This project has received funding from "HORIZON 2020" the European Union's Framework Programme for research, technological development and demonstration under grant agreement no 645220

Identity-based proxy re-encryption security extension for Dedalus

N. Fotiou, I. Thomas, S. Toumpis
Mobile Multimedia Laboratory, Athens University of Economics and Business, Greece
20/11/2018

Introduction

An Identity Based Encryption (IBE) scheme is a public key encryption scheme in which an arbitrary string can be used as a public key. An IBE scheme is specified by the four algorithms, *Setup*, *Extract*, *Encrypt* and *Decrypt*, summarized as follows:

- Setup: it is executed once by a *Secret Key Generator* (SKG). It takes as input a security parameter and returns a *master-secret key (*SK_{SKG}) and some generator specific *public parameters* (PP). The SK_{SKG} is kept secret by the SKG, whereas the PP are made publicly available.
- Extract: it is executed by a SKG. It takes as input PP, SK_{SKG}, and an arbitrary string ID, and returns a secret key SK_{ID}.
- Encrypt: it is executed by users. It takes as input an arbitrary string ID, a message M, and PP, and returns a ciphertext C_{ID}.
- Decrypt: it is executed by users. It takes as input C_{ID} the corresponding private decryption key SK_{ID}, and returns the message M.

An Identity-Based Proxy Re-Encryption (IB-PRE) scheme is a scheme in which a third party is allowed to alter a ciphertext, encrypted using an arbitrary string ID1, in a way that another user that owns a secret key SK_{ID2} can decrypt it. Such a scheme specifies two new algorithms, RKGen and Reencrypt, in addition to the IBE algorithms already discussed:

- RKGen: it is executed by a user that owns a secret key SK_{ID1}. It takes as input PP, the secret key SK_{ID1} and an arbitrary string ID2 and generates a (public) re-encryption key $RK_{ID1 \rightarrow ID2}$.
- ReEncrypt: it is executed by a third party. It takes as input PP, a re-encryption key $RK_{ID1 \rightarrow ID2}$ and a ciphertext C_{ID1} and outputs a new ciphertext C_{ID2}.

The ciphertext generated by the ReEncrypt algorithm can be decrypted only by using SK_{ID2}. The entity that performs the re-encryption learns nothing about the encrypted plaintext or about the secret keys of users ID1 and ID2.
Figure 1 illustrates an IBE-PRE example. There are three users, namely UserA, UserB, and UserC. UserA and UserB have received the secret key that corresponds to their user identity from two different SKGs (step 2). UserC encrypts a message by executing the IBE Encrypt algorithm with the identity of UserA (that is, "UserA") as a key and stores the generated ciphertext in a storage entity. Supposedly, UserB wishes to enable UserA to access the ciphertext generated by UserC: she generates an appropriate re-encryption key and sends it

to the storage entity. The storage entity is now able to re-encrypt the ciphertext in a way that can be decrypted using the private key of UserA.

