Summary Report

Cryptography Protocols – ProVerif project

Name: Aditya Bhardwaj – **BCEW1E**

Topic: Security Verification of Key Exchange in Ciphertext-Policy Attribute Based Encryption

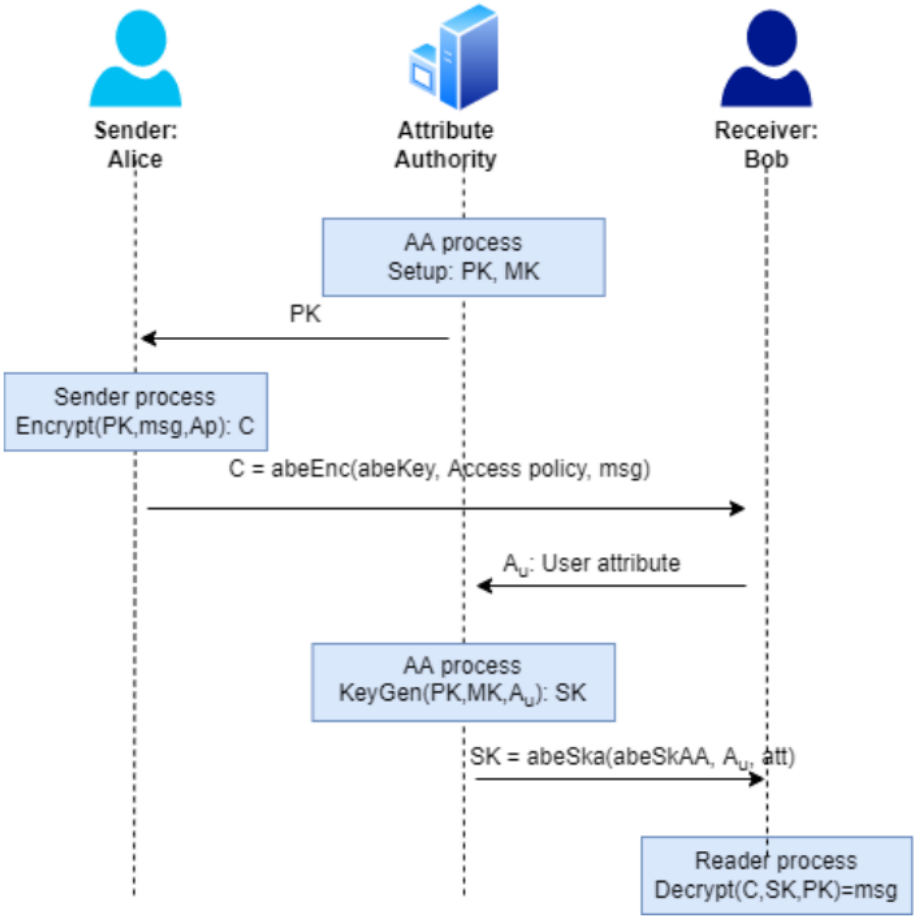
Abstract: The project required us to implement the CP-ABE scheme in ProVerif. In CP-ABE, a user's private key is associated with a set of attributes expressed as strings, and ciphertext is specified by an access policy over attributes. A user will be able to decrypt a ciphertext if that user's attributes satisfy the policy of the corresponding ciphertext. The generated user's private key has to be sent to the corresponding user securely. This is a challenge. We have to formally verify the security of key-exchange in the CP-ABE scheme.

Introduction

Attribute-based encryption (ABE) is an extension scheme of identity-based encryption and public-key encryption using authorisation policies. The secret key of a user and the ciphertext are dependent upon attributes (e.g. their email address, the country in which they live, or the kind of subscription they have). In such a system, the decryption of a ciphertext is possible only if the set of attributes of the user key matches the attributes of the ciphertext.

In CP-ABE, the secret keys are associated with a set of attributes, the ciphertext is associated with the access tree, and the sender can determine the access policy under which the data can be decrypted.

Algorithms in CP-ABE



CP-ABE scheme

CP-ABE scheme consists of four main algorithms: Setup, Encrypt, KeyGen, and Decrypt.

