Journal of Universal Computer Science, vol. 15, no. 3 (2009), 555-583
submitted: 23/8/07, accepted: 29/1/09, appeared: 1/2/09 © J.UCS

A New Fair Non-repudiation Protocol for Secure Negotiation and Contract Signing[*]

Antonio Ruiz-Martínez
(University of Murcia, Murcia, Spain
arm@um.es)

C. Inmaculada Marín-López
(University of Murcia, Murcia, Spain
inmaml@um.es)

Laura Baño-López
(University of Murcia, Murcia, Spain
laurabl@um.es)

Antonio F. Gómez-Skarmeta
(University of Murcia, Murcia, Spain
skarmeta@um.es)

Abstract: The participation of an e-notary, acting as an on-line Trusted Third Party is required in some scenarios, such as Business to Business, Intellectual Property Rights contracting, or even as a legal requirement, in contract signing is frequently necessary. This e-notary gives validity to the contract or performs some tasks related to the contract, e.g. contract registration. In the abovementioned contracting scenarios, two important additional features are needed: the negotiation of the e-contract and confidentiality. However, until now, e-contract signing protocols have not considered these issues as an essential part of the protocol. In this paper, we present a new protocol which is designed to make negotiation and contract signing processes secure and confidential. Moreover, compared to other previous proposals based on an on-line Trusted Third Party, this protocol reduces the e-notary's workload. Finally, we describe how the protocol is being used to achieve agreements on the rights of copyrighted works.

Keywords: fair exchange, contract signing protocol, Intellectual Property Rights contracts, secure negotiation, confidentiality
Categories: C.2.2, K.4.4, K.6.5

1 Introduction

Nowadays, a part of the research in e-commerce and business transactions focuses on the electronic signing of contracts. In these contracts, the parties involved gain from their relationship. These gains could be goods, services or others. The contracts are particularly important in Customer-to-Business (C2B) and Business-to-Business (B2B) commerce when establishing long-term relationships or offering services.

[*] This is an extended version of a paper presented at the PST 2006 conference in Markham, Ontario, Canada.

In contract signing protocols, Trusted Third Parties (TTPs) provide security and confidence to the system. However, if the TTP has to participate in a lot of transactions, a bottleneck can ensue. For this reason, except in certain circumstances, it is desirable that the TTP participates as little as possible in the execution of the protocol. Thus, it is preferable to have an off-line TTP instead of an on-line TTP. Nevertheless, an on-line TTP is still needed in some scenarios [AAVV, 05], [Angelov, 05], [Kötz, 97], [Ruiz, 03], [Yang, 05]. In such scenarios, to obtain a valid contract it is mandatory that the contract is signed by a TTP, which acts as an e-notary. For example, Spanish or French laws establish that some contracts, such as those related to royalties or legacies, must be signed by an e-notary [AAVV, 05], [Kötz, 97]. Thus, the e-notary validates the contract and records it. Similar scenarios could be the agreement of rights of copyrighted works or a B2B contract update.

Some proposals such as [Yang, 03], [Yang, 05], [Zhou, 96] have been put forward for these scenarios. However, these proposals present some problems, such as not guaranteeing abuse freeness, or the over important participation of the TTP in the protocol. The aim is that the protocol reduces the TTP's load to the minimum possible, i.e., that TTP participates in very few messages in the protocol and makes the minimum number of cryptographic operations. Moreover, it is also important that the cryptographic operations have the least computational workload. Neither do these proposals take into account two important requirements in business/DRM transactions: secure negotiation and confidentiality. Confidentiality is important to avoid parties not involved in the protocol knowing the agreement between two parties, their behaviour, etc. On the other hand, in general, before agreeing on a contract there is a negotiation [Darko, 06], [Delgado, 01],[Limthanma, 00]. The security during this process is also important and we should protect the information exchanged, maintaining its integrity, confidentiality and protecting the parties against attacks from parties not involved in the negotiation [Darko, 06], [Delgado, 01], [Limthanma, 00].

We propose a new protocol that is based on an on-line TTP. However, in our protocol, unlike the best of the previous proposals based on a TTP on-line, the TTP does not participate in all the messages exchanged between the parties. In our proposal the TTP only participates during the last phase of the protocol, in order to sign the agreement reached. Its main added value against other proposals is the incorporation of important features such as confidentiality and the secure negotiation of the contract. In this protocol, the contract negotiation information has been included as part of the protocol. At the same time, we have reduced the TTP's overload from two important aspects, compared to other works. On the one hand, as regards the number of cryptographic operations to perform and on the other hand as regards the number of messages in which the TTP participates. Furthermore, we have designed the protocol so that it works with any public key cryptosystem that provides signature and ciphering (RSA, ECDSA,…). Thus, users could utilize the protocol with the keys and certificates that they own, because it is not necessary either to generate new keys or to engage in a registration process with a TTP, unlike [Garay, 99], [Park, 03], [Wang, 05].

The paper is organized as follows. In section 2, we introduce contract signing protocols and related work. Section 3 details our proposal. Then, in Section 4, its security is analyzed and compared with previous works. In section 5, we introduce a protocol variation to facilitate contract management. Finally, in section 6, we present conclusions and future work.

2 Contract signing protocol requirements and related work

A paper-based contract is a signed document where two or more parties express an agreement. Contract signing protocols appeared to allow two or more parties to establish a contract over a network in a fair way. By fair it is meant that each honest party sending a signed contract is assured that if the other party obtains it, then he will also obtain the necessary signed contract of that party. In a contract signing protocol, the signing parties, at the very least, must participate. Additionally, other parties can participate, e.g. a TTP. The role of the TTP is to guarantee fairness and provide confidence to the system, because the signing parties might not be trustworthy.

As stated in [Kermer, 02], depending on the grade of implication of the TTP in the protocol, they can be classified as: in-line, in the case that the TTP participates in the delivery of each message; on-line, when the TTP only participates once in the run of the protocol; off-line or optimistic, when it only participates if something goes wrong; and transparent, in those cases when it is not possible to determine if the TTP is participating or not. In this kind of protocol there could be other parties (malicious entities) in the network that want to interfere in the signing of the contract by either modifying the content of the contract or by impersonating one of the parties or replaying old messages or by seeking to obtain confidential information about the contract. Basically, these attacks may pursue several goals, such as preventing a successful agreement being reached, obtaining confidential information, being detrimental to another party and obtaining the benefits of a contract by impersonating another party.

In this section we introduce the requirements that we need for electronic contract signing protocols in order to negotiate and sign contracts in a secure way and make use of a TTP on-line for scenarios where the electronic signature of a TTP is essential, for example, in some business-to-business or DRM scenarios [Angelov, 05], [Jalali, 00], [Ruiz, 03], [Yang, 03], [Yang, 05] or by legal requirement [AAVV, 05], [Kötz, 97], as was stated in the introduction. We therefore analyze related work.

2.1 Requirements

With the aim of signing a contract in a fair way and of addressing the abovementioned threats, we define the basic features required for a contract signing protocol: non-repudiation, fairness, efficiency, completeness, viability and timeliness. These requirements are briefly defined as follows:

- *Non-repudiation*. This feature aims to ensure that participants in a contract signing protocol cannot deny having participated therein [Kremer, 02], [Zhou, 01].
- *Fairness*. This requirement seeks to guarantee that no party gains an advantage over another at any moment during the running of the protocol. The protocol would not be fair, for example, if one of the parties obtained the signed contract without the other being able to do likewise.

- *Efficiency*. The number of cryptographic operations and the time used to execute them should be the least possible.
- *Completeness*. The protocol should be robust against adversaries that try to abort it without the consent of any of the parties.
- *Timeliness*. This requirement is given if any entity can stop the protocol in a finite amount of time while guaranteeing fairness [Gürgens, 05].

These features are discussed in more detail in [Gürgens, 05], [Kremer, 02], [Markowitch, 02]. These criteria are the classical features that we should require for a protocol that guarantees fair exchange and, therefore, for a contract signing protocol included in fair exchange. Furthermore, a new important feature for the contract signing protocols was introduced in [Garay, 99]:

- **Abuse freeness**: "if it is impossible for a single player at any point in the protocol to be able to prove to an outside party that it has the power to terminate (abort) or successfully complete the contract" [Garay, 99].

Other additional requirements introduced and considered by S. Yang et al. in [Yang, 03], [Yang, 05] are:

- *Trust dependency on a third party*. A TTP that knows the content of the messages is heavily dependent on the confidence in a third party. On the other hand, a TTP that does not know the contents presents a lesser degree of dependency as regards the trust to be deposited in the third party. We could also require low dependency for a contract signing protocol unless, in some circumstances, the TTP needs to know the content in order to perform certain operations (for example, an e-notary who has to record the contract and assure its validity).
- *Existence dependency*. The protocol should generate evidences in such a way that, in case there is a subsequent dispute, the result can be determined without recurring to the TTP.
- *Recipient role*. Normally, the protocols are sender-or-requester-oriented. They give this role more control and responsibility and might not be a good solution for the efficient processing of service requests. However, in e-commerce applications, the service provider has to take an active role in the execution of the protocol to obtain both system efficiency and integration.

