



Privacy and forensics investigation process: The ERPINA protocol

Giannakis Antoniou^{a,*}, Leon Sterling^a, Stefanos Gritzalis^b, Parampalli Udaya^a

^a Department of Computer Science and Software Engineering, The University of Melbourne, 111 Barry Street, Carlton, Victoria 3053, Australia

^b Department of Information and Communication Systems Engineering, University of the Aegean, Samos, 83200 Greece

Abstract

The rights of an Internet user acting anonymously conflicts with the rights of a Server victim identifying the malicious user. The ERPINA protocol, introduced in this paper, allows an honest user communicating anonymously with a Server through a PET, while the identity of a dishonest user is revealed. Prior research failed to distinguish objectively between an honest user and an attacker; and a reliable and objective distinguishing technique is lacking. The ERPINA protocol addresses the reliability issue efficiently by defining from the beginning of the communication what is considered as malicious and what is not.

© 2007 Elsevier B.V. All rights reserved.

Keywords: RPINA; Network Forensics; Accountability; Privacy; PET

1. Introduction

An honest Internet user desires to communicate anonymously with a Server while the communication offers integrity and confidentiality. Internet users can readily perform malicious actions through the Internet because the current technologies can help hide their identities [4,1]. Currently, digital forensics investigations procedures are time-intensive and costly, and the reliability and the collection method of evidence is a concern [9]. Furthermore, a victim may well avoid performing digital forensics investigation because the victim's devices, such as a web server, may need to stop functioning for a period of time. An appropriate automatic mechanism could reduce the duration and cost and increase the reliability of an investigation, encouraging victims to prosecute attackers.

A Privacy Enhancing Technology (PET) offers privacy [8] (including communication anonymity) to Internet users. Network Forensics tries to reveal the privacy of Internet users, in

order to discover the identity of attackers. However, a PET may hide the identity of attackers [7] while Network Forensics may violate the privacy of honest users. These opposite-goal technologies can be used in an appropriate way in order to build an Internet society, which discourages attackers participating and respects the privacy of honest users. To the best of our knowledge, currently the only known protocol which offers privacy to honest users and accountability to attackers is the Respect Private Information Not Abuser (RPINA) protocol [2]. However, the reliability of the RPINA protocol is currently not guaranteed. The main goal of this paper is to increase dramatically the level of the protocol's reliability. The goal is achieved with the proposed protocol, which is called Enhanced-RPINA (ERPINA).

The Fig. 1 illustrates the participated entities of the ERPINA protocol. The policy definition entity (PDE) is responsible to publish the desire policy of each entity. The directory service (DS) and the forensic investigation entity (FIE) have a very strong relation and they can be considered as one entity. The DS/FIE issues tickets to the anonymous user (AU) and performs forensic investigation. Before the AU contacts with the Server, the AU needs to contact with the PDE in order to publish his/her acceptable policy and the DS/FIE in order to get a ticket. The AU hides his/her communication identity from the Server by using a PET. In case the Server detects an attack from the AU,

* Corresponding author.

E-mail addresses: gant@csse.unimelb.edu.au (G. Antoniou), leon@csse.unimelb.edu.au (L. Sterling), sgritz@aegean.gr (S. Gritzalis), udaya@csse.unimelb.edu.au (P. Udaya).

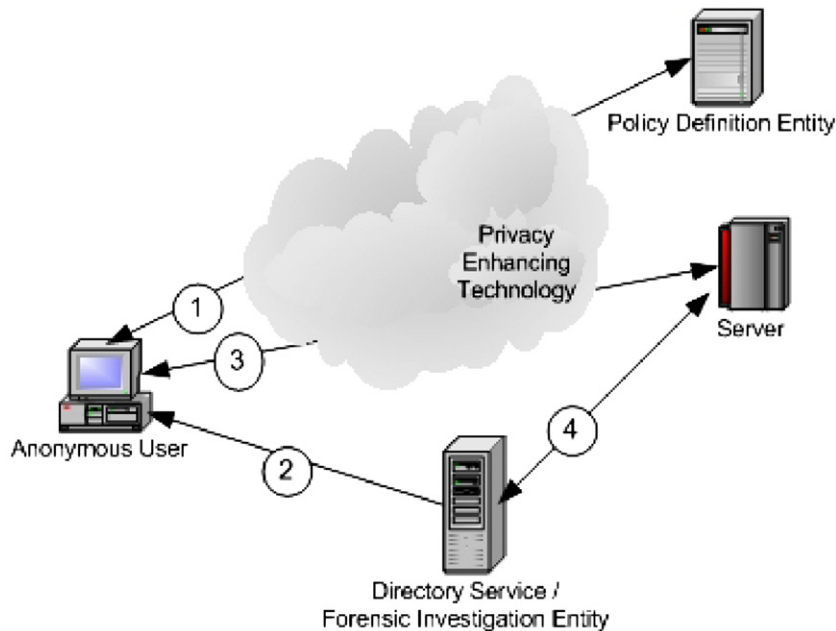


Fig. 1. The architecture of the ERPINA protocol.

the Server can provide to the FIE the received messages sent by the AU and the FIE reveals the identity of the AU to the Server. Further detail analysis can be found in the rest of the paper.

