End-to-Same-End Encryption: Modularly Augmenting an App with an Efficient, Portable, and Blind Cloud Storage

Long Chen, *Institute of Software, Chinese Academy of Sciences;* Ya-Nan Li and Qiang Tang, *The University of Sydney;* Moti Yung, *Google & Columbia University*

https://www.usenix.org/conference/usenixsecurity22/presentation/chen-long

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

E2SE is a system for securely storing private data in the cloud with the help of a key server (an App server). Our E2SE artifact is a prototype in Java including both the client and key server implementation. The software requirements includes JDK 8[1] or later, Maven 3.8.1 or later [2] and some dependencies which could be automatically downloaded by maven, and OpenSSL 1.1.1[3] with *libssl-dev*. To reproduce the evaluation results, the hardware requirements include an AWS EC2 t3.xlarge instance in Seoul for running the client, an AWS EC2 t2.micro instance in Osaka for running the key server, and a AWS S3 cloud server in Tokyo. We provide the EC2 instances satisfying the software requirements and S3 cloud server access for the evaluation.

The key server could be run to provide assistance for secure storage. Given a plain file, the client could run to securely deposit the file to cloud storage and securely retrieve it later. The client will output the time cost of each procedure. The average statistic result should be consistent with the efficiency part of our paper.

A.2 Artifact check-list (meta-information)

- **Algorithm:** OPRF, AES, KDF, SHA256
- **Program:** Siege[4], an open-source benchmarking tool used to test the performance of web server, is needed. The throughput test of key server in our paper is done with Siege 4.0.4.
- **Compilation:** JDK 8 or later, Maven 3.8.1 or laer
- **Run-time environment:** ubuntu 18.04 TLS.
- **Hardware:** An AWS EC2 t3.xlarge instance, an AWS EC2 t2.micro instance, and AWS S3 server are needed.
- **Run-time state:** It is network sensitive as the client needs to communicate with the key server and cloud server (AWS S3 in our implementation). Both the network delay between the client and key server & cloud server will affect the time cost. The client transfers the file to/from the cloud server, where the network speed also affect the measured time cost.
- **Execution:** The running time depends on the the file size and network delay and speed. In our experiment described in the paper, the time cost of running the whole procedure 25 times for each file varies from several minutes to one hour with the file increasing from 10mb to 300mb.
- **Metrics:** Running the compiled jar package with calling the key server, it provides service in port 20202 to help the client. Running the compiled jar package with calling the client, the execution time for each procedure is reported. Using Siege to test the key server performance, the throughput is reported.
- **Output:**
  1. For the efficiency test, the outputs are the running time of IBOPRF, Give, Take, optimized secure deposit and retrieve, secure deposit and retrieve, plain deposit and retrieve. The statistically average of the outputs should be consistent with the efficiency part of the paper, including the Figure 8,9 and the Table 2.
  2. For the key server throughput test. The output is the throughput of key server, say the number of transactions per second (trans/sec). When the key server is deployed on devices with different processing cores and memory, the throughput increases almost linearly with the processing core increasing.
- **Experiments:** The full preparation is described in the README.md instruction of the open source code https://github.com/yananli117/E2SE. We provide two well-prepared EC2 instances. The reviewer could upload the code to the instances, and follow the Run instruction and test instruction to get the results.
- **How much disk space required (approximately)?:** It depends on the data size. We test the files from 10mb to 300mb, so the required disk should be less than 1GB.
- **How much time is needed to prepare workflow (approximately)?:** Prepare from scratch, it probably costs 2 hours. We provide a well-configured instances in EC2 to run, only cost 10 minutes.
- **How much time is needed to complete experiments (approximately)?:** Several minutes are needed to test whether the artifact works. If redo all the experiments to produce the data of the four figures and one table about efficiency and scalability, approximately less than 6 hours are needed.
- **Publicly available:** . Github: https://github.com/yananli117/E2SE.
- **Code licenses:** Our code is under MIT license.
- **Archived (stable URL):** https://github.com/yananli117/E2SE/tree/bd4de7fb1c6c70df96bf89a17c100624fa665d0b

A.3 Description

A.3.1 How to access

We have open sourced our artifact at https://github.com/yananli117/E2SE. To reproduce the performance evaluation results, we provide the EC2 cloud instances with proper configurations and credentials, etc. Since we do not know when the reviewers will execute our code to repeat, to avoid keeping the cloud instances running for a whole month (which is a bit unnecessary waste), please inform us in the system before you plan to test. We will start the well-configured EC2 instances and send you the corresponding IP addresses.

