COMP3334 Computer System Security Assignment 2

Poon Yat Sing (13040015D) <star.poon@connect.polyu.hk>

***Abstract*—This document provides the compilation procedures, execution instructions, design concerns and implementation details of of the Password-Based Authentication System. We will discuss the assumptions and requirements of the system. We will also compare different existing tools and explain the choice of libraries used in the system.**

***Keywords*—Scrypt, key derivation function, Galois/Counter Mode(GCM), AES, RSA, Digital Signature, password storage, salt, Authenticated Encryption (AE)**

I. Compilation

The system depends on `Openssl` and `Libcrypt`. Make sure Openssl and Openssl-devel packages is installed. For x64 Linux machine, `Libcrypt` has already been built. For other systems, you need to build `lib/libscrypt` and put `libscrypt.a` into `lib`. Use the following command to build the system:

```
make
```

To launch a sample test case:

```
make test
```

The compilation and execution of the program have been tested on Fedora22 with kernel version 4.2.3-200.fc22.x86_64, OpenSSL 1.0.1k-fips 8 Jan 2015 and Libcrypt v1.21.

II. Execution

A. Create user module

The `create_user` module is located in `build/create_user`. The program will take arbitrary number of `{id, pw}` pair as the input from stdin. User can enter End-Of-File(Ctrl+D) to terminate the input. The program will print out the id and its corresponding hashed password for verification. When the input is finished, the program will write the encrypted id and hashed password list into `list.txt`. The encryption and integrity protection mechanism will be discussed in Section III.

The input format of `create_user` (> is user input), the system will output the hashed password for verification:

```
>id1 pw1
>id2 pw2
>EOF
```

The following assumptions have been made:

- `id` and `pw` do not contain the space character.
- The number of `{id, pw}` pair is within the memory limit.
- The number of `{id, pw}` pair is within the encryption limit of Galois/Counter Mode ($2^{39} - 256\text{bits} \approx 68\text{GB}$) [4].

B. Authenticate user module

The `create_user` module is located in `build/authenticate_user`. The program will read the file `list.txt`. If the file is not created by `create_user` module or being modified, the system will reject. After the file is successfully loaded, the user can enter the id and password. The input format of `create_user` (> is user input):

```
>id1 pw2
Authentication fail.
>id2 pw2
Welcome to the system, id3!
>EOF
```

III. Design and library choice

A. Hash of password

We decided to use Scrypt[1] plus salt to hash the password and store it in MFC format[2]. The following table quote from Colin's paper[1] summarized the strength of common password storage method. Scrypt has a clear advantage and it also allow user to specify the amount of memory and CPU to use. The actual implementation used is Libscrypt [3].

TABLE I. Estimated cost of hardware to crack a password in 1 year.[1]

KDF	**6 letters**	**8 chars**	**10 chars**
DES CRYPT	<$1	<$1	$1
MD5	<$1	<$1	$1.1k
MD5 CRYPT	<$1	$130	$1.1M
PBKDF2 (5.0s)	<$1	$920k	$8.3B
bcrypt (3.0s)	<$1	$4.3M	$39B
scrypt (3.8s)	<$900	$19B	$175T

B. Authenticated encryption plus RSA signing

To protect the confidentiality and integrity of list.txt, we use AES-256-GCM (AES 256bit Galois/Counter Mode) to encrypt the file. This can protect the username list and potentially protect the weak password used by the users. Each time the `create_list` module generate list.txt, it will randomly generate a key and initialization vector for AES-GCM encryption. Since GCM is an authenticated encryption algorithm, it will generate a tag for message authentication. If the userlist is being modified, the MAC will be able to detect.

Since the attack may be able to obtain the source code or use reverse engineering to attack the system, symmetric encryption won't help. In our design, we use RSA private key to encrypt the key, initialization vector and tag used by

GCM encryption. The private key is stored in the secure `authenticate_user` module. The public key is stored in `create_user` module. The process is explained in the following two graph.

Since the attack may be able to obtain the source code or use reverse engineering to attack the system, symmetric encryption won't help. In our design, we use RSA private key to encrypt the key, initialization vector and tag used by GCM encryption. The private key is stored in the secure `authenticate_user` module. The public key is stored in `create_user` module. The process is explained in the following two graph.Since the attack may be able to obtain the source code or use reverse engineering to attack the system, symmetric encryption won't help. In our design, we use RSA private key to encrypt the key, initialization vector and tag used by GCM encryption. The private key is stored in the secure `authenticate_user` module. The public key is stored in `create_user` module. The process is explained in the Fig. 1 and Fig. 2.