The paper is organized as follows: Section 2 describes related work; Section 3 introduces the ERPINA protocol, overcoming the reliability issue of previous related protocols; Section 4 analyses the characteristics of the ERPINA and illustrates an example where the reliability of the ERPINA protocol is presented; Section 5 concludes the paper and gives directions for further research in relation to the ERPINA protocol.

2. Related work

In [5] a solution is described for revocable anonymous access, where the identity of a user who is hiding his/her identity behind a PET (offering communication anonymity) is revealed by a third entity called Management Entity. However, the solution lets the Management Entity know with whom the AU is communicating. The solution does not describe under what conditions the revocation takes place. There is not even a mechanism describing how the Management Entity is assured about the “guilt” of the AU. Further, with the Management Entity knowing the AUs and the corresponding Servers makes the whole infrastructure vulnerable with a single-point of failure.

The Protect Private Information Not Abuser (PPINA) protocol [3] offers live proactive forensic investigation functionality. It operates over any PET (with anonymous communication capability) protocol, offering to users an end-to-end message integrity and confidentiality. It also offers accountability to users who perform malicious actions against the Servers, while honest users enjoy the privacy offered by the PET. The PPINA protocol lacks strong message integrity while a Server replies to the user. It has also an unnecessary layer of message confidentiality and protection against replay attack. Moreover, revealing an attacker’s

identity is not guaranteed because no specific criteria have been set about what is considered malicious and what is not.

The RPINA protocol, which is an upgraded version of PPINA protocol, enforces message integrity during the communication between an AU and a Server and removes an unnecessary layer of message confidentiality and protection against replay attack. However, the RPINA protocol does not address one major problem; the lack of an entire mechanism to help the Server and AU agree to a common set of criteria, resulting in the RPINA protocol sometimes failing to reveal the identity of an attacker.

3. ERPINA: the conceptual framework

The ERPINA protocol (Fig. 2) is an upgraded version of the RPINA protocol which increases the level of offered reliability.

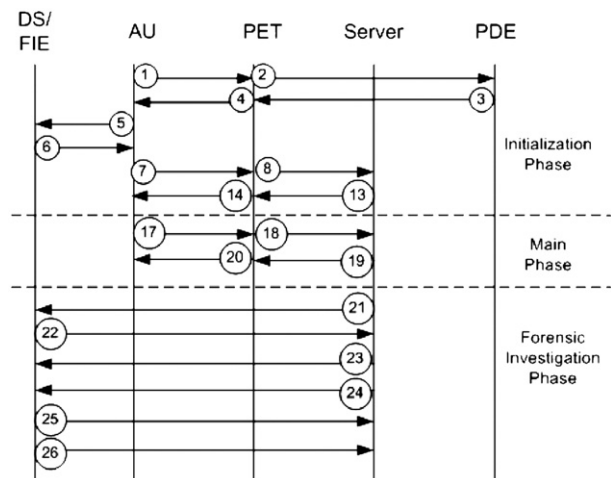


Fig. 2. It illustrates the steps of the Initialization, Main Phase and Forensic Investigation Phase (Steps 9 through 12 and Steps 15 and 16 are not shown).

The participating entities are an AU, a DS, a PET, a Server, a FIE and the PDE.

The AU signs a Policy Definition Document (PDD) and then requests a ticket from the DS. The DS issues the ticket, embedding the PDD, to the AU and the AU sends the ticket to the Server through the PET, which offers communication anonymity to the AU. Once the Server verifies the validity of the ticket and agrees to the PDD, the AU can communicate with a Server through a PET. If the Server detects an attack from that AU based on the PDD, the Server contacts the DS/FIE which has issued the ticket. The Server then sends all the received messages sent by the user to the FIE. The FIE is responsible to review the messages and decide whether the messages violate the agreed PDD. The DS/FIE is obligated to function properly because the DS has signed the ticket. If the messages are malicious (based on the agreed PDD), the FIE will reveal the identity of the user. Otherwise, the FIE will refuse to reveal the identity of the user.

3.1. Function descriptions

The protocol assumes that there is an established public key infrastructure (PKI) where all the participated entities have published their public key and are accessible to the rest of the entities. An additional assumption is that a message which has been signed by using a private key can be linked to the owner of the public key which verifies the related signature.

