TLSNotary Project Final Report

Yiwen Tang Jiayi Zhang Yawei Zhang Rui Zhang

May 1, 2020

1 Introduction

Although TLS protocol is already used to encrypt network for other network protocols such as HTTP protocol[1][2][3], there still exists attacks aiming at SSL/TLS such as man-in-the-middle attack[4]. The attacker would establish separate connections with the server and client, disguise as a legal connection endpoint, and intercept normal network data to alter or insert illegal data without being recognized. Users are currently unable to prove to a third party the content they have observed on a particular website. One of the most popular methods for users to document and share content they watch on the Internet are screenshots that are trivial to falsify, which is inefficient and inconvenient. However, TLSNotary allows a client to provide evidence to a third-party auditor that certain web traffic occurred between himself and a server. The evidence is irrefutable as long as the auditor trusts the server's public key. Apart from web scenarios, TLSNotary can also be applied to the blockchain[5], physical sensors and so on to provide reliable real data.[6]

In this project, we implemented an additional protection mechanism above the Transport Layer Security (TLS) protocol to ensure users to retrieve secure data from the network transfers. For users to get data from web server, this can be done in two ways, from the server side and the client side. The server side would require the modification of the TLS protocol and a third party auditor can be used in the client side to attest a TLS connection. We studied the nature of the network between the server and the client, some detailed descriptions of TLS and HTTP and utilize some properties of the cryptography algorithms to achieve our desired results. After that, we implemented the algorithms, deploy the server and the client side to generate verification between different servers, and finally find ways to improve the performance of the algorithms.

2 TLSNotary

In this section, we will analyse the mechanism of TLSNotary and explain the workflow of two functions: generating a proof file and reviewing a proof file.

TLS protocol operates between the transport Layer and the application layer. It wraps application layer traffic in encryption during transport. Thus, TLS protocol can be utilized to encrypt HTTP channels. HTTP protocol transfers data in plaintext which can bring risks of eavesdropping, tempering and pretending. TLS protocol is designed to solve these problems. It can encrypt all the transmitted data that won't be acquired by third-party.

It has a verification mechanism so that both the sender and the receiver could find out if something has changed. It has an identity certification to prevent identity impersonation.

Asymmetric encryption of RSA between the client and server to obtain the master secret is what the core of TLSNotay lies in, which could be used for generating a session key for data packet exchange.

The multiplicative homomorphic property of RSA encryption is applied to allow each part of the client (Auditee and Auditor) to pass the full pre-master secret to the server without sharing the two-part pre-master secret. The purpose is to make the server receive a complete pre-master secret when both Auditee and Auditor do not know each other's pre-master secret.

2.1 Mechanism

When users send a URL to the TLSNotary, the client initiates a conversation request to the server, so that the client can obtain the data provided by the server. In order to prove that the data obtained by the client is indeed from the source server, the client is divided into two parts - Auditee and Auditor. Auditee performs data acquisition and Auditor is responsible for data verification.

To finish a whole TLS handshake, Auditor separated from Auditee, so they need to interact session. The mechanism is as follows: Because both Auditee and Auditor contain only half of the pre-master secret, Auditor sends half of RSA-encrypted pre-master secret to Auditee.[7] The reason for this is to ensure security by preventing Auditee and Auditor from colluding and Auditee tampering with the full key. Auditee encrypts its pre-master secret with RSA, multiplying the result with the Auditor, and then passes it to the server. Due to the multiplicative homomorphism of RSA,[8] the server can decrypt the complete pre-master password by its own private key. As a result, this process establishes a session where Auditee and Auditor do not share the full pre-master password.

At the same time, the Auditor and the Auditee use the pseudo-random function to generate the Server Mac Key and the other three session keys respectively, because they only have half of RSA-encrypted pre-master.[9] In this way, when Server send encrypted data to Auditee, Auditee can't decrypt it, so Auditee can't modify the data. After Auditor verification, Auditee can generate a proof file.

2.2 Workflow

2.2.1 Gernate a proof file

After they finish the TLS handshake, the server will use Server Mac Key to encrypt requested data. Because Auditee doesn't have Server Mac Key, it can't decrypt the data and modify it. Then, Auditee send encrypted data to Auditor, after Auditor use Server Mac Key to decrypt and verify, it will sign the data. Next, Auditor send the signature to Auditee. At

this time, we finish the audit process. Auditee will write the signature from Auditor and other useful information into a proof file.