Finally, other additional requirements we consider essential and, therefore, which we require for contract signing protocols are:

- *Confidentiality*. The information exchanged between the parties should be known only to them. Most of the protocols suppose a private communication channel. However, this supposition is not realistic enough. A user cannot suppose the underlying network provides this property because the network may not offer it. Furthermore, establishing this private channel requires the exchange of additional messages, and therefore, a greater load for both the users and TTP.
- *Secure Negotiation*. The protocol should allow the secure negotiation of the terms and conditions of the contract since, in general, there is a negotiation phase prior to the contract signing process [Darko, 06], [Delgado, 01],

[Limthanma, 00]. Moreover, according to [Darko, 06], "the process of reaching an agreement, i.e. the negotiation of the contract, is therefore particularly critical to the position and advantage that a partner can draw from participating in the virtual organization. If the negotiation process is not properly secured, a partner may miss the opportunity to conclude an advantageous agreement". Therefore, secure negotiation should be guaranteed even if, in the end, an agreement is not reached. Furthermore, incorporating this feature into the contract signing protocol we will make this kind of business process (negotiation and contract signing) more efficient than if we perform these processes in two different protocols. The efficiency issue is analysed in depth in section 4.9.

In most studies these additional criteria have not been considered. We now provide the justification for our decision to include each of these features as an essential part of a contract signing protocol.

As for confidentiality, some proposals (see [Gürgens, 05], [Kremer, 02]) assume a channel where the information is exchanged in a ciphered way: either using Virtual Private Networks or SSL/TLS connections between the different parties. Apart from the fact that the supposition of a secure channel is not realistic enough, it supposes a higher load (e.g. a SSL/TLS channel needs five messages, at least, to be established [Asokan, 98], [Dierks, 99]). Therefore, until now, in contract signing protocols confidentiality is not supposed as an essential part of the protocol. However, this characteristic is important because we want to prevent parties not involved in the protocol knowing the agreement between two parties, the conditions, their behaviour, and so on.

As regards the secure negotiation of the contract terms and conditions, this is not a feature which any of them incorporates. Negotiation is a fundamental process in any e-commerce or business model, as stated in [Darko, 06], [Limthanma, 00], [Röhm, 98].

Protecting the information exchanged in this process is also fundamental because most of the conditions agreed in the final contract are included in the negotiation. Therefore, the security provided in this phase should be similar to the contract phase and we should maintain the integrity, confidentiality and avoid man-in-the-middle and impersonation attacks during this phase.

However, in the existing protocols, [Abadi, 02], [Kim, 99], [Yang, 03], [Yang, 05], [Zhou, 96], secure negotiation could be achieved by means of either the execution of the contract signing protocol for each offer or by using a previous specific protocol to carry out the negotiation.

The main disadvantage of an execution of the contract signing protocol, for each offer is that it supposes the exchange of a lot of messages, which makes it slow and not efficient. The main disadvantage of the use of a secure negotiation-specific protocol with a contract signing protocol is that in both processes we have to generate similar cryptographic material and use similar operations to obtain a secure communication, which is also inefficient as we explain in section 4.9. In both cases, as an additional drawback, we can mention that the negotiation is not linked to the contract, which is desirable [Delgado, 01], [Röhm, 98]. Thus, by offering a protocol with both features (negotiation and contract signing), we could make the process more efficient than when we use a different protocol for each process. In fact, there are other fields of electronic business/commerce that have followed this approach of combining negotiation with another process. For example, in the field of electronic payments, the negotiation of the price and the payment of an electronic product is combined, as appears in [Cox, 95],[Ruiz, 01].

Therefore, a contract signing protocol that provided all previous requirements, would make the whole lifecycle of the negotiation and contract signing secure and efficient.

2.2 Related Work

To date, several contract signing protocols based on an on-line TTP have been proposed [Abadi, 02], [Kim, 99], [Yang, 03], [Yang, 05], [Zhou, 96]. However, in this section we only refer to [Yang, 05] since this protocol is an improvement on previous ones.

The goal of the protocol proposed by S. Yang et al. in [Yang, 05] is to guarantee non-repudiation in the delivery of the messages, and it could be used for contract signing. In fact, it was proposed for collaborative e-commerce. But since the protocol is conceived for message delivery, if we want to use it to sign contracts we have to execute it twice. In the first execution, Alice sends the signed contract to Bob, and Bob confirms the delivery but does not sign the contract. Then, in the second execution, Bob signs the contract signed by Alice and sends it to her so that she can confirm she has received it.

The protocol offers evidence as to whether one of the parties is not behaving correctly. It also offers confidentiality, and the TTP will never know the content of the contract. However, there are two main drawbacks. The most important is that the protocol does not guarantee the abuse freeness property because Bob obtains a signed contract before Alice obtains it. Therefore, Bob has an advantage over Alice.

Another important problem is that the TTP has a significant participation in the execution of the protocol because it participates in five of its six messages of each iteration. This could lead to the TTP becoming a bottleneck if there were many concurrent executions of the protocol. Furthermore, if we want to negotiate the terms and conditions of the contract, it would be necessary to offer an additional mechanism. This is because if we used this protocol for each message in the negotiation, then the complete process (negotiation plus contract signing) would present a high overload in the system, given the number of messages exchanged and the cryptographic operations made. Finally, one minor problem that could be solved easily is that the protocol does not satisfy the timeliness property.

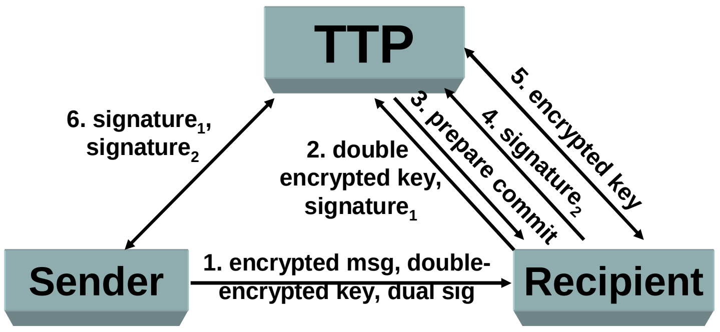
TTP
Sender
Recipient
1. encrypted msg, double-encrypted key, dual sig
2. double encrypted key, signature₁
3. prepare commit
4. signature₂
5. encrypted key
6. signature₁, signature₂

Figure 1: Secure exchange in Yang et als' proposal

3 SURENESS: A new SecURE NEgotiation and contractS Signing protocol

In this section we present a new contract signing protocol which satisfies the requirements introduced in the previous section to provide a robust contract signing protocol. Our protocol is based on an on-line TTP for those situations in which an e-notary has to authenticate a contract by signing it. Our protocol reduces the participation of the TTP to the minimum possible expression as regards the number of cryptographic operations (especially the asymmetric ones, due to their computation cost) and messages sent and received. It is also important to mention that it could be used with any existing public key cryptosystem, as long as this offers signature and ciphering. Thus, it could be used without generating new keys and certificates, unlike other protocols that require either specific keys (e.g., ElGamal keys [Garay, 99]) or the generation of new ones with special features, or a previous registration process with a contract TTP, as required in [Garay, 99], [Park, 03], [Wang, 05]. We have named the protocol SURENESS (SecURE NEgotiation and contractS Signing). In this protocol, we consider that none of the parties is going to act against its own interests. In addition to the primary protocol, we provide, resolve and abort subprotocols to guarantee that every party is able to complete the execution of protocol in time, without having to wait for actions by the other, potentially malicious, party.

3.1 Notation

In this section we provide a description of the notation used to specify the sequence of messages in our protocol specification.

Symbol	**Meaning**
[*Data*]	This indicates that this piece of *data* is optional, and may not be in the message.
H(*Data*)	A message digest of *Data*, obtained using a hash algorithm such as SHA2 [NIST, 04].
$\|Data\|^{K}$	*Data*, encrypted by a symmetric cipher such as AES [NIST, 01] using the key *K*.
$\|Data\|_{K}$	*Data* is authenticated using an HMAC algorithm with a cryptographic key *K*. This represents a message composed of two elements: *Data* and its cryptographic checksum.
$\|Data\|_{K1,K2}$	This is equivalent to $\|\|Data\|_{K1}\|^{K2}$
$\{Data\}_{X}^{-1}$	*Data* is signed using the private key of *X*.
$\{Data\}_{X}$	*Data*, encrypted for *X* using public key cryptography (RSA, ECDH, ECMQH,…). For computational efficiency, this is implemented using either a digital envelope (RSA) or an agreement exchange (ECC) as specified in [Blake-Wilson,02],[Hously,04].
X =>Y	This indicates that *X* sends a message to *Y*.

