protection it is so-called *watermarking* [12], [4]. Watermarking techniques are being used also for the following purposes content archiving, meta-data insertion, tamper detection, digital fingerprinting, broadcast monitoring. In particular, we are focused on watermarking for tamper detection and meta-data insertion to ensure the authenticity of an e-document.

Our research goes beyond watermarking process. It consists in developing a web site where e-documents can be signed and also certified. In particular, we propose to carry out two main activities:

- Providing copyright protection to a digital document (watermarking);
- Verifying authenticity of a signed digital document, i.e. a process for retrieving information that includes the signature of a document, and that it is used to provide evidence of originality.

With the implementing of these processes, it would be possible to determine the authenticity of a document and, thus, to know whether it was forged or not. For now, we do the following consideration: *if a document is edited after being signed, then the signature must be corrupted, and the authenticity of the document disproved.*

In this paper, we describe some implementation details and security aspects about developing a web site able to certify the authenticity of an e-document, based on a symmetric cryptographic protocol.

The rest of the paper is organized as follows:

Section 2 outlines our research project; Section 3 describes safety aspects to be taken into account to sign and verify digital signatures from the web service point of view, or from the client side;
Section 4 introduces some formalism used in Section 5, to explain the authentication protocol and the general protocol introduced in Section 6. In Section 7, we describe some related work and we compare such works with ours; Finally, Section 8 describes the conclusions and future work.

2 OUTLINE

The general objective of this research is to implement a secure web site able to provide two main services:

1. to include a robust signature into a digital document; and
2. to certify the authenticity of a signed digital document.

Figure 1 ilustrates the general process; there, *S* represents the web site (the Server) and *C* represents the client. An arrow denotes a data flow from one participant to the other, and a double arrow means interchange of data between the participants. In addition, a down arrow denotes a local process.

Figure 1 illustrates that the participants need a initial knowledge at the beginning of all processes. Then, an authentication procedure is necessary before the client may sign or verify signatures.

According to our general objetive, we have splitted the security mechanisims in two leves: a) Host and b) Network.

The host level involves the initial knowledge that *C* and *S* own before the run of the protocol, and a local process

either in the client side or in the server side. In this case, to generate *R* from applying process *signing(…)* or *revealing(…)* in order to obtain a signature or reveal the hidden information of the e-document, see Figure 1.

The network level involves three main processes:

i) authentication;
ii) sending and reception of data (*SDD* in Figure 1 denotes sending a digital document); and
iii) sending the result *R* according to the process being requested either signing an e-document or revealing the content of the signature, *signing(…)* or *revealing(…)* respectively.

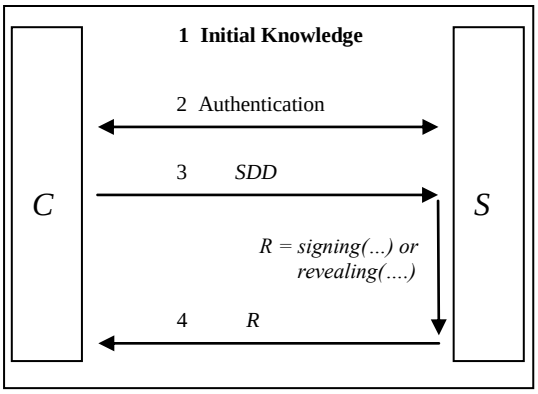

Figure 1: General Process, signing and consulting signatures in digital documents.

3 SECURITY MECHANISMS IN HOST LEVEL TO IMPLEMENT SIGNATURES

3.1 Steganography and Watermarking

Steganography is the art of data covering, data encapsulation or data hiding, [2]. Although steganography is diferent from cryptography, many authors categorize steganography as a form of cryptography, because of their similarities, [11].

The process of steganography generally involves two parts: the message to be hidden and the carrier[11]. The carrier, particularly in computing, is usually a file with a specific format (jpg, bmp, png, pdf, mp3, etc), where one wants to hide the message to be sent.

Steganography is crucial for important applications in the digital world, most notably in digital watermarking.

Watermarking is used primarily for document identification, and involves embedding information in a carrier. Information embedding must not be distinguishable; that it, changes to the carrier should not be evident. Some of the main applications of watermarking described by M. Vidyasagar Potdar, et. al. [23] include protection of copyright, content management, meta-data insertion, broadcast monitoring, detection of documents' violation and fingerprint.

This research is focused on these kinds of applications. Now, let *F* be an e-document (the carrier) and *m* be any message (to be hidden with watermarking purposes); after a steganography process we obtain *F´;* and which must be such that:

$$F \neq F' \wedge F \sim F'$$

Operator ~ denotes that *F* is equal to *F'* according to the human view (in average); since, they are similar.