Here are the functions used in the next sections:

Sencrypt(*plainText*, *secret key*)

The *plainText* is encrypted (symmetrically) by using the *secret key*.

Sdecrypt(*cipherText*, *secret key*)

The *cipherText* is decrypted (symmetrically) by using the *secret key*.

Aencrypt(*plainText*, *public key*)

The *plainText* is encrypted (asymmetrically) by using the *public key*.

Adecrypt(*cipherText*, *private key*)

The *cipherText* is decrypted (asymmetrically) by using the *private key*.

sign(*message*, *private key*)

The *message* is signed using the *private key*. The function returns the value *signedMessage* (where *signedMessage*=signature, *message*).

IsVerified(*signedMessage*, *public key*)

The function returns true if *signedMessage* is equal to *sign*(*message*, *private key*), where the *private key* is related with the *public key*. Otherwise, it returns false.

HashFunction(*message*)

This function returns the hash value of the *message*.

GenerateKey()

The function returns a random number appropriate to be used as a secret key.

GeneratePairOfKeys()

The function returns a random pair of keys— public and private keys.

3.2. The three phases

The ERPINA protocol is divided into three phases, the Initialization, Main and Forensic Investigation Phases. The goals of the Initialization phase are:

- 1) to establish a relation between: a) DS and AU b) Ticket and AU c) DS and Ticket,
- 2) to develop an agreement between the Server and the AU on a common security policy, and
- 3) to establish the responsible entity corresponding to the needs of the Server to reveal the identity of a potential attacker.

The Main Phase serves the need of the AU to send and receive messages with the Server. During this phase, the actual communication is taking place. The goal of the Forensic Investigation Phase is to collect evidence and reveal the identity of the attacker to the Server, in case the user is indeed an attacker.

3.2.1. Initialization phase

The Server and the AU should agree to a common security policy during the Initialization Phase. The security policy defines what the Server considers as malicious and what is not. Instead of the AU and the Server exchanging the whole document, they exchange only the message digest of the document.

From the very beginning of the Initialization Phase, the AU sends a PDD to the PDE through the PET (Steps 1 and 2). The PDE generates the hash value of the PDD ($HashPDD = HashFunction(PDD)$), signs the hash value ($SignedHashPDD, HashPDD = sign(HashPDD, PDEprivate-key)$), and returns back the signed hash value to the AU (Steps 3 and 4).

AU→PET: Aencrypt((PDD, key), PDEpublic-key) (Step 1)

PET→PDE: Aencrypt((PDD, key), PDEpublic-key) (Step 2)

PDE→PET: Sencrypt((signedHashPDD), key) (Step 3)

PET→AU: Sencrypt((signedHashPDD), key) (Step 4)

Before the AU requests a ticket from the DS, the AU needs to generate the request (Step 5).

The ticket is the necessary element which links the future actions of the AU with the AU. For this reason a Server does not accept communicating with any AU without a valid ticket. The AU generates a pair of keys, called the session pair of keys because this pair of keys will be used only for the purpose of one session. At that moment, both keys (session public and private keys) are kept secret.

The AU calculates the hash value of the public key, known as Token, $Token = HashFunction(session\ public\ key)$, and then calculates the hash value of the Token, known as Token2.

$Token2 = HashFunction(Token)$

Token2 and the signedHashPDD are signed by the session private key and encrypted by a random secret key, known as aKey, generated by the AU. The secret key (aKey) and the Token are encrypted by the public key of the DS. The AU signs the whole message by using the private key and sends the message to the DS (Step 5). The below algorithm describes the calculations of the AU in order to make the request for a ticket.

3.2.1.1. Algorithm— RequestForTicket().

```
(AUsession-public-key, AUsession-private-key)
    = GeneratePairOfKeys()
Token    = HashFunction (AUsession-public-key)
Token2   = HashFunction (Token)
X1       = sign((Token2, signedHashPDD), AUsession-private-key)
aKey     = GenerateKey( )
X2       = Sencrypt(X1, aKey)
X3       = Aencrypt((aKey, Token), DSpublic-key)
RequestForTicket = sign((X3, X2), AUprivate-key)
```

The DS verifies the validity of the message's signature (RequestForTicket) and decrypts the aKey and the Token. By using the aKey, the DS decrypts the signed Token2 and verifies whether the Token2 is the hash value of the Token. After the validation, the DS stores the message of the Step 5 and issues the ticket. The DS cannot verify the signature of X1. The Server will be the only entity who can verify it. The DS signs the signed Token2 and the signedHashPDD found in the related ticket's request, and encrypts the whole message by using the public key of the AU.

```
ticket=sign(sign((Token2, signedHashPDD), AUsession-private-key), DSprivate-key)
```

