Formal Analysis of a Private Access Control Protocol to a Cloud Storage

Narjes Ben Rajeb1, Mouhebeddine Berrima2, Matthieu Giraud3, and Pascal Lafourcade3

1LIP2, Université Tunis El Manar, Tunis, Tunisia
2LIP2, Université de Monastir, Tunis, Tunisia
3LIMOS, Université Clermont Auvergne, Aubière, France

1narjes.benrajeb@gmail.com, 2berrima.mouheb@gmail.com, 3{first.last}@uca.fr

Keywords: Cloud storage, formal methods, attribute based signature, attribute based encryption, data and user privacy.

Abstract: Storing data in the Cloud makes challenging data’s security and users’ privacy. To address these problems cryptographic protocols are usually designed. Cryptographic primitives have to guarantee some security properties such that data and user privacy or authentication. Attribute-Based Signature (ABS) and Attribute-Based Encryption (ABE) are very suitable for storing data on an untrusted remote entity. In this work, we formally analyze the Ruj et al. protocol of cloud storage based on ABS and ABE schemes. We clarify several ambiguities in the design of this protocol and model the protocol and its security properties with ProVerif an automatic tool for the verification of cryptographic protocols. We discover an unknown attack against user privacy. We propose a correction, and automatically prove the security of the corrected protocol with ProVerif.

1 INTRODUCTION

Cloud storage refers data storage services hosted over the Internet. The cloud users store data online, so that they or any other authorized users can access them from any location via the Internet. However, the share of the sensitive data on a third party through a public network brings some security challenges. In particular, there are concerns with the privacy of users and data. Protecting privacy in cloud is more difficult than in traditional environments, because sensitive data may be disseminated and stored over many external location, managed by external service providers (Wang et al., 2010), and both cloud and user can be malicious (Mulazzani et al., 2011; Zhang et al., 2012). User privacy is required in many applications when users store sensitive information like financial or health data (Tang et al., 2012). There are two important privacy requirements when a user stores data on the cloud: anonymity and unlinkability. The ISO/IEC standard (governmental organisations, 2009) define anonymity as the property ensuring that a user may use a service or a resource without disclosing his (or her) identity. However, preserving the anonymity property may still release information about a user by allowing an adversary to track several uses of a resource by the same user. Such information might allow an adversary to deduce or at least restrict the possible identities of a user. Therefore, the unlinkability property is required, ensuring that the different uses of a service or a resource for the same user should not be linked by an adversary. On the other hand, the Cloud Service Provider (CSP) must authenticate the user to be sure that he has the right to store data on the cloud, moreover this authentication must be done without reveal any information about his identity. Attribute-Based Signature (ABS) is a cryptographic scheme privacy-preserving authentication. Indeed, in ABS the verifier of a signature can only check if the message is signed by authorized one without knowing any information about its identity.

Data Privacy has been also gained research interest because only authorized users have access to sensitive data on the cloud. Data must be protected when transmitted to CSP and during the storage. The protection is against the unauthorized users as well as the CSP since the cloud is often assumed to be honest but curious (Li et al., 2010; Yu et al., 2010). To ensure data privacy, several works propose the storage of data in encrypted form. Thus, if the storage is compromised, then the leaked information should be protected. Identity based cryptography is not feasible in this situation because the inability of users to share their encrypted data at a fine-grained level. Attribute-Based Encryption (ABE), introduced by Sahai and Waters (Sahai and Waters, 2005), solves the problem of fine grained access control. There are two complementary forms of ABE (Goyal et al.,
2006): Ciphertext-Policy Attribute-Based Encryption (CP-ABE) and Key-Policy Attribute-Based Encryption (KP-ABE). In CP-ABE, the users have given a set of attributes and the data are encrypted under an access policy described as Boolean formula. Only users having the attributes satisfying the access policy can decrypt the ciphertext. In KP-ABE, the situation is reversed: users are associated with access policies and ciphertexts are encrypted with sets of attributes.

