to include security-specific flavors.

According to SSE-CMM¹⁴ model [42] security engineering includes the following:

1. Identify security risks
2. Define security needs (requirements) to counter risks
3. Apply security needs
4. Promote the correctness and effectiveness of the system

¹⁴SSE-CMM: Systems Security Engineering Capability Mature Model.

5. Define operational impact of untreated risks (accepted risks)
6. Understand the trustworthiness of the system through uniting all effort from different disciplines

Capturing security requirements of a system starts with a functional description of the system (specs, legal requirements, and assumptions). This is continued by defining assets and their threats, possible attackers, their motivation, and resources. Next the probability of each threat is estimated and grouped according to some predefined criteria, maybe according to the severity of the threat and the need to eliminate it. These steps generate enough information for arriving at the security requirements.

The proposed steps to be added to the XP planning Game to support security requirement engineering are as follows: [44].

1. Identify security assets
2. Define Abuser stories (threats)
3. Assess risk for Abuser stories (using CORAS, see next section)
4. Negotiate Abuser stories and User stories
5. Define security related Users stories
6. Define coding standards and countermeasures checking

2.5.2 CORAS Risk Assessment Methodology

CORAS is a risk analysis methodology that aims to deliver precise, unambiguous, and efficient risk analysis for security critical IT systems. CORAS is the result of the CORAS project [43]. CORAS has been chosen for risk analysis because the extended XP practices for security requirements derivation do not specify a particular methodology to use. But other features in CORAS contributed to our decision as well:

1. It is based on many well-formed and agreed standards. This makes it more generic and its results more trustworthy. The standards are:
 - RM-ODP UML: Standards for documentation.
 - AS/NZS 4360, ISO/IEEE 13335, 17799: Standards for risk and security management.
 - UP: Standards for systems development.
 - XML: Standards for data integration.

2. CORAS has a library of reusable experience packages. A package includes assessment results and documentations.
3. CORAS has an UML Profile for security assessment which is an extension of the basic UML language targeting security risk assessment.
4. CORAS project has developed an XML mark-up language for representing security risk assessment information. This is due to the lack of a standardized meta-data for representing risk assessment information.
5. Finally, CORAS was chosen because the concepts of trust and risk are interwoven. Trust can be dealt with just as any asset. This is important to TrustCoM concepts and objective. Legal risk analysis is incorporated with the risk analysis methodology (e.g. analysis of concrete legal documents, like contracts, which is the core of our thesis work).

The core of the CORAS risk analysis methodology consists of five consecutive tasks. Other tasks are communication, consulting, monitoring and reviewing the results. In figure 8 we demonstrate five core tasks and their flow.

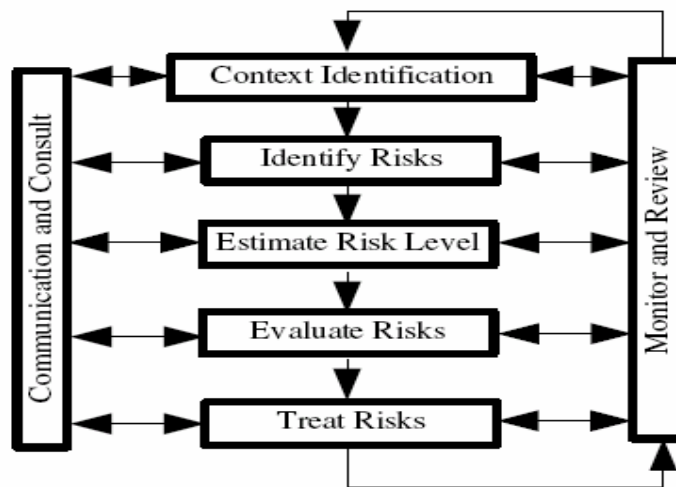


Figure 8: CORAS Methodology[43]

Phase one: Context Identification

In the first phase the characteristics of the target of analysis are mentioned together with identifying the assets and valuing them (asset diagram can be built in this section). Here the risk acceptance level criteria are set.

Phase two: Identify Risks

Through a thorough brainstorming threats to all identified assets along with the asset vulnerabilities are identified.

Phase three: Estimate Risk Level

In this phase the level of severity of each identified risk is determined. Risk level is based on the frequency and the consequences of the risk.

Phase Four: Evaluate Risks

In this phase risks are prioritized according to the risk level. Similar risks are grouped together. Here is the time to decide which risks are to be treated, because not all risks can be eliminated. This is done according to the risk acceptance criteria defined in phase one.

Phase Five: Treat Risks

For each unaccepted risk treatment plans are defined, evaluated, and then prioritized.

3 Deriving Security Requirements

At a Glance:

In this chapter the security requirements relevant for contract signing are derived. Extended XP Practices for Security Requirements Engineering are used, and CORAS method is applied for risk assessment.

At the end of the chapter a set of security requirements that are to be satisfied in the chosen contract signing protocol, is presented.

Building a secure software system is not an easy task. Attention should be paid to all aspects that affect the system. An understanding of the system security requirements is mandatory. The designed system environment needs to be deeply understood and this should be expressed as design requirements. Since the choice of a contract signing system is our ultimate goal, it is crucial to dig out the security requirements that will be reflected in the protocol choice and implementation.

We will use the Extended XP (eXtreme Programming) practices for security requirements gathering [44]. These practices support the agile methods for project management. Extreme Programming (one of the agile methods) supports iterative and gradual progress of the system design and requirements gathering, and promotes inter-disciplinary person-to-person communication. These methods emerged after the sad fact published in the Chaos report [46] stated that less than 50% of all software projects have a successful ending. In the risk assessment phase of the XP Extended Practices CORAS will be used. CORAS is chosen because it is an efficient security risk analysis framework and it focuses mainly on IT security systems (see chapter 2.5 for more information about Extended XP practices and CORAS).

3.1 Context Identification

The system that is studied here is a contract signing system. This system is responsible for the execution of the chosen signing protocol between two parties under the supervision (maybe) of a trusted third party that will be consulted in case of any dispute. The system is to be designed to work in a dynamic environment where signing processes are supposed to run dynamically, automatically, and securely¹⁵ after negotiating the contract. Note that in terms of efficiency no objectives were defined. In our case it is difficult to measure it, since no similar systems were built before that can be used to compare with or measure the efficiency. Still we do consider efficiency as a protocol property and mention it in the protocol choice chapter, but our main concern is the security of the signing protocol.

The contract signing system consists of several components, each with its definition and role. Assets are the components to be protected. They are vital for the successful run of the signing process; assets can also be some sensitive data (e.g. a private key). Each asset is defined and its relationships with the other assets are shown. In the subsequent subchapters we present the system's user stories, followed by the systems critical assets.

¹⁵ By *securely* we mean satisfying all the security properties previously agreed.

3.2 System User Stories

Defining user stories is important for they reveal the main functionality of the system and the important assets involved. Figure 9: System User Stories and Assets - shows participant's resources (assets) and capabilities (user stories)

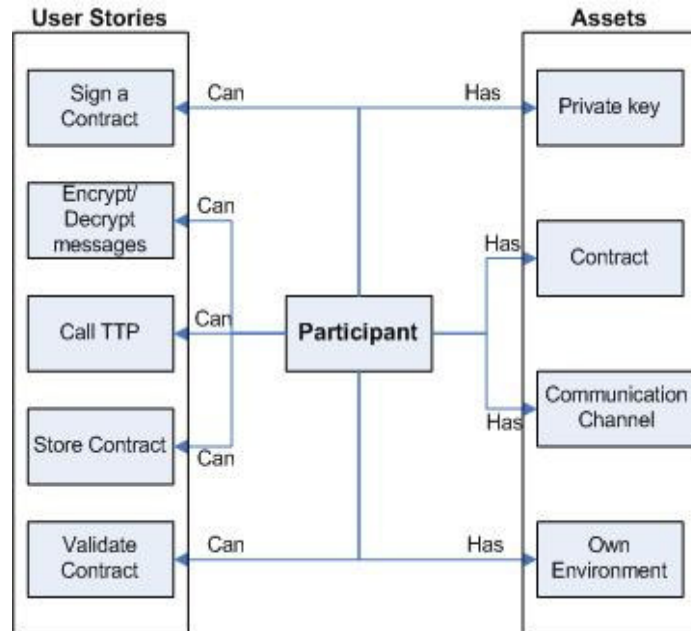


Figure 9: System User Stories and Assets

3.3 Assets Identification

In Figure 9: System User Stories and Assets the following critical assets are depicted:

Contract Text: a contract text is an asset. By the contract text we mean an electronic file of any agreed format and a structure (e.g. XML, TXT...etc) on which agreed-upon clauses and terms are written. At the end of the contract signing process the contract text will be signed with the signatures of both parties to be considered legal and enforcing. According to TrustCoM Architecture Deliverable version 1[34], a contract is defined as “a form of convention that designates the behavior the involved parties commit to”. As previously mentioned, a contract is principally a legal counterpart to an SLA.

Participant's Private Key: This is the secret key with which a participant signs a contract. A document signed with A's private key, for instance, is an evidence that this document is originated from A, because no one else is assumed to know A's private key. The private key of a participant is

crucial in the signing process; any loss or disclosure of it will compromise the participant's situation.

Participant's local environment: This constitutes the front-end of the signing process, from which each party can monitor, view logs or even manually interfere in the signing process. The signing system should not exploit the local environment of any participant.

Communication Channel: This is the physical connection between the parties and the TTP. Over this channel all protocol transactions are executed. In case of TrustCoM contracts are supposed to be signed across Internet by dynamically setting up communication channels between parties to carry out the signing process. Once established the channel should be protected against any information loss or eavesdropping to defend both parties' privacy and the security of the signing process.

<i>Asset ID</i>	<i>Asset Theme</i>	<i>Value</i>
Participant's private key	Software or Hardware (e.g. smart card)	High
Participant's local environment	Software and Hardware	High
Communication Channel	Hardware and software	Medium
Contract Text	Software	High

Table 1: Assets Identified

3.4 Abuser Stories Identification

In identifying abuser stories we assume a very primitive contract signing model, i.e. two parties with no security precautions and without a trusted third party. This is done to extract a maximum number of abuser stories and to assess contract signing process in general (not only in TrustCoM framework).

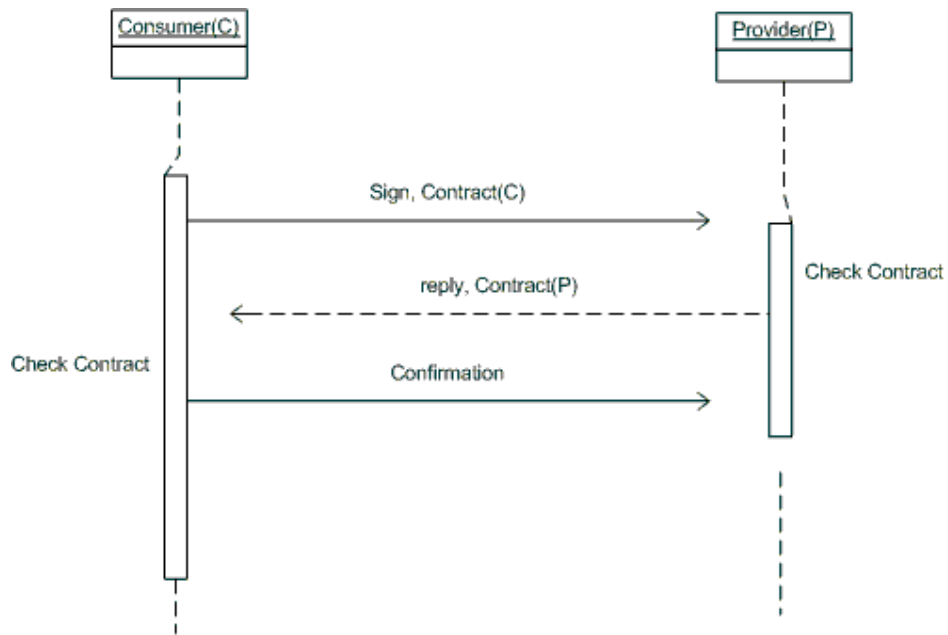


Figure 10: Primitive Contract Signing Model

Abuser Story	Security Concept Abused	Description
<i>Gaining control over party's local environment</i>	<i>Confidentiality Integrity</i>	Vulnerability in the protocol implementation can result in vulnerability in the party's system, where an attacker (can be a competitor) can use it to hack into. This could lead to leakage of participant's private key and secret information.
<i>Eavesdropping the communication channel</i>	<i>Confidentiality</i>	A non-participant (e.g. an interested competitor) can eavesdrop the communication channel between the two parties. This could be used as a mean for industrial espionage and to expose the contract to public.
<i>Modification of the contract</i>	<i>Integrity</i>	As each participant has his own contract copy, a party may change the contract terms to suite his own benefits, thus violating the negotiated SLA.
<i>Modification of the protocol parameters</i>	<i>Integrity, Confidentiality</i>	A party may have access to important protocol configuration parameters, which he can change to adapt it to his own benefit. Configuration parameters can be a time-out value, encryption scheme, ...etc

<i>Abuse of protocol's intermediate outputs</i>	<i>Trust</i>	A party may involve in a contract signing process just with the intention to abuse others and use some intermediate results to present to another party as a proof of good will and reputation (refer to abuse-freeness definition in chapter 4). NRR and NRO are such intermediate outputs of a signing protocol.
<i>Denial of participation (repudiation)</i>	<i>Trust</i>	A party may deny any participation in a signing process. This can be done by a signer that discovered that he can no more be bonded to the contract. This can be done by removing any evidences he has about the protocol run. (e.g. NRO and NRR), or by denying his own signature.
<i>A non participant may impersonate a participating party</i>	<i>Trust, Integrity</i>	A non-participant may act as a participating party and thus carry out the contract for himself. The impersonated party will assume that he didn't sign with his counter party while the other party will not know that.
<i>A party might terminate the protocol prematurely</i>	<i>Integrity, Trust</i>	A party may stop the protocol execution prematurely to gain some benefits. He can e.g. build the counter signature given his higher computation power or just halt because he does not want to sign anymore.
<i>Process Crash</i>	<i>Availability, Trust</i>	The signing process may suddenly crash. This may lead to undesirable consequences. Atomicity of the protocol run must be assured.
<i>Network Error</i>	<i>Availability</i>	A sudden error in the communication channel may interrupt the signing process and leave the participants in vulnerable states.

Table 2: Abuser Stories

3.5 Abuser Stories Risk Assessment – CORAS

In our case we lack quantitative data, so qualitative data are to be used for risk assessment applying CORAS. In CORAS risk is calculated as a function of “Impact” and “Probability”. CORAS defines a scale for each (Impact and Probability). Impact and Probability scales are defined below in two tables (Table 3 : CORAS Impact Scale and Table 4: CORAS Probability Scale).

Impact	Description	Value
Insignificant	Minor delays, no business impact	1
Minor	Loss of project phases or profits	2
Moderate	Loss of a client or a project	3

Major	Loss of some part of the business or closing a department	4
Catastrophic	Out of business	5

Table 3 : CORAS Impact Scale

Probability	Description	Value
Rare (0 – 0.01)	Less than once per 10 years	I
Unlikely (0.01 – 0.05)	Less than once a year	II
Possible (0.05 – 0.2)	About once a year	III
Likely (0.2 – 0.5)	2-5 times a year	IV
Certain(0.5 – 1.00)	More than 5 times a year	V

Table 4: CORAS Probability Scale

Risk level is calculated according to the following formula:

Risk level = Impact * Probability. This can be expressed in a matrix (Table 5: Risk Level Matrix).

		<i>Impact</i>				
		<i>Insignificant</i> 1	<i>Minor</i> 2	<i>Moderate</i> 3	<i>Major</i> 4	<i>Catastrophic</i> 5
<i>Pr ob abi lity</i>	<i>Rare – I</i>	Low	Low	Low	Moderate	Major
	<i>Unlikely – II</i>	Low	Low	Moderate	Major	Major
	<i>Possible – III</i>	Low	Moderate	Major	Major	Extreme
	<i>Likely – IV</i>	Moderate	Major	Major	Extreme	Extreme
	<i>Certain – V</i>	Moderate	Major	Extreme	Extreme	Extreme

Table 5: Risk Level Matrix

After showing the impact, probability, and risk level scales as defined by CORAS we need to assign each abuser story with an impact and probability value. Our risk assessment involves two parties, the service consumer and provider, so abuser stories will have different impacts and probabilities depending on the party referenced. See Table 6: Abuser Stories Risk Assessment.

		<i>Impact to</i>		<i>Probability to</i>		<i>Risk Level</i>	
		<i>Consumer</i>	<i>Provider</i>	<i>Consumer</i>	<i>Provider</i>	<i>Consumer</i>	<i>Provider</i>
<i>A b u s e r S t o r i e s</i>	<i>Gaining control over party's local environment</i>	3	4	III – IV	IV	Major	Extreme
	<i>Eavesdropping the communication channel</i>	4	3	IV	IV	Extreme	Major
	<i>Modification of the contract</i>	3-4	3-4	III – IV	III – IV	Major	Major
	<i>A non participant may impersonate a participating party.</i>	4	3	IV	IV	Extreme	Major
	<i>A party might terminate the protocol prematurely.</i>	3	3	III	III	Major	Major
	<i>Modification of the protocol parameters</i>	2-3	3	III	III	Moderate	Major
	<i>Abuse of protocol's intermediate outputs</i>	2-3	3	III – IV	III - IV	Major	Major
	<i>Denial of participation (repudiation)</i>	3-4	3	IV	III - IV	Extreme	Major
	<i>Process Crash</i>	2	3-4	III	III	Moderate	Major
	<i>Network Error</i>	2-3	3	III	III	Moderate	Moderate

Table 6: Abuser Stories Risk Assessment

After filling the abuse stories impact and probability values, we can calculate the risk level for both, the consumer and the provider using the risk level matrix table (Table 5: Risk Level Matrix). Note that assigning impact and a probability value is a subjective task in a sense that it depends on experience and knowledge.

After assigning the risk levels of all abuser stories a decision should be made whether to:

1. Accept
2. Monitor
3. Or treat the risk by the means of countermeasures

The decision is taken according to the risk level each abuser story has. Table 7: Risk Actions Criteria is a table that defines the criteria upon which decision is taken.

If risk level is	Then
<i>Low</i>	Accept the risk
<i>Moderate</i>	Monitor the risk
<i>Major or Extreme</i>	Take mitigating actions.

Table 7: Risk Actions Criteria

Risks will be dealt with in the following priority: extreme risks, major, and then moderate. In the next section we define security-related user stories to countermeasure the abuse stories. The result is presented in Table 8: Security-related User Stories.

3.6 Security-Related User Stories

In this subchapter we have defined the security-related user stories; these are functions that contribute to the security of the signing system, its environment, and its interactions. In security-related user stories we have tried to be specific about the signing system even though security should be looked upon as a whole. It was difficult to include the whole TrustCoM environment in our analysis for the lack of the security-related information.

Security-Related User Story ID	Description
A	Generate contract digest before encryption
B	Encrypt communication channel
C	Logging and auditing of protocol execution instances
D	Use a Trusted Third Party (TTP) to mitigate real-time errors
E	Securing the party's environment (mainly the private key)
F	Time-out mechanism
G	An Abort Signing mechanism (by TTP)
H	A Resolve Signing conflict mechanism (by TTP)
I	Secure contract storage and encryption
J	Use a standardized and formal contract representation (e.g. XML)
K	Provide a contract comparison mechanism (depends on contract representation)
L	Ensure process Atomicity

M	Use a secure digital signature (e.g XML Digital Signature)
N	Provide authentication mechanism
O	Provide non-repudiation mechanism

Table 8: Security-related User Stories

3.7 Mapping Abuser Stories to Security-Related User Stories

Now that abuser and user stories are identified it is time to see how they relate. Mapping is done according to which user story helps monitoring or mitigating (depending on risk level) an abuser story. The result is presented in Table 9: User - Abuser Stories Mapping.

Abuser Story	Security-related User Story
<i>Attack on party's own environment</i>	BCEI
<i>Eavesdropping the communication channel</i>	BFM
<i>Modification of the contract</i>	AJKM
<i>Modification of the protocol parameters</i>	CD
<i>Abuse of protocol's intermediate outputs</i>	DFGHNO
<i>Denial of participation (repudiation)</i>	DFGHCNO
<i>A non participant may impersonate a participating party.</i>	NEC
<i>A party might terminate the protocol prematurely.</i>	DGHL
<i>Process Crash</i>	ECLGH
<i>Network Error</i>	LGH

Table 9: User - Abuser Stories Mapping

Table 10: Abuser Stories-Security-Related User Stories Mapping is presented to show the mapping between security-related user and abuser stories. A shaded box means that the user story can treat the corresponding abuser story. The signing protocol should be able to address as many security-related user stories as possible.

3.8 Concluded Signing Protocol Requirements

As a conclusion for chapter 31 we present the features/requirements of the desired contract signing protocol.