After that, the DS sends the ticket encrypted to the AU (Step 6). The AU can use this ticket for one communication session at any time in the future to contact with any Server. The AU sends the ticket including the session public key, through the PET (Step 7), to the desirable Server (Step 8). The Server can verify whether or not the sender is the owner of the ticket by verifying the signed Token2 with the session public key, as well as whether Token2 is the hash value of the hash value (Token) of that session public key. Otherwise, the Server rejects the ticket and denies continuing the communication.

```
AU→DS: sign((Aencrypt((aKey,Token), DSpublic-key), Sencrypt(sign((Token2, signedHashPDD), AUsession-private-key), aKey)), AUprivate-key) (Step 5)
DS→AU: Aencrypt((ticket), AUpublic-key) (Step 6)
AU→PET: ticket, Aencrypt(sign(AUsession-public-key, AUsession-private-key), SERVERpublic-key) (Step 7)
PET→Server: ticket, Aencrypt(sign(AUsession-public-key, AUsession-private-key), SERVERpublic-key) (Step 8)
```

If the Server does not know the content of the actual PDD, the Server will request from the PDE the actual PDD, based on the signed hash value of the PDD found in the ticket (Steps 9 and 10). The PDE signs and returns the PDD to the Server (Steps 11 and 12).

```
Server→PET: Aencrypt((signedHashPDD, key), PDEpublic-key) (Step 9)
PET→PDE: Aencrypt((signedHashPDD, key), PDEpublic-key) (Step 10)
PDE→PET: Sencrypt(sign(PDD, PDEprivate-key), key) (Step 11)
PET→Server: Sencrypt(sign(PDD, PDEprivate-key), key) (Step 12)
```

If the Server agrees with the related PDD, the Server replies to the AU positively (Steps 13–14). The key (AUsession-public-key) which the AU uses decrypting the message should also verify the signed message.

```
Server→PET: Sencrypt(sign((Token2, signedHashPDD), AUsession-private-key), AUsession-public-key) (Step 13)
PET→AU: Sencrypt(sign((Token2, signedHashPDD), AUsession-private-key), AUsession-public-key) (Step 14)
```

If the Server does not agree with the related PDD found in the ticket or the ticket does not contain any PDD, the Server rejects the ticket and informs the AU about the accepted PDD (Steps 15 and 16). In such a case, the AU needs to request a new ticket from the DS.

```
Server→PET: Sencrypt((sign((Token2, signedHashPDD), AUsession-private-key), signedHashPDD), AUsession-public-key) (Step 15)
PET→AU: Sencrypt((sign((Token2, signedHashPDD), AUsession-private-key), signedHashPDD), AUsession-public-key) (Step 16)
```

3.2.2. Main Phase

The session public key will be used by the AU and the Server as a secret key during their communication offering message confidentiality. The AU also signs the messages before sending them to the Server by using the session private key.

```
AUtoServerData=sign(Sencrypt((Data,nonce), AUsession-public-key), AUsession-private-key)
```

The Server is able to verify the signature of the messages by using the session private key. Once the AU and the Server agree on the PDD, the Main Phase begins (Steps 17–20)

```
AU→PET: AUtoServerData (Step 17)
PET→Server: AUtoServerData (Step 18)
Server→PET: Sencrypt(sign(Data, SERVERprivate-key), AUsession-public-key) (Step 19)
PET→AU: Sencrypt(sign(Data, SERVERprivate-key), AUsession-public-key) (Step 20)
```

The Server can verify the validity of the received message (Step 18) as follows:

```
3.2.2.1. Algorithm— message validity. If IsVerified(sign(Sencrypt((Data, nonce), AUsession-public-key), AUsession-private-key), AUsession-public-key)=true then
    (Data, nonce)=Sdecrypt(Sencrypt((Data, nonce), AUsession-public-key), AUsession-public-key)
    if nonce already received during this session then
        Invalid message — Replay attack
    else
        Valid message
    end if
else
    Invalid message
end if
```