Table 1: Cryptographic notation

3.2 Definition of a contract in SURENESS

For this protocol, a contract has the following structure:

$$\{NID, A, B, Timestamp, Nonce, H(ContractDoc), H(SC_A), H(SC_B)\}_{TTP}^{-1}, SC_A, SC_B$$

The contract is basically a document signed by the e-notary (TTP) and it contains the following information:

- *NID (Negotiation Identifier)*. This is a unique identifier of the contract. The *NID* identifies the transaction performed between Alice and Bob. Although this identifier may not be globally unique, it is used to distinguish between the different negotiations or transactions performed by the same parties. Thus, the NID is used to identify the transaction to which a message belongs.
- *A*, *B*. These are the identifiers of the parties (Alice and Bob) between whom the contract is signed. This type of identifier is the digest of the party’s public key. It is used to avoid impersonation attacks, as we comment later in section 5.
- *Nonce*. A nonce (randomly generated number) received from Alice. It is introduced to avoid Bob’s having an advantage over Alice at a given moment. This issue is analyzed in the Section 4, in a subsection called abuse freeness.
- *ContractDoc (Contract Document)*. A document that reflects the agreed contract terms between Alice and Bob. This document could be expressed in natural language or it could follow a specified contract language, as proposed in [Tan, 00].

- *H(ContracDoc)*. This is the hash of the document which represents the contract terms.
- *H(SC_A), H(SC_B)*. These represent the hashes of the contract signed by Alice and Bob, respectively. The content of SC_A and SC_B is commented later. We can advance that it is basically the electronic signature of the contract with other information necessary to guarantee the non-repudiation and fairness of the protocol.

By means of this contract, the TTP testifies that Alice and Bob have reached an agreement or contract (identified by the *NID*). This is reflected in a document that contains some information, such as the hash of the contract document (*H(ContractDoc)*), the hashes of the signatures made by Alice and Bob, and the corresponding signatures. It also contains information about when the contract is signed by the TTP (*Timestamp*).

To sum up, our contract is an electronic signature made by the TTP that links the contract terms signed individually by each party. Thus, the contract contains the signatures of the TTP and the participating parties, as supposed.

From this contract, we can not deduce if the TTP (or e-notary) knows the contract content or not. The e-notary will own a copy of the contract if Alice and Bob consider that the e-notary ought to know it. For example, when the e-notary takes the responsibility for monitoring if the conditions of the contract are being complied with or not, or whether the e-notary has to certify, record and save a copy of the contract [Angelov, 05], [Jalali, 00], [Ruiz, 03], as occurs in real estate contracts.

In the following section, we describe the contract signing protocol when there is a phase where the contract terms are previously negotiated. Later, in a subsequent section, we present the description of the protocol when it is not possible, or it is not desirable to negotiate the conditions of the contract. For example, if there is a pre-established contract.

3.3 Normal Mode

In the normal mode, the protocol is composed of the messages that appear in Figure 1. Each type message is described in more detail below.

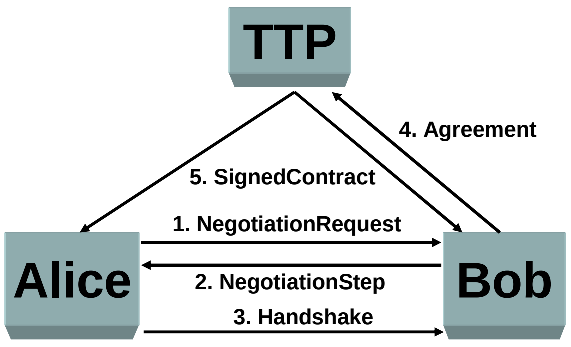

Figure 2: SURENESS messages in normal mode

STEP I. Alice=>Bob: NegotiationRequest

$\{\{\text{NID}, \text{Time}_1, \text{SeqN}, [\text{Cred}], \text{B}, \text{EnKey}, \text{SignKey}, \text{Flag}\}_{\text{A}}^{-1}, \text{ContractDoc}, \text{H(ContractDoc)}\}_{\text{B}}$

Where:

- *NID (Negotiation Identifier)*. The *NID* identifies the negotiation being performed between Alice and Bob. Although this identifier might not be globally unique, it is used to distinguish between the different negotiations or transactions performed by the same parties. Thus, the NID is used to identify the transaction to which a message belongs. This identifier or label could be a randomly generated number. However, in order to provide a high level of security we have decided to follow the principles proposed in [Gürgens,05]. These design principles recommend that the label has the following properties: verifiability, uniqueness and secrecy. Therefore, we have generated this identifier as *H(A,B,TTP,H(EnKey),H(SignKey))*.
- *$Time_1$*. This is the time until which Alice will wait for a response from Bob. In general, we represent with *$Time_X$* the time one party will wait for a response from the other one. This time should include a date and hour. It will delimit the time of a possible response. In this case, if a response does not arrive before *$Time_1$*, Alice will consider that Bob is not interested in continuing with the negotiation. This may be because Bob is not interested in her conditions. Thus, the negotiation is considered finished without an agreement. Therefore, there will be no contract signing process and no more messages are exchanged.
- *SeqN (Sequence Number)*. During the negotiation phase, it is possible to exchange several messages. Every message in this sequence must be unique in order to prevent reply attacks. For this reason, a *SeqN* field is present in the negotiation messages and each party must increment the *SeqN* value after receiving this type of messages.
- *Cred (Credentials)*. This is an optional field which can be used to provide the user's credentials. For example, these could be a SAML Assertion or Artifact [Maler,03].
- *B*. Bob's identifier. This field is the digest of Bob's public key. It identifies the intended receiver of this negotiation request, in this case, Bob. In this way, we can avoid any possible impersonation attack (we analyze this type of attack in section 4).
- *EnKey (Encryption Key)*. This is a symmetric key generated by Alice that is used to provide confidentiality to the following messages exchanged between Alice and Bob. The default symmetric cipher to employ is AES.
- *SignKey (Signing Key)*. This is a symmetric key generated by Alice. It is used to provide integrity to the subsequent messages exchanged between Alice and Bob. The default cryptographic checksum function to employ is HMAC [Krawczky,97] with SHA2 [NIST,04].
- *Flag*. This is used to indicate whether it is the last offer from Alice. If its value is true, it indicates that Alice will not accept counter-offers to this

offer. Then, Bob can either accept the offer or finish the negotiation. The offer is represented by the different terms indicated in the *ContractDoc* field.

- *ContractDoc (Contract Document).* This is a document that reflects a proposal of contract to be negotiated between Alice and Bob. The language used to express the conditions, obligations and rights associated to the contract will be that decided upon by Alice and Bob.

The *NegotiationRequest* message indicates the beginning of the execution of the protocol. The message is used by one of the parties when it decides to initiate the negotiation of the conditions of a contract in order to reach an agreement that will be reflected in a signed contract. The contract terms appear in the proposed contract document in the field called *ContractDoc*.

In this step, the contract documents (*ContractDoc*) and *H(ContractDoc)* are not signed by Alice to avoid the abuse freeness, since if Alice signed these data, then Bob would be able to present them to another party, so gaining an advantage over Alice.

STEP II. Bob=>Alice: NegotiationStep

$|\text{NID, Time}_2\text{, SeqN, [Cred], ContractDoc, Flag}|_{\text{SignKey,EnKey}}$

Bob sends this negotiation message to make a new offer (counter-offer) if he does not agree with the contract terms. If Bob does agree to the conditions, he sends a *Handshake* message instead. Thus, the *Handshake* indicates the end of the negotiation and the initiation of the contract signing process.

The *NegotiationStep* message is used by Alice and Bob on various occasions until one of the following conditions is reached:

- One of them accepts the conditions of the other party in the last *NegotiationStep* message. In this case, a *Handshake* message is sent.
- One of them receives a *NegotiationStep* message containing a last offer (flag is activated) that it does not agree with. In this case, the communication is closed.

STEP III. Alice=>Bob: Handshake

$|\text{RecKey}|^{\text{EnKey}}\text{, SC}_\text{A}$

Where:

- SC_A= $\{\text{NID,Time}_3\text{,B,TTP,}\{\text{Nonce,H(ContractDoc)}\}_{\text{TTP}}\text{, H(RecKey)}\}_{\text{A}}^{-1}$
  SC_A represents the contract signing by Alice.
- *RecKey (Receipt Key).* This is a symmetric key which will be used to receive the contract signed by the TTP. Its value is the result of performing the hash on the *Nonce*.
- *Nonce*. This is an array of randomly generated bytes of variable length.
- *Time$_3$*. This indicates the deadline until which Alice considers the agreement is possible with Bob. This time must be taken into account by Bob and TTP. Even if Bob signed his part of the contract, if it is signed after time *Time$_3$*, the TTP will not sign it and there will not be a valid contract.

This is the step where Alice decides to accept the contract by sending it to Bob. Thus, this message indicates the end of the negotiation process and the initiation of the contract signing process. However, in order to avoid Bob gaining an advantage over Alice, the signed contract is ciphered with the TTP's public key. The hash of the symmetric key, *RecKey*, is also sent signed, so Bob and the TTP could be sure the symmetric key was not changed by another party, because this key will be used to send the signed contract in a ciphered form.