3.2 Cryptography

Cryptography is defined as the art of writing or solving codes.[2] Cryptography is also defined as the study of

[2] Concise Oxford Dictionary definition.

mathematical systems involving two types of security problems: privacy and authentication, [5].

Privacy prevents extraction of information transmitted over a public channel by unauthorized users. With this, the sender of the message is ensured that the message will be read only by the intended recipient. Therefore, we should encode the signature before applying steganography.

Authentication ensures the recipient about the legitimacy of the message issuer. Authentication is usually related to two terms: message authentication and user authentication:

- Message authentication consists in verifying that a message was sent by, who claims to have edited it.
- User authentication consists in verifying that an individual is who claims to be.

In general, cryptography has two main mechanisms:

- Symmetric key cryptography: we can encode and decode a message using the same key. The best known algorithms for this type of cryptography are: Digital Encryption Standard (DES), [7], ; and the algorithm Advanced Encryption Standard (AES), [6]; and
- Asymmetric key cryptography: we must have two keys, one to encode the message and the other to decode it (reciprocally). One of the most known algorithms in for this type of cryptography is RSA, [13].

In this work, we have used AES algorithm.

4 FORMALISM HOST LEVEL AND NETWORK LEVEL

The next two subsections explain the formalisms used in our security protocol, described in the following section.

4.1 Formalism in host level

Symmetric keys will be represented using this notation K_{AB}, where *A* and *B* denote the agents sharing that key. The symbol $\{| m |\}_K$ denotes that a message *m* has been ciphered under the key *K*.

Let *F* be an e-document, *K* be a symmetric key (with implicit cyphering algorithm), *al* be a steganography algorithm, and let {*dp*} be the private data; that is, the information to hide. We apply the following formula to obtain a signed e-document using these data:

$$F' = signing(F, \{dp\}, al, K)$$

where *F'* denotes the new e-document, which provides the following: F≠F' and F ~ F' (see section 3.1 for this property).

Now, in order to know who has signed an e-document, we use the following formula:

$$\{dp\} = revealing(F', al', K)$$

where *al*' is the inverse algorithm of *al*. So, once {*dp*} has been analyzed by the client, it can know who the signer is, and accept or reject the authenticity of an e-document.

So far, we have implemented the previous formalization in a desktop application using two diferent algorithms for symmetric cryptography: a) AES and b) Password-based. The latter algorithm consists in applying a hash function into

a password which is introduced by the user, and after that, applying to it a XORing function.

4.2 Formalism in network level

When one consults any website, the traveling information could be seen by one or more Spies.[3] That means that any non-encrypted message transmitted on the network is considered unsafe. Therefore, it is necessary to implement security mechanisms when Internet applications are carried out. One possibility to get around this problem would be to implement a security protocol; another one, but that does not provide authentication, is to implement secure tunneling, i.e. a virtual private network (VPN). In our approach, we have chosen to do with a security protocol.

A security protocol is a set of rules and conventions whereby one or more agents agree about each other's identity, usually ending up in the possession of one or more secrets [22]. These rules are usually encrypted under a cryptography scheme presented in the previous section.

The formal specification of a protocol depends on the formal language used to verify it. We have verified our authentication protocol using the AVISPA tool web interface (as shown later on in the text). However, for illustration issues, we will use Alice and Bob notation from three views: i) *initial knowledge representation*; *ii) the sending and the reception of messages by the participants*; and iii) *the local process representation.* For this purpose, we will use the following notation:

Initial knowledge:

$A : ik$

This means that initially agent *A* knows *ik.*

Sending and Receiving Messages:

$n. A \rightarrow B: m$

This means that at step *n* agent *A* sends message *m* to agent *B*, which *B* receives.

$n2. A \rightarrow B: m;\{n\}$

This means that at step *n2* agent *A* sends message *m;{n}* to agent *B*, which *B* receives; here, symbol ";" denotes message concatenation.

Local process representation of message flow:

$n. A \rightarrow B: m$

$m2 = f(m)$

$n2. A \rightarrow B: m2$

This means that at step *n2* agent *A* sends message *m2* to agent *B*, which *B* receives; message *m2* was generated via a local process, *f(m)*, run between step *n* and *n2*.

5 PROTOCOL AUTHENTICATION BASE OF THE IMPLEMENTATION

5.1 On the need of formal verification in security protocols

Security protocols comprehend of only a few messages, but amazingly, they are very hard to build correctly. For instance, the detection of a flaw in the 3-message Needham- Schroeder public key (NSPK) protocol took roughly 17 years [17] after its introduction. We can cite another example, the Andrew Secure RPC protocol, which was found to have failures 2 years after its inception and it was amended by Burrows, Abadi and Needham [3]; that version actually was also flawed, and Lowe suggested a new

[3] Sniffers are usually used to capture packets on a network.

version, [18]. There exist a lot of examples of faulty protocols documented in Clark-Jacob library [4] and AVISPA library.[5]

Therefore, security protocol verification has attracted a lot of interest in the formal methods community yielding an abundance of tools and techniques in the last few years. For a survey for formal verification in security protocols see [16].

