REED: Rekeying-aware Encrypted Deduplication Storage

Introduction

REED is an encrypted deduplication storage system with rekeying enabled. Specifically, it can replace an existing key with a new key so as to protect against key compromise and enable dynamic access control. REED builds on a deterministic version of all-or-nothing transform (AONT) for secure and lightweight rekeying, while preserving the deduplication capability. It also exploits similarity to mitigate key generation overhead. We implement a REED prototype with various performance optimization techniques.

Publications

- Jingwei Li, Chuan Qin, Patrick P. C. Lee, and Jin Li.
  Rekeying for Encrypted Deduplication Storage.
  Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2016) (Regular paper), Toulouse, France, June 2016.

- Chuan Qin, Jingwei Li, and Patrick P. C. Lee.
  The Design and Implementation of a Rekeying-aware Encrypted Deduplication Storage System.
  ACM Transactions on Storage (TOS), 13(1), 9:1–9:30, March 2017.

Installation

Dependencies

REED is built on Ubuntu 16.04 LTS with a g++ version of 5.1.0. It depends on the following libraries:

- OpenSSL (https://www.openssl.org/source/openssl–1.0.2a.tar.gz)
- Boost C++ library (http://sourceforge.net/projects/boost/files/boost/1.58.0/boost_1_58_0.tar.gz)
- GMP library (https://gmplib.org/)
- LevelDB (https://github.com/google/leveldb/archive/master.zip)
- CP-ABE toolkit and libbswabe (http://acsc.cs.utexas.edu/cpabe/)
- PBC library (https://crypto.stanford.edu/pbc/)

We pack CP-ABE toolkit (version 0.11), GMP library (version 6.1.2), libbswabe (version 0.9) and PBC library (version 0.5.14) in `dependency/`. LevelDB (version 1.15.0) is also provided in `server/lib/`.

Instructions

Step 1: Run the following commands to install OpenSSL and Boost C++ library.

```
$ sudo apt-get install libssl-dev libboost-all-dev
```

Step 2: The GMP library depends on m4 that can be installed via the following command:

```
$ sudo apt-get install m4
```

Then, compile our provided GMP source to make install.

```
$ tar -xvf dependency/gmp-6.1.2.tar && cd gmp-6.1.2/
$ ./configure
$ make
$ sudo make install
$ cd ../ && rm -rf gmp-6.1.2/
```

Optionally, run `make check` (before removing the extracted GMP directory) to check the correctness of the GMP library.

Step 3: The PBC library depends on flex and bison, both of which can be installed via the following command:

```
$ sudo apt-get install flex bison
```

Then, compile our provided PBC source to make install.

```
$ tar -xvf dependency/pbc-0.5.14.tar.gz && cd pbc-0.5.14/
$ ./configure
$ make
$ sudo make install
$ cd ../ && rm -rf pbc-0.5.14/
```

Step 4: The libbswabe depends on libglib2.0-dev which can be installed via the following command:

```
$ sudo apt-get install libglib2.0-dev
```

Then, compile our provided libbswabe source to make install.

```
$ tar -xvf dependency/libbswabe-0.9.tar.gz && cd libbswabe-0.9/
$ ./configure
$ make
$ sudo make install
$ cd ../ && rm -rf libbswabe-0.9/
```

Step 5: Configure the cpabe package for installation.

```
$ tar -xvf dependency/cpabe-0.11.tar.gz && cd cpabe-0.11/
$ ./configure
```

Informed by the solutions to make error with libgmp and error in linking gmp, you need to make a few changes on the configuration files.

First, add a line `-lgmp` into the definition of `LDFLAGS` (Line 14) in the Makefile. This makes the `LDFLAGS` like:

```
...
LDFLAGS = -O3 -Wall \
    -lglib-2.0  \
    -Wl,-rpath /usr/local/lib -lgmp \
    -Wl,-rpath /usr/local/lib -lpbc \
    -lbswabe \
    -lcrypto -lcrypto \  # remember to add `\` here
    -lgmp # newly added line
...
```

Second, add the missed semicolon in the Line 67 of the file policy_lang.y.

```
...
result: policy { final_policy = $1; } # add the last (missed) semicolon
...
```

Finally, make and install the library.

```
$ make
$ sudo make install
$ cd ../ && rm -rf cpabe-0.11/
```

Step 6: The LevelDB depends on libsnappy-dev, which can be installed via the following command.

```
$ sudo apt-get install libsnappy-dev
```

Then, compile and make the LevelDB that is located at `server/lib/`.

```
$ cd server/lib/leveldb-1.15.0/ && make
```

REED Configurations

Server

We include both datastore (for storing file related data and metadata) and keystore (for storing key information) in REED server. Compile server via the following command:

```
$ cd server/ && make
```

Then, start a REED server by the following command. Here `port_1` and `port_2` direct to the datastore and keystore ports, respectively.

```
$ ./SERVER [port_1] [port_2]
```

Key Manager

We have a key manager for key generation. Run the following commands to maintain key management service on the `port` of a machine.

```
$ cd keymanager/ && make
$ ./KEYMANAGER [port]
```

Client

Edit the configuration file `client/client.conf` to set the server and the key manager information (note the information should be consistent with the IP address and port configured for the server and key manager).