Related Work: (Bertino et al., 2009; Angin et al., 2010; Chow et al., 2012; Ren et al., 2006) proposed approaches to deal with security and privacy. In (Bertino et al., 2009), the privacy of users is preserved with zero-knowledge proof protocols while it is based on anonymous identification in (Angin et al., 2010). Recently, taking advantages of ABS and ABE has emerged as a widely accepted approach by the cloud security community (Ruj et al., 2012; Dahshan and Elkassass, 2014; Wang et al., 2014; Li et al., 2010; Zhao et al., 2011; Ruj et al., 2011). The ABS is used to ensure the authentication while hiding anonymity, and the ABE allows a fine-grained access control to data. The cloud storage protocol proposed by (Ruj et al., 2012) is among the pioneering works to use ABS and ABE. The protocol uses the SSH protocol to secure all the communication between the users and the cloud, and supports reading and writing data stored in the cloud.

Contributions: We analyze the Ruj et al. protocol (Ruj et al., 2012) which we abbreviate RSN’12 protocol. We model it in the applied π-calculus (Abadi and Fournet, 2001). We use ProVerif tool (Blanchet et al., 2001) to analyze cryptographic protocols (Puys and Lafourcade, 2015; Cremers et al., 2009). For sake of simplicity, we consider one attribute in our modeling of the ABE and ABS schemes. We formalize and verify the fundamental security properties of the protocol. In writing mode, we verify the writer authentication and writer privacy which is expressed by the anonymity of writer’s identity and unlinkability, that is a user who stores data on the cloud. While in reading mode, we check the required property that is data privacy. We show that the unlinkability of a writer is not satisfied against an attack in which the adversary delays the messages of some writers. Then, we propose a fix, which prevents this attack. We also discuss some ambiguous aspects of RSN’12 protocol.

Outline: We give a description of SSH protocol, ABS and ABE schemes and RSN’12 protocol in Section 2. We model RSN’12 protocol in Section 3 and analyze the security properties in Section 4. Finally, we conclude in Section 5.

2 RSN’12 PROTOCOL DESCRIPTION

In (Ruj et al., 2012) the authors propose a protocol for reading and writing data stored in the cloud which is based on the decentralized approach of CP-ABE (Lewko and Waters, 2011) and ABS (Maji et al., 2008) where many authorities distribute secret keys associated to attribute. Using ABS the cloud verifies the authenticity of a user without knowing his identity before storing data. Using ABE only valid users are able to decrypt the stored data. The protocol makes the following assumptions:

1. The CSP is honest-but-curious, i.e. it tries to derive some information from the messages he learned during the execution of the protocol, but cannot modify the user’s content.
2. Users can have a read or/and write access to a file stored in the CSP.
3. All the communications between participants are secured by SSH (Secure Shell) protocol.

The RSN’12 protocol involves a user who may be a writer or/and a reader, a Trusty Authority (TA) registering users, one or more Key Distribution Center (KDC) issuing the secret keys associated to users’ attributes, and a CSP. TA and KDC are trusted entities while CSP is semi-trusted. Some users can be malicious and thus are considered as untrusted entities. The protocol is composed of three sub-protocols.

Registering and getting attribute secret keys. In a first phase, a user gets attribute secret keys from the KDCs by presenting his token obtained from TA:
- The user presents his identity to the TA, for instance a federal government (1) in Fig. 1.
- TA registers the user if he is eligible and gives him a token as described in ABS scheme (2) in Fig. 1. TA embeds a random value in the token which will be incorporate in the attribute secret keys for signing to prevents collusion of the users.
- The user on presenting the token to KDC receives secret attribute keys for signing and decryption (3) in Fig. 1. KDC checks the validity of the
token using TA’s public key, and sends the corresponding keys for signing and decryption (Δ in Fig. 1).

**Writing on the cloud.** To store a message \( MSG \) on the cloud, the user proceeds as follows:

- He creates an access policy \( X \) containing all required fields, and encrypts the message \( MSG \) under \( X \) as \( C = \text{Encrypt}(MSG, X) \).

- Then he calculates the message \( C_1 = H(C) \| \tau \) where \( H \) is a hash function, \( \tau \) is a timestamp and \( \| \) is the concatenation operation. The timestamp is used to prevent the user to use stale message back with a valid signature, when his attributes have been revoked. Next, he generates the signature \( \sigma \) of \( C_1 \) with a claim policy \( \gamma \).

- Finally, he sends \( e = (C, \tau, \sigma, \gamma') \) to the CSP. Then CSP verifies, using \text{Verify} \text{algorithm, if the message} \ H(C) \| \tau \ was signed by a user satisfying the claim policy \( \gamma' \).