1. **Support of a TTP** (preferably an offline TTP for its advantages presented in chapter 4).
2. **Support of abuse-freeness** (see chapter 4.3 for the definition of abuse-freeness). This can be implemented as a protocol property or controlled by the reputation system that exists in TrustCoM. For example, intermediate results could have no influence on a party's reputation, what counts is the final signing. Also the party has no authority to control its reputation in the VO, only the reputation system does.
3. **Fairness** of the signing process to both parties. Fairness can be enforced by providing functions simulating rights that a disadvantageous party can practice in case of unfairness.
4. **Non-repudiation**. It can be provided by using the PKI (public key infrastructure) cryptosystem.
5. **Atomicity** of the signing process transactions. This property is related to fairness and abuse of a protocol run. Atomicity ensures that a protocol run ends either with a signed protocol or is considered as it has never happened.
6. Support of a **secure digital signature** scheme (e.g. XML digital signature). This property is also related to non-repudiation because XML digital signatures support PKI cryptosystem.
7. The chosen protocol must **support asynchronous and unreliable** networks. This requirement stems from the heterogeneous network environment the protocol shall run in, which is the Internet.

Abuser Story ID	Abuser Story Description	User Story Description															
		User Story ID	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Gaining control over party's own environment																
2	Eavesdropping the communication channel																
3	Modification of the contract																
4	Modification of the protocol parameters																
5	Abuse of protocol's intermediate outputs																
6	Denial of participation (repudiation)																
7	Process Crash																
8	Network Error																
9	A non participant may impersonate a participating party.																
10	A party might terminate the protocol prematurely.																

Table 10: Abuser Stories-Security-Related User Stories Mapping

4 Contract Signing Protocols

At a Glance:

In this chapter we start talking about contract signing protocols in depth. Particular accent is put on the challenges associated with the signing process and the properties contract signing protocols possess. We are presenting the idea of gradual exchange, probabilistic and trusted third party protocols with descriptions taken from the relevant literature. It is worth noticing that the terms ‘contract’ and ‘SLA’ are used interchangeably and are to be treated as synonyms in this text.

4.1 What is electronic contract signing?

A contract is an agreement between two or more parties where the rules of the game are specified. It allows the mutually suspicious parties to overcome distrust of each other, because their misconduct can be revealed and penalized. Parties can cooperate then with minimal risks. SLA (Service Level Agreement) is a specific type of contract which states the quality of services that has to be maintained. SLAs are concluded between a service provider and a service consumer. See chapter 2.2 for more information on SLAs.

No contract or SLA would be of any use unless it is signed. A signed contract is a contract body signed by both parties, that is, a pair of signatures on a contract text [26]. Contract signing achieves more than just an agreement: besides concluding an agreement the parties also want to be able to prove it afterwards [29]. A signed contract is a convincing certificate to an impartial third party that an agreement has actually been concluded. In our thesis we define **signing** as *a sequence of transactions the VO service provider and service consumer need to execute in order to exchange electronic signatures on the previously agreed SLA*.

Electronic signing should not be confused with electronic/digital/cryptographic signatures. The latter can be thought of as a partial outcome of a signing process. A digital signature on an SLA is a cryptographic unit attached to an SLA text. This cryptographic unit testifies who had signed and what had been signed. Electronic/digital signatures are based on a public-key infrastructure (PKI) and involve PKI certificates. The strength of electronic signatures is related to the cryptographic algorithm used. Electronic signatures provide integrity of a message (SLA in our case) and authentication evidence. An **electronic/digital signature** can be defined as *a construct that authenticates both the origin and the contents of a message in a manner that is provable to a disinterested third party* [50].

Signing contracts is not a novel phenomenon. In fact, signing and various visual marks associated with it, e.g. signatures and seals, have been around for so long, that they have even acquired a cultural meaning that differs from country to country [30]. Signing today has essentially the same function as in ancient times with the exception that it has moved to a digital world: contracts have become ‘electronic contracts’ and signing – ‘electronic signing’. This new world has introduced certain previously unknown challenges described in the next chapter.

4.2 Challenges Associated with Electronic Signing

In a physical world contracts are signed face-to-face. All parties can be sure about the identities of the other parties, that they sign the same contract, that everyone has signed the contract, and that outsiders would not intercept the contents of the contract. They can also sign the contract simultaneously under the observation of some impartial external party. All this is not provided by default in the electronic metaphor. Obviously, the difficulties arise due to the geographic distance between the parties and the unreliable nature of open networks.

The situation when a party A has no way of determining whether party B's failure to respond was malicious or not, and when A has no means of convincing an impartial outsider of B's malicious actions is called a *contract signing problem* [31] or, alternatively, a *fair exchange problem* [28]. The signing process enters at some point an unfair state, in which one party can take an advantage of the other. To overcome this problem, protocols are required to be **fair**, i.e. at the end of the protocol execution either both parties have each other's valid signatures on a contract or none of them does [25]. Fairness should hold even when one of the parties acts maliciously.

For example, consider the situation when the terms of a contract have been negotiated and everyone seems to be content with the situation, the only thing that remains is to sign it. Who is to start signing? Should it be some external trigger that instructs everyone to sign at the same time, or should it be lottery-based, time-stamped etc.? The point is that no one wants to take the risk to start first, because initially there is no guarantee that everyone will sign and act honestly.

Further challenge in electronic signing is the provision of **atomicity** [31]. Atomicity implies at the execution level that either all planned series of transactions take place or none of them does. In our case it means, either all transactions involved in the signing protocol will occur as specified, or the whole signing process will be annulled. Atomicity is necessary but not sufficient for a fair protocol. It means that a protocol may provide atomicity but may be flawed at a logical level and not provide non-repudiation or fairness.

One more challenge associated with electronic signing is that signing is expected to be a quick process. Even if a perfectly secure signing protocol exists, it is expected to perform relatively quickly and **efficiently** (see protocol property nr. 9 in 4.3) when implemented in a network environment.

4.3 Properties of Contract Signing Protocols

What properties do contract signing protocols possess to deal with the mentioned challenges? What are their characteristics? To start with, it is reasonable to make an assumption that no party acts against its own interests [15]. We talk about protocols with this assumption in mind. In this paper only two-party protocols are considered.

After exploring material on contract signing protocols (see the protocol references at the end of this document) we have come to a list of properties that can be used to evaluate each contract signing protocol.

1) TTP (Trusted Third Party)

TTP stands for a Trusted Third Party. Trusted third party is an external party that may be used by signatories to enforce trust in their interactions. The primary task of a TTP is to facilitate message exchange between two parties that do not fully trust each other but trust the TTP.

There are protocols that do not make use of a TTP. They either follow a probabilistic approach or belong to the equal-computational-power approach. Probabilistic protocols provide probabilistic security, i.e. security with some probability p . Protocols demanding equal computational power are based on gradual exchange of signatures (bit-by-bit by taking turns) and demand equivalent computational power of entities, which is quite unrealistic [21, 29, 30]. If one of the signatories suddenly halts prematurely, both entities will have the same fraction of the peer's signature and, hence, the same amount of work to complete the contract offline.

A VO environment can be quite complex and involve many communicating parties. Gradual exchange protocols are cheap on the one hand (in a sense that no extra party is involved), but, on the other hand, they necessitate an extremely large number of messages and computational overhead, so that it gets expensive with respect to communication and computation [22, 39]. Probabilistic protocols give only probable security, which is not always acceptable; especially not in cases where financial liabilities are involved.

There are different types of protocols that use a TTP. A TTP involvement can be *inline* (intermediary in each transmission), *online* (intervenes in each session), *offline* (does not intervene unless a problem occurs; protocols with offline TTP are also called *optimistic*), *neutral* (the

assistance the TTP brings is not conditioned by the knowledge of the information exchanged), and *transparent* (it is impossible to decide at the end of the protocol whether the TTP did intervene or not). In addition, there can be *not-so-trusted third party* protocols. These protocols assume a TTP, but do not have to trust it as much as in other TTP protocols, because a not-so-trusted TTP is not involved in resending any contractual parameters at any time during the protocol [31].

The main drawbacks of having a TTP are the creation of bottlenecks (when it is used abundantly) and the need to trust the TTP. In practice a TTP should not be fully trusted, should not have any interest in the contract under question, and should not be able to derive private keys of the parties [22, 26]. Using a TTP implies extra costs for each involvement [23]. In addition, finding a TTP can be a problem [31].

2) Stateless vs. State-keeping TTP

This property concerns protocols with TTP only. A *stateless* TTP does not keep track of any information during or after the protocol execution. A TTP that is *state-keeping* has to keep records of diverse information, concerning each protocol run, in databases, e.g. messages it forwards, what protocol moments were completed, etc. Management of such databases represents a significant security risk, because sensitive information is centralized at one site [15, 25]. A positive side of a state-keeping TTP is that in case of disputes proofs are more complete.

3) Fairness

Fairness implies that the parties finish the protocol in a fair state: either both get the signed contract and the corresponding non-repudiation evidences or none of them gets anything valuable. A fair contract signing protocol guarantees that A gets B's signature if and only if B gets A's signature on the contract. Fairness can be *weak*, *probabilistic*, *true* or *strong*. [15].

Weak fairness (W) ensures that if one party does not obtain its evidence, while the other party does, the first party will receive a proof of this fact.

Probabilistic fairness (P) concerns protocols without a TTP. It guarantees fairness with some probability p .

Strong fairness (S) implies that at the end of the protocol either the sender got the non-repudiation of recipient evidence together with the signed contract and the receiver got the corresponding non-

repudiation of origin evidence together with the signed contract, or none of them got any valuable information.

True fairness (T) is the same as strong fairness plus the non-repudiation evidences produced during the successful execution of the protocol are independent of how the protocol was executed. Transparent TTPs provide true fairness.

It has been proposed that a contract signing protocol that is fair should fulfill the following requirements [29]:

Correct Execution: Consider that two correct signatories A and B decide to sign a previously unsigned contract. The contract signing protocol outputs “signed” if and only if the contract texts of both signatories are identical or else the signing is rejected.

Unforgeability of Contracts: No valid contract can be produced without the participation of a correct signatory (A or B). If a correct signatory did not carry out the signing procedure till the end so far, a correct verifier (notary, a TTP) will not be able to confirm the contract as signed.

Verifiability of Valid Contracts: A contract that was ever declared “signed” can not be invalidated again. It means that if a correct signatory A or B has completed the signing procedure with the affirmative result, a correct verifier will always confirm it. The same requirement has been also called *durable* in [26] in the sense that once the agreement is reached there is no way to undo it.

No Surprises with Invalid Contracts: A contract that was ever declared “rejected” cannot be declared “signed” afterwards. If a correct signatory A or B has rejected the signing of a particular contract, no correct verifier can declare it as “signed”.

Termination on Synchronous Network: A correct signatory A or B will either complete signing the contract or reject the signing after a fixed number of rounds.

Termination on Asynchronous Network: A correct signatory A or B will either complete signing the contract or reject the signing before a wakeup signal (a local time-out value or an input from the user that signals a time-out).

4) Non-repudiation

Whenever a message is sent over the Internet, there is no guarantee that it will be delivered to the intended recipient. Even if the message has been delivered, the recipient may repudiate this fact. This may be unpleasant in the situations where contracts with liabilities are involved.

To overcome the risk of repudiation many protocols provide both the *non-repudiation of origin* (NRO) and *non-repudiation of receipt* (NRR) evidences. NRO implies that a protocol generates a

non-repudiation of origin evidence, destined to the receiver, which can be presented to an adjudicator, who can unambiguously decide whether the sender is the author of a given message or not [15]. NRR means that a protocol generates a non-repudiation of receipt evidence, destined to the sender, which can be presented to an adjudicator, who can unambiguously decide whether the receiver received a given message or not [15]. Usually, in non-repudiation protocols, a message (e.g. a signed contract) together with a NRO is exchanged against a NRR.

Non-repudiation assures that a contract can not be broken after the signing protocol is over. If any party would like to withdraw the other party must give its consent [31].

The challenging part of non-repudiation protocols is to insure that no party is able to cheat. In other words, if a protocol provides non-repudiation, it does not automatically mean that it is fair. For example, protocols ISO/IEC 13888-1¹⁶, ISO/IEC 13888-2¹⁷, ISO/IEC 13888-3¹⁸, proposed by International Organization of Standardization support non-repudiation but do not support fairness [15].

A non-repudiation protocol can be seen as a special instance of a fair exchange protocol. But there are some inherent differences pointed out in [15]. In a fair exchange protocol the description of the items is known in advance, while in non-repudiation protocol the recipient does not expect a particular message and the description of the message is revealed only at the end of the protocol. Moreover, the recipient does not exchange the item, but only the evidence of receipt. In a fair exchange protocol both the item and the evidence are required. This implies that non-repudiation protocols can be implemented more efficiently than fair-exchange protocols.

5) Confidentiality and secure digital signature algorithm

Confidentiality implies that the signing protocol neither reveals what it signs, nor with what it signs, i.e. the encryption key.

In our context an SLA agreement is being signed. Partners in a VO conclude individual SLA agreements corresponding to their specific role. Each partner may conclude several agreements –

¹⁶ISO/IEC 13888-1, Information Technology – Security techniques – Non-repudiation – Part 1: General (1997).

¹⁷ISO/IEC 13888-2, Information Technology – Security techniques – Non-repudiation – Part 2: Mechanisms using symmetric techniques (1998).

¹⁸ISO/IEC 13888-3, Information Technology – Security techniques – Non-repudiation – Part 3: Mechanisms using asymmetric techniques (1997).

each per role and, perhaps, with different VOs. Therefore, it is important that the contents of the agreement are not intercepted by an outsider. Hence, an encryption of the agreement should be provided by the protocol.

Confidentiality is realized by secure digital signature algorithms. A secure digital signature scheme is crucial for the implementation of the secure signing protocol [24]. The result of signing is a digital signature. A party's commitment to an SLA is basically determined by his/her digital signature to it; "digital contract signing is essentially implied by fair exchange of digital signatures between two potentially mistrusted parties". [25]. It is also desirable that the signature algorithm is standard/compatible. Then it is possible to integrate the protocol into the existing software [25]. However, the contract signing problem (see 4.2 for the definition) can not be solved by cryptography alone [31].

We do not take into account any specific digital signature algorithms in our solution. Nor do we analyze any for their security strengths. We assume that a digital signature algorithm used in the implementation of the protocol is secure and efficient. Information about various digital signature algorithms based on verifiable encryption technique and their efficiency is provided in [29, 30].

6) Communication Channel

Protocols are often assumed to work on a specific type of a communication channel. We consider three types of communication channels [15]: *unreliable channel (U)* (no security or performance assumptions need to be made, e.g. the Internet), *resilient channel (R)* (delivers correct data after an unknown but finite amount of time), and *operational channel/reliable channel (O)* (delivers correct data after a known amount of time). Since VOs operate in highly heterogeneous networks, we assume that the SLA signing protocol needs to operate in an unreliable channel.

Communication channels can be classified as synchronous or asynchronous [29]. *Synchronous* means that something is coordinated in time. On synchronous networks, messages are guaranteed to be delivered within a "round", so that the recipient of the message can unambiguously decide whether the message was sent or not. *Asynchronous* implies that data are not being synchronized with respect to an external clock. There is no notion of global time and no time-limit on the time needed for message transmission. The clocks of the sender and the receiver are independent and not synchronized either. A recipient of any message on an asynchronous network cannot decide whether the message was sent or not since messages may be reordered arbitrary and delayed for an

unknown amount of time. This implies that the termination of the protocol cannot be guaranteed on asynchronous networks unless the protocol logic deals with that.

7) Timeliness

Timeliness insures that the protocol will be finished after a finite amount of time, i.e. it will not be pending for ever. By definition, a protocol provides timeliness “if and only if all honest parties always have the ability to reach, in a finite amount of time, a point in the protocol where they can stop the protocol while preserving fairness” [15]. To decide whether the protocol supports timeliness, one can ask the question: Can A and B at each moment in the protocol take an action to force a fair termination? [15].

To provide timeliness in asynchronous networks a wakeup call is used for the protocol to stop waiting for pending messages and terminate with a correct output. Wakeup can be implemented by a local time-out or by an interaction with the user [29]. In synchronous networks timeliness is provided.

8) Abuse-Freeness

Abuse-freeness means that even “if the protocol is not executed successfully, none of the two parties can show the validity of the intermediate results to others” [25]. It implies that at no stage during the protocol’s execution can parties show what the terms of the contract are to an outsider, until after the contract is signed [31].

A dishonest party may want to engage in the signing process without really meaning to, and instead show the partial commitments to an outsider in order to obtain some benefits. For example, consider a situation of job offers. If A offers a job to B with a certain salary, B can turn around and show the offer to C. C may decide to raise the salary to B and create a counteroffer. B then may take this counteroffer and show it to A again, etc.

9) Efficiency

Last but not least, protocols differ greatly when it comes to efficiency [31]. By efficiency we understand a number of messages exchanged between the parties and TTP (in case of with TTP

protocols), the computation time required to process the messages received, the number of rounds, and the feasibility of implementation of the protocol under question.

The time-complexity of a synchronous protocol is the number of rounds required for its execution. The time-complexity of an asynchronous protocol is the time required for its execution if transmission of each message requires one time unit and local computations require no time [29].

It has been proven that no optimistic contract signing scheme with only two messages exist [29]. The message-optimal optimistic scheme for synchronous networks requires three messages and three rounds in the optimistic case using a stateless TTP. A round-optimal synchronous scheme requires 2 rounds but 4 messages in the optimistic case and a stateless TTP. A time-optimal asynchronous scheme terminates in time 3 and requires 2×3 messages in the optimistic case. The TTP is state-keeping. An asynchronous optimistic contract signing protocol with 3 messages in the optimistic case does not exist. There is no asynchronous optimistic contract signing protocol with time 2 either. However there is an asynchronous contract signing protocol with 4 messages and time 2 but with an inline TTP. The latter is an optimal non-optimistic scheme according to [29]. For information concerning the proof of the facts presented in this paragraph see [29].

5 Contract Signing Protocol Survey

At a Glance:

In this chapter a survey of contract signing protocols is presented. The survey is structured according to the involvement of a TTP. We start with no TTP protocols, and then move to the protocols with an inline, online and, finally, offline optimistic TTP.

TTP protocols are also called ‘optimistic’ protocols. Because of their proven practical advantages a number of the optimistic contract signing protocols described in the survey is greater than those of other types.

The result of our survey is summarized in the protocol-property matrix. In the matrix each protocol is depicted in relation to the contract signing protocol properties described in chapter 4.3.

5.1 Contract Signing Protocols with no Trusted Third Party

Contract signing protocols with no TTP are signing protocols where no TTP is used to mediate or resolve any execution or network error. Even with the lack of a TTP this kind of contract signing protocols or signature exchange protocols try to achieve security (fairness, non-repudiation, for instance) but with less certainty. The best-case scenario of such a protocol execution is: most probably the correct contract contains both parties' correct signatures. Thus, these kinds of protocols are usually referred to as probabilistic protocols.

Eliminating the need of a TTP has non-trivial advantages. First, it makes the signing process more efficient and faster, since a TTP will be representing a bottleneck when many signing processes are run under its supervision. This is true because some types of TTPs are extensively involved in the signing process (inline and online TTPs). Another advantage of eliminating the TTP is freeing the parties from the burden of fully trusting a TTP. Not any TTP can be blindly trusted since it is difficult to assess the risks introduced by using a TTP [16]; especially if it is a state-keeping TTP (i.e. stores information about the signing protocol instance). This might be a serious issue for the privacy of the involved parties and the confidentiality of the contract.

Having no TTP and gaining the mentioned advantages does not come for free, but at the expense of accepting probabilistic contract correctness¹⁹. It has been shown by Yacobi and Even [13] that there is no non-deterministic²⁰ protocol where the participation of the TTP is not required. It is so, because there is a stage in the protocol execution where one party will have his counterparty's signature without committing himself to the contract. So, if the advantageous party stops the execution the principle of fairness is violated. Interestingly, it has been noted that signing protocols without TTP appeared long after the protocols with TTP, since using a TTP as a center of cancellations in case of any unfair state is a natural solution. It is quite logical, since early trials of contract signing were done in private business environments, where a TTP could be trusted and would not act as a bottleneck. In the beginning of the 90's, the Internet started to play an important role in building up complex business and commercial environments [15]. Complex business transactions were built involving multiple parties. This implied that transactions needed to be faster and more efficient. One way to realize that was by eliminating the TTP.

¹⁹'Correctness' in this context implies a correct contract, with the true signatures of parties involved in it.

²⁰In [13], 'non-deterministic' is a synonym to 'probabilistic'.

There are plenty of contract signing protocols that use no TTP, but not all of them are presented in this thesis. The reason for this is that many no TTP protocols assume equal computation power of participating parties. Because of this assumption we are not considering them in the end choice. Next we will present some contract signing protocols that use no TTP. The presentation will follow the chronological order, from older to newer.

5.1.1 Even (1982)

Definitions:

In his protocol [14], Shimon Even describes a contract signing protocol that uses randomization; it relies on the infeasibility of some number operations and on public key cryptography which is commonly regarded as safe. This protocol uses two concepts, Rabin's signature concept [17], and Merkle's concept of a puzzle [18]. The puzzle is a message pair $(m, F_k(m))$, where F is an encryption function that encrypts the message m using the key k . It is required to find k . The key k must be unique with very high probability. This can be done by choosing a short key relative to m . The puzzle is solved by exhaustion; controlling hints (e.g. revealing some bits) about the key.