Setup. The setup algorithm takes as input the security parameter and produces the public parameters PK and a master key MK.

Encrypt. The encryption algorithm takes as input the public parameters PK, a message M, and an access structure A. The algorithm encrypts message M and produces a ciphertext CT as output. The ciphertext CT, which contains an access policy A, can be decrypted only if a user has attributes that match the access structure.

KeyGen. The key generation algorithm takes as input the master key MK and a set of attributes S which describe the key. It produces a private key SK as output, a private key for a set S of attributes.

Decrypt. The decryption algorithm takes as input the public parameters PK, a ciphertext CT, and a private key SK. If the set S of attributes satisfies the access structure A, the algorithm will decrypt the ciphertext and return a message M.

The Challenge

Every user has its attributes. They send their attributes to Authority, and then the key generation algorithm is worked in the Authority part to output the secret key.

After getting the private key, a receiver can decrypt the message if that user's attribute matches the access policy.

"But the keys still must be sent to the corresponding user."

Keeping the key-exchange secure

In the paper [1], authors have recommended the use of either KDC or PKI to send the generated keys.

KDC or a Key Distribution Centre, is the way to automatically distribute keys to support arbitrary connections between pairs of users. The users can be a computer, a process or applications. Each user shares a unique key with the KDC, known as the master key.

On the other hand, PKI or Public key infrastructure is an arrangement that binds public keys with respective identities of entities. The binding is established through a process of registration and issuance of certificates at and by a certificate authority (CA)

➔ **I have chosen to add the Needham-Schroeder Public Key protocol to the ProVerif implementation of CP-ABE scheme.**

Step 1: Running only the CP-ABE scheme

The code available in the paper was enough to understand the working of the scheme.

I added it all together and then to run it, I needed to add the main process in ProVerif code. It was missing some declarations, but important parts were present already.

Then I used the online ProVerif tool here:
http://proverif20.paris.inria.fr/index.php

Some things which I added:

1. `free c: channel.`
2. `adec()` #I added all the declaration from Needham-Schroeder PKI protocol
3. `fun userid(pkey):bitstring.`
4.

Also, there was a mistake in the code in the paper.

`in( c , abeSkAtt1:bitstring )` is written in the paper but later it is calculated like `let abeSkAtt1 = adec(encAbeSkAtt1, skReceiver)`

So, I removed `in( c , abeSkAtt1:bitstring)` as it is not sent on channel.

Also added secrecy queries.

```
(* Secrecy queries *)

free pMsg: bitstring [private].

noninterf pMsg.

query attacker(pMsg).
```

The one major modification I made was, I changed one equation to a rewrite rule in the form of `reduc`. The reason being it kept showing an error that a block of equation is not linear and could not be proved convergent. I had to make them simpler.

```
Error: the following sets of equations
abeDec(abeEnc(abePk(sk1),AccessPolicy,msg),abeSka(sk1,uid,att)) = abeEval
(AccessPolicy, msg, abeCheckKey(abePk(sk1),abeSka(sk1,uid,att), uid, att))

and

abeCheckKey(abePk(sk1),abeSka(sk1,uid,att),uid,att) = pass

use common function symbols.
Error: Blocks of equations marked [convergent] or [linear] should use function
symbols disjoint from each other.
```

Figure 1

I converted the equation to a *reduc* as follows:

```
(* fun abeCheckKey(abeKey,bitstring,bitstring,bitstring) :bitstring. *)
(* forall sk1:abeKey, uid:bitstring,att:bitstring;
abeCheckKey(abePk(sk1),abeSka(sk1,uid,att),uid,att) =pass; *)

(* changed to rewrite rule so used bool and they can not be used in
other equations or rewrite rules *)
reduc forall sk1:skey, uid:bitstring,att:bitstring;
abeCheckKey(abePk(sk1),abeSka(sk1,uid,att),uid,att) = true.
```

Instead of returning *pass*, a bitstring, I declared the return type to be *true*. And the first equation of abeDec() was simplified to:

```
abeDec(abeEnc( abePk(sk1), AccessPolicy, msg),
abeSka(sk1,uid,att) )= abeEval(AccessPolicy, msg, true).
```

After all the changes, the code ran fine. I have shared the code for this state as ABE-scheme – v1.pv

The output is shown:

```
--------------------------------------------------------------
Verification summary:

Non-interference pMsg cannot be proved.

Query not attacker(pMsg[]) is false.

--------------------------------------------------------------
```

Step 2: Add the PKI protocol

I copied all the code from corrected Needham-Schroeder Public Key protocol.