**Reading from the cloud.** A user can access at any time to the data and requests a ciphertext, then the CSP sends the requested ciphertext using SSH. Note that, the authors do not propose any revocation model, but it is still possible to incorporate it. The protocol is clear but contains some ambiguities. We discuss these minor problems and explain how to fix them.

**Timestamper.** They are used to prevent the writing when the attributes and keys of a writer have been revoked, since a timestamp informs when the message was created. However a writer signs its message along with a timestamp generated by himself. Then a verifier cannot be really sure since the signer may include an arbitrary timestamp. In order to address this problem it is possible to use a trusted timestamping described in RFC 3161 (Adams et al., 2001). The signer sends the hash of its message to a trusted third party (TSA-Time-Stamping Authority) which concatenates a timestamp to the hash and calculates the hash of this concatenation. This hash is then signed with the private key of TSA. This role of TSA can be ensured by TA in the RSN’12 protocol.

**Writer and SSH connection to CSP.** In writing access, the protocol uses SSH connection between users and the CSP which is assumed to be semi-trusted. However, when establishing SSH connection the CSP knows the user’s identity following the execution of SSH authentication sub-protocol (Ylonen and Lonvick, 2006) which compromises the user privacy against CSP. This ambiguity can be addressed by configuring the SSH server of CSP to allow log in without any user authentication.

**Reader and SSH connection to CSP.** In reading access the SSH connection is useless because the messages are encrypted using ABE and only authorized users can decrypt them. Hence, we can drop the SSH connection between a reader and the CSP.

### 3 MODELING RSN’12 PROTOCOL

We describe the main process modeling the RSN’12 protocol. It is specified as the parallel composition of the processes modeling the roles of writers, readers, TA, KDC and CSP.

**The main process.** It is specified in Figure 2. First, the fresh secret keys \( skTA, skKDC \) and \( skCSP \) are re-generated and their corresponding public keys are published. The secret key of writer \( skW \) and reader \( skR \) are made under replication to model an infinite number of writers and readers. The processes writer and reader are under replication, because one user may establish many sessions with the CSP. In our modeling, we use public keys of asymmetric encryption as identities of participants.

![MainProcess](image-url)

**The writer process.** The writer process given in Figure 3 models the role of a writer. The secret keys \( KWTA, KWKDC \) and \( KWCL \) are the secret shared keys established by SSH transport protocol respectively with TA, KDC and CSP. After the receipt of a token form TA, the writer sends a request to KDC to get attribute secret key for signing, this request is encoded by the
Afterwards, the writer encrypts his message msg and signs it using the attribute secret key absSkA.

\[
\text{Writer}(skW, TPK, absAPK, abeAPK, pkTA, pkKDC, pkCSP) \triangleq
\text{Clt-SSH-Auth}(skW);
\text{in}(ch, encToken);
\text{let token = sdec(KWTA, encToken) in}
\text{Clt-SSH-Trans}(pkKDC); \text{Clt-SSH-Auth};
\text{out}(ch, senc(KWKDC, (token, write)));
\text{in}(ch, encAbsSkA);
\text{let absSkA = sdec(KWKDC, encAbsSkA) in}
\text{let abeEncMsg = abeEnc(abeAPK, AccessP, msg) in}
\text{Clt-SSH-Trans}(pkCSP);
\text{let sigMsg = absSign(TPK, absAPK, token, absSkA, abeEncMsg, ClaimP) in}
\text{out}(ch, senc(KWCSP, (abeEncMsg, sigMsg))).
\]

**Figure 3: Writer process.**

The reader process. The role of a reader is modeled by reader process given in Figure 4. At first, a reader behaves as a writer by requesting a token from the TA and an attribute secret key for decryption from the KDC. Next, it has access to the CSP, without secure communication, to read a message stored on the cloud. Finally, he decrypts the message read from the CSP using his attribute secret key abeSkA, and behaves as RestOfReader with the received message.

\[
\text{Reader}(skR, pkTA, pkKDC) \triangleq
\text{Clt-SSH-Trans}(pkTA); \text{Clt-SSH-Auth}(skR);
\text{in}(ch, encToken);
\text{let token = sdec(KWTA, encToken) in}
\text{Clt-SSH-Trans}(pkKDC); \text{Clt-SSH-Auth}(skR);
\text{out}(ch, senc(KWKDC, (token, read)));
\text{in}(ch, encAbeSkA);
\text{let abeSkA = sdec(KWKDC, encAbeSkA) in}
\text{in}(c, encMsg);
\text{let msg = abeDec(encMsg, abeSkA) in RestOfReader.}
\]