Assumptions:

This protocol assumes that the two involving parties are interested in the exchanging transaction and that items exchanged (signatures, in our case) are of equivalent values to both parties. The cost of computation is assumed to be the same for both parties, and as mentioned above, the availability of a secure conventional cryptosystem is assumed also.

Execution:

The execution is presented between parties: Alice (A), and Bob (B), where A is assumed to be the initiator.

A prepares $2N$ puzzles. N is a value (e.g. $N=50$), and S is a known sample message, both S and N are known to all. K is the key to be used to encrypt ($x=F_k(m)$) and decrypt ($F_k^{-1}(x)$).

1st Step – A prepares $2N$ puzzles, $F_{k_1}(S), F_{k_2}(S) \dots, F_{k_{2N}}(S)$ where N is a known number in the network (e.g. $N=50$), and $2N$ encoded signed copies of a contract C , $F_{k_1}(D_A(C,1)), F_{k_2}(D_A(C,2)), \dots, F_{k_{2N}}(D_A(C,2N))$. $D_x(C,i)$, is the i^{th} signed copy (signed by party 'x') of contract C .

where $(C,i) 1 \leq i \leq 2N$ is the i^{th} copy of C , and formed by appending the bit string representing i to C . Both the $2N$ puzzles and signed copies of C are then sent to B. B does the same operation and sends his generated puzzles and signed copies to A.

2nd Step – A chooses any i , $1 \leq i \leq 2N$, and asks B to reveal his k_i . B does so, and A can verify if the received key is right by decrypting the i^{th} puzzle of B, and then try after that decrypting B's signature of the i^{th} copy. B acts similarly.

This step is repeated $N+1$ times, because it is conventionally made that B is formally committed to A on a contract c , if B has received any $N+1$ of $2N$ signatures sent by A to him.

The $\{ F_{k_i}(S), F_{k_i}(D_A(C,i)) \}$ is called the i^{th} unit of a message sent from A to B in step 1. A unit received by B will by default considered *bad*, if:

- The puzzle key size is different from that agreed upon before; i.e. A is making the puzzle harder, and thus cheating.
- k_i is not the solution of A's i^{th} puzzle. Or, the signature is not verified upon decoding.

Once a bad unit is detected, B will terminate the protocol execution on the assumption that A is trying to cheat.

Protocol Properties:

In a correct and honest execution of this protocol, both parties will have each other signatures on the contract and both are acknowledged with that. If the protocol is stopped prematurely with no reason (cheating, or other problem) the uncooperative party will have a slight advantage over the other in computing the other party's signature. This advantage can be kept very small fraction of V (V is the contract value), by choosing an enough large N (for instance, for $N=50$, then this advantage can be expressed as 4% of V). If a party tries sending 'm' bad units purposely, then the probability that his cheating will be exposed during the first k exchanges is $1 - P$;

$$\text{where } P = \frac{\binom{2N - m}{k}}{\binom{2N}{k}}$$

5.1.2 Markowitch- Roggeman (1999)

The goal of this protocol is to design a probabilistic non-repudiation protocol accepting probabilistic fairness for the benefit of avoiding a trusted third party. The Markowitch-Roggeman protocol is iterative in the sense that no party can get any benefit (gain more privileges) except at the last iteration.

Definitions:

The authors define non-repudiation services, based on the “repudiation” definition in [20]; it was defined as the “*denial by one of the entities involved in a communication of having participated in all or part of the communication*” [20]. In transaction protocols (e.g. contract signing protocols), non-repudiation resembles two ensuring actions: Non-repudiation of origin, and non-repudiation of receipt, both defined in the following²¹:

Definition 1: *Non-repudiation of Origin (NRO):* To make sure that the originator of the message cannot deny the fact that he has sent the message.

Definition 2: *Non-repudiation of Receipt (NRR):* To make sure that the recipient of the message cannot deny the fact that he has received the message.

The problem of non-repudiation of receipt is similar to the problem of fair message exchange, where the originator waits for an acknowledgment that the message was received. So, in this protocol the recipient needs a NRO, while the originator needs NRR for every message exchanged. To know if this problem is similar to the fair exchange problem, one must know what is meant by “fair” exchange in addition to other well know properties that must be respected in any fair exchange protocol. These are fairness, time-bounded (timeliness), and viability.

Definition 3: *Fair:* A protocol is said to be fair, if at each step of the execution, either both parties receive the expected items, or none receives any (or any information of value concerning the expected items).

Definition 4: *Time-bounded:* A protocol is said to be time-bounded (also referred to as respecting timeliness) if whenever at least one party behaves correctly, then the protocol finishes within a finite amount of time.

Definition 5: *Viable:* A protocol is said to be viable, if both parties behave correctly and complete the protocol run normally, and at the end of the execution each party will receive his expected item.

Protocol Description:

This protocol is to run between the two parties A, and B. During the protocol, the following evidences will be generated: (l is the protocol unique ID, i is the iteration step number, and v_i is the value sent at i^{th} iteration)

- Evidence of Origin of a ciphertext c : $\text{EOO} = \text{Sig}_A(B, l, c)$
- Evidence of Receipt of ciphertext c : $\text{EOR} = \text{Sig}_B(A, l, c)$
- Evidence of Origin of a value v_i : $\text{EOO}_i = \text{Sig}_A(B, l, i, v_i)$
- Evidence of Receipt of a value v_i : $\text{EOR}_i = \text{Sig}_A(A, l, i, v_i)$

²¹These definitions are mentioned here for they are assumed by the protocol authors, even though they are similar to the definitions in Chapter 4. The rest of the thesis assumes the definitions of Chapter 4.

Then the NRO and NRR will be:

$$\text{NRO} = \{ \text{EEO}, \text{EEO}_i \} \text{ and } \text{NRR} = \{ \text{EOR}, \text{EOR}_i \}.$$

Before initiating the protocol, A goes through a setup phase where she chooses a random number n according to a geometric distribution. This number is kept secret by Alice during the whole protocol run. n will denote the number of iterations of the protocol. After that, $n-1$ random, independent and equi-distributed²² values r_i and a key k are chosen, both r_i and k are of the same size.

Suppose that A wants to send a message m to B. A initiates the protocol first by sending $c = C_k(m)$, which generates the ciphertext of the message to be sent. The cipher is accompanied with a non-repudiation of origin evidence. B acknowledges the received message by sending an evidence of non-repudiation of receipt. Then A sends value r_1 along with its NRO_1 , and then B acknowledges with NRR_1 , then r_2 , etc...

The process continues until A receives the NRR_{n-1} of r_{n-1} . A sends the key k , and B acknowledges it with NRR_n . Then B will wait for the next message from A, but there will be none. After a time-out period B knows that that was the last message, i.e. the key k . Then B is able to decipher c using k .

It is apparent that B gets nothing useful before the last step, so he has no interest in terminating the protocol prematurely. The time required by each party to wait for a message must be estimated correctly. A will not wait B long enough for him to try r_i on c . After A's timeout elapses and A does not receive the acknowledgment (NRR), A will consider that B is cheating by trying to decipher c using the sent r_i .

Protocol Execution:

<i>Step</i>	<i>Message Direction</i>	<i>Message Content</i>
1	A ->B	B, l, c, EOO
2	B ->A	A, l, EOR
3	A ->B	B, l, l, r1, EOO _{k,l}
4	B ->A	A, l, EOR _{k,l}
...

²²Equi-distributed means that r 's have values of equal frequency of occurrence.

<i>Step</i>	<i>Message Direction</i>	<i>Message Content</i>
2n-1	A ->B	$B, l, n-1, r_{n-1}, EOO_{k,n-1}$
2n	B ->A	$A, l, EOR_{k,n-1}$
2n+1	A ->B	$B, l, n, k, EOO_{k,n}$
2n+2	B ->A	$A, l, EOR_{k,n}$

Any incorrect message received by any party will be considered as an error (cheating or network failure), so the recipient will have to stop the protocol. The same case applies for any party sending a message, if transmission did not start after some (maybe publicly known) deadline, the recipient can consider this delay as a cheating attempt or a network problem, and can thus stop running the protocol.

The number of iterations n , as mentioned before, is secret. It is assumed that B cannot deduce n from A's messages or by any other means. But still there is a probability that B may guess n , this probability is denoted as $\hat{\delta}$; $\hat{\delta} = P(i=n)$. If this is the case, B can decipher c and refuse to send back EOR_n . Thus, B will have the non-repudiation of origin evidence of m , whereas A will not get any non-repudiation of receipt evidence of m and, hence, can not create the EOR.

Protocol Properties:

It should be noted that this protocol's fairness heavily depends on two things: time-out, and cryptosystem used. The time-out period must be enough to get a message sent taking into consideration network characteristics. Using the deadline (time-out) concept for sending and receiving makes it possible to use this protocol in unreliable networks. When it comes to the cryptosystem, the ciphering should be such that it is impossible to partially decipher the ciphertext and get a meaningful partial plaintext. Cryptosystem to be used must force the deciphering of all messages before getting anything of value.

At the $(2n+1)^{\text{th}}$ iteration, if the protocol is stopped, no party can gain anything of value (i.e. EOR and EOO). The probability that B will stop accidentally at the $(2n+2)^{\text{th}}$ step is $\hat{\delta}$. Therefore, we can say that this protocol is $\hat{\delta}$ -fair.

5.1.3 Mitsianis (2001)

In his paper "A New Approach to Enforcing Non-Repudiation of Receipt" [19], J. Mitsianis proposed a protocol that is similar to Markowitch-Roggeman in [16]. The only difference is that n

(the number of iterations or protocol rounds) is static to all protocol runs, while in Markowitch-Roggeman it is up to the sender to stop or continue the protocol after any completed iteration.

Assumptions:

It is assumed that both users share a common session key K_π . Encryption is assumed to be *all-or-nothing* mode, where the cipher can not be decrypted unless all key bits are known.

Protocol Description:

In this protocol A wants to send B a message m . A encrypts m with a secretly chosen session key K_1 , then sends the cipher C to Bob. A waits for B to send his NRR (see **Definition 2**). After A receives the NRR, she appends a padding x to the session key K_1 to have a new key K'_1 . The size of x is secretly chosen by A. Then using the shared (between A and B) session key K_2 , A encrypts the new padded session key to get a new key, K_3 , and sends it to B. The mode of ciphering done at this phase is all-or-nothing mode. After A chooses randomly and secretly n - the number of protocol rounds. Using this value A splits K_3 into n k_i parts of different random sizes. The real protocol execution part involves n 2-way transactions (send k_i and receive NRR_i). At the end, B concatenates K_i 's to get K_3 and then decrypt it with K_2 . Knowing the size of K_2 , B can extract the padding x . Then the left bits constitutes K_1 , used to decipher C and get m .

Note that even when Bob knows the size of K_1 , he needs to concatenate all K_i 's to get K_1 . This is due to the all-or-nothing mode used in ciphering, in addition to the secrecy of the padding size. The probability that Bob can read m without sending back the last NRR (NRR_i) is $1/n$. Each message (whether a K_i or an NRR_i) has a deadline or time-out after which the waiting party can assume a cheating attempt or network error and terminate the execution. In 错误! 未找到引用源。 the event flow is sketched.

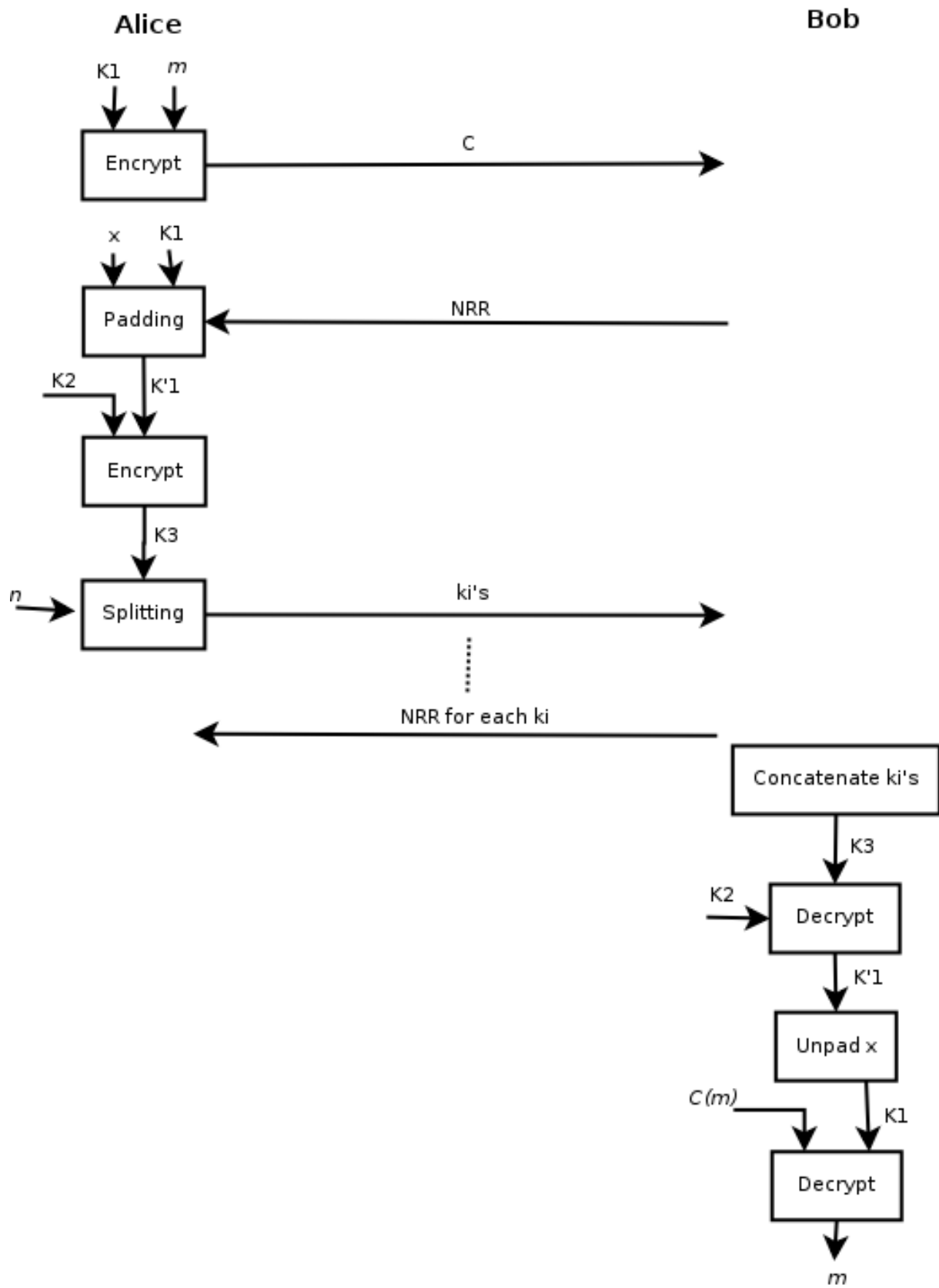


Figure 11: Mitisianis Protocol

5.2 Contract Signing Protocols with a Trusted Third Party

In this section we describe a number of contract signing protocols that require the intervention of a TTP. For the definition of a TTP see chapter 4.3.

5.3 Protocols with an Inline TTP

If a TTP is involved in each message's transmission during the protocol execution it is said to be an *inline* TTP [15].

5.3.1 Coffey and Saidha protocol (1996)

In this non-repudiation protocol [15] the TTP is used as a non-repudiation server. It means that the TTP collects non-repudiation evidences and transmits them to two signing parties, called A and B hereafter. The protocol uses *partial evidences*, i.e. the evidences are the part of the final non-repudiation evidences. It makes use of time-stamps produced by a time-stamping authority. The time-stamping authority adds a time-stamp to partial evidence signed by the entity with whom it communicates. The time-stamping authority does not consider the contents of the received messages.

In order to distinguish between different protocol sessions using the same TTP, a random value is associated with each protocol session, which is produced and transmitted by the TTP to the signing parties. A and B must present this random value to the TTP during each communication with it.

Protocol Description

1. A initiates the protocol. A submits to the time-stamping authority the signed partial non-repudiation of origin evidence to be dated and signed by this authority.
2. If A's signature is valid, the time-stamping authority replies with the non-repudiation of origin evidence, ciphered for A.
3. If this non-repudiation of origin evidence is valid, A requests the TTP to initialize a non-repudiable communication and sends the complete non-repudiation of origin evidence as well as the partial non-repudiation of receipt evidence of the same message. A computes the partial non-repudiation of receipt evidence to the TTP in order to decrease load of the TTP.
4. If A's message and signature are valid, the TTP sends the partial non-repudiation of receipt evidence to B.

5. B signs this partial non-repudiation of receipt evidence and sends it to the time-stamping authority.
6. If B's signature is valid, the time-stamping authority responds with the complete non-repudiation of receipt evidence to B.
7. B submits the complete non-repudiation of receipt evidence to the TTP.
8. After checking, the TTP sends the non-repudiation of origin evidence to B and the non-repudiation of receipt evidence to A.

The protocol has:

- State-keeping TTP;
- strong fairness, because A and B never communicate directly but, instead the TTP collects all necessary information and then forwards to A and B;
- confidentiality of the message exchanged (the message is the SLA agreement in our case);
- resilient communication channels between each of the entities A and B and the TTP;
- If the deadline is fixed for each sending, then the protocol respects timeliness.

Disadvantages of the protocol:

- Since the TTP is state-keeping, the management of large databases by the TTP is required. The databases contain sensitive information like the forwarded messages and the times of each event. The management of such centralized databases with sensitive information represents a significant security risk.
- A TTP involved in each transmission creates a maximal potential bottleneck.

Advantages of the protocol:

- ❖ An inline TTP can provide information about the time a message was sent or received as evidence to a judge.

5.3.2 An Optimal Asynchronous Non-Optimistic Scheme (1998)

This protocol [29] is a message- and time-optimal asynchronous protocol with an inline TTP. It is based on the TTP storing and forwarding the contract signatures to the signatories. The protocol requires four messages in two time units. As mentioned before, time in an asynchronous protocol is the time required for its execution where the transmission of each message requires one unit of time

and the local computation requires none [29]. Four messages of this protocol take two time units because they are sent in parallel two and two.

Notation \mathbf{tid}_x means Transaction Identifier and is a unique identifier for the protocol run.

Protocol Description

1. Each signatory A and B sends a signed message $m_{1A} := \text{Sig}_A(C_A)$ and $m_{1B} := \text{Sig}_B(C_B)$ respectively containing the contract text C to the TTP.
2. If a wakeup call occurs (since the network is asynchronous, timeliness is provided by a wakeup call), a message $m_3 := \text{sig}_A(\text{wakeup})$ or $m_3 := \text{sig}_B(\text{wakeup})$ is sent.
3. The TTP waits for m_{1A} and m_{1B} and verifies whether $C_A = C_B$ and $\mathbf{tid}_A = \mathbf{tid}_B$. If this is the case the TTP sends $m_2 := \text{sig}_T(m_{1A}, m_{1B})$ to A and B. If the checks fail or the wakeup call is received before the m_{1A} and m_{1B} , the TTP sends $m'_2 := \text{sig}_T(\text{rejected})$, meaning that the signing procedure has failed. Schematically this protocol is presented in the figure below.

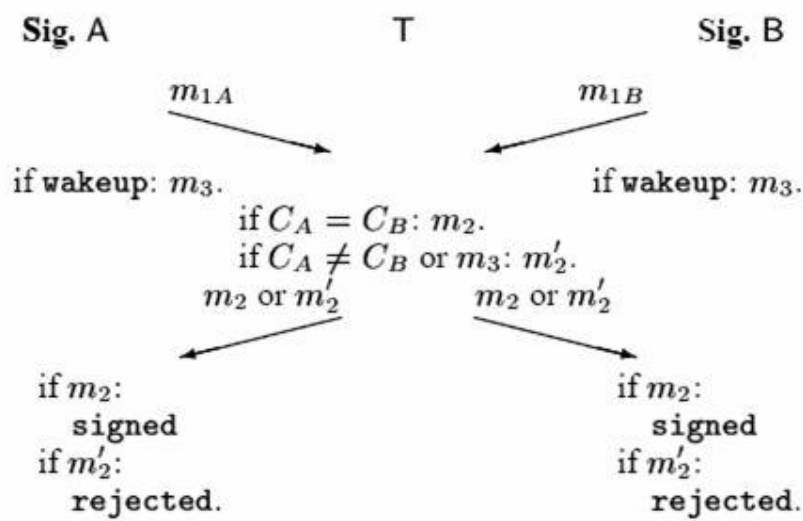


Figure 12: Behavior of the Optimal Asynchronous Scheme with inline TTP

The protocol supports:

- State-keeping TTP
- Strong fairness
- Confidentiality if the contract is encrypted
- Asynchronous communication model, resilient channel between all parties
- Timeliness property
- Optimal efficiency for an asynchronous fair contract signing protocol with an inline TTP

Disadvantages of the protocol:

- A TTP involved in each transmission creates a maximal bottleneck.
- A TTP is state-keeping and thus can be a potential weak point both from the point of view of an attacker and with respect to trust. It stores sensitive information, i.e. contract signatures.