2.2.2 Review a proof file

Our system not only can verify the data and generate a proof file, it also can review a proof file. Main steps are as follows: When the auditor receives the proof file, first it should verify whether the public key of the server matches the public key in the proof file. This verification is divided into two steps: 1) verifying whether the digital certificate in the proof file has a trusted CA signature.[10] 2) The public key is extracted from proof file and compared with the public key of the server. After verifying the public key, Auditee need verify the information contained in the proof file step by step. First, Auditee should extract the certificate from the uploaded file (acquire the RSA encrypted modulus and server domain name) and verify that the auditee provides a valid Audito[7]r's signature. After verifying the valid signature, Auditee verifies that data hash provided by the proof file matches the data provided by the server. If it is correct, the certificate is credible and the data is from the original server and has not been tampered with. Otherwise, the verification of the proof file fails and the data does not match the original server.

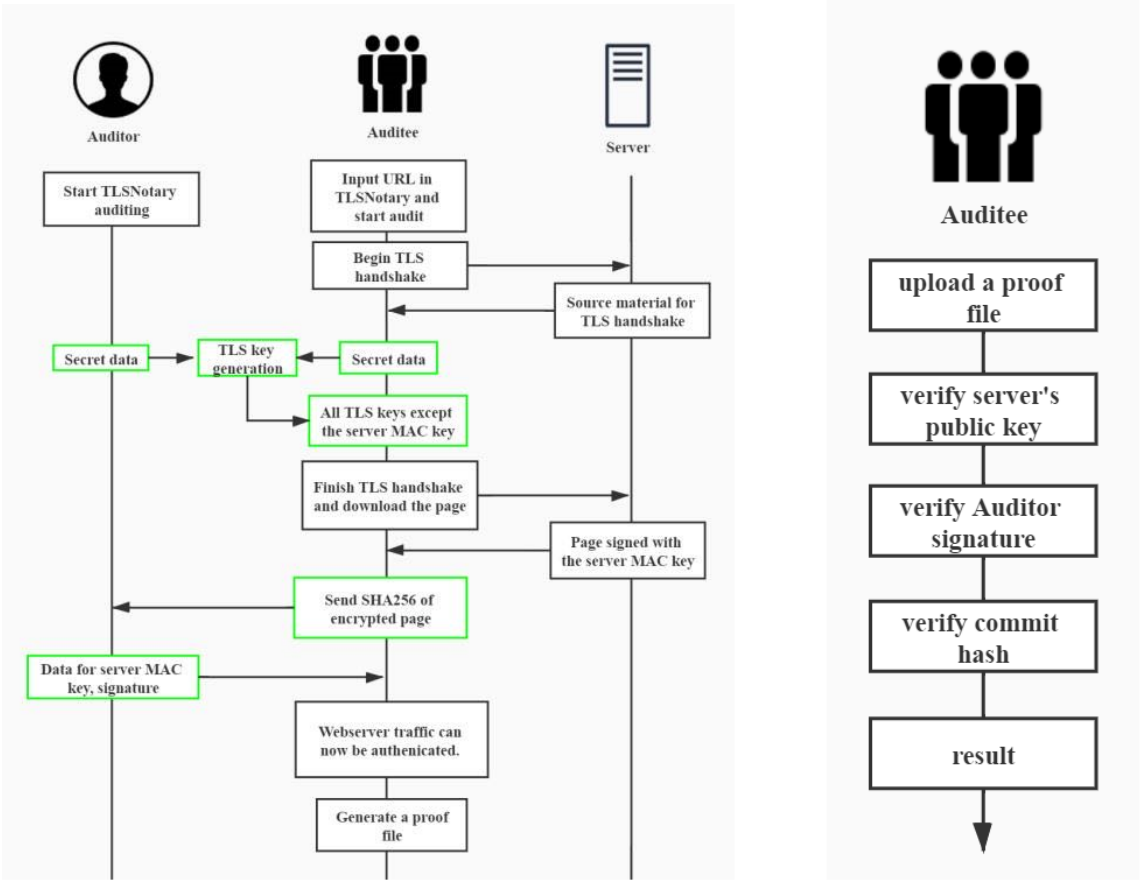


Figure 1: The Workflow of TLSNotary

3 Implementation

Based on our algorithms and mechanisms that have been discussed above, we implemented functionalities for Auditor and Auditee, deployed Auditor at AWS and developed web applications using Flask for TLSNotary.