**Figure 4: Reader process.**

The TA process. Trustee authority process is given in Figure 5. After the establishment of the shared key for symmetric encryption KWTA by SSH transport protocol, and the authentication of the user by SSH authentication protocol, TA generates a token and sends it to the user encrypted with KWTA.

\[
\text{TA}(skTA, TSK, pkClt) \triangleq
\text{Ser-SSH-Trans}(skTA);
\text{Ser-SSH-Auth}(pkClt);
\text{new base};
\text{event DelivToken(pkClt)};
\text{out}(ch, senc(KWTA, absToken(TSK, base, pkClt))).
\]

**Figure 5: Trusted Authority process.**

The KDC process. The KDC process is given in Figure 6. When receiving a request from a user, KDC checks the correctness of the token using the public key of user pkC, used as its identity, which was authenticated during SSH authentication protocol. If the token is valid, it issues an attribute secret key for encryption or signing following the value of AccessMode, which has two possible values: “write” or “read”.

\[
\text{KDC}(skKDC, TPK, absASK, abeASK, pkClt) \triangleq
\text{Ser-SSH-Trans}(skKDC); \text{Ser-SSH-Auth}(pkClt);
\text{in}(ch, encToken);
\text{let (token, AccessMode) = sdec(KWKDC, encToken) in}
\text{if absTokenCheck(TPK, pkC, token) = true then}
\text{if AccessMode = write then}
\text{event DelivKeySign(pkC);}
\text{out}(ch, senc(KWKDC, absSkA(absASK, absGetBase(token), att)));
\text{else if AccessMode = read then}
\text{out}(ch, senc(KWKDC, abeSkA(absASK, pkC, att)))}
\text{else 0.}
\]

**Figure 6: Key Distribution Center process.**

The CSP process. The CSP which is responsible of the storage of user data, is modeled by the process in Figure 7. SSH connection without user authentication is established between writers. If the signature is valid with respect to the claim policy, the CSP stores the message that becomes immediately accessible by the readers. Since in reading mode, there is no secure communication between the reader and the CSP, in our modeling the CSP outputs the incoming messages from the writers on a public channel.

\[
\text{CSP}(skCSP, TPK, absAPK) \triangleq
\text{Ser-SSH-Trans}(skCSP); (+SSH with a writer*)
\text{in}(ch, encMsg);
\text{let (msg, sigMsg) = sdec(KWCSP, encMsg) in}
\text{if absSignCheck(TPK, absAPK, msg, ClaimP, sigMsg) = true then}
\text{event AcceptSign;}
\text{out}(ch, msg).
\]

**Figure 7: Cloud Server Provider process.**

4 SECURITY ANALYSIS

We analyse the security properties of the protocol, namely the authentication and privacy of a writer, and the confidentiality of the data. All proofs of our propositions are not presented because they are directly implied by our ProVerif codes*.

*Our ProVerif codes are available under request via the PC and will be publicly available if the paper is accepted.
4.1 Confidentiality

It means that a user without valid access policy cannot decrypt the data stored on the cloud. In applied \(\pi\)-calculus this property can be expressed as a secrecy property: it should be impossible for an adversary, interacting with the protocol and without valid attribute secret key, to learn a message which is encrypted and stored on the cloud.

**Definition 1.** Given an access policy \(AP\), a cloud storage protocol ensures confidentiality if a secret message stored on the cloud by an honest writer is not deducible by an attacker without attribute secret key satisfying \(AP\).

Proving secrecy property is expressed by the reachability notion. We request ProVerif to check that a private message, encrypted using a public access policy, cannot be deduced by the attacker. Proverif proves this property in less one minute.

**Proposition 1.** RSN’12 protocol ensures the confidentiality property.

This result confirms the fact that SSH communication between CSP and a reader is useless for confidentiality, since our modeling does not use it and ProVerif is able to prove the secrecy of the message.

4.2 Writer Authentication

A user can only write in the cloud if he has the attribute validating the claim policy. Moreover, an invalid user cannot receive attribute from a KDC, if does not have the token from TA. Authentication property can be captured as a correspondence assertion.