Advantages of the protocol:

- ❖ This protocol was proven to be the most efficient of its kind.

There are other inline TTP protocols, e.g. ZG-3 [24]. We refer to the protocol matrix to see the summary of its properties.

5.4 Protocols with an Online TTP

If a TTP is involved in each session of the protocol but not in each message's transmission, it is said to be *online* [15].

In this subchapter we have described several protocols with an online TTP. Notice how the function of the TTP differs in those.

5.4.1 Rabin's Random Beacons (1983)

This protocol is the first exchange protocol that uses a TTP [21, 39]. The protocol is based on the use of random beacons (a TTP in this case). TTP in this protocol is a machine that picks a random message called *beacon* at every time interval t and broadcasts it to the world. Beacon is signed and is composed of n public keys, a *deciphering key* D_k that can decipher one of the public keys sent in the previous broadcasting and a value j indicating which previous public key it can decipher ($j \in \{1, 2, 3, \dots, n\}$).

A and B have to realize their complete round of the protocol between two broadcastings of the TTP, i.e. between the time intervals zt and $(z+1)t$, where $z \in \mathbb{N}$. The point of the protocol is that A and B must both guess the same number at the same time interval.

Protocol Description

1. A sends to B a message ciphered with the session key k .

2. B randomly chooses an integer i ($i \in \mathbb{N}$), time-stamps it, and sends i to A with a NRR evidence for the message (sent in 1).
3. A sends to B the session key k ciphered with the i^{th} public key broadcasted in the last beacon. This whole message is signed by A.
4. The TTP broadcasts the next time-stamped beacon.
5. If the value $j=i$ is true, then B can retrieve k and decipher A's message. The NRO and NRR are composed of the signed messages sent during this round associated with the beacon.
6. If $j=i$ is not true, then A and B will have to wait to the next beacon is broadcasted and try protocol execution again.

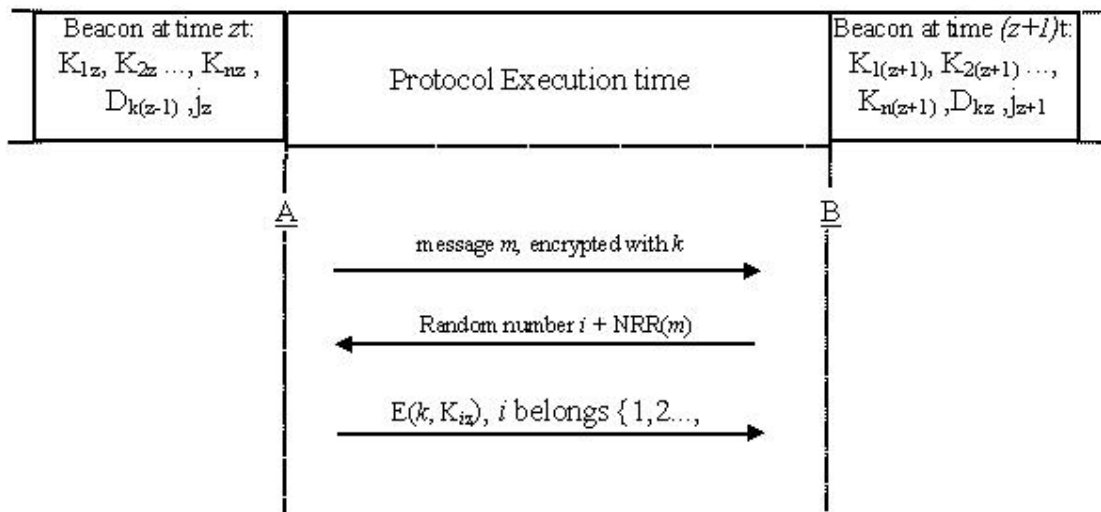


Figure 13: Rabin's Beacon

The protocol supports:

- A stateless TTP
- Probabilistic fairness. Fairness is broken when B and TTP choose the same integer, i.e. $i=j_{z+1}$. This can happen with the probability $1/n$. This way, B can ignore sending the generated i and NRR, and use i to decrypt the message directly.
- Probabilistic version of timeliness. The probability that $i \neq j_{z+1}$ decrease for each round but is never zero. The expected number of rounds is n .
- Confidentiality. The message initially sent remains confidential and can be read only by B.
- No abuse freeness, since any party can decide any time to stop the protocol execution and use available intermediate results.

Disadvantages of the protocol:

- The man in the middle attack is possible. To prevent it A and B can add the recipient's identity when they send their signed messages [15].
- A, B and TTP need to be synchronized. This is a heavy constraint to realize. The deadline between two broadcastings of the TTP has to be calculated in such a way, that even the entity with the smallest computing power can provide the necessary message within the interval.

Advantages of the protocol:

- ❖ This protocol belongs to the category of not-so-trusted TTP protocols. This implies that the TTP does not need to be trusted very much and can still be used. Particularly, this TTP is unable to forge a contract.

This protocol suits to the setting where the recipient does not need to know the contents of the message. Rather the fact that the message was exchanged in a confidential manner is of importance.

5.4.2 Zhang and Shi protocol (1996)

Protocol Description [15]

A transmits to B a message ciphered with a session key k and ciphered again with B's public key K . The TTP publishes in a public board the session keys that are provided to the TTP by A. The time when this happens is decided by B and agreed by A. These session keys are used to retrieve A's message.

In this protocol the TTP manages the database containing the keys and records time when these keys are added in that database. In case of disputes the judge contacts the TTP. For this reason the TTP can not delete any information from the database.

The protocol supports:

- State-keeping TTP
- In case of unreliable channel the protocol does not support fairness.
- Confidentiality with respect to B of the message (signed contract) transmitted since it is ciphered twice, and confidentiality with respect to the TTP
- Timeliness
- No abuse-freeness

Disadvantages of the protocol:

- The database that the TTP has to manage grows infinitely and is the major security hole.

Advantages of the protocol:

- ❖ The protocol provides double confidentiality of the message: to B and to the TTP

5.4.3 Zhou and Gollmann protocol 1 (1996)

The assumption of this and the next protocol is that A and B do not play against themselves.

In this protocol [ZG-1 in 25] c is the cipher of the contract m , i.e. m is encrypted under the key k . l is a unique label for the protocol run, and $S_X(z)$ is a message z signed by a participant X . The symbol “->” means “sends”. $A \leftarrow TTP: Z$ means that A fetches Z from B using “ftp get” operations or some web browser operations. f_{xxx} is a flag denoting the purpose of each step in the protocol.

The main idea of this protocol is that c is sent first as a commitment that the protocol will be executed to its end. Then k is released to unlock the message.

The following shortenings are used:

$NRO = S_A(f_{NRO}, B, l, c)$ – non-repudiation of origin evidence

$NRR = S_B(f_{NRR}, A, l, c)$ – non-repudiation of recipient evidence

$sub_K = S_A(f_{SUB}, B, l, k)$ – evidence of submission of the key

$con_K = S_{TTP}(f_{CON}, A, B, l, k)$ – confirmation evidence for the key issued by the TTP

Protocol Description

1. $A \rightarrow B: f_{NRO}, B, l, c, NRO$
2. $B \rightarrow A: f_{NRR}, A, l, NRR$
3. $A \rightarrow TTP: f_{SUB}, B, l, k, sub_K$
4. $B \leftarrow TTP: f_{CON}, A, B, l, k, con_K$
5. $A \leftarrow TTP: f_{CON}, A, B, l, k, con_K$

A initiates the protocol by sending the encrypted contract c in step 1. It can be an encrypted signed SLA for example. B confirms the receipt but cannot read c . In step 3 A submits the key to the TTP, and the sub_K is the proof of this submission. The TTP stores the tuple (A, B, l, k, con_K) in some read-only directory accessible to public. In step 4 B gets the key and in step 5 this is confirmed to A. These two last steps can be performed in any order.

The protocol supports:

- State-keeping TTP
- Strong fairness
- Non-repudiation through the non-repudiation evidences
- Confidentiality of the message (signed contract), but no confidentiality of the encryption/decryption key
- No timeliness because the protocol does not specify any time limits, it has no deadline. It is not clear when the protocol can be considered as terminated, aborted or in progress. That is why it is not clear when the adjudication process can be initiated.
- No abuse-freeness, i.e. intermediate results can be shown to an external party.

Disadvantages of the protocol:

- The protection of the encrypted SLA sent in the first message lies in the strength of the symmetric encryption algorithm. If the encryption algorithm and the key are weak, B might guess the message (signed SLA) after step 1 and not reply in step 2. Since timeliness is not provided B can use infinite time to try out possibilities, e.g. use a brute force attack. In other words, B can just break c , drop the protocol, get the signed contract and repudiate having done so.
- The key is revealed to the public, so an intruder may intercept it and abuse it if the same key is used at some later step of the application. Even malicious B can eavesdrop the key in step 3 and then revoke the signature sent in step 2.
- The TTP stores sensitive information in its database that can be compromised by an intruder. Moreover the DB can grow infinitely which implies extra costs for the TTP.

5.4.4 Zhou and Gollmann protocol 2 (1996)

The idea behind this protocol [21 and ZG-2 in 25] is to reduce the work of an online TTP to a minimum. If, during the protocol, an incorrect message arrives or an expected message does not arrive at all, the potential recipient (A or B) stops the protocol execution. The protocol looks very much like the previous one, but supports the timeliness property.

Protocol Description

1. A \rightarrow B a message ciphered with the session key k , a label identifying the protocol session, a time-out value before which the session key must be submitted to the TTP and after which the TTP can be consulted, and the signed NRO (for the ciphered message).

2. If B accepts the time-out value, B \rightarrow A signed NRR for the ciphered message.
3. A \rightarrow TTP a signed copy of the session key k .

The TTP checks A's signature and the time-out value. The TTP accepts only one submission from a party (A or B) during a protocol session.

4. After the time-out B can get the session key k and the NRO for the k . The NRO for the k is necessary in order to build a complete NRO for A's message.
5. Similarly, A consults TTP to build A's complete NRR: a NRR send by B in 2 and the NRO of the k .

Here is the summary of the protocol taken from [25, ZG-2]

$NRO = S_A(f_{NRO}, B, l, T, c)$ – non-repudiation of origin evidence

$NRR = S_B(f_{NRR}, A, l, T, c)$ – non-repudiation of recipient evidence

$sub_K = S_A(f_{SUB}, B, l, T, k)$ – evidence of submission of the key

$con_K = S_{TTP}(f_{CON}, A, B, l, T, T_0, k)$ – confirmation evidence for the key issued by the TTP (i.e. non-repudiation evidence for the key)

T is the time-out value on the TTP's clock and T_0 is the time the confirmed key has been made available to the public. The key remains available to the public until T.

1. A \rightarrow B: f_{NRO}, B, l, T, c, NRO
2. B \rightarrow A: f_{NRR}, A, l, NRR
3. A \rightarrow TTP: $f_{SUB}, B, l, T, k, sub_K$
4. B \leftrightarrow TTP: $f_{CON}, A, B, l, T_0, k, con_K$
5. A \leftrightarrow TTP: $f_{CON}, A, B, l, T_0, k, con_K$

The protocol supports:

- State-keeping TTP. Both A and B get their complete evidences (NRO and NRR) through the TTP.
- Strong fairness, if the channel between the TTP, A and B is resilient.
- Non-repudiation through the non-repudiation evidences.
- Confidentiality of the signed contract but not of the encryption/decryption key.
- Resilient channel between the TTP and the entities A and B. Only if the channel between all parties is resilient the strong fairness property is preserved.
- Timeliness, if the channel between the A, B and the TTP is resilient.
- No abuse-freeness

Disadvantages of the protocol:

- Since the TTP holds the sensitive information (the session key k) it automatically constructs a possible attack target.
- The protection of the encrypted SLA sent in the first message lies in the strength of the symmetric encryption algorithm. If the encryption algorithm and the key are weak, B might reveal the message (signed SLA) after step 1 and not reply in step 2, thus getting around the non-repudiation and fairness.
- The key is sent unencrypted and is revealed to the public, so an intruder may intercept it and abuse it if the same key is used at some later step of the application. Even malicious B can eavesdrop the key in step 3 and then revoke the signature sent in step 2.
- An attack compromising fairness is possible if A is malicious. This is referred to as “termination problem” in [24]. Since $T_0 < T$, A might delay the step 3 up to the last moment before T, so that A can perform the step 5 while B will miss the step 4.

There are more protocols with an online TTP presented in the literature. We refer to the protocols ZG-5 and ZG-6 in [24]. The summary of the properties of these protocols is given in the protocol matrix.

5.5 Protocols with an Offline/Optimistic TTP (including transparent TTP)

An optimistic/transparent/offline TTP does not participate in honest executions at all; it only intervenes in case of problems like cheating or communication stop [23]. When such a problem occurs, A or B invokes the TTP to help them to finish the protocol in a fair way. Optimistic/offline/transparent TTP protocols are based on the assumption that most of the time the entities behave honestly and do not need the TTP’s intervention. A *transparent TTP* produces evidences (NRO and NRR) that are indistinguishable from the evidences A and B produce in a faultless case. It means that at the end of the protocol, by only looking at the produced evidences, it is impossible to say whether the TTP did intervene or not. This can be especially useful in the context of electronic commerce in order to avoid bad reputation [15].

The protocols with an optimistic/offline/transparent TTP usually consist of three sub protocols. The first is the main protocol. This one is used in a normal execution and is sufficient if nothing unexpected happens during the protocol flow. The second one is the recovery protocol. It is used when some problem occurs and the TTP gets involved to provide evidences to A and B. The third is

the abort protocol. It is used when the party that has initiated the protocol run wants to stop it fairly. It also informs the TTP about this intention. The recovery and the abort protocols are mutually exclusive.

Before we proceed to protocols' description, let's introduce some common notations [15].

A->B: transmission from entity A to entity B; read A "sends" to B;

h(): a collision resistant one-way hash function;

E_k(): a symmetric-key encryption function under key k;

D_k(): a symmetric-key decryption function under the key k;

E_X(): a public-key encryption function under X's public key;

D_X(): a public-key decryption function under X's private key;

S_X(): the signature function of X;

m: message sent;

k: the session key;

c=E_k(m): the cipher of m under k;

l=h(m, k): a label that in conjunction with (A,B) uniquely identifies a protocol run;

f: a flag indicating the purpose of a message;

5.5.1 Fair Non-Repudiation Protocol Respecting Timeliness

This is a generic version of a fair non-repudiation protocol using an offline TTP [15]. This protocol respects the timeliness property. A simpler version without the timeliness property can be found in [21, p.17-19] and its properties are outlined in the protocol matrix.

The protocol consists of three sub protocols: the main, recovery and abort. Both A and B can launch the recovery sub protocol, while only A (the initiator) can launch the abort sub protocol at any moment.

The following evidences are generated in this protocol:

the evidence of origin for the cipher: **E_{OO}**=S_A(f_{E_{OO}}, B, TTP, l, h(c));

the evidence of receipt for the cipher: **E_{OR}**=S_B(f_{E_{OR}}, A, TTP, l, h(c));

the submission evidence for key k: **Sub**=S_A(f_{Sub}, B, l, E_{TTP}(k));

the evidence of origin for key k: **E_{OO_k}**=S_A(f_{E_{OO_k}}, B, l, k);

the evidence of receipt for key k: **E_{OR_k}**=S_B(f_{E_{OR_k}}, A, l, k);

the recovery request (where X and Y are A or B): **Rec_X**=S_X(f_{Rec_X}, Y, l);

the confirmation evidence for key k: **Con_k**= S_{TTP}(f_{Con_k}, A, B, l, k);

the abort request: $\mathbf{Abort} = S_A(f_{\text{Abort}}, B, l)$;

the abort confirmation evidence: $\mathbf{Con}_a = S_{TTP}(f_{\text{Con}_a}, A, B, l)$;

Protocol Description

Main protocol

1. A->B: $f_{\text{EEO}}, f_{\text{Sub}}, B, \text{TTP}, l, c, E_{\text{TTP}}(k), \text{EEO}, \text{Sub}$

If A times out, then A starts the abort protocol

2. B->A: $f_{\text{EOR}}, A, \text{TTP}, l, \text{EOR}$

If B times out, then B starts the recovery protocol

3. else A->B: $f_{\text{EEO}_k}, B, l, k, \text{EEO}_k$

If A times out, then A starts the recovery protocol

4. else B->A: $f_{\text{EOR}_k}, A, l, \text{EOR}_k$

In the steps 1 and 2 the cipher c (of the message m encrypted with the key k), the evidence of origin for c and the evidence of receipt of c are exchanged. The key k is encrypted with the TTP's public key and sent over to B in step 1. In the steps 3 and 4 the key k , the evidence of origin for k and the evidence of receipt for k are exchanged. A and B choose themselves their time-out values. If the second message does not arrive to A within the chosen amount of time, A executes the abort protocol. If the messages 3 or 4 do not arrive to B and A respectively, they can execute the recovery protocol.

Abort protocol

1. A->TTP: $f_{\text{Abort}}, l, B, \text{Abort}$.

If the protocol has already been aborted or recovered, then stop

2. Else TTP->A: $f_{\text{Con}_a}, A, B, l, \text{Con}_a$

3. TTP->B: $f_{\text{Con}_a}, A, B, l, \text{Con}_a$

In the abort protocol the TTP checks if the current protocol has not yet been aborted or recovered. If this is the case then the TTP informs both A and B that the current protocol has been aborted. As it was introduced under common notations the protocol run is identified by the label l and the identities of A and B. It is worth noticing that it is possible to complete a faultless main protocol and then launch an abort protocol. The abort evidence does not mean that the exchange did not take place. It merely means that the recovery protocol can not be launched by any of the entities any more.

The goal of the recovery protocol is to substitute the missing evidences for the entities A or B. Either A or B can start launching the recovery protocol. B can start it after the step 1 in the main protocol, while A after the step 2. B will then get the missing k and a substitution for $E_{OO_k} - Con_k$, and A will get EOR and a substitution for $E_{OR_k} - Con_k$.

Recovery protocol

Assume that X and Y is either A or B

1. $X \rightarrow TTP: f_{RecX}, f_{Sub}, Y, l, h(c), E_{TTP}(k), Rec_X, Sub, EOR, EOO$

If the protocol has already been aborted or recovered, then stop

2. Else $TTP \rightarrow A: f_{Conk}, A, B, l, k, Con_k, EOR$

3. $TTP \rightarrow B: f_{Conk}, A, B, l, k, Con_k$

After the step 1 of the main protocol B can either stop the protocol (does nothing), continue the main protocol or launch the recovery protocol. If B stops the protocol no party will obtain correct evidence. A would not stop the protocol after the step 1, because it would brake our basic assumption that no party acts against its own interests (see the section “Contract Signing Protocols’ Properties”, paragraph 1). At any time A can start the abort protocol. After step 2 both A and B can launch the recovery protocol. If the entity A wants to cheat by providing a wrong key k in $E_{TTP}(k)$ (step 1, main protocol), the evidence Con_k in the recovery protocol will be invalid.

The protocol supports:

- State-keeping TTP
- Strong fairness
- Non-repudiation
- No confidentiality. Confidentiality, however, can be provided easily by either ciphering c and k with B’s public key or by using SSL/VPN mechanisms.
- Channel between the TTP and A and between the TTP and B is resilient; channel between A and B is unreliable
- Timeliness, since both A and B at any time can terminate the protocol preserving fairness. This is achievable due to the fact that the channels between the TTP and the entities A and B are resilient. Otherwise, if the channels were unreliable, the timeliness would not be supported.
- No abuse-freeness. B can take c and EOO and show it to someone else.

Disadvantages of the protocol:

- The evidences that the parties get from the TTP differ from the ones they obtain during the normal protocol run. The case when no party was malicious but had to launch the recovery protocol due to e.g. network problems could damage the party's reputation. This is especially relevant in the case of TrustCoM where the reputation of the potential service providers is checked and constantly updated.

If we “translate” this protocol into our need to sign an SLA then the signed SLA is c . The protocol has to be executed 2 times in parallel to exchange the signatures from both parties.

5.5.2 Non-Repudiation Protocol with Transparent TTP (based on Markowitch-Kremer protocol)

In this protocol [15] a GPS (Girault-Poupard-Stern) signature scheme is used²³. The signature is issued in two phases. In the first phase the signer u produces a committed signature $\text{ComSig}_u(m)$. This committed signature is turned into a final signature $\text{FinalSig}_u(m)$ by the signer or by the TTP. The verifier v can check the validity of the committed and final signature.

The protocol consists of four sub protocols: the main, recovery, abort and error. Both A and B can launch the recovery sub protocol, while only A (the initiator) can launch the abort sub protocol at any moment. Error sub protocol is launched from the recovery sub protocol in case the hash values of the key do not correspond.