3.1 Auditor deployment

We deployed the Auditor and Signing Server in image Ubuntu Server 16.04 LTS (HVM), SSD Volume Type on AWS. The cloud instance exports 10011 as a public TCP port for connection with Auditee.

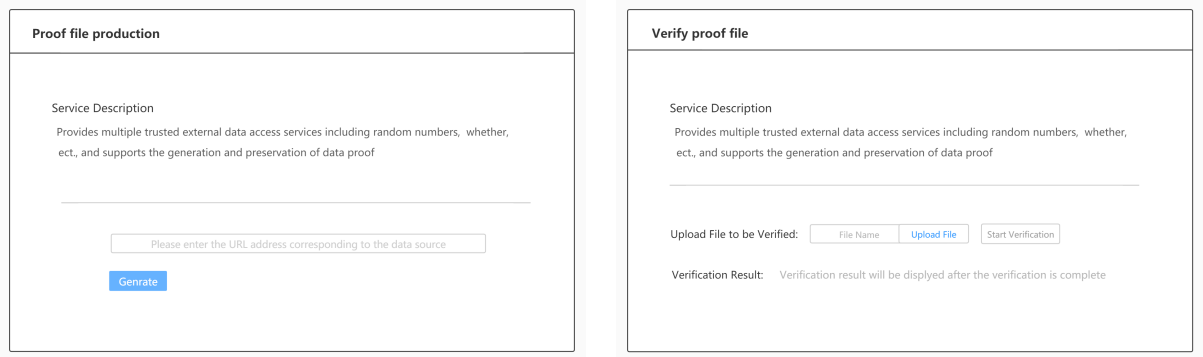
3.2 Web design

The web application for TLSNotary has two webpages: generate and review.

Generate webpage is designed to generate a proof file for the requested URL. Users only need to enter a URL in the input field to generate a proof. If the proof is generated successfully, the generate webpage will display the proof file name and the web browser will open the requested webpage.

Review webpage is designed to verify a proof file that has been generated through our website. Users could upload a pgsg file and our system will verify the file format, server public key, server certificate chain, auditor signature, encrypted and decrypted server response.

Figure 2 shows wireframes of the generate webpage and review webpage.



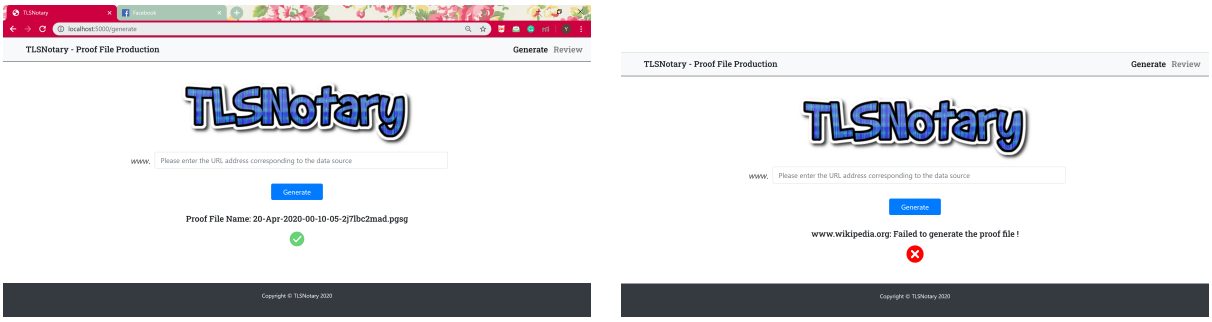
(a) Wireframe of generating proof file

(b) Wireframe of verifying proof file

Figure 2: Wireframe of website.

When the processes of generation and review failed, webpages won't tell you detailed reasons for failure. In this way, we could prevent attackers from utilizing error messages to fabricate legally requested data and compromising our system.