In next section, we describe the authentication protocol, which is the base of our implementation. This protocol has been verified formally and we explain such details.

5.2 The authentication protocol

As a reference point, we have taken Andrew Secure RPC protocol like base of our proposal. The RPC protocol aims to distribute a symmetric session key and to authenticate the participants so that later the participants will have a secure connection using the session key. The protocol that we present in Alice-Bob notation is a modified version of the Lowe version, [18]. We have already introduced this protocol in a previous work and here we provide a full specification, [1]. The protocol is as follows:

$C : K_{CS}$
S : Key , $K_{CS} \in$ Key

1. $C \rightarrow S : C; N_c$
2. $S \rightarrow C : \{\!|N_c; K'_{CS}; S; N_s|\!\}_{K_{CS}}$
3. $C \rightarrow S : \{\!|N_c; N_s|\!\}_{K'_{CS}}$

Generally speaking, in step 1, C begins a session with S by sending him a plain-text containing its identification, so-called agent name[6] *C*, and a nonce[7] N_c. Upon message reception *S* sends back a ciphered message, which only C could have decrypted, containing the received nonce *Nc* (as a challenge-response), a new session key K'_{CS}, his agent name *S* (as a proof of his identity), and a new challenge-response used in the following step. Having interpreted the message of step 2, C sends back (in step 3) a message ciphered under the new session key K'_{CS}; this message must be interpreted by *S* like a proof of authentication. Now, *C* and *S* are now ready to further communication using the distributed session key.

We have verified this new protocol using The AVISPA Tool Web Interface.[8] See figures 1, 2, 3, and 4 for the definitions and roles in HLPSL specification. We have proved that the protocol provides agreement, the fourth level of authentication according to the hierarchy of Lowe, [19]; in our proof agent *C* authenticates *S* under nonce N_c , and S authenticates agent C under session key K'_{CS}, as shown in figure 4.

[4] Clark-Jacob library is available in http://www.lsv.ens-cachan.fr/spore/index.html

[5] AVISPA library is available in http://www.avispa-project.org/

[6] Agent names are used to refer to the identity of a participant.

[7] A nonce is a random, unguessable number that has not been used before. This newness is usually referred to as freshness in the literature.

[8] AVISPA web tool is available via http://www.avispa-project.org/

```
role alice(C,S:agent,
           Kcs: symmetric_key,
           Snd_CS, Rcv_CS: channel (dy))
played_by C
def=

  local State: nat,
        Nc, Ns: text,
        Kpcs: symmetric_key

  const kpcs, alice_bob_nc, bob_alice_kpcs: protocol_id

  init  State := 0

  transition

%% Start of the protocol
  step1.        State = 0  /\ Rcv_CS(start)  =|>
%1 C --> S : C, Nc
                State':= 1 /\ Nc':= new()
                           /\ Snd_CS(C.Nc')

  %% Receving the first response of S
%2 S --> C : {Nc, Kpcs, S, Ns}_{Kcs}
  step2.        State = 1  /\ Rcv_CS({Nc.Kpcs'.S.Ns'}_Kcs) =|>
%3 C --> S : {Nc, Ns}_{Kpcs}
                State':= 2 /\ Snd_CS({Nc.Ns}_Kpcs')
                           /\ witness(C,S,bob_alice_kpcs,Kpcs')
                           /\ request(C,S,alice_bob_nc,Nc)


end role
```

Figure 1. Role definition of the client, in this case with name alice.

```
role   bob(C,S:agent,
           Kcs: symmetric_key,
           Snd_SC, Rcv_SC: channel (dy))
played_by S
def=

  local State: nat,
        Nc, Ns: text,
        Kpcs: symmetric_key

  const kpcs, alice_bob_nc, bob_alice_kpcs: protocol_id

  init  State := 0

  transition

%1 C --> S : C, Nc
  step1.        State = 0  /\ Rcv_SC(C.Nc') =|>
%2 S --> C : {Nc, Kpcs, S}_{Kcs}
                State':= 1 /\ Kpcs':= new() /\ Ns':= new()
                           /\ Snd_SC({Nc'.Kpcs'.S.Ns'}_Kcs)
                           /\ witness(S,C,alice_bob_nc,Nc')
                           /\ secret(Kpcs',kpcs,{S,C})


%3 C --> S : {Nc, Ns}_{Kpcs}
  step3.        State = 1  /\ Rcv_SC({Nc.Ns}_Kpcs) =|>
                State'= 2
                           /\ request(S,C,bob_alice_kpcs,Kpcs)

end role
```