To define the authentication of a writer, we annotate the protocol by the following events:

- **AcceptSign:** This event is placed inside CSP’s process and emitted if the signature is valid, i.e. \(\text{absCheckSign} = \text{true}\).
- **DelivKeySign(IdUser):** This event is placed inside KDC’s process and emitted when the KDC issues an attribute secret key for signing to a user with identity \(IdUser\).
- **DelivToken(IdUser):** This event is placed inside TA’s process and emitted when TA delivers a token to a user with identity \(IdUser\).

**Definition 2.** A cloud storage protocol ensures the authentication of a writer with identity \(Id\) if for every execution trace of the protocol each occurrence of the event \(\text{AcceptSign}\) is preceded by an occurrence of \(\text{DelivKeySign(Id)}\) which is preceded by an occurrence of \(\text{DelivToken(Id)}\).

This property can be expressed in ProVerif in terms of nested correspondence (Blanchet, 2009) which allows us to order events. ProVerif can automatically prove the corresponding nested correspondence in less one second:

\[
\text{event(AcceptSign)} \Rightarrow (\text{event(DelivKeySign(pkwriter))} \\
\Rightarrow \text{event(DelivToken(pkwriter))})
\]

**Proposition 2.** RSN’12 protocol satisfies the authentication of a writer.

4.3 Writer Privacy

In the context of cloud storage, writer privacy is expressed by two properties; anonymity and unlinkability. Anonymity of a writer’s identity is ensured if it is not possible for anyone, even the CSP, to learn the writer’s identity of a stored message. Unlinkability means that no one can link the messages stored on the cloud, more precisely no one is able to decide if two messages were stored by the same writer, or not.

**Anonymity:** A cloud storage system ensures anonymity if it keeps the writer’s identity secret from everyone. Hence, anonymity can be formalized as a secrecy property: no one can deduce the identity of a writer who store a message on the cloud. Since the identities of the writers are known values, anonymity is captured by the concept of strong secrecy. Strong secrecy means that the adversary cannot distinguish two instances of the same protocol with two different values of the secret. For the precise definition, we refer the reader to (Blanchet, 2004). In ProVerif, strong secrecy is expressed by diff-equivalence defined between processes that share the same structure and differ only in the choice of terms representing the secret values (Blanchet et al., 2008).

**Definition 3.** A cloud storage protocol ensures anonymity of a writer’s identity if for any two writers with identities \(IdW_1, IdW_2\) and for any message \(msg\), an adversary cannot distinguish whether \(msg\) comes from \(IdW_1\) or \(IdW_2\).

We request ProVerif to check if

\[
C[Writer(IdW_1, msg)] \approx C[Writer(IdW_2, msg)].
\]

with \(C[\_\_]\) is an evaluation context modeling the whole cloud storage protocol as described in main process with a hole for a writer process, and the process Writer(IdW, msg) models a writer with identity IdW storing a message msg on the cloud. ProVerif succeeds to prove this request in 3 seconds.

**Proposition 3.** RSN’12 protocol preserves anonymity of writer’s identity.
Unlinkability: Informally, in cloud storage context, unlinkability holds when the different stored messages of the same writer cannot be linked by an attacker even a dishonest user (writer or reader). Thus, unlinkability can be viewed as the secrecy of link between writer and its messages stored on the cloud. The definition of unlinkability is similar to the definition of voter privacy in e-voting protocol (Kremer and Ryan, 2005) in the sense that we must consider at least two honest writers. To understand this assumption, consider the case where all the writers are dishonest except one, as the stored messages on the cloud are published by the CSP, the dishonest writers can collude and determine the message of the honest writer.

Definition 4. A cloud storage protocol ensures unlinkability if for any two writers with identities $Id_W^1$, $Id_W^2$ and for any two messages $msg^1$ and $msg^2$, an adversary cannot distinguish the situation in which $Id_W^1$ stores $msg^1$ and $Id_W^2$ stores $msg^2$ from the situation in which $Id_W^1$ stores $msg^2$ and $Id_W^2$ stores $msg^1$.

In applied $\pi$-calculus this definition can be formalized as the following equivalence:

$$C[Writer(Id_W^1, msg^1)] \approx C[Writer(Id_W^2, msg^2)]$$

where $C[.]$ is an evaluation context modeling the whole protocol with a hole for two writers. In ProVerif, the above pair of process can be expressed as single biprocess as follows:

$$C[Writer(Id_W^1, choice[msg^1, msg^2])] \parallel C[Writer(Id_W^2, choice[msg^2, msg^1])]$$

ProVerif finds an attack, in which a man-in-the-middle attacker selectively delays or delete some messages sent to the CSP by one writer until he can link a message to somebody.