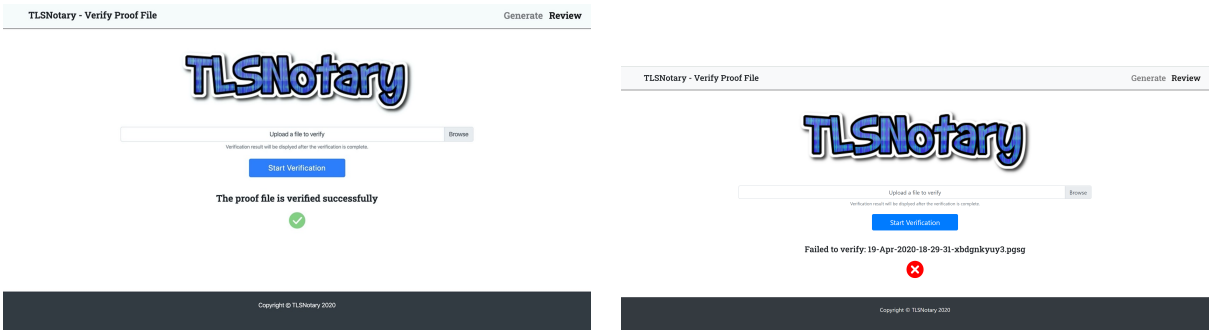
Figure 3(a) shows an example of successfully generating a proof file for `facebook.com` and open the requested URL. Figure 3(b) shows an example of failing to generate a proof file for the requested `wikipedia.org` .



(a) Generation success (b) Generation fail

Figure 3: Generate results

Figure 4(a) shows an example of successfully verifying a proof file and figure 4(b) shows an example of failing to verify a proof file.



(a) Successfully verify proof file (b) Failed to verify proof file

Figure 4: Review results

4 Experiments and Result Analysis

In this part, we will discuss our experiment design, results and analysis. We mainly conduct the experiments from three aspects: whether our mechanism can be widely used in different websites; how the latency is produced; tradeoff for the security.

4.1 Functional Experiment

We choose the top 500 websites from `https://www.alexa.com/topsites` and post generate proof file requests on them. We repeat the requests for all websites twice and 614 of the 1000 experiments success. Our mechanism can work on the majority of websites.

4.2 Latency Experiment

From experiments, we find that the latency of generating the proof file is large and find a way to decrease latency will have great improvement in the user's experience. We first analyze the possible reasons that cause the latency, and then do experiments with different reliable sites and sites in different server locations accordingly.

4.2.1 Latency Analysis

The first part latency comes from the interaction between the Auditee and Auditor. They have multiple interactions to generate a session key and sign the hash value obtained from the server. Therefore, throughout the process, Auditee and Auditor need to interact multiple times. If the network quality is poor and the network distance is long, the delay time of multiple interactions would be long.

The second part latency comes from the interaction between the Auditee and server. The multiplicative homomorphic property of RSA is used when sending the pre-master secrets to the server and this property requires a padding sequence with no zero bytes being generated. Therefore, a reliable site is needed before talking to the real server. We can test on the site of whether there are zero bytes in the padding and regenerate a new sequence if there is any. This extra communication to a reliable site will bring some latency. Same as the first part, if the network quality is poor and the network distance to the server is long, the delay time of multiple interactions would be long.

4.2.2 Latency with Different Reliable Sites

We choose the top 50 websites from `https://www.alexa.com/topsites` and test the latency of generating proof files with different reliable sites. We choose 6 reliable sites, which are `worldofwarcraft.com`, `y8.com`, `king.com`, `wildtangent.com`, `networksolutions.com` and `thawte.com`. The average latency from communicating with the reliable site and total latency are calculated for each reliable site and shown below.

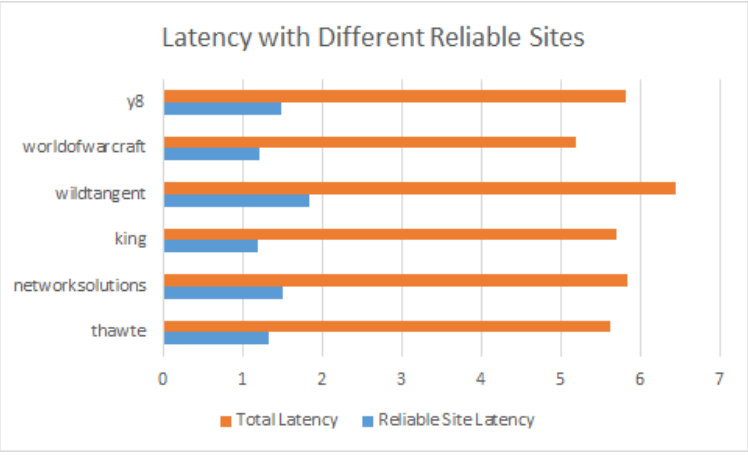


Figure 5: Latency with Different Reliable Sites

From Figure 5, we can see that the latency from the different reliable sites have some differences and we can decrease the total latency from a great choice of a reliable site.