STEP IV. Bob=>TTP: Agreement

There are two possibilities in this step:

a) $SC_A,\{SC_B\}_{TTP}$ or

b) $SC_A,\{SC_B\}_{TTP,}$ $|ContractDoc|^{RecKey}$

Where:

- $SC_B = \{NID,Time_4,TTP,H(ContractDoc)\}_B^{-1}$
  This represents the signing of the contract by Bob.
- *$Time_4$*. This indicates the time until which Bob will wait for the contract to be signed. This time should not be greater than *$Time_3$*. This is because the TTP also has to take into account that Alice, in *$Time_3$*, indicated that she wants the contract to be signed before this time. If Bob put a greater time, it would be useless because the TTP would have to sign the contract before *$Time_3$* indicated by Alice. If Bob agrees with the time specified by Alice he should put the same time. *$Time_4$* has been introduced in case Bob wants to further limit the deadline for signing the contract by TTP.

This message represents the agreement between the two parties for a contract. It contains part of the information received in the *Handshake* message (step III), i.e. the signing of the contract by Alice. Furthermore, it includes signed information by Bob that is sent ciphered to the TTP with its public key. This information reflects Bob's conformity with the contract. It is sent ciphered to the TTP in order to avoid Alice intercepting the message, which she could show to another party to gain an advantage over Bob. There are two versions of this message. In the first, Alice and Bob decide that the TTP does not need to know the content of the e-contract (case a). But, in the second, they deem it necessary that the TTP knows the e-contract (case b), e.g. in those scenarios where it is required that the TTP certifies, records and saves the contract. In order to avoid outside entities knowing the content of the contract, it is sent ciphered with the symmetric key (*RecKey*). When the TTP receives this message, then the TTP will check that the signatures are valid and that the contract document signed by each party is the same. In this case, the protocol continues in the following step, otherwise it finishes.

STEP V. TTP=>Alice, Bob: SignedContract

$|\{NID,A,B,Timestamp,Nonce,H(ContractDoc), H(SC_A),H(SC_B)\}_{TTP}^{-1}, SC_B|^{RecKey}$

Where *Timestamp* indicates the date and the time when the TTP signed the contract, once Alice and Bob had reached an agreement. This time should be less than the minimum between $Time_3$ and $Time_4$. That is, the TTP would choose the most restrictive time. As commented in the previous message, there are two possibilities: either both $Time_3$ and $Time_4$ are equal or $Time_4$ is less than $Time_3$.

The message represents the approval of the signed contract between Alice and Bob. It is signed by the TTP in order to prove that both entities agreed to the contract reached. Bob's signature is included so that Alice can have a copy of the signed contract by Bob. Thus, in case of dispute, the TTP will not be necessary.

3.4 Abort Subprotocol

In this section we present the abort subprotocol. We have defined it taking into account that it is possible that in some circumstances, a party that has signed a contract wants to cancel it prior to its being signed by the TTP ([Zhou, 01], [Gürgens, 05]). Thus, a party that has initiated the contract signing process can abort it fairly. As commented previously, the signing process is initiated with the *Handshake* message, it continues with the *Agreement* message and it finishes with the *SignedContract* message. Therefore, if a party wants to abort this signing process once initiated, he/she has to execute the abort protocol before the *SignedContract* message is generated by the TTP. For this abort protocol, the messages that we have needed to define are:

STEP I. Alice=>TTP: Abort

$\{NID, B, TTP, \{Nonce, H(ContractDoc)\}_{TTP}\}_{A^{-1}}$

This message is used by Alice in order to inform the TTP that she wants to abort the protocol. If the contract has not yet been signed, step II of the abort protocol is executed. Otherwise, the last message of the execution of the protocol is re-sent (*SignedContract*).

STEP II. TTP =>A,B: ConfirmedAbort

$\{NID,Timestamp,A,B,Abort\}_{TTP^{-1}}$

This message is received as confirmation that the protocol was aborted. This is indicated by the flag *Abort*.

3.5 Subresolve Subprotocol

It could occur that when one party, after having sent its contract signature to the other one or to the TTP, after the deadline, does not receive any answer. We will comment on the sequences of messages to exchange supposing that it was Alice who did not receive the answer.

STEP I. Alice=>TTP: Resolve

$\{\{NID,Time_X,B,TTP,Nonce,H(ContractDoc), RecKey\}_{A^{-1}}\}_{TTP}$

Where *Time*$_X$ is the time indicated in the *Handshake* or *Agreement* message. The time, when this message is sent, should be later than *Time*$_X$. Depending on the messages sent (or not) to the TTP, in the execution protocol, Alice could receive one of the following messages:

a) **STEP II. TTP =>Alice: SignedContract**

This message is the same as we commented in step V of the protocol and it would be received if the contract was signed and sent to the TTP by Bob.

b) **STEP II. TTP =>Alice: ConfirmedAbort**

$\{NID,TimeStamp,A,B,Abort\}_{TTP^{-1}}$

This message would be received if Bob cancelled the signing of the contract.

c) **STEP II. TTP =>Alice: NoContract**

$\{NID,A,B,Timestamp\}_{TTP^{-1}}$

If after the *Time*$_X$, the TTP has not received an abort or a signed contract, TTP sends a signed message to Alice indicating that there was no agreement.

3.6 Aggressive mode

The aggressive mode is useful in those cases where there is a predefined contract whose content is already known to the parties, but where some minimal details, e.g. dates, names of the parties and so on, have to be filled in. In this mode, the messages would be the following:

STEP I. Alice=>Bob: AgreementRequest

$\{\{NID,Time_1,[Credentials],B,EnKey, SignKey\}_{A^{-1}}\}_B,|ContractDoc|_{SignKey,EnKey},SC_A$

In this message, Alice sends an agreement request with the signed contract.

STEP II. Bob=>TTP: Agreement

a) $SC_A,\{\{NID,Time_2,TTP,H(ContractDoc)\}_{B^{-1}}\}_{TTP}$ or

b) $SC_A,\{\{NID,Time_2,TTP,H(ContractDoc)\}_{B^{-1}}\}_{TTP}, |ContractDoc|^{RecKey}$

STEP III. TTP=>Alice,Bob: SignedContract

$|\{NID,A,B,TimeStamp,Nonce,H(ContractDoc),H(SC_A),H(SC_B)\}_{TTP^{-1}}, SC_B|^{RecKey}$

4 Analysis and comparison with related work

In this section we are going to analyze the protocol from different points of view: security, requirements mentioned in section 2, and comparison with related work.

4.1 Replay Attacks

Replay attacks are avoided thanks to the use of the fields *NID* and *SeqN*. *NID* is the identifier of the transaction and represents an execution of the protocol, while *SeqN* is the number of negotiation messages within the execution. Each time a new message is received, the *SeqN* number is increased. Thus, in the same transaction or negotiation, if a message with an inferior *SeqN* value to the expected value is received, it will be rejected. Similarly, if a message with a correct sequence number, but from another negotiation, is received (a message with a previously used *NID*), the message will be rejected. In the same transaction, messages with the expected *SeqN* value can not be generated by an outside party unless one party has revealed the keys used (symmetric and/or asymmetric keys).

4.2 Timeliness

The protocol has the ability to stop, in a finite amount of time, its execution while at the same time preserving fairness. This property is assured thanks to the $Time_X$ fields that we have included in the messages of the protocol in order to limit the reception time of a message. Thus, when an entity receives a message, it checks that the actual time is later than the time indicated in $Time_X$, then the entity discards the message. The contract signing process is also atomic, since the contract is either valid or not valid at all after the time-window $Time_X$ has expired.

4.3 Impersonation

Abadi and Needham postulated some basic engineering practices for cryptographic protocols in [Abadi,96]. One of these principles is related to naming: "if the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message". Impersonation attack tries to convince some protocol party that the communication is being performed only between entities Alice and Bob, although there is a third party participating in that communication (impersonating Alice or Bob). We have included the identifier of the parties in the messages, so as to avoid this problem. If we did not use these identifiers, like in messages shown below, the following situation (M is a malicious party) could occur.