The following evidences are generated in this protocol:

the evidence of origin: $\mathbf{EOO}=\text{ComSig}_A(f_{\text{NRO}}, B, \text{TTP}, l, h(c), h(k));$

the evidence of receipt: $\mathbf{EOR}=\text{ComSig}_B(f_{\text{NRR}}, A, \text{TTP}, l, h(c), h(k));$

the non-repudiation of origin evidence: $\mathbf{NRO}=\text{FinalSig}_A(f_{\text{NRO}}, B, \text{TTP}, l, h(c), h(k));$

the non-repudiation of receipt evidence: $\mathbf{NRR}=\text{FinalSig}_B(f_{\text{NRR}}, A, \text{TTP}, l, h(c), h(k));$

the evidence of submission for the key k : $\mathbf{Sub}=\text{S}_A(f_{\text{Sub}}, B, l, E_{\text{TTP}}(k));$

the abort request: $\mathbf{Abort}=\text{S}_A(f_{\text{Abort}}, B, l);$

the recovery request: $\mathbf{Rec}_X=\text{S}_X(f_{\text{Rec}_X}, Y, l);$

²³For more explanation see [21 p.23], G. Poupard, J. Stern, “Security Analysis of a Practical “on the fly” Authentication and Signature generation” in Advances in Cryptology: Proceedings of Eurocrypt’98, Vol. 1403 of Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 422-436 and M. Girault, “Self-Certified Public Keys” in Advances in Cryptology: Proceedings of EuroCrypt’91, Vol. 547 of Lecture Notes in Computer Science, Springer-Verlag, 1991, pp. 490-497.

the abort confirmation: $\mathbf{Con}_a = S_{TTP}(f_{Cona}, A, B, l)$;

the error confirmation: $\mathbf{Con}_e = S_{TTP}(f_{Cone}, A, B, l)$;

Main protocol

1. A->B: $f_{EOO}, f_{Sub}, B, TTP, l, h(k), c, E_{TTP}(k), EOO, Sub$
If A times out then A launches the abort protocol
2. B->A: f_{EOR}, A, TTP, l, EOR
If B times out then B launches the recovery protocol
3. A->B: f_{NRO}, B, l, k, NRO
If A times out then A launches the recovery protocol
4. B->A: f_{NRR}, A, l, NRR

In step 1 of the main protocol A transmits the cipher c of the message m (an SLA) encrypted with the session key k , the hash of the session key, the session key ciphered with the TTP's public key, the committed signature EOO and the evidence of the session key k 's submission. In step 2 B checks the EOO by using the GPS algorithm and checks Sub. Then B sends his committed signature EOR. If A does not receive message in step 2, A times out and initiates the abort protocol. As in the previous protocol the time-out values are chosen individually by A and B. If step 2 is executed correctly, A sends the key k and the final signature (the final non-repudiation of origin evidence) in step 3. Likewise, B sends his final signature in step 4. In case B times out and does not receive the third message or the third message is incorrect, B calls for the recovery protocol. A does the same in case the fourth message does not come or is incorrect.

As in the previous protocol, A can launch the abort protocol at any time. As for the recovery protocol, B can initiate it already after the first message, while A – after the second message. B can stop (do nothing) the protocol at any time, while A can stop it after the message 2 has arrived.

Abort protocol

1. A->TTP: $f_{Abort}, l, B, Abort$

If the protocol has already been aborted or recovered, then stop

2. else TTP->A: f_{Cona}, A, B, l, Con_a
3. TTP->B: f_{Cona}, A, B, l, Con_a

A would usually launch the abort protocol if A does not receive message 2 in the main protocol. This would prevent B from initiating the recovery protocol later. Neither A nor B receive the final signatures (neither NRR nor NRO), so the protocol remains fair.

Recovery protocol

1. X->TTP: $f_{\text{Rec}_X}, f_{\text{Sub}}, Y, l, h(c), h(k), E_{\text{TTP}}(k), \text{Rec}_X, \text{Sub}, \text{EOR}, \text{EOO}$

If $h(k) \neq h(D_{\text{TTP}}(E_{\text{TTP}}(k)))$ then launch the error protocol

If the protocol has already been aborted or recovered, then stop

Else

2. TTP->A: $f_{\text{NRR}}, A, l, \text{NRR}$

3. TTP->B: $f_{\text{NRO}}, B, l, k, \text{NRO}$

The point behind the recovery protocol is that both entities obtain their counterpart's final signature as if nothing wrong has occurred. The initiator X sends to the TTP the hash of the ciphered message, the hash of the session key k , the session key ciphered for the TTP with the TTP's public key and the four signatures $\text{Rec}_X, \text{Sub}, \text{EOR}, \text{EOO}$. The TTP checks all four signatures first. If at least one signature is incorrect, the recovery request is ignored. B can not cheat here by sending a wrong session key k , because Sub and EOO are signed by A. Then the TTP verifies the hash of the key k by comparing the hash taken from the EOO with the hash it computes on its own (the key is extracted by decrypting $E_{\text{TTP}}(k)$). If the values are not equal, A is trying to cheat. In that case the TTP launches the error protocol which informs B that A is trying to cheat. Otherwise the TTP checks whether neither the abort nor the recovery protocol have yet been performed. If not, the TTP converts the committed signatures into the final signatures using its private key and sends them to A and B. In this way, the protocol remains fair. It is worth noting that the data always arrive to A and B thanks to the resilience of the channels between the TTP and A/B.

Error protocol

Aborted=true

1. TTP->A: $f_{\text{Con}_e}, A, B, l, \text{Con}_e$

2. TTP->B: $f_{\text{Con}_e}, A, B, l, \text{Con}_e$

The goal of the error protocol is to inform B that A is trying to cheat and to inform A that A's attempt to cheat has been detected (optional message). Neither A nor B will get the final non-repudiation evidences, so the protocol remains fair.

The protocol supports:

- State-keeping TTP
- True fairness, because only by looking at the evidences at the end of the protocol it is impossible to decide whether the TTP did intervene or not;
- Non-repudiation through NRO and NRR evidences.
- No confidentiality of the key that can be easily provided however, see matrix (step 3, main protocol)
- Channel between the TTP and A and TTP and B is resilient; channel between A and B may be unreliable
- Timeliness. Both A and B have their time-out mechanisms and the channel between the TTP and both A and B is resilient.
- No abuse-freeness. After getting the first message B can stop the protocol (the protocol will be eventually aborted by A) to show this first intermediate result to an outsider.

Another protocol that makes use of a transparent TTP has been proposed by Markowitch and Saaednia²⁴. It is claimed to be the most efficient protocol for fair exchange with transparent TTP [15]. Unfortunately we could not find this protocol in a free version. The protocol just described is based on Markowitch-Saaednia method.

5.5.3 Optimistic Fair Contract Signing Protocol (Based on the “Optimistic Fair Exchange of Digital Signature” by N. Asokan, V. Shoup, M. Waidner²⁵)

The idea of this protocol was proposed by N. Asokan, V. Shoup, and M. Waidner and taken up in [26]. This protocol is claimed to be simple, elegant and efficient.

It is assumed that m is the body of the contract. TTP is always trusted and eventually reachable even if there are network problems. A, B and TTP can be mutually authenticated. The main idea behind this protocol is to exchange the *precontracts* first. A precontract is a commitment that the contract will be signed. Once A receives B’s precontract after sending hers, A knows that B has committed to signing the contract and A can now sign the real contract without a risk. If B would

²⁴ O. Markowitch, S. Saaednia, “Optimistic Fair Exchange with Transparent Signature Recovery” in 5th International Conference, Financial Cryptography 2001, Lecture Notes in Computer Science, Springer-Verlag, 2001.

²⁵ N. Asokan, V. Shoup, M. Waidner, “Optimistic Fair Exchange of Digital Signature”, IEEE Journal of Selected Areas in Communication, vol. 18, no. 4, 2000

not sign the real contract in return, A can go to the TTP. The TTP can produce the signed contract on the basis of two precontracts.

A precontract is defined as follows:

A's precontract: $=S_A(m, A, B, TTP)$

B' precontract: $=S_B(m, A, B, TTP)$

The protocol distinguishes between a *standard contract* and a *notarized contract*. A standard contract is completed without the TTP's interference while the notarized one with the TTP's help. Both types of contracts are equally valid. The recovery and abort protocols are mutually exclusive, i.e. if one of them has already been executed, the other will be rejected.

Main protocol

If A and B behave as expected and no messages get lost then the protocol is executed without the involvement of the TTP (see Figure 14: Overview of Optimistic Fair Exchange). The main protocol's execution results in a standard contract $=\{S_A(m, A, B), S_B(m, A, B)\}$.

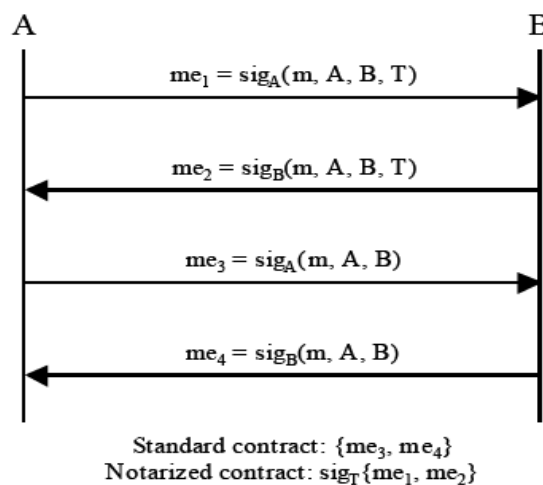


Figure 14: Overview of Optimistic Fair Exchange

When is the time of the agreement formation for each party? In other words, at what point in time the contract signing process cannot be reversed? For B it is the time of receiving message me_3 . As soon as B gets me_3 B can create his own signature and have a valid contract. For A it is when it receives me_2 . By having the two precontracts A can always obtain a notarized contract.

Recovery protocol

If something goes wrong during the execution of the main protocol, a party A or B who has both precontracts can ask the TTP for assistance. If the contract digest and signatures are correct, the

TTP can issue an alternative form of a valid contract by signing the two precontracts. In such a way a notarized contract is created (see Figure 15: Recovery Subprotocol).

Notarized contract= $S_{TTP}(S_A(m, A, B, TTP), S_B(m, A, B, TTP))$.

For example, if B sends his precontract me_2 but does not receive me_3 , B can turn to the TTP and ask to resolve the situation. The TTP checks whether the protocol has been aborted before, and if so returns a signed abort message. Otherwise, the TTP returns a notarized contract. The similar scenario happens if A sends me_3 but does not receive me_4 .

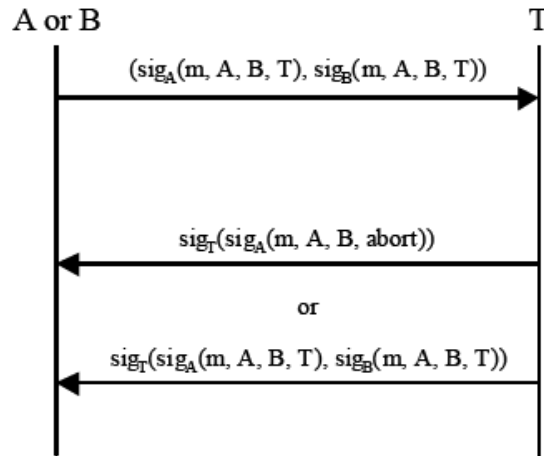


Figure 15: Recovery Subprotocol

Abort protocol

If A sends me_1 to B, but does not receive me_2 , A can ask the TTP to abort the protocol (see Figure 16: Abort Subprotocol). The TTP checks whether the recovery protocol has been executed prior to that. If so, the TTP returns a notarized contract to A. Otherwise the TTP records the abort of the contract and returns an abort message to A.

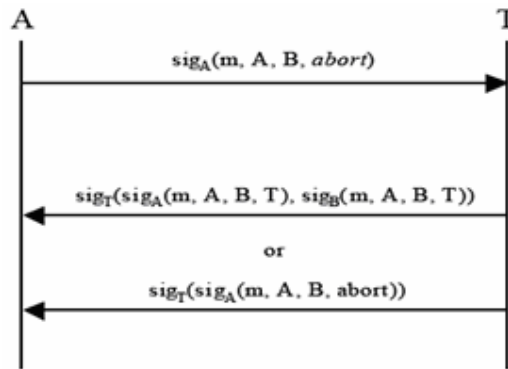


Figure 16: Abort Subprotocol

It is worth noticing that only A can abort the protocol. This is reasonable because it is A who makes the commitment by sending the precontract first. It is A who puts itself in a vulnerable position. Because B cannot abort the protocol, A can be sure that the contract is agreed on the arrival of m_{e2} .

The protocol supports:

- State-keeping TTP, the TTP has to keep track of which protocols were aborted, recovered or signed
- Strong fairness
- Non-repudiation evidences
- Confidentiality if the contract is encrypted
- Resilient channel between the TTP and the entities A and B, unreliable channel between A and B
- No timeliness
- No abuse-freeness
- High efficiency and simplicity

Disadvantages of the protocol:

➤ The timeliness property is not taken into account. Eventual time-out mechanisms should be implemented.

Advantages of the protocol:

❖ The main advantages of this protocol are its efficiency and simplicity. There is also material available on how to implement it using Web Services.

5.5.4 A Time-Optimal Asynchronous Scheme

This protocol is a time-optimal asynchronous contract signing scheme. The proofs of this are described in [29]. It terminates in three time units and requires 2×3 messages totally in the optimistic case. As mentioned before, “time” in the asynchronous network model is the time required for the execution of the protocol if transmission of each message requires one time unit and the local computation time can be ignored.

In this protocol C denotes the contract text and m_x a message in a round x.

Main protocol (see Figure 17: Optimistic Behavior of the Time-Optimal Asynchronous Protocol)

1. The signatory A starts the protocol by sending a signed contract in a message $m_{1A} = S_A(C_A)$.
2. If A receives m_{1B} from B with an identical contract, A sends $m_{2A} = S_A(m_{1A}, m_{1B})$.
3. If A receives m_{2B} , A sends $m_{3A} = S_A(m_{2A}, m_{2B})$.

4. After receiving m_{3B} A can claim the contract signed.

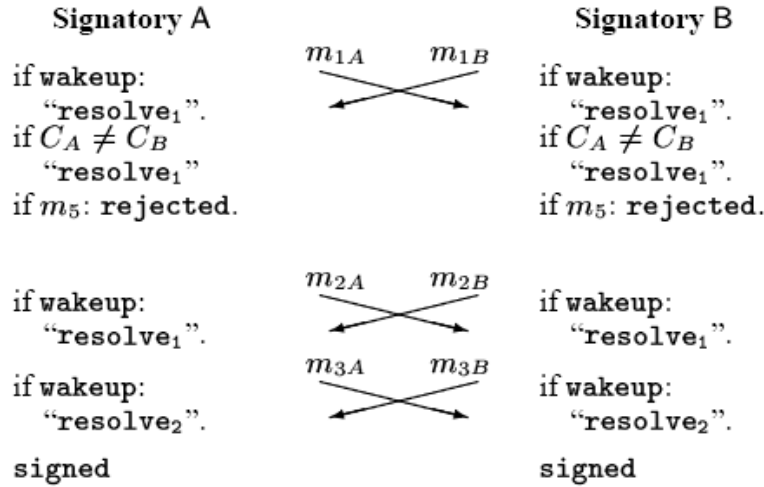


Figure 17: Optimistic Behavior of the Time-Optimal Asynchronous Protocol

If the signatory B is trying to cheat by sending a wrong contract in m_{2B} or m_{3B} , where $C_A \neq C_B$, B's messages are ignored.

In the asynchronous network messages can be reordered arbitrary. If m_{2B} is received before m_{1B} , A sends both m_{2A} and m_{3A} . If m_{3B} is received before m_{2B} , m_{3A} is sent and the contract is declared as signed. If a m_{1B} is received with a different contract before m_{2B} or a wake-up signal occurs at A's site before A sends m_{2A} , the abort protocol is started by sending $m_{4A} = S_A(m_{1A})$. If a wake-up signal occurs after m_{2A} has been sent but before m_{3A} , the recovery protocol is initiated by sending $m'_{4A} = S_A(m_{2A})$. Otherwise the recovery protocol is started by $m''_{4A} = S_A(m_{3A})$.

Abort protocol

1. The signatory A sends m_{4A} to the TTP to abort the protocol.
2. If the protocol has already been aborted or recovered, the TTP resends the previous decision. Otherwise the TTP sends an abort acknowledgement $m_5 = S_{TTP}(m_{4A})$ and changes to the aborted state of the protocol.

Recovery protocol (when started by m'_{4A})

1. The signatory A sends m'_{4A} to the TTP.
2. If the protocol has already been aborted or recovered, the TTP resends its previous decision. Otherwise the TTP creates an affidavit (a substitution of the signatory's signature) $m'_5 = S_{TTP}(m'_4)$.

3. After receiving m'_5 the signatory A treats the protocol as signed and the TTP changes the protocol status to “recovered”.

Recovery protocol (when started by m''_{4A})

This recovery protocol is used to complete the contract when it is clear that the signatories have committed to it.

1. A signatory (A or B) sends a message m''_{4A} to the TTP.
2. The TTP check whether the protocol has already been aborted or recovered and, if so, resends the previous decision. If this is not the case the TTP produces an affidavit $m''_5 = S_{TTP}(m''_{4A})$ and saves the protocol as recovered.
3. After receiving m''_5 the signatory treats the protocol as signed.

The protocol supports:

- State-keeping TTP. The TTP has to keep track of which protocols were signed, aborted or recovered.
- Strong fairness
- Non-repudiation
- Confidentiality if the contract is encrypted
- Asynchronous channel between all entities
- Timeliness
- No abuse-freeness
- High efficiency that has been proven

Disadvantages of the protocol:

- It is not clear what channels there are (resilient/unreliable) between the entities and the TTP.
- There is no info on implementation.

Advantages of the protocol:

- ❖ This protocol is proven to be a time-optimal asynchronous optimistic contract signing protocol.

There are more protocols with offline TTP. We present their properties in the protocol matrix.

For example, there is a message optimal protocol for a synchronous network model. In this protocol a contract is signed using 3 messages in 3 rounds. The TTP is stateless. There is a generic protocol for fair exchange [21], Optimistic Fair Contract Signing [23], Zhou and Gollmann 4 [24], Abuse-Free Fair CSP Based on the RSA [25] and many more.

5.6 Protocol Matrix

In the protocol matrix some protocols are typed in black and some in grey color. Those that are in black have been described in our survey, while those in grey have been analyzed but not described in detail due to space restrictions. A notation N* means that the property is originally not provided but with some simple modifications in the protocol the property can be provided.