4.2.3 Latency with Sites in Different Server Locations

According to the experiment results in Section 4.2.2, we choose `worldofwarcraft.com` and `king.com` as our reliable sites because they have the least latency among all sites. Each experiment in this section will choose a reliable site between them randomly. We choose the top 50 websites from `https://www.alexa.com/topsites` and repeat 10 times of generating proof files for each website to get a stable latency value. We select and show the average total latency for 30 websites in the figure.

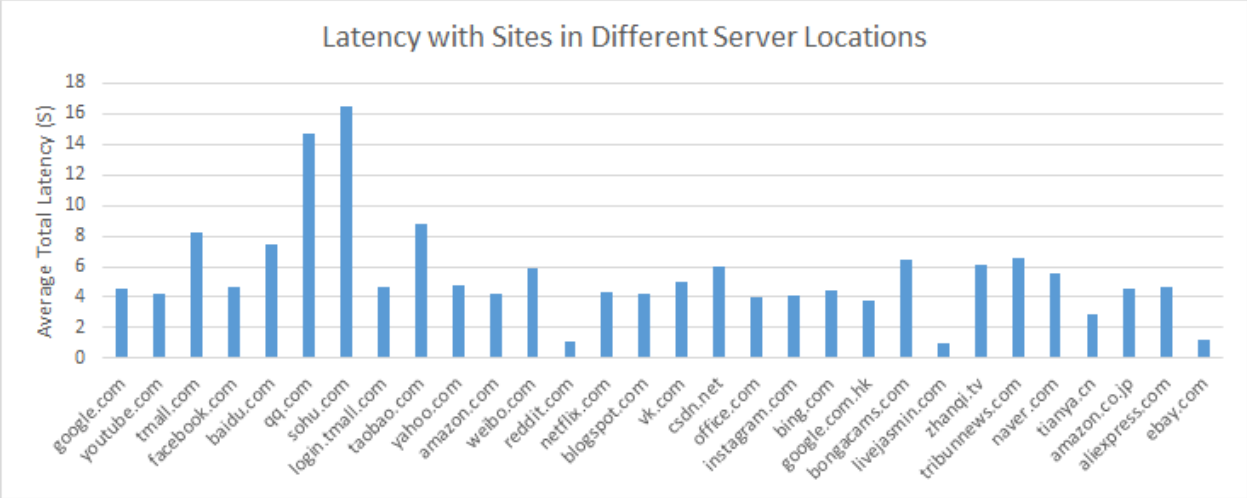


Figure 6: Latency with Sites in Different Server Locations

From Figure 6, we can see that the total latency from different servers has a great difference. For example, the latency from `qq.com` and `sohu.com` are much larger than other websites. This may because the webserver of these two websites locate in China. The extra communication time between the server and clients makes the total latency large from the long distance between them.

4.3 Security Experiment

At this project, we use RSA encryption. It is possible that the encrypted pre-master secret will be rejected if any of the bytes created in the RSA encryption padding region are zero,[7] and the auditee cannot know this in advance. Therefore, we need to attempt with the reliable sites before we conduct the formal trail. That's why we conduct latency experiments on different reliable sites. And we find that the latency of attempt with the reliable sites does matter. So, to reduce the latency, we should reduce the number of trials, so we need to reduce the possibility of padding containing zero bytes.

Consider an extreme case, if we manipulate the padding part all with '\$x01$' so that the final product form must not contain zero bytes, then the trial and error step can naturally be omitted. Therefore, it can be found that the possibility of zero bytes depends on the certainty of the padding sequence. The higher the certainty of the padding sequence, the less likely it is to generate zero bytes. From analysis above we can draw: The more random bytes, the lower the certainty of the padding sequence, the more likely it is to generate zero bytes, and the more times you need to grab a reliable-sites attempt. Therefore, the best way to reduce this latency is to reduce random bytes.

After the above analysis, we roughly came up with a way to reduce latency. However, the method of reducing random bytes proposed may cause security problems. As we know, the more random bytes, the higher security.

This is easy to understand. For example, there is an attack type called short message attack.[11] If I know that the plaintext space on both sides of the encryption and decryption is a four-byte string, then I can compare all the correspondences by observing the plaintext corresponding ciphertext. Enumeration, you can know the plaintext corresponding to this ciphertext without a key. If the same plaintext is different every time the ciphertext is different, there is no way to achieve this kind of attack. It can be said that the more random bytes, the more ciphertext generated by the same plaintext, the more difficult it is to be attacked.