STEP I. *Alice => M*: NegotiationRequest

$\{\{\text{NID}, \text{Time}_1, \text{SeqN}, [\text{Cred}], \text{EnKey}, \text{SignKey}, \text{Flag}\}_{A^{-1}}, \text{ContractDoc}, \text{H(ContractDoc)}\}_M$

STEP II. *M => Bob*: NegotiationRequest

$\{\{\text{NID}, \text{Time}_1, \text{SeqN}, [\text{Cred}], \text{EnKey}, \text{SignKey}, \text{Flag}\}_{A^{-1}}, \text{ContractDoc},$

H(ContractDoc)$\}_B$

STEP III. *Alice* <=> *M* <=> *Bob*: NegotiationStep

|NID,Time$_2$,SeqN,[Cred],ContractDoc, Flag|$_{SignKey,EnKey}$

In this scenario, the malicious attacker gains access to all the information exchanged during the negotiation phase (he can learn the negotiation strategy of both parties, the contract conditions, etc.), Alice is unaware that she is not talking to Bob as she thought, and Bob does not know that there is a man-in-the-middle. However, in our protocol, to avoid this attack, we have included the identifier of the parties. Thus, any forwarded *NegotiationRequest* message in which the identity of the recipient does not match with the identity specified in the message can be interpreted as an attack. Furthermore, we also avoid Alice's being able to use Bob's name (or vice versa) in her communications with the TTP because the messages to the TTP are signed and include the identification of the parties. For this attack to be successful Alice would have to know Bob's private key (and vice versa).

4.4 Confidentiality

During the protocol execution, we have used both symmetric and asymmetric ciphering algorithms to ensure the confidentiality of the information exchanged. In all the messages, as far as possible, we have used symmetric cryptography for the sake of efficiency, especially in the different steps of the negotiation (step II). It is also used in the *Handshake* message that is used to finish the negotiation (step III) and in the *SignedContract* message to receive the signed contract (step V). As symmetric cipher we propose AES because since it was introduced no significant security problems have been revealed. In the cases where there had been no previous contact between the entities, it was necessary to use asymmetric cryptography (steps I, III and IV).

4.5 Abuse Freeness

Each party sends the TTP his/her signed agreement to the contract, ciphering it with the TTP's public key. Therefore, neither is Alice able to show to an outside party that Bob signed the terms of the contract, nor can Bob prove that Alice signed the same contract as he did. In step III of the protocol, Bob receives a signed message; however, the signature can not be linked with the content of the contract since the hash of the contract is ciphered with the TTP's public key. In order to show it to an outside party, Bob could try to make the envelope with the hash of the contract that he knows, which is the same as Alice knows. However, he does not know the field *Nonce*, so the envelope would not match and he would not be able to show it to an outside party. The same occurs if Alice intercepts the message between Bob and the TTP.

The *Nonce* is secure enough if it is generated by using a cryptographically secure pseudo-random number generator that takes into account the considerations mentioned in [Eastlake, 97], [Kürtz,07]. Thus, we are sure of the uniqueness and freshness of the number generated. Therefore, the enveloped information is secure because the cost of obtaining that information is equivalent to a brute-force attack [Kürt, 07], [Schneier, 95].

The TTP is the only entity that can create the contract from the individual signature of each party, showing that each party really signed it. As far as the protocols presented in section 2 are concerned, none satisfy this requirement. However, the new protocol presented here guarantees that none of the parties gets a copy of the contract until the TTP has signed it.

4.6 Non-repudiation and Dependency of Existence

Once the TTP receives the contract signature from Alice and Bob, it generates a signature to validate the transaction and to relate the information signed by the entities. The TTP signs the transaction identifier, a timestamp indicating the moment of the registry of the contract, the nonce contained in the envelope of the signature of the contract by Alice, and the hash of the contract signature by Alice and Bob. With this information, even if the TTP were not available, a third party would be able to check the validity of the contract. The steps would be the following: firstly, it would verify the TTP's signature; secondly, it would verify the Bob's signature. After that, it would check that the contract hashes are the same. Then, from the nonce inserted in the TTP's signature, and from the hash of the contract, it could calculate the envelope of the TTP in which Alice's signature is. Finally, it would check Alice's signature.

4.7 Recipient Role

The contract negotiation is initiated by one of the parties. However, depending on the steps followed in the negotiation, the recipient could become the sender if the recipient accepts the contract proposal. In this case, the recipient could send message III to the sender, thus becoming a sender. The role of recipient and sender are therefore symmetric in our protocol. Both have control of the protocol since, without both signatures the final contract is not possible. Thus, our protocol is not sender-oriented and the recipient can play an important role in the protocol because he/she is able to finish the protocol without involving the sender. Furthermore, we avoid extra messages in the protocol, which in turn, improves efficiency. Therefore, our protocol satisfies the recipient role requirement.

4.8 Secure Negotiation

The protocol, unlike those mentioned in section 2, incorporates the possibility of negotiating the contract in a secure way and does not allow either party to gain an advantage over the other. This security is provided by means of asymmetric signature and encryption in the *NegotiationRequest* message. The asymmetric encryption allows only the recipient to decipher the content. The information signed in this message is used to authenticate the user, confirm the keys that will be used in the following messages and avoid both replay (see also section 4.1) and impersonation (see also section 4.3) attacks. In *NegotiationStep* messages the authentication is based on symmetric cryptography by means of HMAC codes and the information is sent in a confidential way (see also section 4.4) by means of the encryption of the information using a symmetric cipher as AES.

Furthermore, the negotiation does not suppose an excessive overload, since it uses symmetric cryptography. We could separate negotiation and contract signing processes and carry them out with different protocols. However, as we comment in the following section this process would be more inefficient.

4.9 Efficiency

As commented in the introduction, our goal is to provide a protocol for those scenarios where the TTP has to participate in the contract signing, for example, by legal requirement. Since the TTP has to participate, our aim is that the number of cryptographic operations (especially the asymmetric ones, due to their computation cost) as well as the messages to be sent and received by the TTP be the minimum possible. At the same time we have also to satisfy all the security requirements established in section 2.1. Thus, our solution is more efficient than previous work if it satisfies two conditions. First, we use fewer messages and cryptographic operations. Second, we provide better security properties with these operations and messages.

In order to have a reduced computational cost, given the cryptographic operations, the protocol uses mainly symmetric cryptography (cipher) and hash functions, except in those operations related to non-repudiation or when it is necessary to send information to other parties that have had no previous contact, when asymmetric cryptography is used instead.

We present two tables (Table 2 and Table 3) where the cryptographic operations made for each party, and for each protocol, appear. Table 2 shows the comparison of the number of asymmetric cryptographic operations performed. In table 3 we compare the number of symmetric encryption/decryption and hash operations made.

Protocol	**Entity**	**Signature & Verification**	**Dual Signature & Verification**	**Encryption & Decryption**
[Yang,05]	A	4	2	4
	B	6	2	2
	TTP	6	2	2
SURENESS Normal mode	A	4		2
	B	4		2
	TTP	3		2
SURENESS Aggressive mode	A	4		2
	B	4		2
	TTP	3		2

Table 2: Asymmetric cryptographic operations of the protocols in a contract signing

For both protocols we have supposed that the set of cryptographic algorithms used are the same. Thus, we suppose both are using the same set of algorithms for hash, symmetric encryption and asymmetric (public key) encryption. In this case, we have supposed that the algorithms used are SHA2, AES and RSA, respectively. We have also supposed that the keys used in these algorithms have the same length.

In the Table 2 and Table 3, the protocol proposed by Yang et al. [Yang, 05] was introduced in section 2.2.1 and improves other previous works commented in that section. We compare this protocol with both modes of our SURENESS protocol (normal mode and aggressive mode). In Table 2, although operations related to signature are based on the encryption/decryption of a hash, we have separated the operations related to encryption and decryption, because in our proposal, asymmetric encryption is based on creating a digital envelope, that is, generating a symmetric key, ciphering the content with that symmetric key, and finally, ciphering the symmetric key with the asymmetric key. Therefore, the computational cost of the signature operation is less than the computation cost of the digital envelope. Furthermore, with this separation we clarify the different operations the protocol performs.

As a result of this comparison we realize that, in the protocol proposed by Yang et al., the number of public key operations made by the TTP is higher than SURENESS in any of the modes. If we supposed the same cost for all the operations of this kind, Yang et al.'s proposal needs ten operations, unlike ours, which only needs five. As for symmetric and hash operations (see Table 3) we can see that in the TTP the number of these operations made in both protocols is almost the same. Therefore, we can conclude that we have reduced the overload of the TTP and our protocol is more efficient as regards cryptographic operations.

We can also compare the two modes defined in SURENESS. As can be seen, the cost of public key operations is the same since the primitives used to make the contract signing process are the same. If we analyse the symmetric and hash operations we can see that the normal mode performs more operations. This is due to the fact that there are messages of negotiation which are based on this kind of cryptography. In this comparison we have only supposed one negotiation step. The more negotiation steps are used the more symmetric cryptography operations are needed. Therefore, the difference in the number of operations would be increased as the number of negotiation steps increases.

The SURENESS protocol, as far as the messages exchanged between all the parties is concerned, is less than in other protocols mentioned in related work. In our protocol, six messages are sent and received in the normal mode (if we suppose only one negotiation step in the comparison) or four messages are sent and received in the aggressive mode (without negotiation), unlike Yang et al.'s protocol (which needs two executions to sign the contract) where this number is twelve. Furthermore, if we compare the number of messages in which the TTP participates, we can see that in our protocol, the TTP sends two messages and receives only one message in both modes. But in the protocol proposed by Yung et al., the TTP receives four messages and sends six messages.