Proposition 4. RSN’12 protocol does not ensure unlinkability property.

For this attack we consider an attacker who is a semi-honest reader with valid attribute secret keys, who wants link the messages to a writer. In a real cloud storage environment, to achieve the attack, an attacker performs the following steps:

- Access to CSP and memorize all the files stored in the cloud.
- Listen to the network and wait for a message send to the CSP.
- When a new message $MSG$ is sent, it identifies its sender $Id_W$ and blocks all the messages sent to CSP after the message $MSG$. He now has just to wait until $MSG$ becomes available on the CSP, i.e. CSP appends $MSG$ to the previous files.
- Then, he can access to the files and then learn $MSG$ by comparing the current contents of files with the previous contents. Thus, he concludes that $MSG$ was sent by a writer with identity $Id_W$ and can link a file to somebody.

Fixed protocol: The previously discovered attack against unlinkability is based on the fact that an attacker can instantaneously access to CSP to learn a message just after it was sent by a writer. To fix this problem, a solution is that CSP simultaneously publishes at least two incoming messages from different persons. However, the messages are accessible from a file, so if the messages are written on the file in a deterministic order, for instance following arriving time of the messages, the adversary can link a message with its writer by inspecting the order of the sent messages to the CSP on the network. Therefore, the CSP must write the incoming messages on the files in non-deterministic way. The new role of the CSP is given in the Figure 8.

\begin{verbatim}
FixedCSP\{ skCSP, TPK, absAPK\} ≜ Ser-SSH-Trans\{skCSP\};
\text{in}(ch,\text{enMsg1});
let (msg1,sigMsg1) = sdec(KWCSP,\text{encMsg1}) in
if absSignCheck(TPK,APK,msg1,ClaimP,sigMsg1)=true
then \text{in}(ch,\text{enMsg2});
let (msg2,sigMsg2) = sdec(KWCSP,\text{encMsg2}) in
if absSignCheck(TPK,APK,msg2,ClaimP,sigMsg2)=true
then
\{ sync 1; \text{out}(ch,\text{msg1}) | sync 1; \text{out}(ch,\text{msg2}) \}
\end{verbatim}

Figure 8: Fixed Cloud Server Provider process.

The synchronisation command $\text{sync 1}$ in the last line of the above CSP process is introduced to synchronize CSP process. This means that the CSP process waits until the two $\text{sync 1}$ are reached before publishing the received messages. Therefore, the outputs $\text{out}(ch,\text{msg1})$ and $\text{out}(ch,\text{msg2})$ of the two received messages are executed in parallel. This parallel execution captures the non-deterministic behaviour of the writing of the messages on the file, because the semantic of a parallel composition $P \parallel Q$ allows simultaneously and independently execution of $P$ and $Q$. Note that, in this case synchronisation helps to automatically prove diff-equivalence by ProVerif, and hence the observational equivalence of applied $\pi$-calculus, because it allows to swap data between processes at the synchronisation points. In fact, the diff-equivalence is stronger than observational equivalence. In particular, when proving equivalence between processes that contain several parallel components, e.g., $P \parallel Q$ and $P' \parallel Q'$, diff-equivalence requires that $P$ is equivalent to $P'$ and $Q$ is equivalent to $Q'$. This constraint can be relaxed by swapping data.
between parallel processes at synchronisation points. For more details, we refer the reader to (Blanchet and Smyth, 2016). Fortunately, ProVerif succeeds to prove observational equivalence with the new role of CSP in 28 seconds, and therefore we can conclude the security of the fixed protocol.

**Proposition 5.** The revisited RSN’12 protocol ensures unlikability property.

5 CONCLUSION

In this paper, we revisit the security of the protocol of (Ruj et al., 2012). We use ProVerif to prove automatically claimed security properties by the authors in the original paper. ProVerif helps us to discover a flaw in this protocol for the unlinkability property. We then give a correction and prove the security of the modified version with ProVerif. The next step is to use our framework to model and analyze more protocols using ABE and ABS in order to discover flaws or formally prove the security of these protocols.

REFERENCES