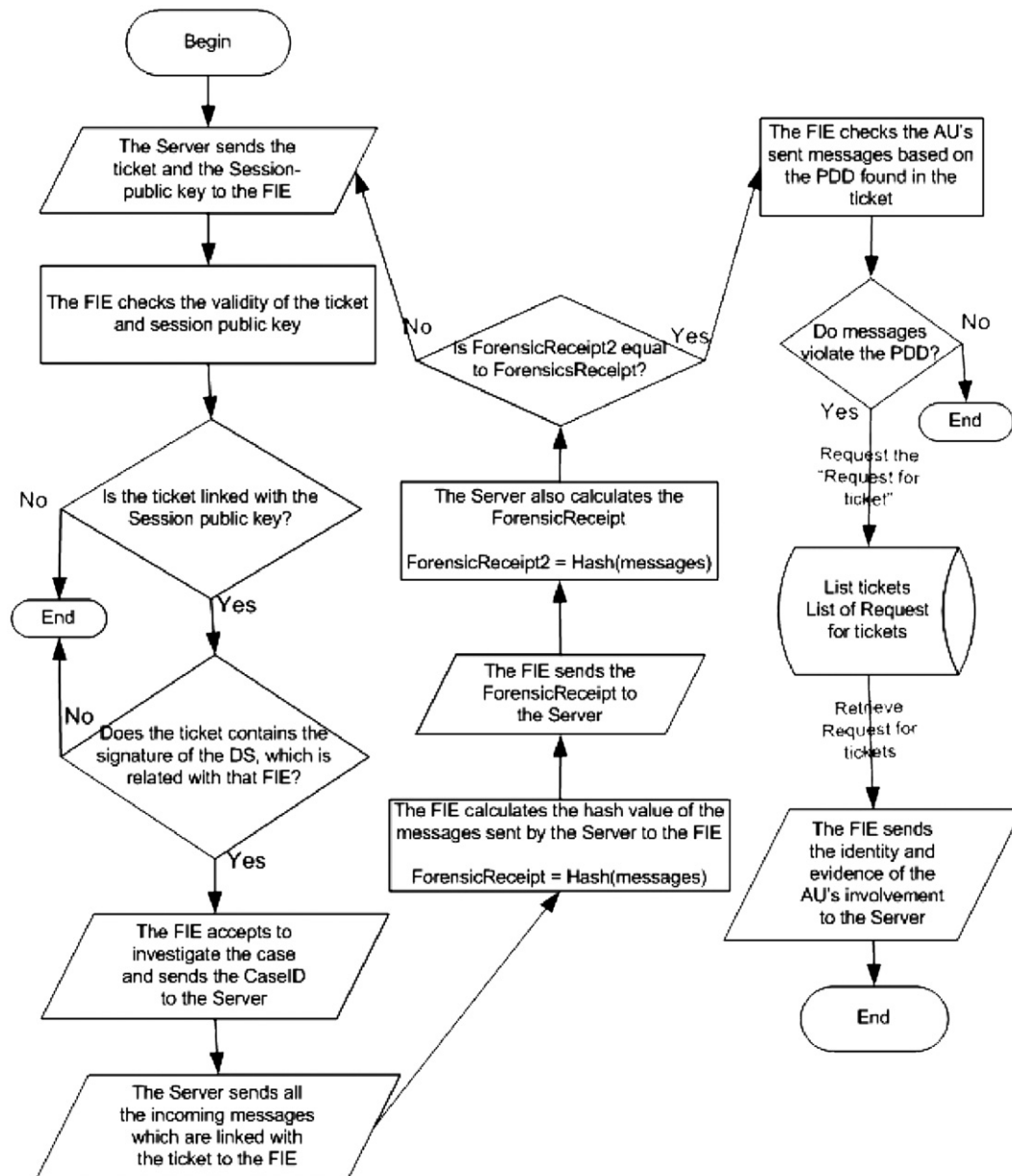


Fig. 3. The flowchart of the Forensic Investigation Phase.

3.2.3. Forensic Investigation Phase

The Forensic Investigation Phase (Fig. 3) begins when the Server detects malicious messages sent by an AU. The transition from the Main Phase into the Forensic Investigation Phase is built into the design of the protocol. It is up to the Server to exit from the Main Phase and enter into the Forensic Investigation Phase (Fig. 4). The Server contacts the FIE and sends all the relevant information, captured during the Initialization and Main Phases, in order to reveal the identity of the AU (Steps 21–26). Once the Server sends all the related messages to the FIE, the FIE issues a ForensicReceipt. The ForensicReceipt gives the opportunity to the Server to check whether or not the communication between the FIE and the Server was offering message integrity. Moreover, the FIE cannot argue that it hasn't received all the messages submitted by the Server (during the Forensic Investigation Phase), in case that the FIE wants to protect the identity of a malicious AU.

ForensicReceipt = the hash value of the data of Steps 21–23
evidence = $\text{sign}(\text{signedPDD}, \text{AUsession-public-key}, \text{ForensicReceipt}, \text{aKey}, \text{IP Address}, \text{sign}(\text{Aencrypt}(\text{aKey}, \text{Token}), \text{DSpublic-key}), \text{Sencrypt}(\text{sign}(\text{Token2}, \text{signedHashPDD}), \text{AUsession-private-key}), \text{aKey}))$, *AUprivate-key*)

The Server can be sure that the FIE will reveal the identity of the AU. Otherwise, the DS/FIE will be accused because the DS issues the ticket (there is DS's signature on the ticket).