Even, if we compared SURENESS protocol with only one iteration of Yung et al.'s protocol (which fulfils similar features to those proposed here), our protocol has the same asymmetric cryptographic operations, and the number of messages is fewer. In SURENESS the TTP participates in three messages unlike the protocol proposed by Yang et al., where the TTP participates in five messages.

Moreover, SURENESS protocol satisfies all security requirements defined in section 2, unlike Yung et al.'s proposal. As commented in related work (see section 2.2) Yung et al.'s protocol does not satisfy abuse freeness, unlike SURENESS (see section 4.5). Therefore, our proposal improves the previous ones because it offers more security properties and it has a lower overload both in the number of messages sent and cryptographic operations.

Protocol	**Entity**	**Encryption & Decryption**	**Hash**
[Yung,05]	A		2
	B		
	TTP	2	3
SURENESS Normal Mode	A	3	6
	B	2	5
	TTP	2	3
SURENESS Aggressive Mode	A	2	5
	B	3	4
	TTP	2	4

Table 3: Symmetric cryptographic operations of the protocols in a contract signing

We also are going to justify the proposal of a protocol that offers both negotiation and contract signing (the SURENESS normal mode) from the efficiency point of view. We are going to compare this single protocol with the use of different protocols for each process. That is to say, a protocol for carrying out the negotiation and the aggressive mode (which does not incorporate negotiation) proposed here for the contract signing process. Our purpose is to show that our integral solution is not only justified from the business point of view but also from that of efficiency.

As we commented in section 2.1, in the negotiation we also have to provide security to avoid different kind of attacks. There are several possibilities for this negotiation process.

As a first option, we could use the existing non-repudiation protocols analysed in the related work [Abadi, 02], [Kim, 99], [Yang, 03], [Yang, 05], [Zhou, 96] to make the negotiation secure. Thus, we would execute one of these protocols for each offer. The main disadvantage of an execution of one of these protocols, for each offer, is that it supposes the exchange of a lot of messages, which makes it slow and not efficient. For example, with [Yang, 05], each offer would need the exchange of five messages between the different parties. Furthermore, the participation of the TTP is required (performing five public key operations), which is not needed in this phase. Therefore, we can see that this option is rather inefficient.

Secondly, we could make use of SSL/TLS with client authentication (to avoid impersonation). With this option, we would need five messages for SSL/TLS handshake plus the messages of the negotiation (two if we suppose a single step of negotiation). In the SSL/TLS handshake, apart from symmetric operations, three asymmetric operations are made (two signatures and one asymmetric encryption). We have considered only these because they are very time-costly. To sign the contract, this SSL-based negotiation with the aggressive mode supposes more messages and more cryptographic operations than our normal mode. In fact, eleven messages are needed, unlike SURENESS protocol, in which the whole protocol is executed in six messages. In the negotiation phase, in this normal mode, the parties only perform two asymmetric operations. However, in this option we have presented the cost is three asymmetric operations. Thus, in this SSL/TLS option, the participation of the TTP is the same but the parties have more messages and operations to perform.

The third option is to use a secure negotiation protocol such as [Darko, 06] combined with the aggressive mode. In the negotiation phase, the protocol [Darko, 06], which requires a coordinator, would need six messages and eight cryptographic operations (if we only suppose a single negotiation step). Therefore, the cost would also be higher than in the SURENESS normal model. Furthermore, the negotiation protocol proposed in [Darko, 06] does not guarantee abuse freeness property.

As a last option, we could have decided to define a negotiation protocol similar to the messages involved in the negotiation phase of the normal mode (two public key-based operations). This new negotiation protocol could be used with the aggressive mode. In this situation, the cost in messages would be the same for all parties (the number of messages in which the TTP participates is the same as the normal mode). However, in the whole process, Alice and Bob would perform two more public key-based cryptographic operations.

The analysis of these different combinations of using two different protocols for the negotiation and the contract signing process shows that SURENESS protocol, by combining both features, is more efficient. We can also mention than the aggressive mode (when no negotiation is needed) is also more efficient than previous works.

As a conclusion to the security analysis of our protocol (in the previous sections) and the different aspects related to the number of messages exchanged and the number of cryptographic operations made, we can point out that our protocol, in both modes, offers better security with less computational cost than previous work. Therefore, our protocol is more efficient than previous work.

4.10 Trust dependency on a third party

Unless the signers of the contract decide that the TTP should know the contract, the TTP will only know the contract hash. Thus, in our case, the trust dependency on a third party is minimal, unless the signers decide to deposit more trust or it is a requirement of the application or the environment in which they are working [Angelov, 05], [Jalali, 00], [Ruiz, 03].

4.11 Availability

Apart from these features, and since we are considering cases where the presence of a TTP on-line is mandatory (most of the cases by legal requirement), we could have problems in the service if the TTP is either not available or if it has to support many concurrent contract signings. Thus, the TTP is a possible single-point-of-failure in a contract signing protocol. Nowadays, this problem could be solved with replication techniques, load balancing servers or solutions based on grid [Rabinovick, 02], [Schroeder, 00], [Zegura, 00]. In any case, if the TTP is not available at the moment of signing the contract, the parties could try to send the messages later. When the deadline indicated in the $Time_X$ arrives, the protocol is considered aborted. This mechanism allows us to preserve fairness and timeliness properties.

4.12 Formal validation

We have carried out a formal validation of the different protocols and sub-protocols proposed using the Automated Validation of Internet Security Protocols and Applications (AVISPA) tool [AVISPA, 07], [Armando, 05]. The validation process is the following. First, we specify our protocol in the High Level Protocol Specification Language (HLPSL) [Chevalier, 04]. Then, the AVISPA tool translates it into the Intermediate Format (IF) specification [AVISPA, 07], [Armando, 05]. Finally, this IF specification is analyzed invoking state-of-the-art back-ends that this tool provides, which are currently: On-the-Fly Model Checker (OFMC) [Armando, 05],[Basin, 03], Constraint-Logic-based Attack Searcher (CL-AtSe) [Armando,05],[Turuani, 03] SAT-based Model Checker (SATMC) [Armando, 03], and Tree Automata-based Protocol Analyzer (TA4SP) [Boichut, 04]. These back-ends allow us to check a set of automatic analysis techniques such as protocol falsification or abstraction-based verification. Concretely, AVISPA allows us to check if the machine of the protocol is correctly designed (non-deterministic protocols), replay attacks, confidentiality, impersonation, secrecy and authentication (weak and strong). In AVISPA non-repudiation properties are specified as a set of authentication goals as mentioned in [Santiago, 06]. Furthermore, the back-ends of this tool follow the standard Dolev-Yao model, in which the intruder is assumed to have control over the network. Thus, the intruder is able to perform several tasks. First, he can receive all messages and store them. Second, if he has the key used to cipher the messages, he tries to decrypt them and obtain the different information exchanged. Third, he builds new messages (based on the knowledge he has) and sends them to any other agent.

In the process of validation with AVISPA, the most important step is the specification of the SURENESS protocol by means of HLPSL. The process followed to build that specification is the following. First, we defined the different roles for each party that can participate in the system: Alice, Bob and TTP. For each role, we specified the information that each initially knows and the different transactions that take place since the protocol is modelled as a finite state automata. Thus, each transaction is fired when a message is sent or received. Specifically, in each state of each role we defined the information to send or receive as well as the different information that should be authenticated and/or maintained in secret between the parties. In HLPSL, the secrecy is specified by means of *secret* events, which define a security goal to be satisfied. In each state the authentication goals are specified through *witness* and *request* directives. With *witness*, we declare what a party asserts and what it wishes to communicate to another party, e.g., for role TTP, a witness is the signed contract that he sends to Alice and Bob. On the other hand, with *request*, a party declares the belief in some information specified in a *witness*. Continuing with the same example, for Alice and Bob, the signed contract is specified as a request. Second, once we have declared each role, then, these roles are composed together in sessions. In a session we explain the different information that is shared between the different roles that participate in that session (e.g., TTP's public key, the contract to negotiate and sign, etc). We have defined a session with all parties with honest behaviour. We have also defined another session where one of the parties has a dishonest behaviour (the intruder represents the dishonest agent). In this specification this is achieved through the role named *environment*. Thus, as a third and last step, in the *environment* we specify both the knowledge of the intruder and the different sessions. We have also specified the different goals of authentication to be satisfied. In our case, the final goal is to achieve the authentication of the signed contract and proof that both parties have received it.

This specification has been tested with the different back-ends mentioned above. As a result, these back-ends return attacks (if any) in a readable output format. In our case, no security flaws were revealed after the tests.