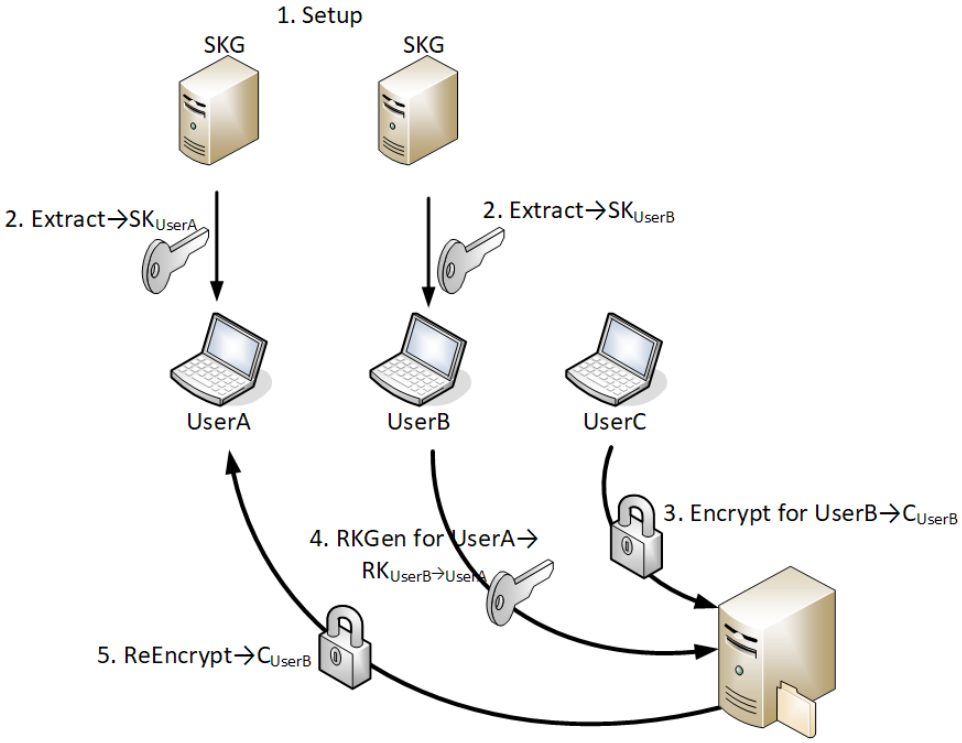

Figure 1: Identity-Based Proxy Re-Encryption (IB-PRE) example

Implementation

For the purposes of the project we have developed a file encryption application using the Green and Ateniese IBE-PRE scheme[1] as implemented by the Charm-Crypto library.[2] The current implementation[3] is composed of several python scripts. In our application, we consider as users the testbed nodes. Furthermore, each testbed node is identified by an identifier, denoted by ID_{node}.

Usage

Our file encryption application is composed of the following processes.

Setup

The Setup process is executed only once, and it generates the master secret key and the public parameters. Setup is simply executed as follows:

$ python Setup.py

[1] M. Green, G. Ateniese, "Identity-Based Proxy Re-Encryption," in Applied Cryptography and Network Security. Springer Berlin/Heidelberg, 2007

[2] https://github.com/JHUISI/charm

[3] Located at https://gitlab.com/mmlab-group/projects/unsurpassed/tree/master/security%20module

The outputs of this script are stored in the *setup* folder.

Node key generation

As already discussed, it is assumed that each (testbed) node is identified by an identifier. The secret key that corresponds to that identifier is generated using the Extract script. For example:

$ python Extract.py node1

The output of this script is stored in the *userkeys* folder.

File encryption

A file is encrypted using an ID_{node} as the (public) key. File encryption is implemented in the Encrypt.py script, which is invoked by supplying a ID_{node} and a filename as input parameters. For example:

$ python Encrypt.py node1 Hello_world.txt

The output of this script is stored in the *ciphertexts* folder. A file is encrypted using the following steps:

- A random symmetric encryption *K* is generated.
- The file is encrypted using a symmetric encryption algorithm (e.g., AES) and *K*
- *K* is encrypted using the IBE Encrypt algorithm

File decryption

A file encrypted using the previous process is decrypted using the Decrypt.py script. This script requires as input parameters the ID_{node} and the file name used during the encryption process. Using the provided input parameters, the script retrieves the corresponding secret key and ciphertext from the appropriate folders. An example of a call to that script follows.

$ python Decrypt.py node1 Hello_world.txt

The decrypted file is stored in the *plaintexts* folder.

Re-encryption key generation

A re-encryption key is used for re-encrypting a ciphertext generated using an ID_{node} *A* as a public key, such that it can be decrypted using the private key that corresponds to an ID_{node} *B*. Such a re-encryption key can be generated using the Re-keygen.py script. For example:

$ python Re-keygen.py node1 node2

The generated re-encryption key is stored in the *re-keys* folder.

File Re-encryption

An encrypted file can be re-encrypted using a re-encryption key. Fil re-encryption is implemented by the Re-ecnrypt.py script. This script requires as inputs two ID_{node}s and the name of the encrypted file. It then retrieves the appropriate re-encryption key and ciphertext. For example:

$ python Re-encrypt.py node1 node2 Hello_world.txt

The re-encrypted file is stored in the *ciphertexts* folder.

This project has received funding from "HORIZON 2020" the European Union's Framework Programme for research, technological development and demonstration under grant agreement no 645220

Decryption of a re-encrypted file

A re-encrypted file is decrypted using the Re-decrypt script and the identifier used for the re-encryption process. For example:

$ python Re-decrypt.py node2 Hello_world.txt

Performance

In our experiments IBE-PRE is applied, in essence, for protecting the symmetric key used for the file encryption. The size of this key is 128bits. We are using an elliptic curve that achieves a security level equivalent to RSA with a key size of 1024 bits. The size of our solution's public parameters is 2048 bits. Furthermore, the size the encrypted symmetric encryption key is 2048 bits, and the size of a re-encryption key is 2816 bits. In the following table the time required for each IB-PRE algorithm is presented, as measured in a Raspberry Pi 3 Model B, using the Charm-Crypto library.

Algorithm	Time in ms
Encrypt	156
Decrypt	96
Re-encryption key generation	180
Re-encrypt	19

Table 1: IB-PRE performance