Server → FIE: $\text{sign}(\text{Aencrypt}(\text{AUsession-public-key}, \text{ticket}), \text{FIEpublic-key}, \text{SERVERprivate-key})$ (Step 21)

FIE → Server: $\text{sign}(\text{CaseID}, \text{FIEprivate-key})$ (Step 22)

Server → FIE: $\text{sign}(\text{Aencrypt}(\text{AUtoServerData}, \text{FIEpublic-key}), \text{SERVERprivate-key})$ (Step 23)

Server → FIE: $\text{sign}(\text{CaseID}, \text{SERVERprivate-key})$ (Step 24)

FIE → Server: $\text{sign}(\text{ForensicReceipt}, \text{CaseID}), \text{FIEprivate-key})$ (Step 25)

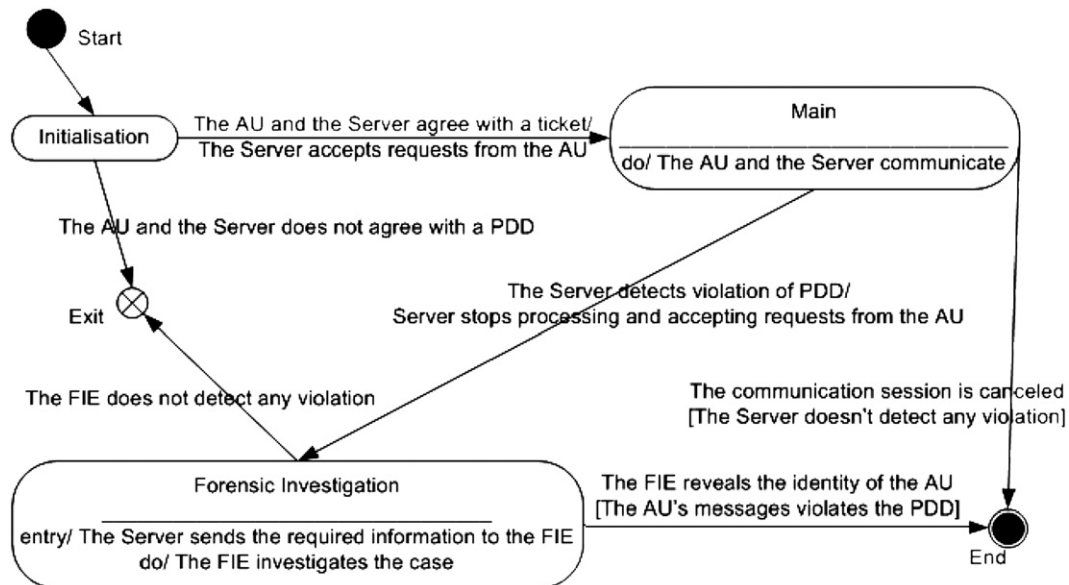


Fig. 4. The state diagram for the three phases of ERPINA protocol.

FIE→Server: $\text{sign}(\text{Aencrypt}(\text{evidence}, \text{SERVERpublic-key}), \text{FIEprivate-key})$ (Step 26)

X→PDE: $\text{Aencrypt}(\text{signedHashPDD}, \text{key}), \text{PDEpublic-key})$
PDE→X: $\text{Sencrypt}(\text{sign}(\text{PDD}, \text{PDEprivate-key}), \text{key})$.

A ticket is linked with the messages sent by a user. The Server can link the ticket with the received messages and with the DS who issues the ticket. The Server expects from the FIE a proof that an AU has requested the specific ticket from the DS. Therefore, DS/FIE is the only entity which can link a ticket with the owner of the ticket. For this reason the Server needs the help of the FIE. The FIE has the obligation to reveal the identity of the user only in case that the Server proves to the FIE that the owner of the ticket violates the related PDD, which is found in the ticket. The Server cannot only identify the attacker, but can also prove the attacker's malicious actions to any third entity, such as a judge in a court.

3.3. The functions of the policy definition entity

The PDE is responsible signing the PDD sent by other entities, like a DS, an AU and a Server, and making them available to the public. The PDE also generates and signs the message digest of a PDD. A PDD contains all the actions that are considered as malicious and the conditions when considered malicious.

For a given PDD, the PDE:

- signs the PDD (sPDD)
- generates the hash value of the signed PDD (HashPDD)
- signs the hash value of the signed PDD (signedHashPDD)
- returns the output to the requested entity

Let X be any entity other than PDE

X→PDE: $\text{Aencrypt}(\text{PDD}, \text{key}), \text{PDEpublic-key})$
PDE→X: $\text{Sencrypt}(\text{signedHashPDD}, \text{key})$

For a given signed hash value of a signed PDD, the PDE returns the related signed PDD

3.4. The Policy Definition Document

The PDD needs to have an appropriate structure in order to define precisely and accurately the preferred policy. The document is divided into two sections, the Security Policy (SP) – which is mostly focused upon in this paper – and the Privacy Policy (PP). The SP defines what is considered to be malicious for the Server and the PP defines the privacy policy of the Server. The SP must be respected by the AU whereas the PP must be respected by the Server.