4.13 Conclusion

As a conclusion to this analysis, we can affirm that we have proposed a protocol that satisfies the requirements established in section 2. Furthermore, our protocol is more efficient than the previous works commented on in section 2. Efficiency is measured from both points of view. First, with regard to the number of cryptographic operations and the number of messages that the TTP has to take part in. Second, from that of the security provided. From the tables and the analysis made in this section (in the efficiency part), we can conclude that we have reduced the workload of the TTP compared to the previous proposals and, at the same time, we have improved security of the proposals. Additionally, we could use replication techniques, load balancing, server and cluster solutions to achieve greater availability and a better service in the TTP.

5 Variations to the Protocol

In this section, we provide some variations to the protocol proposed in section 3. These modifications are aimed at facilitating the subsequent contract management by the end-users and other parties without having to develop new primitives to verify the contract.

It is clear that for the execution of the protocol, the user needs a SURENESS-compliant implementation. Although it can be based on standards like CMS (Cryptographic Message Syntax) [Hously, 04], or XML (Extensible Markup Language) Encryption and Signature [W3Cb, 02] that are supported for most of the cryptographic libraries, a further requirement is the building of supplementary primitives to create and verify the contract from these standards. The structure of the contract that we introduced in section 3 has three parts: the contract signed by the TTP, the contract signed by Alice, and the contract signed by Bob. Each part of the contract represents the (CMS/XML) signature of a byte array which contains DER-encoded contract information (not the contract document itself). Thus, to verify the contract we need to verify the CMS or XML signatures. Then, we have to parse these DER structures and, finally, to verify that the contract document matches with the information contained in the three separated structures. Therefore, for the last two steps we need to develop some additional SURENESS-compliant primitives to verify the contract.

However, it would be desirable if once the parties have signed the contract they were able to manage the contract without additional SURENESS-compliant primitives - only with CMS/XML Signature ones. It would also be interesting that the end-user could (if he/she wants) send the contract to another party or to another e-notary. Thus, this new party could use the generic cryptographic tool that he/she uses normally. Currently, many end-user-oriented applications only offer commands to create and verify signatures in the abovementioned standards but they are not able to process these additional structures. They are only able to know that the DER encoded expression is correctly verified. On the other hand, if the document contract was directly included in a CMS/XML signature (detached or enveloping), without any special DER structure, the user would only have to use his/her application as usual. For example, the majority of Linux distributions include the *openssl* command (from OpenSSL cryptographic library). In this tool the processing of the CMS signature is provided. However, the original format of the contract proposed here would involve the development of new code. Furthermore, the majority of the libraries support CMS formats but some of them cannot work with ASN.1 structures in a suitable way. In a similar way, we find the same problems when we consider an XML signature.

We propose some variations to the protocol so that the final signed contract is directly included in the CMS/XML signature. As mentioned, the main advantage we obtain with this modification is that the user could process the contract in an easier way. On the other hand, the main disadvantage is that the messages are more complex from the point of view of computational cost since the structures are more complex. The contract now has the following format:

Contract: $\{\text{ContractDoc}\}_{A^{-1},B^{-1},TTP^{-1}} + {}_{TST^{-1}}$

Basically, the contract is now a CMS/XML signature which contains the information of three signers in the *SignerInfos* field (if we are using CMS) or in the *SignedInfo* and *Object* elements (if we are using XML). Additionally, ,associated to each *SignerInfos* field, the TTP includes a Timestamp according to the RFC 3161 format. Therefore, this structure could be processed by any application that supports the verification of CMS/XML signatures.

This signature that represents the signed contract could be expressed in CMS or XML depending on the choice indicated in the first message of the protocol that is described below. Similarly, the contract document could be within the signature or not, depending on the choice indicated in a new field named *AdditionalInfo*. The contract also contains a timestamp (Timestamp Token as appears in RFC 3161) issued by a Timestamp Authority or by the e-notary acting as a timestamp authority. This new contract format implies some changes in the messages described in section 3. Below, we detail the new fields included in each message involved in this variation:

STEP I. A => B: NegotiationRequest.

$\{\{\text{NID},\text{Time}_1,\text{SeqN},[\text{Credentials}],\text{B},\text{EnKey},\text{SignKey},\text{Flag}\}_{A^{-1}},$
$\text{ContractDoc},\text{H(ContractDoc)}, \underline{\text{AdditionalInfo}}\}_B$

The *AdditionalInfo* field contains information about the type of signature (CMS or XML) and its options (enveloping, detached…).

In this variation, *NegotiationStep* message (step 2) requires no changes at all.

STEP III. A=>B: Handshake.

$|RecKey|^{EnKey}$,{NID,Time$_3$,B,TTP,{Nonce, $\underline{\text{SignatureValue}}$}$_{TTP}$, H(RecKey)}$_A{}^{-1}$

In this message, unlike that in Section 3, the *H(ContractDoc)* value has been replaced by the *SignatureValue* value (underlined) where *SignatureValue* field contains the value of the signature according to the specified format in the *AdditionalInfo* field. Thus, in this message *SignatureValue = {ContractDoc}*$_A{}^{-1}$.

STEP IV. B=>TTP: Agreement.

a) SC$_A$, {NID,Time$_4$,TTP,$\underline{\text{SignatureValue}}$}$_B{}^{-1}$ or
b) SC$_A$, {NID,Time$_4$,TTP,$\underline{\text{SignatureValue}}$}$_B{}^{-1}$, $|ContractDoc|^{RecKey}$

This pair of messages has only one difference with the original ones. In these messages, the *H(ContractDoc)* value has been replaced by the *SignatureValue* value (also underlined) containing Bob's signature according to the format indicated in the *AdditionalInfo* field. Thus, in this message *SignatureValue = {ContractDoc}*$_B{}^{-1}$.

STEP V. TTP=>A,B: SignedContract.

$|\{ContractDoc\}_{A^{-1},B^{-1},TTP^{-1}+TST^{-1}}|^{RecKey}$

This new message, instead of having three different parts, each with a separated signature, only contains, as data, the contract document. Therefore, any user could easily verify the signed contract, the signers and the timestamp. Furthermore, the number of evidences to store for long-term signature validation processes is less than in the original protocol. On the other hand, as a main drawback, the number of electronic signature operations to make and to verify is higher.

6 Conclusions and Future Work

The use of contract signing protocols based on a TTP is required in some B2B and DRM systems, or by legal requirement, in order to give validity to e-contracts. Current proposals for this kind of protocol present two main drawbacks. They do not take into account the incorporation of some fundamental aspects for B2B and DRM: confidentiality and the secure negotiation of the contract conditions and, in some protocols, the TTP has an important participation in the protocol which might cause bottlenecks when there are many concurrent executions of these protocols. Another problem found in some solutions is that they require the generation of new keys and certificates.

Confidentiality is important to achieve the privacy of the information exchanged between the different parties that sign a contract. Thus, only the interested parties know the information about the contract conditions. On the other hand, the negotiation phase is intrinsic to the contract signing, and this phase must be carried out safely, guaranteeing aspects such as confidentiality, integrity and avoiding replay-attacks. Furthermore, the integration of the negotiation with the contract signing protocol would guarantee higher efficiency than when using two different protocols for each process.

As a response to these problems we have presented a new protocol named SURENESS. Our protocol improves upon previous ones with the new requirements and with good efficiency for the cryptographic operations and the number of messages. Furthermore, the protocol could be used without generating specific keys or new ones with special features, or a previous registration with a TTP.

There are a number of future research directions. One issue is the extension of the contract negotiation to a multi-party contract environment while maintaining the idea of the involvement of the e-notary to give validity to the transaction. One possible scenario would be the buying of real estate between several clients and several vendors, and where the e-notary validates the transaction by signing it as specified by the laws.

Acknowledgements

We would like to thank the anonymous reviewers for their comments and suggestions. They have significantly contributed to improve the quality of this paper. We also want to thank Rafael Marín and Óscar Cánovas for their valuable comments on some issues of this paper.

This work has been partially funded by "Programa de Ayuda a los Grupos de Excelencia de la Fundación Séneca 04552/GERM/06".

References

[AAVV, 05] AA.VV., Código Civil, Boletín Oficial del Estado (26ª Ed). 2005.

[Abadi, 96] M. Abadi, R. Needham, Prudent engineering practice for cryptographic protocols, IEEE Transactions on Software Engineering, 22(1):6-15, January 1996.

[Abadi, 02] M. Abadi, N. Glew, B. Horne, B. Pinkas, Certified email with a light on-line trusted third party: design and implementation, The Eleventh International World Wide Web Conference, Honoloulu, Hawaii, USA, 2002.

[Aknine, 04] S. Aknine, S. Pinson, M. F. Shakun, An Extended Multi-Agent Negotiation Protocol, Journal of Autonomous Agents and Multi-Agent Systems, 8, 5– 45, Springer Science+Business Media B.V., Formerly Kluwer Academic Publishers B.V. 2004.

[Angelov, 05] S. Angelov, S. Till, P. Grefen, Dynamic and Secure B2B E-contract Update Management, Proceedings of the 6th ACM Conference on Electronic Commerce (EC'05), Vancouver, Canada. 2005.