A.3.2 Hardware dependencies

- To test the artifact is workable, two processes deployed in one or two devices are needed for the key server and client.

[1] https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html
[2] https://maven.apache.org/download.cgi
[3] https://www.openssl.org/source/
[4] https://github.com/JoeDog/siege

- To reproduce the evaluation results, we provides two EC2 instances for running the client and key server. The client is deployed in EC2 t3.xlarge instance in Soul, and the key server is deployed in EC2 t2.micro instance in Osaka.
- AWS S3 as cloud storage server (During the artifact review, we provide the access credential to access it). To apply to other cloud services, the code should be tuned a bit for the different cloud APIS.

A.3.3 Software dependencies

The software dependencies include JDK 8 or later version, Maven 3.8.1 or later versions, OpenSSL 1.1.1 and *libssl-dev*. (Some dependencies could be automatically installed in the compilation using Maven.) To test the throughput, the key server is implemented as a web server, so Tomcat + nginx framework are needed for the key server. The Siege tool in the test server is needed.

A.3.4 Data sets

N/A

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

N/A

A.4 Installation

When you plan to test the artifact, please do the following steps:

- Inform us to open the two instances and we will return the two ip addresses of the two instances for your access.
- Download all the credentials of AWS S3 with the link we provide only for artifact evaluation. For other users, please login in your own AWS account and get the security credential following the link https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html.
- Open the terminal, securely and remotely control the EC2 t3.xlarge instance in Seoul via SSH for running the client

```
ssh -i CredentialPath/EC2_Client_Seoul.cer
    ubuntu@ip_client
```

- Open another terminal, and securely and remotely control the EC2 t2.micro instance in Osaka via SSH for running the key server

```
ssh -i CredentialPath/EC2_keyServer_Seoul.cer
    ubuntu@ip_client
```

- Clone the git repository and change to the root directory for both the client and key server

```
git clone https://github.com/yananli117/E2SE.git
cd E2SE/E2se4j
```

- Follow the instruction in README.md shown in https://github.com/yananli117/E2SE to config, compile, run and test our artifact and use our prototype for protecting data.

A.5 Experiment workflow

A.6 Evaluation and expected results

In our paper, we have two main claims efficiency and scalability.

A.6.1 Claim on efficiency

We do experiments to demonstrate that our design is efficient. We mainly measure the time cost of each procedure during the secure storage, including the register, give, take, deposit and retrieve procedures. We also compare the time cost of plain deposit/retrieve with the time cost of secure and optimized secure deposit/retrieve to show that the overhead of secure deposit/retrieve is very small, which could be seen in Figure 8,9 and Table 2.

When running the client with a specified plain file, 25 users run the whole procedure sequentially as follows: register to the system, run the give protocol to share the data encryption key (ibOPRF + give), encrypt the specified file and deposit the ciphertext to S3 in an optimized way (secureDepOpt), run the take protocol to reconstruct the data encryption key (ibOPRF + take), retrieve the ciphertext and decrypt it in an optimized way (secureRetOpt), encrypt the specified file and deposit the ciphertext to S3 (secureDep = Enc + DCT), retrieve the ciphertext and decrypt it (secureRet = RCT + Dec), deposit plain file to S3 (plainDep), retrieve plain file from S3 (plainRet), encrypt a plain file (Enc) and Decrypt the encrypted file (Dec).

We need to keep the key server running and run the client 8 times by specifying plain files of different sizes from 10mb to 300mb shown in the paper. To produce a file with specific size, we add the generation code in testGuide/ComFile1.java. Please follow the RREADME.md instruction to generate the file with a specific size.

With the output in the client terminal, we can calculate the average time cost for each procedure and the breakdown to form the Figure 8,9 and Table 2.

Since the time costs mainly comes from communication between the client and two servers, they could vary depends on the network delay between the deployed client to the deployed key server and the specified S3 server. The network speed could also affect the time cost especially when the size of file is large. So we cannot give the range of time cost for different network environments. We just claim that the test results could be reproduced if the experiments are the same as ours shown in the paper.

A.6.2 Claim on scalability

. We claim that our key server is scalable. We observe that in secure storage, the key server overhead mainly comes from interacting with the client to run the IBOPRF, which could affect the scalability. To demonstrate our key server is scalable, we deploy the key server as a web server with nginx + tomcat framework. We use Siege as the throughput benchmarking tool to test how many IBOPRF requests the key server could handle at per second. The client use Siege to sends 400 parallel https requests on IBOPRF to the key server and iterates 250 times. The specific commands are shown in README.md. only providing the IBOPRF service.

A.7 Experiment customization

A.8 Notes

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.