The PP describes the privacy policy that the Server should follow regarding the private information that the AU gives intentionally to the Server. In this PP the P3P policy document [6] could be embedded.

It is vital that the SP is well-structured because it can help an entity, like a Server, to process and conclude as quickly as possible whether the SP fulfils the required security safeguards or not.

4. Analysis of ERPINA

During the Forensic Investigation Phase, neither the DS/FIE nor the Server needs to reveal their private key in order to help in the investigation. Additionally, the Server:

- Doesn't need to stop functioning until the investigators examine the Server's machines.
- Only needs to contact with the appropriate FIE and sends the related messages. The forensic investigation procedure is specific and straight forward.
- Doesn't need to spend money or time during the investigation.
- Doesn't need to have large storage devices to store everything for a long period of time.
- Needs to store the messages of the "live" sessions only.

- f) Doesn't need to bother about the integrity of the messages, because the messages are signed.
- g) Cannot ask from a FIE that an AU has violated the PDD without evidence about the actions of the AU.

In case the Server agreed with the AU for a PDD which does not make unacceptable a specific attack, then the AU has effectively been permitted to perform this attack and avoid revealing his/her identity. For this reason, the PDD should not only describe what actions are considered malicious/illegal, but also what are the only actions that are considered legal. Anything not explicitly legal is considered as illegal.

4.1. Analysis of the evidence

Although the FIE knows the IP Address of the AU, it is possible to know the wrong IP Address. The identity of the AU is based on the digital signature that it is found inside the evidence, and is not based on the IP Address.

The messages of Steps 8, 18 and 26 can prove the actions of the AU.

- a) We need to prove the relationship between those messages.

During Step 8, the Server receives the session public key (AUsession-public-key) which verifies the signature of the Step 18. Therefore, the messages of Steps 8 and 18 belong to the same communication session.

From Step 26, the session public key (AUsession-public-key) is the same as the session public key included from Step 8. Therefore, the messages of Steps 8 and 26 refer to the same communication session.

- b) We need to prove that the claims (Step 26) of the FIE are accurate.

The message in Step 26 contains the request for ticket (RFT) that the AU did. The RFT is signed by the AU. How can we relate the RFT with the messages of Steps 8 and 18?

The FIE reveals the secret key (aKey). By using the aKey, we get the $[sign((Token2, signedHashPDD), AUsession-private-key)]$, which can be verified by the session-public-key (AUsession-public-key). If the session public-key verifies the signed message, then the sender of the RFT is the same with the sender of the messages (Steps 8 and 18). Moreover, the Token2 (found in the RFT) must be the digest message of the digest message of the session public-key (AUsession-public-key)

- c) We need to prove that the AU has acted maliciously.

The ticket (found in Step 8) contains the signedHashPDD. Based on that, we request the full PDD from the PDE. We verify that the signedHashPDD (found in Step 8) is indeed the signed message digest of the given PDD. Based on that PDD, we can determine whether the messages on Step 18 violate the agreement between the AU and the Server.

4.2. The strength of the ticket

The DS issues the ticket without knowing which Server the AU is going to communicate with. Before issuing the ticket, the DS verifies the validity of the ticket's request (Step 5) as follows:

4.2.1. Algorithm— request for ticket's verification

```

If IsVerified(sign((Aencrypt((aKey,Token), DSpublic-key),
Sencrypt(sign((Token2, signedHashPDD), AUsession-private-
key), aKey)), AUprivate-key), AUpublic-key)=true then
  (aKey, Token)=Adecrypt(Aencrypt((aKey, Token), DSpublic-
key), DSprivate-key)
  sign((Token2, signedHashPDD), AUsession-private-key)=Sde-
crypt(Sencrypt(sign((Token2,
signedHashPDD), AUsession-private-key), aKey), aKey)
  if Token2=HashFunction(Token) then
    The ticket's request is Valid
  else
    The ticket's request is Invalid
  end if
else
  The ticket's request is Invalid
end if

```

If the ticket's request is valid and the DS issues the related ticket, the DS accepts that an encrypted message which has been encrypted by using a secret key where $[Token=HashFunction(secret\ key)]$ belongs to that AU. The message must also be signed and be able to be verified by that secret key (AUsession-public-key). The Server receives a message (Step 8) during the Initialization Phase. The Server verifies the validity of the message as follows.