[Asokan, 98] N. Asokan, V. Shoup, M. Waidner, Asynchronous protocols for optimistic fair exchange, Proceedings of the IEEE Symposium on Research in Security and Privacy, 1998.

[Armando, 03] A. Armando, L. Compagna, P. Ganty, SAT-based Model-Checking of Security Protocols using Planning Graph Analysis, In Proc. of the 12th International Symposium of Formal Methods Europe (FME), LNCS 2805, Springer-Verlag 2003.

[Armando, 05] A. Armando, A. et al., The Avispa Tool for automated validation of internet security protocols and applications, Proc. Of Computer Aided Verification, Lecture Notes in Computer Science (LNCS) 3576. Springer Verlag, 2005.

[Avispa, 07] Avispa Project: “Automated Validation of Internet Security Protocols and Applications (AVISPA)”. http://www.avispa-project.org/

[Basin, 03] D. Basin, S. Mödersheim, L. Viganò, An On-The-Fly Model-Checker for Security Protocol Analysis, In E. Snekkenes and D. Gollmann, editors, Proc. Of ESORICS’03, LNCS 2808, Springer-Verlag 2003, pp. 253-270.

[Boichut, 04] Y. Boichut, et al., Improvements on the Genet and Klay Technique to Automatically Verify Security Protocols, Proc. of AVIS’04, ENTCS.

[Blake-Wilson, 02] S. Blake-Wilson, P. Lamber, Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS), Request for Comments 3278, 2002.

[Cox, 95] B. Cox, J. D. Tygar, M. Sirbu. NetBill security and transaction protocol. In Proceedings of the 1st conference on USENIX Workshop on Electronic Commerce - Volume 1, 77-88. 1995.

[Darko, 2006] S. Darko-Ampem, M. Katsoufi, P. Giambiagi, Secure Negotiation in Virtual Organizations. In Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops. IEEE Computer Society. 2006.

[Delgado, 01] J. Delgado, I. Gallego, X. Perramon. Broker-Based Secure Negotiation of Intellectual Property Rights. In Proceedings of the 4th International Conference on Information Security, pp. 486-496. Springer-Verlag. 2001.

[Dierks, 99] T. Dierks, C. Allen, The TLS Protocol Version 1.0. RFC 2246. January 1999.

[Eastlake, 94] D. Eastlake, S. Crocker, J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[Garay, 99] J. Garay, M. Jakobsson, P. MacKenzie, Abuse-free optimistic contract signing, In proceedings of CRYPTO'99, Lecture Notes in Computer Science 1666, pp. 449-466. 1999.

[Gilbert, 03] H. Gilbert, H. Hanschuh, Security analysis of SHA-256 and sisters, Selected Areas in Cryptography 2003 (SAC 2003), Otawa, Canada (2003).

[Gürgens, 05] S. Gürgens, C. Rudolph, H. Vogt, On the security of fair non-repudiation protocols, International Journal of Information Security, volume 4, issue 4, 253-262. 2005.

[Jalali, 00] M. Jalali, G. Hachez, C. Vasserot, FILIGRANE: a security framework for trading of mobile code in Internet, Autonomous Agents 2000 Workshop: Agents in Industry, Barcelona, Spain, 2000.

[Kim, 99] K. Kim, S. Park, J. Baek, Improving fairness and privacy of Zhou-Gollmann’s fair non-repudiation protocol, Proceedings of 1999 ICPP Workshops on Security (IWSEC), IEEE Computer Society, Sep. 21-22, pp.140-145. 1999.

[Kremer, 02] S. Kremer, O. Markowitch, J. Zhou, An intensive survey of fair non-repudiation protocols, Computer Communications, 25(17): 1606-1621. Elsevier, Nov. 2002.

[Kötz, 97] H. Kötz, European Contract Law: Formation, Validity, and Content of Contracts; Contract and Third Parties, (Tony Weir trans., Hein Kötz & Axel Flessner eds., 1997.

[Kürtz, 07] K. O. Kürtz, R. Küsters, T. Wilke. Selecting theories and nonce generation for recursive protocols. In Proceedings of the 2007 ACM workshop on Formal methods in security engineering, pp. 61-70. Fairfax, Virginia, USA. 2007.

[Limthanmaphon, 00] B. Limthanmaphon, Y. Zhang, Z. Zhang. An Agent-Based Negotiation Model Supporting Transactions in Electronic Commerce. Proceedings of the 11th International Workshop on Database and Expert Systems Applications, IEEE Computer Society, 2000.

[Maler, 03] E. Maler, P. Mishra, R. Philpott, Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1, September 2003. OASIS Standard.

[Markowitch, 02] O. Markowitch, D. Gollmann, D., S. Kremer, On fairness in exchange protocols, In Pil Joong Lee and Chae Hoon Lim, editors, 5th International Conference on Information Security and Cryptology (ICISC 2002), volume 2587 of Lecture Notes in Computer Science, pages 451-464, Seoul, Korea, November 2002.

[NIST, 01] National Institute of Standards and Technology, Specification for the Advanced Encryption Standard (AES), FIPS 197. November 26, 2001.

[NIST, 04] National Institute of Standards and Technology (NIST), Federal Information Processing Standards (FIPS), Publication 180-2, Secure Hash Standard (SHS). 2004.

[Park, 03] J. J. Park, E. Chong, H. Siegel, I. Ray, Constructing fair exchange protocols for e-commerce via distributed computation of RSA signatures, In Proc. of 22th Annual ACM Symp. on Principles of Distributed Computing (PODC'03), pp. 172-181. ACM Press, 2003.

[Rabinovick, 02] M. Rabinovick, W. Spatscheck, Web caching and replication, Addison Wesley, 2002.

[Röhm, 98] A. W. Röhm, G. Pernul, G. Herrmann, Modeling Secure and Fair Electronic Commerce. In 14th Annual Computer Security Applications Conference (ACSAC '98), 1998.

[Rosenzweig, 05] D. Rosenzweig, D. Runje, W. Schulte. Model–Based Testing of Cryptographic Protocols. In Trustworthy Global Computing. 2005.

[Ruiz, 01] A. Ruiz, G. Martínez, O. Cánovas, A. F. Gómez-Skarmeta, SPEED Protocol: SmartCard-based Payment with Encrypted Electronic Delivery, Proc. Information Security Conference, Málaga, Spain, 2001, 446-461

[Ruiz, 03] A. Ruiz, L. Baño, C.I. Marín, O. Cánovas, A. F. Gómez, A Secure Infrastructure for Negotiation and Purchase of Intellectual Property Rights (SECURingIPR). Proceedings International Conference on Communication, Network, and Information Security, New York USA, 2003.

[Santiago, 06] J. Santiago and L. Vigneron, Automatically Analysing Non-repudiation with Authentication, Proceedings of 3rd Taiwanese-French Conference on Information Technology (TFIT), 541--554, Nancy, France, 2006.

[Schneier, 95] Schneier, B. (1995). Applied cryptography (2nd ed.): protocols, algorithms, and source code in C (pág. 758). John Wiley & Sons, Inc.

[Schroeder, 00] T. Schroeder, S. Goddard, B. Ramamurthy, Scalable Web Server Clustering Technologies, IEEE Network, May/June 2000, pp. 38-45.

[Tan, 00] Y. Tan, W. Thoen. DocLog: An Electronic Contract Representation Language. In Proceedings of the 11th International Workshop on Database and Expert Systems Applications. IEEE Computer Society. 2000.

[Turuani, 03] M. Turuani, Sécurité des Protocoles Cryptographiques: Décidabilité et Complexité, Phd, Université Henri Poincaré, Nancy, December 2003.

[Wang, 05] G. Wang, An abuse-free fair contract signing protocol based on the RSA signature. Proceedings of the 14th international conference on World Wide Web, 412 – 421, Japan. 2005.

[W3Cb, 02] Word Wide Web Consortium (W3C). XML-Signature Syntax and Processing. W3C Recommendation. 12 February 2002.

[Yang, 03] S. Yang, S.Y.W. Su, H. Lam, A Non-Repudiation Message Transfer Protocol for E-commerce. IEEE International Conference on E-Commerce Technology (CEC'03), 2003.

[Yang, 05] S. Yang, S.Y.W. Su, H. Lam, A non-repudiation message transfer protocol for collaborative e-commerce, International Journal of Business Process Integration and Management 2005 - Vol. 1, Nº.1. pp. 34 – 42.

[Zegura, 00] E. Zegura, M. Ammar, Z. Fei, S. Bhattacharjee, Application Layer Anycasting: A Server Selection Architecture and Use in a Replicated Web Service, IEEE/ACM Transactions on Networking, Vol. 8, No. 4, August 2000, pp. 455-467.

[Zhou, 96] J. Zhou, D. Gollmann, A Fair Non-repudiation Protocol, proceedings of 1996 IEEE Symposium on Security and Privacy, pp. 55-61, Oakland, CA, May 1996.

[Zhou, 01] J. Zhou, Non-Repudiation in Electronic Commerce. Artech House Publishers, 2001.