Figure 1. Role definition of the server with name bob.

```
role  asrpoJC(C,S: agent,
              Kcs: symmetric_key)

def=

  local Se_CS, Re_CS, Se_SC, Re_SC: channel (dy)

  composition
          alice (C,S,Kcs,Se_CS,Re_CS)
       /\ bob   (C,S,Kcs,Se_SC,Re_SC)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% The main role
role environment()
def=

  const c, s : agent,
        kcs  : symmetric_key

  intruder_knowledge = {c, s, i}

  composition
          asrpoJC(c,s,kcs)
       /\ asrpoJC(c,s,kcs)
       /\ asrpoJC(s,c,kcs)
       /\ asrpoJC(s,c,kcs)

end role
```

Figure 3. Role definition of the participants and the environment.

```
%% Properties to verify
goal

        secrecy_of kpcs
        authentication_on alice_bob_nc
        authentication_on bob_alice_kpcs
%%      authentication_on bob_alice_kcs
%%      weak_authentication_on bob_alice_ns


end goal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Call of the main role
environment()



%% AVISPA Tool Summary

%% OFMC    : SAFE
%% CL-AtSe : SAFE
%% SATMC   : SAFE
%% TA4SP   : SAFE

%% Refer to individual tools output for details
```

Figure 4. Properties verified and the result of the protocol verification.

6 GENERAL PROTOCOL FOR SIGNING AND AUTHENTICATING E-DOCUMENTS AND IMPLEMENTATIONS.

This section explain two scenaries for signing and authenticating e-documents: the first scenary consists in the security protocol, its scopes and limitations; the second one is a protocol without security and its implications.

6.1 Signing and authenticating e-documents with security protocol

Figure 5 illustrates the complete protocol, which consists of fourth stages:

Initial Knowledge; Authentication; Signing and Secure Consulting process.

Stage	Process
I	Initial Knowledge $C : K_{CS}, un$ S : Key , $K_{CS} \in$ Key , AE
II	Authentication 1.-C $\rightarrow$ S : $C; N_c$ 2.-S $\rightarrow$ C : $\{\!\|N_c; K'_{CS}; S; N_s\|\!\}_{K_{CS}}$ 3.-C $\rightarrow$ S : $\{\!\|N_c; N_s\|\!\}_{K'_{CS}}$
III	Signing 4.-C $\rightarrow$ S : $\{\!\|F; \{dp\}\|\!\}_{K'_{CS}}$ $F' = signing(F, \{dp\}, al, K_{CS})$ 5.-S $\rightarrow$ C : $\{\!\|F'; \text{Hash } (F)\|\!\}_{K'_{CS}}$
IV	Secure Consulting process 4.-C $\rightarrow$ S : $\{\!\|F'; un\|\!\}_{K'_{CS}}$ $\{dp\} = revealing(F', al', K_{CS})$ 5.-S $\rightarrow$ C : $\{\!\|\{dp\}; \text{Hash } (F')\|\!\}_{K'_{CS}}$

Figure 5. Symmetric Cryptography Protocol for Signing and Authenticating E-Documents.

The initial knowledge represents the knowledge that each agent owns at the beginning of the protocol. In this stage client must be previously registered with the server; it means that only authorized users will be able to sign and consult signatures. In the registration process each client must specify its steganography algorithm. Since, the Server knows all shared keys and all steganography algorithms.
All symmetric keys known by the server are being stored in its database. We assume that each client knows its own key.

Authentication describes a run of the protocol where client, *C*, begins a session with server, *S*. The purpose of the protocol is to obtain mutual authentication between participants and to obtain a session key, K'_{CS}, which will be used to encode and decode later communications (signing and consulting process). The implementation of this stage has been a little bit more complicated, we have implemented some actions in the client side using javascript and java applets and we have decided to use a default encryption algorithm AES, which will be coded in jsp (for the server side); however, this is an ongoing work.

Signing begins once the participants have been authenticated. In this stage, client uses session key K'_{CS} to encrypt the submitted e-document, *F*, and personal data $\{dp\}$. Server signs the e-document using formula

$$signing(F; fdpg; al; K_{CS}),$$

which it also uses its initial knowledge and makes a relation with the username, the steganography algorithm pre-selected by the client and its secret shared key.
Here, we are working in jsp for the server side. We have implemented some classes in java and we can sign and obtain signatures in formats like BMP, and PNG. Right now, we are working with an applet implementation for the client side in order to encrypt all messages interchanged between the client and the server with the same key and the same algorithm, since AES.

Secure Consulting process begins after the stage II or III has been completed. An authenticated client can consult the signature of any previously signed e-document. The server maps the username *un* with the corresponding algorithm *al* and shared key K_{CS} to use the function $revealing(F'; al'; K_{CS})$ in order to reveal the personal data $\{dp\}$ that was included in the submitted e-document F'. The server sends back $\{dp\}$ and the hash message of F' like a proof of that the data was extracted from