4.2.2. Algorithm— initial verification

```

sign(AUsession-public-key, AUsession-private-key)=Ade-
crypt(Aencrypt(sign(AUsession-public-key, AUsession-pri-
vate-key), SERVERpublic-key), SERVERprivate-key)
if IsVerified(sign(AUsession-public-key, AUsession-private-
key), AUsession-public-key)=true and IsVerified(sign
((Token2, signedHashPDD), AUsession-private-key), AUses-
sion-public-key)=true and IsVerified(sign(sign((Token2, sign-
edHashPDD), AUsession-private-key), DSprivate-key),
DSpublic-key)=true and Token2=HashFunction(HashFunction
(AUsession-public-key)) then

```

```

  The ticket belongs to the entity who made the request to the
  Server
else
  The ticket may not belong to the entity which made the
  request to the Server.
end if

```

It is important to mention that the ticket can only be used by the entity which generates the request for a ticket, because the ticket has a mathematical connection with the session public and private keys. A malicious entity who wants to use the ticket of another, must calculate the public key based on its hash value and additional, he must find the private key based on the public key (mathematically infeasible).

4.3. Consideration and impact on existing RPINA

In the RPINA protocol the FIE and the Server could have different criteria and what may be considered as malicious for the Server may not be considered as malicious for the FIE. Additionally, the AU is able to know whether the message to be sent is considered as malicious (by the Server) or not, before even sending the message. However, this has not always positive results. An agreed PDD which does not include a specific type of attack can be used by a malicious AU for attacking. In such a case, the Server cannot prove to the FIE that the AU acts maliciously, because the PDD does not consider this action as malicious.

The RPINA protocol lacks an objective technique to determine whether a set of messages are malicious or not. Without such a technique, the reliability of the RPINA protocol to reveal the identity of an abuser is in doubt. The ERPINA protocol allows the AU and the Server to negotiate and agree in a common set of criteria.

5. Conclusions and future work

The ERPINA protocol fulfils the need of an honest user communicating anonymously through the Internet with a Server, and the need of a Server–victim identifying the attacker. In order to increase the reliability of the protocol, a mechanism has been proposed.

Future work is to design a mark-up language for describing Policy Definition Documents. We need, also, a technique preventing a Server collaborating maliciously with the FIE/DS and revealing the identity of honest users, who haven't violated the agreement (PDD). The use of a standard threshold schemes or a group signature could reduce the threat of a malicious DS acting maliciously. A step forward is to use a technique where an AU can prove to any third entity that the Server didn't respect the

agreement with the AU violating the privacy policy section of the agreed PDD.

References

- [1] G. Antoniou, S. Gritzalis, in: W. Hutchinson, C. Valli (Eds.), *Uncontrollable privacy—the right that every attacker desires*, proceedings of the 4th Australian Information Security Management Conference, Australian Computer Society, Mount Lawley, Australia, ISBN: 0-7298-0625-1, December 2006, pp. 23–31.
- [2] G. Antoniou, S. Gritzalis, in: A. Taguchi, et al., (Eds.), *RPINA: network forensics protocol embedding privacy enhancing technologies*, proceedings of the ISCIT 2006 International Symposium on Communications and Information Technologies, IEEE Press, Bangkok, Thailand, ISBN: 0-7803-9741-X, October 2006.
- [3] G. Antoniou, C. Wilson, D. Geneiatakis, in: H. Leitold, E. Markatos (Eds.), *PPINA— A Forensic Investigation Protocol for Privacy Enhancing Technologies*, Proceedings of the 10th IFIP CMS'06 Communications and Multimedia Security Conference, Iraklion, Greece, CMS, 2006, pp. 185–195, LNCS 4237.
- [4] H.L. Armstrong, P.J. Forde, *Internet anonymity practices in computer crime*, Emerald, *Information Management & Computer Security* Vol. 11 (5) (2003) 209–215.
- [5] J. Claessens, C. Diaz, C. Goemans, B. Preneel, J. Vandewalle, J. Dumortier, *Revocable anonymous access to the Internet?* Emerald, *Internet research: Electronic Networking Applications and Policy*, Vol. 13, Num. 4, 2003, pp. 242–258.
- [6] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, J. Reagle, *The platform for privacy preferences 1.0 (p3p1.0) specification*, <http://www.w3.org/TR/P3P/>, April 2002.
- [7] D. Forte, *Advances in Onion Routing: Description and backtracking/investigation problems*, Elsevier, *Digital Investigation*, Vol. 3, Issue 2, June 2006, pp. 85–88.
- [8] A. Pfitzmann, M. Hansen, *Anonymity, unlinkability, unobservability, pseudonymity and identity management— a consolidated proposal for terminology*, <http://dud.inf.tu-dresden.de/literatur/>, .
- [9] W. Shih-Jeng, *Measures of retaining digital evidence to prosecute computer-based cyber-crimes*, Elsevier, *Computer Standards & Interfaces* 29 (2) (February 2007) 216–223.