Protocols	TTP	Stateless/state-keeping TTP	Fairness	Non-repudiation	Confidentiality	Channel between A and B	Channel to TTP	Timeliness	Abuse-freeness	Efficiency & implementation-related notes
Even (1982) [14]	-	-	P		Y	?	?	?Y	N	No assumption
Markowitch-Roggeman (1999) [16]	-	-	P	Yes	Y	u/o	-	Y	N	Not very effective, because of the large number n of iterations decided by the sender.
Mitsianis (2001) [19]	-	-	P	Yes	Y	U	-	Y	N	A number of iterations n is static.
Contract signing based on PSE (1985) [22]	-	-	P	?	Y	?	-	?	?	
Coffe & Saidha (1996) [15]	Inline	SK	S	Y	Y	R	R	N	N	TTP uses time-stamps. A risk of bottlenecks created by the TTP.
Optimal Asynchronous Non-Optimistic Scheme (1998) [29]	Inline	SK	S	?	Y	U/Asyn	U/Asyn	Y	N	Inline protocols are generally not very efficient because of the bottlenecks they create.
Zhou & Gollmann 3 (1996) [ZG-3 in 24]	Inline	SK	S	Y	N	?	?	Y	N	Strong signing algorithm is essential for this protocol bec. the message is sent in plaintext. Otherwise the protocol would break fairness and non-repudiation properties.
Rabin's beacons (1983) [15]	Online	SL	P	Y	Y	O	O	prob.	N	Since the number of rounds is not set, the proto time can be very long.
Zhang & Shi (1996) [15]	Online	SK	S	?	Y	R	R	Y, if R channel	N	The TTP has to support infinite DB and be synchronised with A/B.
Zhou & Gollmann 1 (1996) [ZG-1 in 24]	Online	SK	S	Y	Y	?	?	N	N	The strength of the protocol lies in the symmetric encryption algorithm and the key. The key is sent non-encrypted and

										can be intercepted. The info in the DB of the TTP can be compromised.
Zhou & Gollmann 2 (1996) [21, ZG-2 in 24]	Online	SK	S	Y	Y	R	R	Y	N	The strength of the protocol lies in the symmetric encryption algorithm and the key. The key is sent non-encrypted and can be intercepted. The TTP holds and stores sensitive information. The protocol is susceptible to the “termination problem”.
Zhou & Gollmann 5 (1998) [ZG-5 in 24]	Online	SK	S	Y	Y	?	?	?	N	Rests on the validity of digital signatures insured e.g. by a trusted time-stamping authority (i.e. the TTP). The TTP checks the validity of A’s and B’s PKI certificates and publishes the key at time T. 5 messages required.
Zhou & Gollmann 6 [ZG-6 in 24]	Online	SK	S	Y	Y	?	?	Y	N	B retrieves key sent by A via a TTP based on the Diffie-Hellman public key distribution scheme. 3 different time-limits are use. 5 messages required.
Fair non-repudiation 1 (2002) [15]	Offline	SK	S	Y	N*	U	R	N	N	3 messages in the main and 3 in the recovery sub protocol are required. Key is sent in clear. A has no chance to abort the protocol, risk that the session will be pending infinitely. Has to be executed x2 times.
Fair non-repudiation 2 (2002) [15]	Offline	SK	S	Y	N*	U	R	Y	N	Key is sent in clear. 4 messages in the main, 3 in the abort and 3 in the recovery sub protocols. Has to be executed x2 times.
Non-repudiation with transparent TTP (2001) [15]	Transparent	SK	T	Y	N*	U	R	Y	N	Issues signatures in 2 phases: committed and final. Requires the signer and the TTP to choose large integers (multiples of primes) in the initialization phase. 4

										messages - main, 3 – abort, 3 –recovery, 2 - error. x2 times execution needed.
Generic for fair exchange (1997) [21]	Offline	SK	W	Y	N*	U, synchr.	R, synchron.	Y, has <i>active time</i> limit at TTP's site.	N	The protocol is not symmetrical. Guarantees only weak fairness for A if no item exchanged is revocable or generatable by the TTP. For B and otherwise fairness is strong. Uses random values as commitments. Clocks of A, B and TTP have to be synchronized. 5+ mess. - main, 4-recovery1, 3 – recovery2. The TTP provides a reliable channel between A and B in the recovery sub protocol.
Optimistic fair contract signing [23]	Offline	SK	S	Y	Y	U	?	Y	?	Proved to be unfair in [6, 7], though claimed to be fair.
Zhou & Gollmann 4 [ZG-4 in 24]	Offline	?	S	Y	Y	U	?	Y	N	
Abuse-Free Fair CSP Based on the RSA [25]	Offline	SL	S	Y	Y	U	O	Y	Y	
Optimistic Fair Contract Signing (2003) [26]	Offline	SK	S	Y	Y	U	R	N	N	Has two types of contracts: standard and notarized. A precontract is created before the final contract. The protocol is simple and efficient.
Garay-Jakobsson-Mackenzie protocol based on PCS (2001) [27]	Offline	SK	S	Y	Y	Asynch., U	Write-protected, asynch., O	Y	Y	The DB maintained by the TTP is visible to the intruder. The abuse-freeness in this protocol is referred to as 'balance'.
A time-optimal asynchronous scheme [29]	Offline	SK	S	Y	Y	Assynch./U	Assynch./U	Y	N	High efficiency has been theoretically proven.
A message-optimal protocol [29]	Offline	SL	S	Y	Y	U	U	Y	N	
Fair Exchange of Digital Signatures with	Offline	SK, need	S	? Y thro	Y	O	O	Y, if channel	N	The protocol is based on 'verifiable encryption' of digital signatures. It means

Efficient Verifiable Encryption (1998) [29]		s to store key		ugh EQ_ DLOG				is reliable, otherwise N		that the signature is encrypted under a designated public key in such a way that it is possible to prove subsequently that the resulting ciphertext indeed contains such a signature. The signature protocols using v.e. are claimed and proven in [29] to be extremely efficient. Several signature schemes are analyzed for efficiency. Only 4-message main protocol for contract sign. is presented, no abort or recovery, which makes it incomplete.
---	--	----------------	--	--------------	--	--	--	--------------------------	--	--

Table 11: Protocol Survey Matrix

6 The Choice

At a Glance:

Now that we have presented different protocols with their properties and the environment where our ultimate protocol is supposed to run – TrustCoM SLA subsystem - it is time to make a choice. In this chapter we reflect on the security requirements proposed in 3.8 and argue why and how we have arrived at our choice of the contract signing protocol.

As it was mentioned several times before it is important to keep in mind that the protocol should be compatible with the requirements imposed by the application environment. The protocol should be acceptable or unacceptable on its own terms. If the protocol terms (assumptions, properties it provides) do not match the specific situation, the protocol is incompatible with that situation but is neither wrong, nor flawed [24]. In our case the environment is TrustCoM architecture in general and TrustCoM SLA signing subsystem in particular. By analyzing its security setting with the help of abuser and user stories we have come to a set of security requirements the SLA signing protocol should support (see chapter 3). These are:

1. The existence of the TTP
2. Abuse-freeness provided by the protocol or by the TrustCoM environment
3. The protocol should be fair
4. The parties should not be able to repudiate their involvement in signing
5. The signing process should be atomic
6. The digital signature algorithm involved in the signing should be secure
7. The protocol should be able to run on unreliable and asynchronous networks, which is what the Internet is.

To the above criteria it can be added that

8. The protocol should be simple, implementation feasible and efficient.

The last criterion is derived on the basis the general information about VO environment (see chapter 2.3). This is not a security-related requirement as such but it is related to the practical usefulness of the protocol. If the protocol provides the optimal security but is too troublesome to comprehend the security holes in the implementation are almost guaranteed. In addition the optimal security might not be needed in a particular environment.

Let us reflect on each requirement in turn and see how it affects our choice.

The involvement of the TTP guarantees that the fairness is weak, strong or true. In protocols with no TTP the parties need to possess equal computational power (that was claimed to be unrealistic in 4.3 and 5.1) or the fairness provided is probabilistic. In our case the probabilistic fairness simply would not do as we are signing an SLA and have to be sure that in the end it is either fully signed or not signed at all.

We have mentioned in 2.4.2 that TrustCoM framework presumes the involvement of a notary in the signing process. It has also been mentioned that the decision to use the notary was not final. The very idea of the notary in the TrustCoM architecture promoted us to choose a protocol with a TTP.

Since we propose a protocol with a TTP, the need for a notary as an architecture component should be reconsidered by the TrustCoM developers.

We have decided neither to choose an inline nor an online TTP protocol. This is because inline and online TTPs always create greater bottlenecks in the network than optimistic/offline/transparent TTPs do. Relying on many TTPs, as a possible solution to that, is not advisable for TTPs should be trusted with cautiousness [23]. Inline/online TTPs are also more expensive to run: they must offer services on a 24-hour basis, and maintain a bandwidth capable of handling a potentially enormous traffic [23]. They should carry substantial liability insurances, because being involved in every session or transmission such TTPs need to use highly volatile memory and other precautions. For instance, a computer crash at the TTP's site may cause a complete stop in the traffic with disastrous consequences for the signatories, which the TTP would have to pay for [23]. Moreover, we have assumed that most of the parties participating in the VO creation would behave honestly and the involvement of the TTP would be rarely needed. It is in the parties' interests to act honestly, because otherwise their reputation will hinder them to create alliances with other partners in the future. *Therefore, our choice protocol will have an offline/transparent TTP.* We remind the reader that protocols with an offline TTP are also called optimistic.

It has been proven in [29] that no asynchronous optimistic contract signing scheme with a stateless TTP exists. It means that although a state-keeping TTP is a potential weak site from the point of view of an attacker, we cannot avoid it because we want to have an optimistic protocol. *Therefore, the TTP in our protocol will be state-keeping.*

There are different degrees of fairness (see 4.3). We have already said that the probabilistic fairness will not do in our case, nor will the weak fairness. We cannot allow the parties to end the contract signing in a state where one party has obtained the other's signature while the counterparty has not, even if the proof of this fact was sent and received. *Therefore, our choice protocol will possess strong or true fairness.*

Abuse-freeness implies that the parties cannot abuse the intermediate results of the signing process (see chapter 4.3). If we look at TrustCoM environment and the way the subcomponents function, we will see that the reputation plays a central role in the partner selection procedure and is constantly monitored and updated. The situation where a party would cheat by abusing the intermediate results of the signing procedure to get a better SLA within the same enterprise network (EN) is very unlikely, because by the time a cheater would approach another party for a better bid

the cheater's reputation would be updated and no party would like to deal with it. So, in this way abuse-freeness is provided by the TrustCoM environment. That is why *when choosing a protocol we will not consider abuse-freeness as a necessary property*, which, besides, makes any protocol more cumbersome to implement.

The signatories should not be able to repudiate their involvement in the signing process at any time after the protocol run is completed. *The chosen protocol should provide non-repudiation evidences to both parties*. As we can see from the Table 11: Protocol Survey Matrix, all optimistic contract signing protocols surveyed provide non-repudiation.

The atomicity of the contract signing process at a logical level is guaranteed by the protocol's strong fairness. However, the atomicity at the transaction level can only be achieved if the fair protocol is implemented properly, i.e. there are no security holes in the programming code and the modules are integrated correctly. In other words, *the chosen protocol should be strongly fair and should be atomic in its implementation*.

As it was mentioned in 4.3 we do not make any particular assumptions about the digital signature algorithm. The only thing *we assume is that digital signature algorithm used in our chosen protocol will be secure and efficient*. Which particular scheme is to be used, can be a future research avenue.

The main characteristic of VOs is that they operate in heterogeneous networks. In TrustCoM SLAs are supposed to be signed across the Internet by dynamically allocating communication channels between signatories. This points out that *our chosen protocol should support the unreliable communication channel between the signatories*.

There are two protocol properties that have not been embraced by the security requirements generated in the analysis: timeliness and confidentiality (see 4.3).

We think that the *confidentiality of the contract text transmitted in the unreliable network should be ensured*. If not, there is a risk that a malicious party will intercept the SLA. There is a risk for an impersonation attack as well. It is also a plus if the confidentiality of the contractual text with respect to the TTP is provided. This will exclude the possibility that the TTP will modify the contract.

The protocol run should not be pending for ever. In other words, *the chosen protocol should provide the timeliness property.*

There are several protocols that satisfy our demands (consult Table 11: Protocol Survey Matrix). We have the TTP protocols Fair non-repudiation 1 [15], Fair non-repudiation 2 [15], Non-repudiation with transparent TTP [15], Optimistic fair contract signing [23], ZG-4 [24], Abuse-Free Fair CSP Based on the RSA [25], Optimistic fair contract signing [26], A time-optimal asynchronous scheme [29], A message-optimal protocol [29], etc.

We have not only considered to what degree the protocols fulfill the properties discussed, but also how implementation-feasible, simple and efficient the protocols are. As the protocol matrix shows there are several protocols that could be chosen. However, when one goes deeper and tries to understand how one or another protocol should be implemented, many practical questions arise. (Refer to the protocol references at the end of this thesis for more details). For these reasons we have decided to choose Optimistic Fair Contract Signing [26], which is easy to understand, is elegant and efficient. Parts of the protocol have been implemented in Java and XML. To see the code, we refer to the appendix.

6.1 The Protocol Modifications

The chosen protocol supports all our requirements except timeliness. However, by modifying the protocol, i.e. inserting time-out values in each round, this property can be provided. The time-out values can be decided by each party individually.

Let us briefly remind the reader the initial version of this protocol [26].

The protocol consists of the main subprotocol and recovery and abort sub-protocols. The recovery and abort sub-protocols are executed only in case of an error. The error can be that the party does not receive a message, receives an incorrect message or a network crash occurs.

The main idea behind the protocol is to exchange the *precontracts* first, which serve as the commitment that the contract will be signed and as the non-repudiation evidences. A *standard contract* is obtained when the TTP does not interfere. In the opposite case a *notarized contract* is produced.

The three initial sub protocols are depicted below.

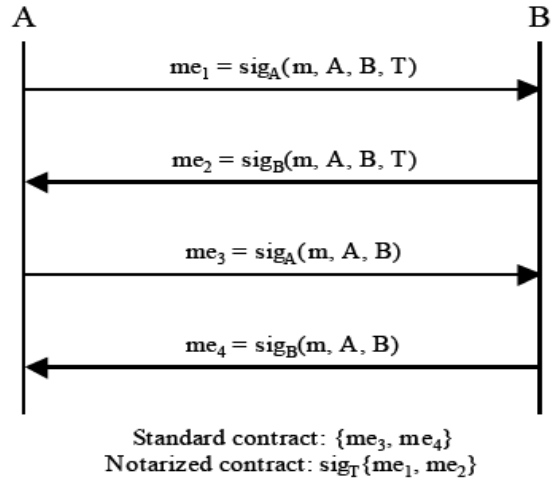


Figure 18: Main sub protocol (initial)

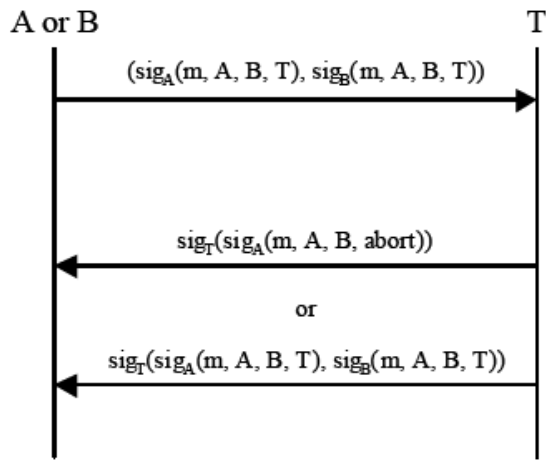


Figure 19: Recovery sub protocol (initial)

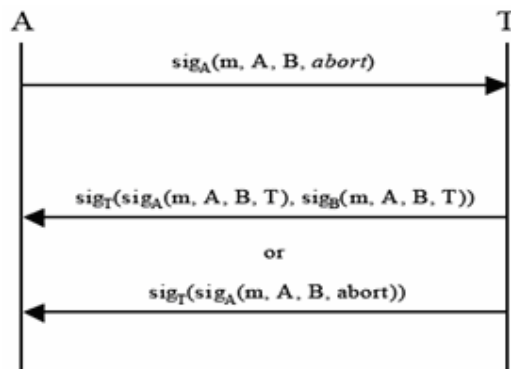


Figure 20: Abort sub protocol (initial)

We propose several improvements to this protocol. The time-out values are included into the initial messages in the following manner:

$$me_1 = SIG_A(m, A, B, TTP, T_B)$$

$$me_2 = \text{SIG}_B(m, A, B, \text{TTP}, T_B, T_A)$$

$$me_3 = \text{SIG}_A(m, A, B, T_B, T_A)$$

$$me_4 = \text{SIG}_B(m, A, B, T_B, T_A)$$

T_B is the time-out value estimated by A in relation to B and T_A is the time-out value estimated by B in relation to A. The timeout values (T_A , and T_B) are assumed to be estimated by A and B at run-time through a certain system call (for instance, ping). We can think of a timeout as the average of a several ping messages round-trip times. The timeout values are estimated at the initialization phase of the protocol execution. Note that timeout mechanism is not modeled in our prototype.

The standard contract of the improved model then is:

$$\{\text{SIG}_A(m, A, B, T_B, T_A), \text{SIG}_B(m, A, B, T_B, T_A)\},$$

which is the same as $\{me_3, me_4\}$.

The notarized contract is:

$$\text{SIG}_{\text{TTP}}(\text{SIG}_A(m, A, B, \text{TTP}, T_B), \text{SIG}_B(m, A, B, \text{TTP}, T_B, T_A)),$$

which corresponds to $\text{SIG}_{\text{TTP}}\{me_1, me_2\}$.

The principal of exchange in the main sub protocol remains the same.

For the recovery sub protocol we adopt the following notations (see 错误! 未找到引用源。):

$$m_{\text{RT1}} = \{me_1, me_2\}$$

$$m_{\text{AT2}} = \text{SIG}_{\text{TTP}}\{ \text{SIG}_A(A, B, \text{abort}) \}; \text{ which is the signed abort signal.}$$

$$m_{\text{RT3}} = \text{SIG}_{\text{TTP}}\{me_1, me_2\}; \text{ which is the notarized contract.}$$

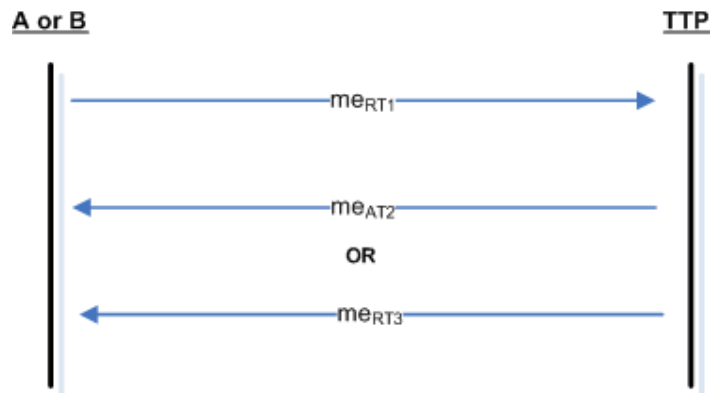


Figure 21: Recovery sub-protocol (improved)

For the abort sub protocol we adopt the notations (错误! 未找到引用源。):

$$me_{\text{AT1}} = \text{SIG}_A\{m, A, B, \text{abort}\}$$

$$me_{\text{AT2}} = \text{SIG}_{\text{TTP}}\{ \text{SIG}_A(m, A, B, \text{abort}) \}$$

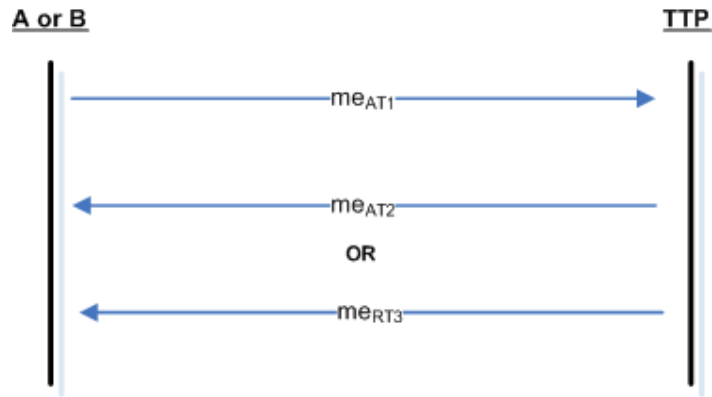


Figure 22: Abort sub protocol (improved)

To enhance the understanding of the security of the improved protocol, we present all possible states the parties A, B and TTP can encounter during the protocol run using the state machine diagrams. By that we prove that the protocol has no dead links, i.e. it always finishes in a fair state for A and B.

$\textcircled{X} \xrightarrow{m_1/m_2} \textcircled{Y}$: Indicates that a machine receives a message m_1 at state X, then it sends out a message m_2 and changes to state Y.

\longrightarrow : Indicates a normal execution path, while the *dashed* lines denote the exception handling execution paths (e.g., when an expected message has not arrived).

$\neg m$: means ‘message m is not received’.

Party A’s state machine diagram is depicted in [错误! 未找到引用源。](#). Party B’s in [错误! 未找到引用源。](#) and the TTP’s in [错误! 未找到引用源。](#).