Our system are adopting a defense in depth approach, Table II summarized the protection features by our system.

References

[1] Colin Percival. Stronger Kye Derivation via Sequential Memory-hard Functions. `BSDCan'09`.

[2] Assurance Technologies. *Modular Crypt Format*[Online]. Available: https://pythonhosted.org/passlib/modular_crypt_format.html

[3] libscrypt. `https://github.com/technion/libscrypt`

[4] David A. McGrew and John Viega. The Galois/Counter Mode of Operation (GCM). `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf`

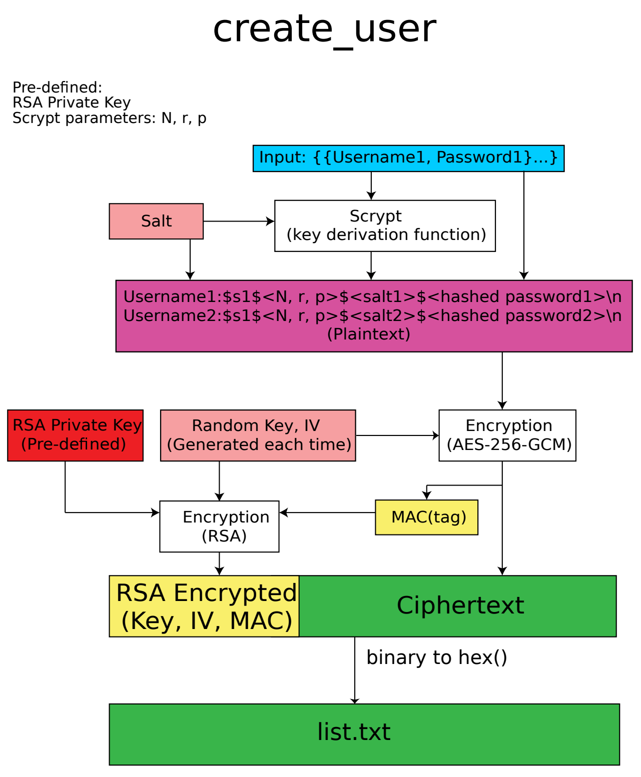

Fig. 1. Workflow of create_user module

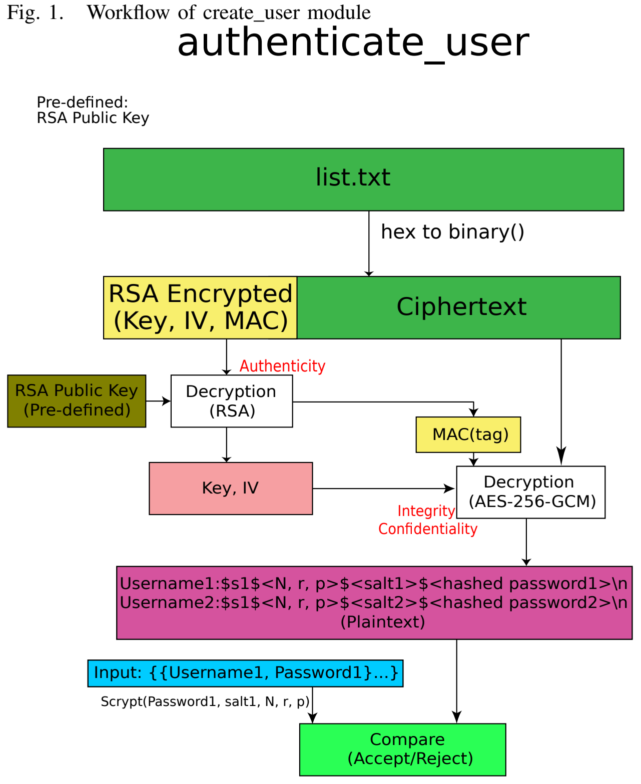

Fig. 2. Workflow of authenticate_user module

TABLE II. DEFENSE IN DEPTH

Attac\Protect	**Pw hashing**	**RSA Encryption**	**AES GCM Encryption**	**Level of protection**
Brute force	Secure	Secure	Secure	Can protect weak password since username are encrypted
Source code/ Reverse engine	Secure	Secure (integrity)	Broken	list.txt is not modified, however the username and hashed password are known to attacker
Source code + Brute force	Rely on strong password	Secure (integrity)	Broken	list.txt is not modified, howeverthe username and hashed password are known to attacker User using strong password are still secure