- Line 1 specifies the number of servers in usage; currently we only support one server that combines the datastore and keystore.
- Line 2 specifies the IP address and port of the configured key manager.
- Line 3 specifies the IP address and port of the configured server (including both the datastore and keystore, see above).

Compile and generate an executable program for client.

```
$ cd client/ && make
```

Usage Examples

After configuring all entities, we can use REED client for typical commands:

```
// keygen
$ ./CLIENT -k [attribute] [private key filename]

// upload file
$ ./CLIENT -u [filename] [policy] [secutiy type]

// download file
$ ./CLIENT -d [filename] [private key filename] [secutiy type]

// rekeying file
$ ./CLIENT -r [filename] [private key filename] [new policy] [secutiy type]

// parameters
// [filename]: full path of file
// [policy]: encryption policy of CA-ABE
// [attribute]: attribute for generating CP-ABE private key
// [security type]: {HIGH} AES-256 & SHA-256; {LOW} AES-128 & SHA-1
// [private key filename]: file used to store CP-ABE private key
```

Key Generation

Before using REED, run key generation to generate private keys. To generate an ABE private key for user 1, run the example keygen command:

```
$ ./CLIENT -k 'id = 1' sk_1
```

You can get the key file `sk_1` (located in `client/keys/sk/`) that stores the private key related to attribute `id = 1`. Note that the attribute is in the form of string (quoted by ' ') and a blank space is necessary before and after =. The format is consistent with the documentation of the CP-ABE toolkit.

File Upload

Suppose we want to upload a file (say `file`) with a policy that requires only user 1 or user 2 can access the file. Run the following command to use advanced protection of REED.

```
$ ./CLIENT -u file 'id = 1 or id = 2' HIGH
```

When specifying the policy, make sure you own the private key whose attribute satisfies the policy. For example, in the above example, the private key with `id = 1` can decrypt the ciphertext under the policy `id = 1 or id = 2` successfully, but another private key with `id = 3` cannot.

File Download

Type the following command to download `file` with the private key `sk_1`:

```
$ ./CLIENT -d file sk_1 HIGH
```

The downloaded file will be renamed to be `file.d` automatically.

Note that, the security type (e.g., `HIGH` in the above example) should be consistent with the pre-defined security type when uploading the file.

Rekeying

To update the policy (e.g., `id = 1 or id = 2`) of `file` to a new policy `id = 1 or id = 3`, run the following command.

```
$ ./CLIENT -r file sk_1 'id = 1 or id = 3' HIGH
```

This revokes the access privilege of user 2 and grants user 3 to access `file`. Like file download, the security type should also be consistent with the pre-defined security type when uploading `file`.

After rekeying, you can generate a new private key (to simulate the action of user 3) and download `file` using the key.

```
$ ./CLIENT -k 'id = 3' sk_3
$ ./CLIENT -d file sk_3 HIGH
```

However, if using the private key of `id = 2`, you cannot decrypt `file` successfully.

```
$ ./CLIENT -k 'id = 2' sk_2
$ ./CLIENT -d file sk_2 HIGH

// outputs
...
cannot decrypt, attributes in key do not satisfy policy
...
```

Limitations & Known Bugs

- In an ABE cryptosystem, a trusted party (e.g., authority) maintains an ABE master secret to generate ABE private keys. In REED, we do not implement the authority, and assume all REED clients own the system-wide master secret (that has already been generated). Each client can use the secret to generate its private keys (e.g., via the keygen interface).

- We test REED with a special case of CP-ABE: (i) assign a single attribute (e.g., id) with private key and (ii) express policy in access tree with an OR gate connecting all authorized identifiers. We cannot guarantee REED works well with generic tree-based access control (that is supported by the CP-ABE toolkit).

- We do not expose interface for lazy revocation (but we implement key regression for policy update). Currently, REED only supports active revocation that immediately revokes the access privileges of old keys.

- In our test, REED works well with most of files. However, for a few files, we face chunking crashes in upload or integrity check failures (missing the data chunks in the last container) in download. The bugs possibly depend on the content of test files.

Maintainers

- Origin maintainer:
  - Chuan Qin, CUHK, chintran27@gmail.com
- Current maintainers:
  - Yanjing Ren, UESTC, tinoryj@gmail.com
  - Jingang Ma, UESTC, demon64523@gmail.com