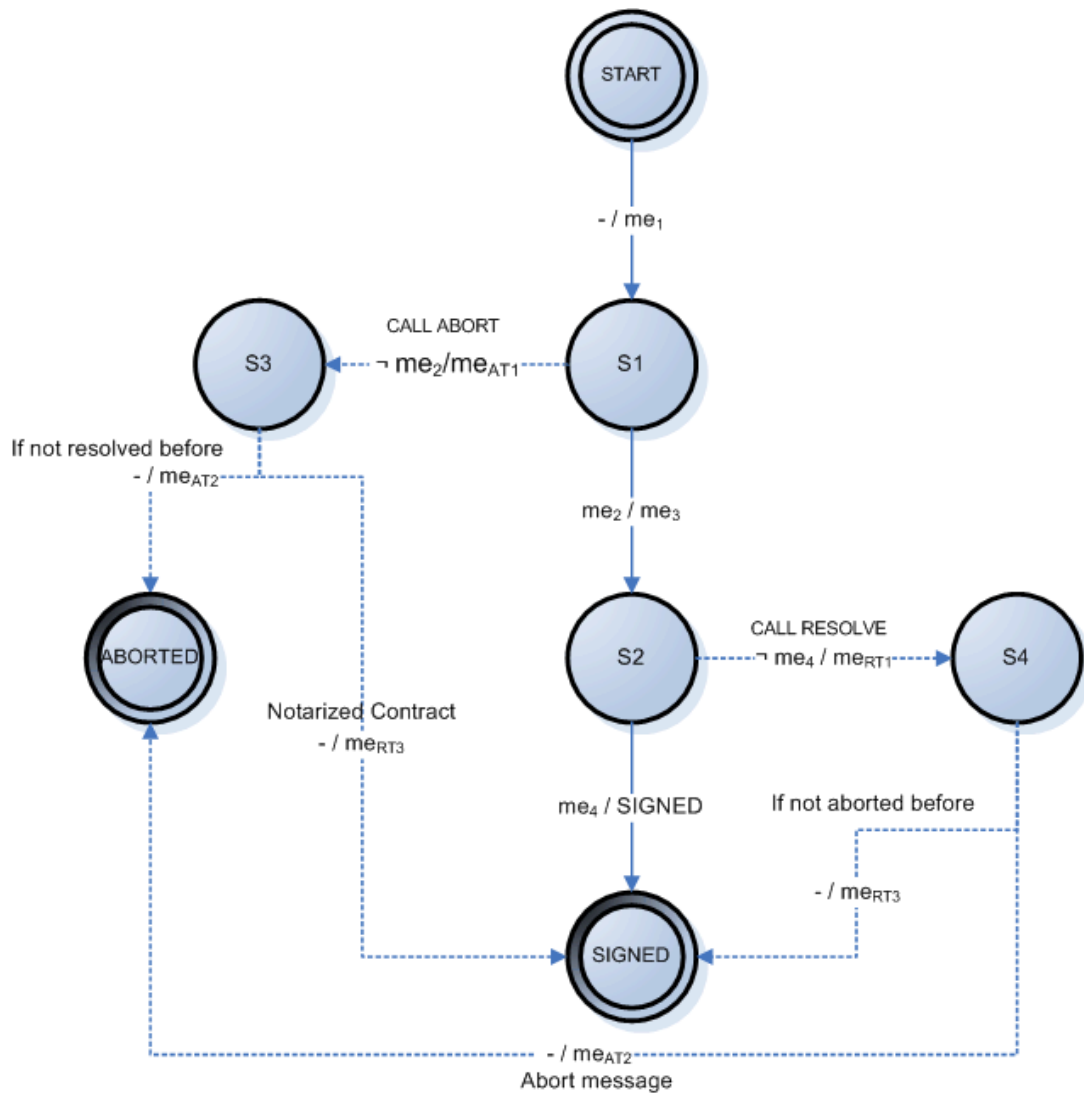


Figure 23: A's State Machine Diagram

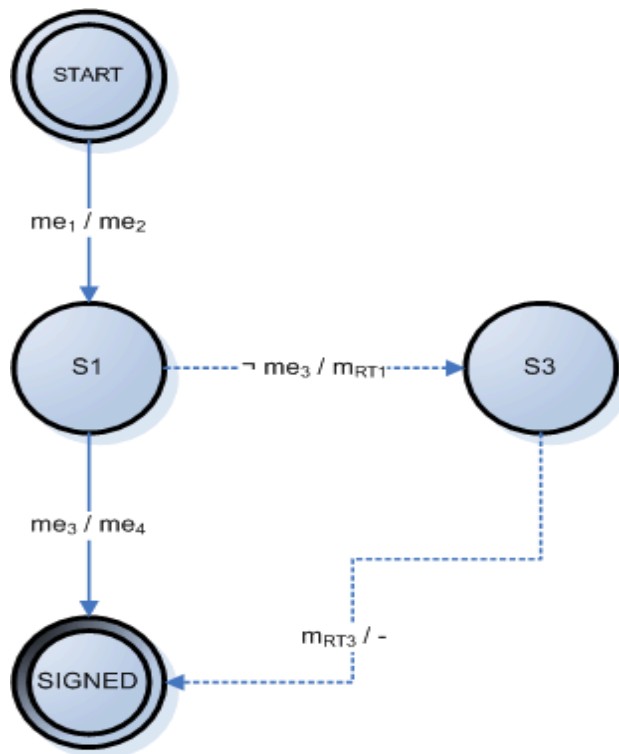


Figure 24: B's State machine Diagram

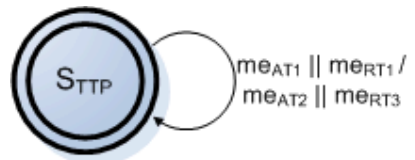


Figure 25: TTP's machine Diagram

7 Conclusion

At a Glance:

In this chapter we summarize the results of our thesis, reflect over the degree of fulfillment of the initial thesis goal, discuss the difficulties encountered along the way, present the limitations and, finally, give suggestions for future research.

The main idea of this thesis work was to conduct a survey of contract signing protocols in order to choose one that suits best for the SLA signing in dynamic virtual organizations (VOs). TrustCoM is a trust and contract management framework that enables secure collaborative business processing in on-demand created, self-managed, scalable and dynamic VOs. We have used TrustCoM documentation to arrive at a set of security requirements relevant for the SLA signing. On the basis of these requirements the SLA signing protocol was chosen from the pool of the surveyed protocols.

The path towards the ultimate goal was not obstacle-free. Along the way we discovered that TrustCoM documentation did not contain much information related to the SLA signing. Therefore, a row of assumption about the application environment had to be made. It was also a challenge to arrive at a final set of contract signing protocol properties. There was no source that presented cohesively the properties that contract signing protocols may possess. The articles focused rather at some and omitted others. In addition, properties with identical definitions had distinct names in different sources or vice versa. Last but not least, when conducting a survey, there is always a question how to determine when enough data has been analyzed. In our case, many more protocols could have been analyzed and, perhaps, this would result in a different final solution. However, due to the time limits and the scope of this thesis we had to stop somewhere.

Generally, we consider that we have managed to fulfill the thesis goal. The properties that a contract signing protocol should possess in a VO environment to carry out SLA signing were outlined, the protocol was chosen, the choice was argued and the prototype was implemented. The protocol survey was conducted and the survey result presented in the protocol-property matrix. The VO environment was analyzed with XP Extension for Security Requirements Engineering and CORAS Risk Assessment Methodology. The partial results of our ongoing work were integrated into TrustCoM framework.

There are several recommendations to make concerning the future research. Firstly, there are many new protocols appearing all the time and the pool of protocols to survey is enormous. Secondly, we have not analyzed the protocols in relation to the efficiency requirement in a detailed way. It would be interesting to implement any contract signing protocol using different signature algorithms and see how its efficiency changes. Thirdly, the solution is implemented as a prototype. It would be interesting to see how it performs in a real life implementation (web services). Because our requirements gathering omitted the legal risks, including them to the risk analysis would add more realistic legal requirements that can be supported by the implementation. This further work is very possible specially that CORAS risk assessment method support legal risks.

References

Service Level Agreements

-
- [1] TeleManagement Forum, <http://www.tmforum.org>, Accessed on 22nd Nov,2005
-
- [2] D. C. Verma, "Service Level Agreements on IP Networks", Proceedings of the IEEE, Vol. 92, No. 9, Sept. 2004.
-
- [3] International Telecommunication Union (ITU), <http://www.itu.int/ITU/>, Accessed on 22nd November, 2005
-
- [4] W. Sun et al., "The Role of XML in Service Level Agreements Management", Network Management Research Center, Beijing Jiaotong Univeristy, IEEE, 2005.
-
- [5] E. Wustenhoff, "Service Level Agreement in the Data Center", Sun© BluePrints™ Online, April, 2002. Available at www.sun.com/blueprints/0402/sla.pdf, Accessed on 22nd November, 2005
-

Service-Oriented Architecture

-
- [6] "<http://webservices.xml.com/lpt/a/ws/2003/09/30/soa.html>", "What is Service Oriented Architecture", Hao He. Accessed on 9th November, 2005.
-
- [7] XML Information Set, WorldWide Web Consortium (February 2004), <http://www.w3.org/TR/>. Accessed on 15th Nov, 2005
-
- [8] [XML Protocol Working Group, WorldWide Web Consortium \(2004\), http://www.w3.org/2000/xml/Group/](http://www.w3.org/2000/xml/Group/), Accessed on 20th December, 2005
-
- [9] XML Schema, WorldWide Web Consortium, <http://www.w3.org/XML/Schema>, Accessed on 20th December, 2005
-
- [10] IBM Systems Journal, Grid Computing, Vol.43, No. 4, 2004. "Evolution of grid computing architecture and grid adoption models", J. Joseph, M. Ernest, and C. Fellenstein, "<http://www.research.ibm.com/journal/sj/434/joseph.html>", Accessed on 10th November, 2005.
-
- [11] "Building Web Services with Java", 2nd Edition, Developer's Library, 2004, Chapter 1, pg1316.
-
- [12] "<http://www.w3.org/TR/ws-arch/>, W3C Working Group, Web services Architecture, Accessed on 11th November, 2005
-

No TTP Protocols

-
- [13] Even, S., and Yacobi, Y., "Relations among Public Key Signature Systems", Technical Report # 175, Computer Science Dept., Technion, Haifa, Isreal, March 1980.
-
- [14] Even S., "A Protocol for Signing Contracts", Computer Science Dept. Technion, Haifa, Isreal, 1983
-
- [15] S. Kermer, O. Markowitch, J. Zhou, "An Intensive Survey of fair Non-Repudiation Protocols".
-
- [16] O. Markowitch, Y. Roggeman, "Probabilistic Non-repudiation without Trusted Third Party"
-
- [17] Rabin, "Digitalized Signatures", in Foundation of a secure Computation, Academic Press, 1978, pp. 155-168.
-
- [18] Merkle, "Secure Communication over Insecure Channel", Communication of the ACM, Vol.21, April 1978, pp.294-299.
-
- [19] Mitsianis, A New Approach to Enforcing Non-Repudiation of Receipt, manuscript (2001).
-

- [20] R. H. Deng, et al. "Practical Protocols for Certified Electronic Mail", Journal of Network and System Management 4(3), 1996, pages: 279-297.
-

with TTP Protocols

- [21] N. Asokan, Matthias Schunter, Michael Waidner, "Optimistic protocols for fair exchange", April 1997, ACM
-
- [22] Shimon Even, Oded Goldreich, and Abraham Lempel, "A Randomized Protocol for Signing Contracts", June 1985 Volume 28 Number 6, ACM
-
- [23] Silvio Micali, "Simple and Fast Optimistic Protocols for Fair Electronic Exchange", July 2003, ACM
-
- [24] Panagiotis Louridas, "Some Guidelines for Non-repudiation Protocols", October 2000, ACM
-
- [25] Guilin Wang, "An Abuse-Free Fair Contract Signing Protocol Based on the RSA Signature", May 2005, ACM
-
- [26] Hiroshi Maruyama, Taiga Nakamura, Tony Hsieh "Optimistic Fair Contract Signing for Web Services", October 2003, ACM
-
- [27] R. Chadha, M. Kanovich, A. Scedrov, "Inductive Methods and Contract-Signing Protocols", November 2001, ACM
-
- [28] Giuseppe Ateniese, "Verifiable Encryption of Digital Signatures and Applications", February 2004, ACM
-
- [29] Birgit Pfitzmann, Matthias Schunter, Michael Waidner, "Optimal Efficiency of Optimistic Contract Signing", 1998, ACM
-
- [30] Vicky Liu, William Caelli, Ernest Foo, Selwyn Russell, "Visually Sealed and Digitally Signed Documents", 27th Australasian Computer Science Conference, The University of Otago, Dunedin, New Zealand. *Conferences in Research and Practice in Information Technology, Vol. 26.*, 2004
-
- [31] David Molnar "Signing Electronic Contracts", *Crossroads, Vol. 7*, Sept.2000, ACM
-
- [32] Directive 1999/93/EC of the European Parliament and of the Council, of 13 December 1999 on a Community framework for electronic signatures, Official Journal, http://www.e-podpis.sk/laws/eu_ep_dir93_1999.pdf
-

TrustCoM

- [33] Theo Dimitrakos, David Golby, Paul Kearney "Towards a Trust and Contract Management Framework for Dynamic Virtual Organisations", 2004
-
- [34] TrustCoM, TrustCom Reference Architecture, v1.0, Deliverable 9(D09), Available at <http://www.eu-trustcom.com/index.php?page=Documentation>, Accessed in December, 2005
-
- [35] Giambiagi, P. et al, "01 - The TrustCoM Framework v0.5"
-

Virtual Organizations

- [36] Bob Travica, "Virtual Organization and Electronic Commerce", the Database for Advances in Information Systems (Vol. 36, No. 3), summer 2005
-
- [37] Alvaro E. Arenas, Ivan Djordjevic, Theo Dimitrakos, Leonid Titkov, Joris Claessens, Christian Geuer-Pollmann, Emil C. Lupu, Nilifer Tuptuk, Stefan Wesner, Lutz Schubert, "Towards web Services Profiles for Trust and Security in Virtual Organizations".

[38] Adomas Svirskas, Alvaro Arenas, Michael Wilson, Brian Mattheus “Secure and Trusted Virtual Organization Management”, ERCIM News No. 63, October 2005, http://www.ercim.org/publication/Ercim_News/enw63/wilson.html

[39] Mikko Ahonen, Christian Bechheim, Pierre Goux, Andreas Schöler, “Virtual Organizations Concepts and Connection to Relationship Marketing”, *Research paper for virtual marketing*, 2000

Others

[40] Matt Bishop “Computer Security, Art and Science”, ISBN 0-201-44099-7, Pearson Education, 2003

[41] Michael Waidner, Keynote in ERCIM News No. 63, October 2005, http://www.ercim.org/publication/Ercim_News/enw63/keynote.html, Accessed in December 2005

[42] Fabio Martinelli, Jean-Jacques Quisquater, “Security and Trust Management”, ERCIM News No. 63, October 2005, http://www.ercim.org/publication/Ercim_News/enw63/intro.html, Accessed in December 2005

[43] SSE-CMM, Systems Security Engineering Capability Maturity Model, Model Description Document Version 3.0. URL: <http://www.sse-cmm.org/docs/ssecmmv3final.pdf>. Accessed in March 2006

[44] Lund, M.S, Vraalsen, F. “Analyzing Trust, Security, and Legal Issues using CORAS”, CORAS Project site <http://www2.nr.no/coras>, SINTEF, iTRUST05. Accessed in March 2006

[45] Böstrom, G. et al., “Extending XP Practices to Support Security Requirements Engineering”, published at the 28th International Conference on Software Engineering (ICSE 2006), Shanghai.

[46] Standish Group, “*The Chaos Report*”, West Yarmouth, MA: The Standish Group, 1995.

Appendix A - Implementation

For implementing the prototype of the chosen protocol [26], Java was chosen. The functions used for the creation of protocol messages (pre-contract, contract, abort, resolve) of this protocol are simulated, i.e. no real digital signature or encryption is implemented, but creating files with certain pattern to resemble the message or output of an operation.

Initially, each party (Alice and Bob) have a contract copy. For Alice this is a file named "initial_con_alice" and for Bob "initial_con_bob".

The code of all mentioned classes will be presented in the end of this Appendix.

Classes:

Contract:

This class resembles the initiation of the contract signing process. It creates a file named "instance_id" and writes to it a random number as the instance id of the process. The file "instance_id" can later be accessed to retrieve the id.

This class can only be called as Alice class, since Alice is always assumed to be an initiator. The initial contract for both parties contains only the word "contract". Each party (including TTP) has a directory structure; this is used to simulate their local environment.

```
<directory>
  <alice>
    |      |- intial_con_alice
    |      |-pre_con_alice
    |      |-con_alice
    |
    <bob>
    |      |- intial_con_bob
    |      |- pre_con_bob
    |      |- con_bob
    |
    <TTP>
    |      |- aborted
    |      |- resolved
    |
    |- notarized
```

At the first stage only bold files exist. The rest of the files are created during the signing process. "aborted" and "resolved" files in the TTP directory are initially blank. If the contract is aborted or resolved its id and the party's name will be written in the files for later reference. Note that in this protocol the TTP is state-keeping.

alice

This class represents an initiator of the signing process. It implements all attributes and functions needed to accomplish the signing of the contract.

alice methods:

initialize() :

The method initiates the signing process by instantiating a Contract object and thus creating an `instance_id` (refer to the Contract class).

createpreContract():

This method is called to simulate the creation of a pre-contract. It reads Alice's initial contract `-initial_con_alice`, and appends to it `"-pre(alice)"` and the contract instance id. The output of this class is a file called `"pre_con_alice"` which will contain:
`contract-pre(alice)-ConID=377`

Note that number 377 is only for the demonstration.

`pre(alice)` resembles Alice's signature on the pre-contract.

After creating the pre-contract it is assumed that it is accessible by Bob (simulating the sending function).

createContract() :

This method is called to simulate creating the final contract. It reads `initial_con_alice` and creates a new file named `con_alice` by appending `-con(alice)-ConID=377`

`con(alice)` resembles Alice's signature on the real contract.

As for the pre-contract, after creating the contract, it is assumed that Bob can access it and thus model the sending function.

check() :

This method checks if a pre-contract or a contract is correct; i.e. contains Bob's signature `pre(bob)` in case of a pre-contract, and `con(bob)` in case of a contract.

After checking, if any error is found the TTP is called for abortion or resolving, depending on whether it is a pre- or real contract. In case it is a pre-contract and an error is found, `abort()` method is called, else the `resolve()` method is called. Abort and resolve methods are discussed later in the TTP class.

bob:

Bob class represents the other party in the contract signing process. It is exactly the same as alice class, except that it does not have the `initialize()` method, since we assumed that only Alice can initiate the contract signing. Bob's signatures are:

On a pre-contract: `pre(bob)-ConID=377`

On a contract: `con(bob)-ConID=377`

Bob's pre-contract file is called: `pre_con_bob`, and that of a contract is called: `con_bob`.

TTP:

This class represents a state-keeping trusted third party used in this protocol. It implements 4 basic methods: `Abort()`, `resolve()`, `checkAbort()`, and `checkResolve()`.

This class's constructor, `TTP()`, reads the instance id from the `instance_id` file upon creating a new TTP object.

checkAbort () :

This method checks `./TTP/aborted` file to check if the *pre-contract* called to be aborted has previously been aborted. An aborted pre-contract will have its `instance_id` saved in this file for later referencing.

`checkAbort()` accepts no input parameter. The return value is true if the `instance_id` exists in the file and false otherwise.

checkResolve () :

This method checks `./TTP/resolved` file to check if the *contract* called to be resolved is previously resolved. A resolved contract will have its `instance_id` saved in this file for later referencing.

`checkResolve()` accepts no input parameter. The return value is true if the `instance_id` exists in the file and false otherwise.

Abort () :

This method commits the abort action. It accepts the party's name that has sent the abort message. At the beginning this method checks if the contract called to be aborted has previously been aborted or resolved through calling `checkAbort()` and `checkResolve()` methods. If at least one of the checks returns a true value, then the contract can not be aborted. Else the caller of `Abort()` is notified that the contract has been aborted and the `instance_id` file is deleted.

Resolve () :

This method commits the resolve action. It accepts the party's name that has sent the resolve message. At the beginning this method checks if the contract called to be resolved has previously been aborted or resolved through calling `checkAbort()` and `checkResolve()` methods. If at least one of the checks returns a true value, then the contract can not be resolved. Else the caller of `Resolve()` is notified that the contract has been resolved by creating a new notarized contract which is available for both parties and the `instance_id` file is deleted.

In addition to this prototype three other classes were created to implement XML digital signature creation and validation in addition to comparing two XML files. Following is the explanation of these three classes.

GenEnvelopedSig():

This class creates an enveloped XML signature over the root element of an XML file contract. It takes the party calling it as a parameter to determine the processing configurations. Configurations include path of the contract to be signed, path to the private key specification file in order to be able to re-build the private key, and the path to which to save the signed contract. Public keys of the parties are assumed to be saved in a publicly accessible directory. The key pairs (private and public) are created by another standalone main class to simulate the key manager. The key generator saves the specifications of the two parties' keys in predefined locations accessible to them.

After parsing the XML contract file, several elements are created to be added to the signature element - `KeyValue` and `KeyInfo` elements. Following is a demonstration of an XML contract before and after signing. Added elements are written in bold characters.

XML unsigned contract: invoice.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<invoice>
  <name>Rabih</name>
  <amount>1250.00</amount>
  <CreditCardNumber>1234-5678-9012-3456</CreditCardNumber>
  <Date>April 3, 2006</Date>
</invoice>
```

After signing, invoice.xml will look as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<invoice>
  <name>Rabih</name>
  <amount>1250.00</amount>
  <CreditCardNumber>1234-5678-9012-3456</CreditCardNumber>
  <Date>April 3, 2006</Date>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments" />
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
      <Reference URI="">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>S0WxfaNZEIgmSxPlJDffdx8U8mo=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>eDmQ4pTsmecMN5wh+Dm5mSpLM6x6nE0KqQ5J49RbeLVOKR6OKvot0EzP0HVkOzZNk4
    HajeT2aGn3
    7fet0kyG1ByJ1kHUFoPaNbgtk1UCD140VD14ibIttR6o/k7YmwjFZuN4+zYYw1b0Rjt4+cpkh/ew
    plsJEefi/jsE3BuHrqA=</SignatureValue>
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
          <Modulus>veYIqzNRA94YhXMYxyVW1+TtAYHZku+5GhmNQyTRp/P2uFIrtAm
          b0XiTG5X6XiKUB+X+SVSVoj9J
          Kz3QUOUPVPq/F1YklVnc/TZ1E20ZKHytT5f6M94ovacRH9OclEjbfNpWpXkgWc
          Qc1KtOAbQe36yD bj9k9Hw8yCVOR/Qf7qU=</Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</invoice>
```

Note that the public key of the sender is embedded in the signature for the validation of the signature; this is presented in the `KeyInfo` element. This element can also be validated with the sender's public key available in the public directory for public keys.

ValidateSig():

This class implements the validation function of the signature and is called by the recipient after receiving a signed contract. It searches for the signature element and calls the `validate()` function.

ComapreContracts():

This class compares the received contract with the local party's copy. It is called after validating the signature and parsing the whole received contract. It uses the xmlDiff class from oracle.xml package to carry out the comparison.

Note that the GenEnvelopedSig(), ValidateSig(), and CompareContracts() need extra packages to be loaded to the java classpath. Those extra packages will be bold and italic in the source code presented in the last chapter of this appendix.