And modified the main process in ProVerif to include the `AccessPolicy` in Sender and attributes `att1` in Receiver.

```
new skS: sskey; let pkS = spk(skS) in out(c, pkS);
Sender(pkSender,abePkAA,AccessPolicy) |
Receiver(skReceiver, pkReceiver, pkS, att1) |
AA(pkS, abeSkAA) | processS(skS) | processK
```

Before the PKI protocol:

```
Sender(pkSender,abePkAA,AccessPolicy) |
Receiver(skReceiver, pkReceiver, abePkAA, att1) |
AA(pkReceiver, abeSkAA)
```

My main idea was:

The Receiver and Attribute Authority (AA) can act as Alice and Bob respectively, in the PKI protocol, where Alice makes a request to the Certification Authority (CA) first, to communicate with Bob.

This is similar to Receiver wanting to communicate with AA. The code fit without any major changes.

I kept trying the code online for fast results and made modifications accordingly.

More *Authentication* and *Secrecy* queries were added as they are already present in the PKI protocol.

As the Receiver and AA are authenticated, as they exchanged keys signed with key of CA, they were also able to establish two secret nonce values: `Na` and `Nb`.

They can be used as session keys.

Hence, the final change in code was to 1. symmetrically encrypt the identity (`pkReceiver`) of Receiver as well as the attributes as shown below.

```
out( c, senc(userid(pkReceiver),NX)); (* NX is Nb - Secret of Attribute Authority *)
out( c, senc(att1,Na));               (* Na - Secret of Receiver *)

in( c , encAbeSkAtt1:bitstring );
let abeSkAtt1 = adec(sdec(encAbeSkAtt1, Na), skReceiver) in
if abeCheckKey(pkX,abeSkAtt1,userid(pkReceiver),att1) = true
then let message = abeDec(abeEncMsg, abeSkAtt1) in
event e.
```

Figure 2

Then 2. AA receives the encrypted identity and attributes, 3. decrypts them, 4. generates the secret key for attributes **`att1`** for Receiver, encrypts it with nonce and then public key of Receiver.

```
in(c, userX:bitstring);
in(c, encAtt1:bitstring);

let att1 = sdec(encAtt1,NY) in (* NY is Na - Secret of Receiver *)
event f;
let userA = sdec(userX,Nb) in  (* Nb - Secret of AA *)
if userA=userid(pkY) then
out( c, aenc(senc(abeSka(abeSkAA, userA, att1),NY),pkY)).
```

Figure 3

5. In the **figure 2**, Receiver gets the encrypted secret key, decrypts them and then 6. the key is checked if it is valid, if true, he can decrypt and get the message

Final Output

All the authentication and secrecy queries were proved.

```
--------------------------------------------------------------
Verification summary:

Query inj-event(endBparam(x)) ==> inj-event(beginBparam(x)) is true.

Query inj-event(endAparam(x)) ==> inj-event(beginAparam(x)) is true.

Query not attacker(secretANa[]) is true.

Query not attacker(secretANb[]) is true.

Query not attacker(secretBNa[]) is true.

Query not attacker(secretBNb[]) is true.

Query not attacker(pMsg[]) is true.

Non-interference pMsg is true.

--------------------------------------------------------------
```

I have shared the code for this as ABE-scheme-final.pv

References

[1] B. Bat-Erdene, Y. Yan, M. B. M. Kamel and P. Ligeti, "Security Verification of Key Exchange in Ciphertext-Policy Attribute Based Encryption," 2022 7th International Conference on Signal and Image Processing (ICSIP), Suzhou, China, 2022, pp. 377-381, doi: 10.1109/ICSIP55141.2022.9887218.