UML Diagrams:

Following we will present UML sequence diagrams for the protocol execution. Figure 17 depicts the honest case where no party is cheating, so no TTP is called and that is why it is not shown.

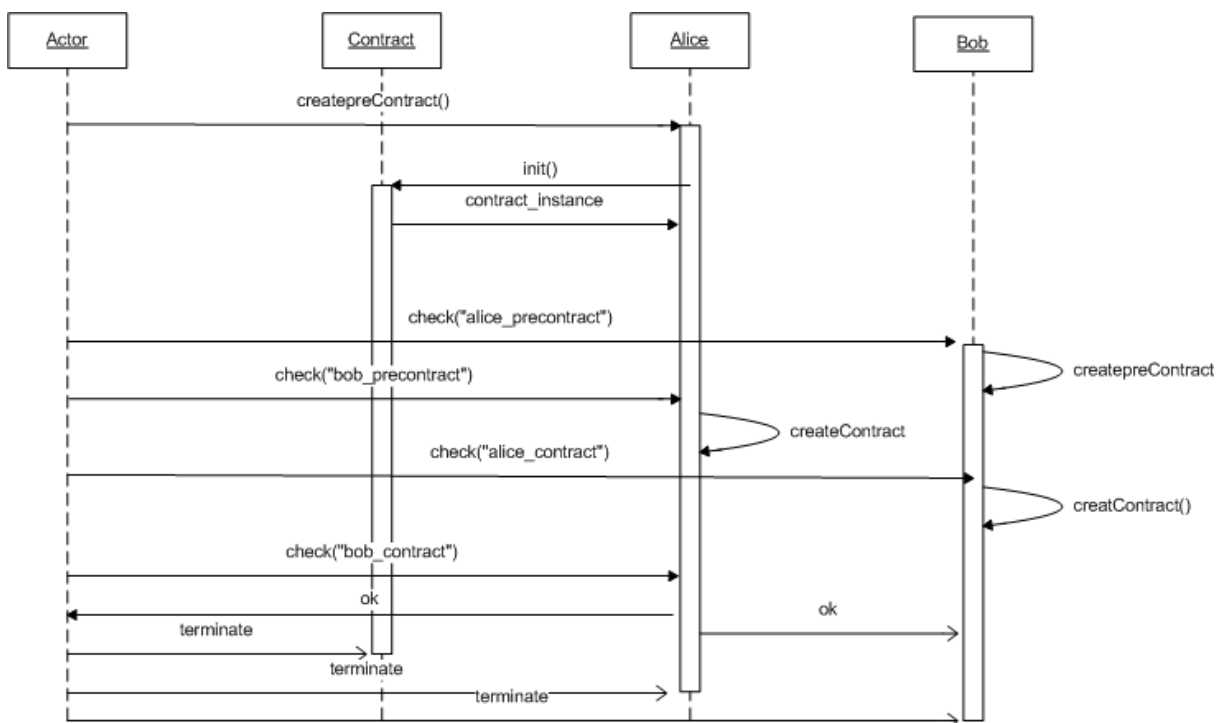


Figure 26: An Honest Execution Sequence Diagram for the chosen protocol

Next, this sequence diagram shows four dishonest scenarios, where four different terminations are presented. The former two end up as aborted contracts, hence no contract is signed. The latter two end up with a notarized contract produced by the TTP after a resolve call from a party.

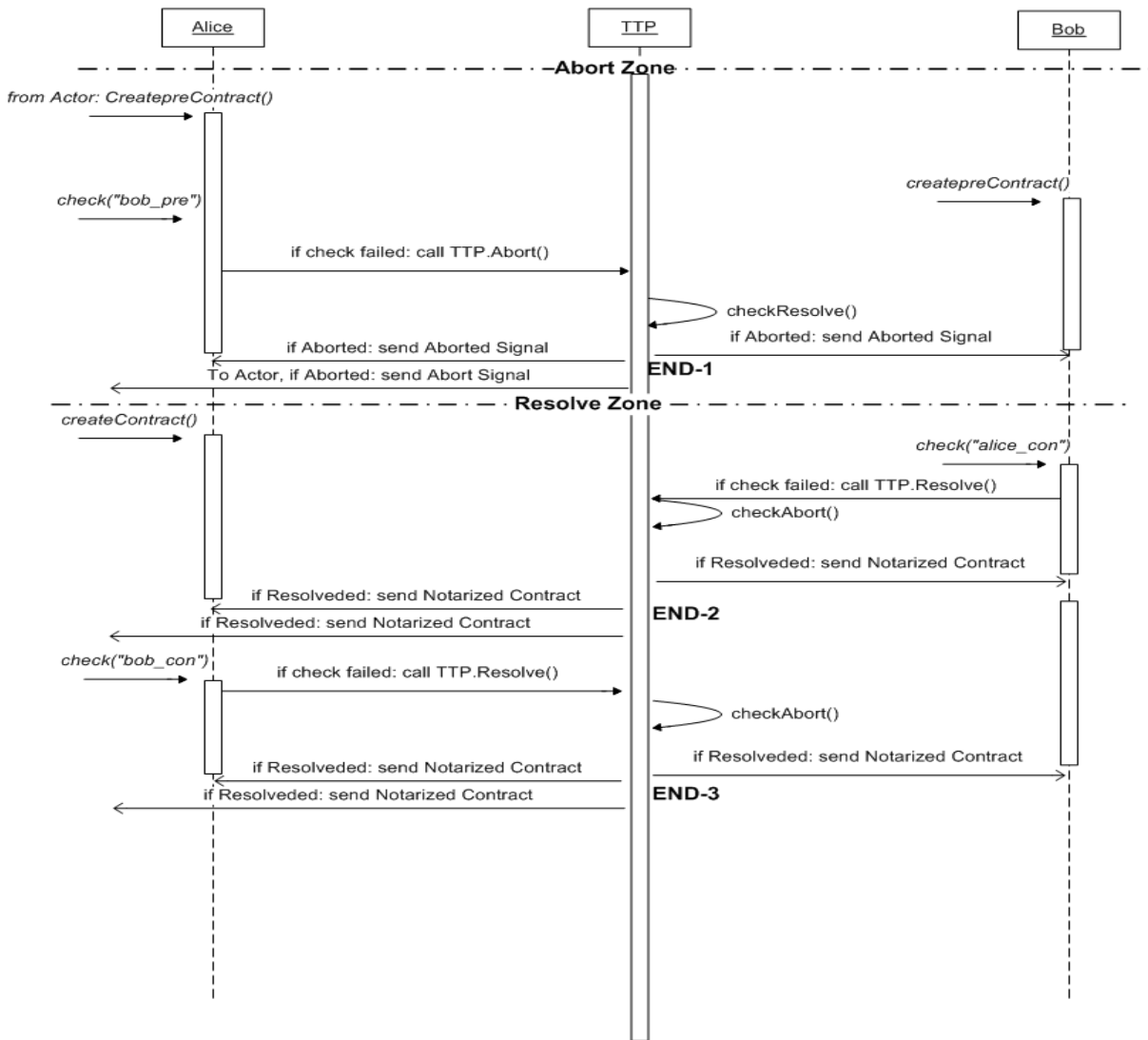


Figure 27: Four Dishonest Scenarios of Protocol Execution

Source Pseudo-Code

Alice Class:

Create pre-contract

```

Create contract instance ID
Create, save and send pre-contract
  
```

Create Contract

```

If (pre-contract exists)
    Read local contract copy
    Sign with alice's signature
    Save and send contract to Bob
Else
    ERROR, exit code 113
  
```

Check contract or pre-contract

```

If(pre-contract or contract is recieved)
    If (contents are correct)
        Return successful checking
    Else
  
```

```

        Call TTP.Abort() if it's a pre-contract, otherwise call
        resolve
        Exit, code 111
    Else
        Call TTP.Abort() if it's a pre-contract, otherwise call
        resolve

```

Bob Class:

Create pre-contract

```

    If (contract instance exists)
        If (alice's pre-contract is received)
            Create and send pre-contract
        Else
            ERROR, Exit code 116
    Else
        ERROR, Exit code 112

```

Create Contract

```

    If (pre-contract exists)
        Read local contract copy
        Sign contract with Bob's signature
        Save and send contract to Alice
    Else
        ERROR, Exit code 115

```

Check Contract or pre-contract

```

    If (pre-contract or contract received)
        If (contents are correct)
            Return successful checking
        Else
            call TTP.Resolve() if the checked is alice contract
    Else
        Call TTP.Resolve()

```

TTP Class

Intiate TTP

```

    If (contract instance exists)
        Continue
    Else
        ERROR, Exit code 118

```

Check Abort

```

    If(pre-contract ID exists in .TTP/aborted)
        Return true
    Else
        Return false

```

Check Resolve

```

    If(contract ID exists in .TTP/resolved)
        Return true
    Else
        Return false

```

Abort

```

    If (checkResolve)
        Contract can not be aborted

```

```

Else
    Add contract and abort caller ID to .TTP/aborted
    Delete contract ID and terminate its instance

```

Resolve

```

If (checkAbort)
    Contract can not be resolved
Else
    Add contract and abort caller ID to .TTP/aborted
    Create notarized contract and send to
    both alice and bob
    Delete contract instance and ID

```

Source Code for the Signing, Signature Validation, and contract comparison Classes

GenEnvelopedSig Class: (in Java language)

```

package org.xmlsig;

/**
 * <p>Title: Enveloped Signature Generator</p>
 *
 * <p>Description: This class generates an XML signature over the root element
 * of the contract, and "envelope" it within the contract
 * </p>
 *
 * <p>Copyright: Copyright (c) 2006</p>
 *
 * <p>Company: SICS</p>
 *
 * @author <a href="mailto:rabih@sics.se">Rabih Ghannoum</a>
 * @version 1.0
 */

import javax.xml.crypto.dsig.*;

import javax.xml.crypto.dsig.dom.DOMSignContext;
import javax.xml.crypto.dsig.keyinfo.*;
import javax.xml.crypto.dsig.XMLSignatureFactory;
import javax.xml.crypto.dsig.spec.*;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.security.*;
import java.util.Collections;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;

import org.xmlsig.*;

/**
 * This is a file generates an Enveloped XML
 * Signature using the JSR 105 API.
 */

public class GenEnvelopedSig {

    /** Reads the local contract copy to be signed, signs it and saves the signed
     * contract locally to be encrypted later before sending
     * @param String requestor
     * @return File signed_contract
     */

```

```

    public static void GenEnvelopedSig (String requestor) throws Exception {
/* Here, as a class implementation, not main, this class invoker should provide
his identity, e.g. as a String "consumer" or "provider", to create the
corresponding keys and save the corresponding signed contract in the right
place!! */

    PrivateKey kp=null;
    PublicKey KP=null;
    String contract=null, output=null;

    if (requestor=="consumer"){

        Consumer cc = new Consumer();

        kp= (PrivateKey) cc.getPrivateKey();

        KP = cc.getPublicKey();

        contract = new Consumer().getContract();

        output=cc.signed_output;
    }

    if (requestor=="provider"){

        provider pp = new provider();

        kp = (PrivateKey) pp.getPrivateKey();

        KP = pp.getPublicKey();

        contract = pp.getContract();

        output=pp.signed_output;

    }

    // Create a DOM XMLSignatureFactory that will be used to //generate the
enveloped signature

    String providerName = System.getProperty
        ("jsr105Provider","org.jcp.xml.dsig.internal.dom.XMLDSigRI");
    XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM",
        (Provider) Class.forName(providerName).newInstance());

    /* Create a Reference to the enveloped document (in this case we are
signing the whole document, so a URI of "" signifies that) and also
specify the SHA1 digest algorithm and the ENVELOPED Transform. */

    Reference ref = fac.newReference
        ("", fac.newDigestMethod(DigestMethod.SHA1, null),
        Collections.singletonList
            (fac.newTransform
                (Transform.ENVELOPED, (TransformParameterSpec) null)),
            null, null);

    // Create the SignedInfo
    SignedInfo si = fac.newSignedInfo
        (fac.newCanonicalizationMethod
            (CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS,
                (C14NMethodParameterSpec) null),
            fac.newSignatureMethod(SignatureMethod.RSA_SHA1, null),
            Collections.singletonList(ref));

    // Create a KeyValue containing the RSA PublicKey that was //generated

    KeyInfoFactory kif = fac.getKeyInfoFactory();
    KeyValue kv = kif.newKeyValue(KP);

    // Create a KeyInfo and add the KeyValue to it

```

```

        KeyInfo ki = kif.newKeyInfo(Collections.singletonList(kv));

        // Instantiate the document to be signed
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        Document doc =
            dbf.newDocumentBuilder().parse(new FileInputStream(contract));
        System.out.println(requestor+"'s"+ " Contract Loaded...");

        // Create a DOMSignContext and specify the DSA PrivateKey and
        // location of the resulting XMLSignature's parent element

        DOMSignContext dsc = new DOMSignContext
            (kp, doc.getDocumentElement());

        // Create the XMLSignature (but don't sign it yet)
        XMLSignature signature = fac.newXMLSignature(si, ki);

        // Marshal, generate (and sign) the enveloped signature
        signature.sign(dsc);

        OutputStream os = new FileOutputStream(output);

        TransformerFactory tf = TransformerFactory.newInstance();
        Transformer trans = tf.newTransformer();
        trans.transform(new DOMSource(doc), new StreamResult(os));
    }
}

```

ValidateSig Class:

```

package org.xmlsig;

import javax.xml.crypto.*;
import javax.xml.crypto.dsig.*;
import javax.xml.crypto.dom.*;
import javax.xml.crypto.dsig.dom.DOMValidateContext;
import javax.xml.crypto.dsig.keyinfo.*;
import java.io.FileInputStream;
import java.security.*;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

import org.xmlsig.*;

/**
 * <p>Title: XML Digital Signature - TrustCoM Project</p>
 *
 * <p>Description: This class validates an XML
 * Signature using the JSR 105 API. It assumes the key needed to
 * validate the signature is contained in a KeyValue KeyInfo element. The
 * requestor id is passed, and a boolean is passed back for success or failure
 * of signature.
 * </p>
 *
 * <p>Copyright: Copyright (c) 2006</p>
 *
 * <p>Company: SICS</p>
 *
 * @author <a href="mailto:rabih@sics.se">Rabih Ghannoum</a>
 * @version 1.0
 */
public class ValidateSig {

    Consumer cc;

```

```

provider pp;
String input, req1;
boolean state=false;

public ValidateSig(String req) throws Exception {

    req1=req;

    if (req1=="consumer"){

        cc = new Consumer();
        input = cc.c_recieved_signed;

    }
    if (req1=="provider"){
        pp=new provider();
        input = pp.p_recieved_signed;
    }

    // Instantiate the document to be validated
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    dbf.setNamespaceAware(true);
    Document doc =
        dbf.newDocumentBuilder().parse(new FileInputStream(input));

    // Find Signature element
    NodeList nl =
        doc.getElementsByTagNameNS(XMLSignature.XMLNS, "Signature");
    if (nl.getLength() == 0) {
        throw new Exception("\t"+req1+": Cannot find Signature element");
    }

    // Create a DOM XMLSignatureFactory that will be used to unmarshal the
    // document containing the XMLSignature
    String providerName = System.getProperty
        ("jsr105Provider", "org.jcp.xml.dsig.internal.dom.XMLDSigRI");
    XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM",
        (Provider) Class.forName(providerName).newInstance());

    // Create a DOMValidateContext and specify a KeyValue KeySelector
    // and document context
    DOMValidateContext valContext = new DOMValidateContext
        (new KeyValueKeySelector(), nl.item(0));

    // unmarshal the XMLSignature
    XMLSignature signature = fac.unmarshalXMLSignature(valContext);

    // Validate the XMLSignature (generated above)
    boolean coreValidity = signature.validate(valContext);

    // Check core validation status
    if (coreValidity == false) {
        System.err.println("\t"+req1+": Signature failed core validation");
        boolean sv = signature.getSignatureValue().validate(valContext);
        System.out.println("\t"+req1+": signature validation status: " + sv);
        // check the validation status of each Reference
        Iterator i = signature.getSignedInfo().getReferences().iterator();
        for (int j=0; i.hasNext(); j++) {
            boolean refValid =
                ((Reference) i.next()).validate(valContext);
            System.out.println("ref["+j+"] validity status: " + refValid);
        }
    } else {
        System.out.println("\t"+req1+": Signature passed core validation");
        state=true;
    }
    //return state;
}

/**

```

```

* KeySelector which retrieves the public key out of the
* KeyValue element and returns it.
* NOTE: If the key algorithm doesn't match signature algorithm,
* then the public key will be ignored.
*/
private static class KeyValueKeySelector extends KeySelector {
    public KeySelectorResult select(KeyInfo keyInfo,
                                   KeySelector.Purpose purpose,
                                   AlgorithmMethod method,
                                   XMLCryptoContext context)
        throws KeySelectorException {
        if (keyInfo == null) {
            throw new KeySelectorException("Null KeyInfo object!");
        }
        SignatureMethod sm = (SignatureMethod) method;
        List list = keyInfo.getContent();

        for (int i = 0; i < list.size(); i++) {
            XMLStructure xmlStructure = (XMLStructure) list.get(i);
            if (xmlStructure instanceof KeyValue) {
                PublicKey pk = null;
                try {
                    pk = ((KeyValue)xmlStructure).getPublicKey();
                } catch (KeyException ke) {
                    throw new KeySelectorException(ke);
                }
                // make sure algorithm is compatible with method
                if (algEquals(sm.getAlgorithm(), pk.getAlgorithm())) {
                    return new SimpleKeySelectorResult(pk);
                }
            }
        }
        throw new KeySelectorException("No KeyValue element found!");
    }

    static boolean algEquals(String algURI, String algName) {
        if (algName.equalsIgnoreCase("DSA") &&
            algURI.equalsIgnoreCase(SignatureMethod.DSA_SHA1)) {
            return true;
        } else if (algName.equalsIgnoreCase("RSA") &&
            algURI.equalsIgnoreCase(SignatureMethod.RSA_SHA1)) {
            return true;
        } else {
            return false;
        }
    }
}

private static class SimpleKeySelectorResult implements KeySelectorResult {
    private PublicKey pk;
    SimpleKeySelectorResult(PublicKey pk) {
        this.pk = pk;
    }

    public Key getKey() { return pk; }
}
}

```

CompareContracts Class:

```

package org.xmlsig;

/**
 * <p>Title: CompareContracts </p>
 *
 * <p>Description: This class implements the comparison between the local
contract
 * copy of a participant with a received one. oracle.xml package is used here,
because XMLDiff class was needed.</p>
 *
 */

```

```

* <p>Copyright: Copyright (c) 2006</p>
*
* <p>Company: SICS</p>
*
* @author <a href="mailto:rabih@sics.se">Rabih Ghannoum</a>
* @version 1.0
*/

import oracle.xml.parser.v2.*;
import oracle.xml.differ.*;
import java.io.*;
import org.w3c.dom.*;
import org.xmlsig.*;

public class CompareContracts extends XMLDiff{

    //public static CompareContracts xmlDiff;

    Consumer cc; provider pp;
    String input_rec, input_local,requestor;

    /** Reads the received, but decrypted contract (after validation). then it
erases
    * the <signature> tag to bring the contract back as it was.
    * @param String validated_contract, requestor
    * @return XMLDocument
    * */
    private XMLDocument fixContract(String req, String input)throws Exception{

        DOMParser parser=new DOMParser();
        InputStream is1 = new FileInputStream(input);
        parser.parse(is1);
        XMLDocument xdoc1 = parser.getDocument();

        // Remove <signature> node to return file as is, for accurate comparison

        Element element = (Element)
xdoc1.getElementsByTagName("Signature").item(0);
        element.getParentNode().removeChild(element);

        return xdoc1;
    }

    /** Constructor; Sets some parameters accordingh to requestor (provider or
consumer),
    * then access the XMLDiff class to carry out the comparison
    * @param String validated_contract, requestor
    * @return boolean
    * */

    public CompareContracts(String req) throws Exception
    {
        requestor = req;

        if (requestor =="consumer"){
            cc=new Consumer();
            input_rec= cc.c_recieved_signed;
            input_local=cc.getContract();
        }
        if(requestor=="provider"){

            pp=new provider();
            input_rec=pp.p_recieved_signed;
            input_local = pp.getContract();
        }

        /* Eleminate signature element after validation to enable comparison
the signature node can be removed and saved as a NRR.
        */
        XMLDocument xdoc1 = fixContract(requestor, input_rec);

```

```

InputStream is2;
DOMParser parser = new DOMParser();

is2 = new FileInputStream(input_local);
parser.parse(is2);
XMLDocument xdoc2 = parser.getDocument();

XMLDiff xmlDiff = new XMLDiff();

xmlDiff.setDocuments(xdoc1,xdoc2);

boolean diff = xmlDiff.diff();

if (diff)
    System.out.println("\t"+requestor+": Contract is not Correct");
// its also possible to write difftree. e.g.
//xmlDiff.printDiffTree(1,BufferedWriter)
// CALL TTP ABORT, for instance!
else
    System.out.println("\t"+requestor+": Contract is Correct");
}
}

```