Then go back to our original proposition: reduce latency by reducing random bytes. By reducing random bytes to improve the success rate of generating padding sequences that do not contain zero bytes, the delay time of web page trial and error is reduced, but this will inevitably affect the security of RSA encryption and increase the probability of being attacked. Therefore, There is a trade-off between latency and encryption security (represented by random bytes).

To test this tradeoff, we conduct an experiment of latency and security. The figure is shown as Figure 7.

It can be seen from the figure that when random bytes is greater than 15, the average latency during interaction with the reliable site will exceed 1s. In our previous test on latency, we know that the average value of the reliable site detection stage should not exceed 1s. In order to ensure that the latency is within the acceptable range, and the number of random bytes reaches the maximum, we choose 15 as our system random bytes.

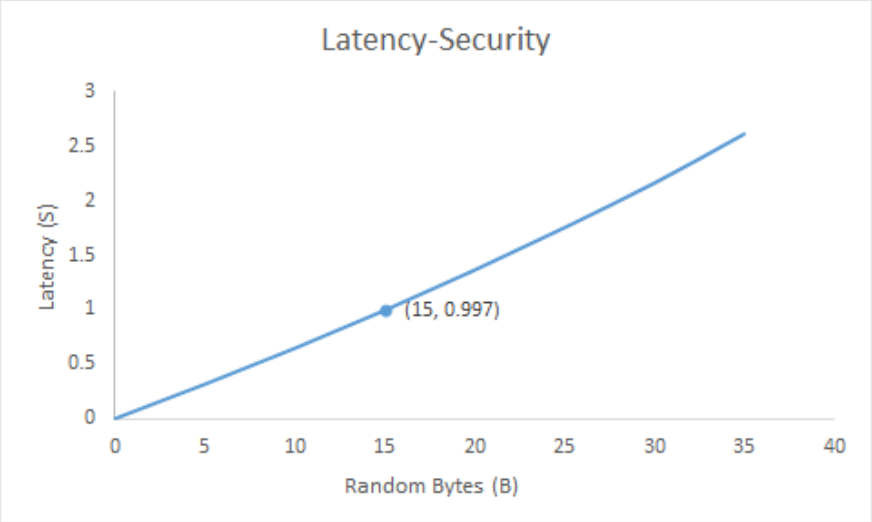


Figure 7: Latency-Security Tradeoff

5 Conclusion

5.1 Project Conclusion

We implemented the TLSNotary system by ourselves and wrote a web application to run the system. Also, we made 3 experiments separately on functional, latency and security. We have analyzed the latency and security problem of TLSNotary. As a result, we reached a trade-off between latency and security.

TLSNotary is a better solution because there is no need for the web server to make any changes. Our solution not only can be used to verify the data, also it can be applied in many situations, such as providing trusted external trigger conditions for smart contracts and trusted data for the blockchain.[12]

After studying and practicing throughout the semester, we not only have a deeper understanding of the OSI network model,[13] but also have a deep understanding of the main content of the TLS protocol.[14] When implementing the TLSNotary system, we get to know the process of the three handshake of TLS and encryption, deepening the understanding of RSA.[7] In addition, we learned new knowledge: RSA multiplication homomorphism[8] and pseudo-random function.[9] Both of these technologies are very important in the field of network and have a wide range of applications.

5.2 Group Work

During the project, each member actively participates in every meeting with TA, and finishes the group work together, such as writing proposals and midterm/final reports. They also have a significant contribution in playing different roles in the project.

- **Yiwen Tang** For the coding part, I am responsible for writing the main code of

Auditor. That is, using a socket to set up a connection with Auditee. Auditor also needs to deal with requests and send responses, including 3 requests. First, Auditor send the encrypted pre-master secret to Auditee. Next, Auditor generates Server Mac Key. At the last phase, when Auditor gets the encrypted data from Auditee, Auditor needs another signning-server to sign the data. After finishing the whole project, I also take part in doing experiments. We decide to conduct 3 experiments: functional test, latency test and security test. After we finish the experiments we analyse the cause of latency and find the tradeoff between the latency and security.

- **Yawei Zhang** For the coding part, I'm responsible for implementing the reviewer part of Auditee. (1) Verify TLS server's public key and certificate; (2) Verify signed proof from Auditor side: extract cert in DER form from the notarization file. Then, initialize a TLSNClientSession and extract cert, modulus and server name; (3) Verify commitment hash; (4) Decrypt html and check for mac errors. Response data and MAC code from Auditee will be checked by the MAC key from Auditor. Also, I'm responsible for building the frontend of TLSnotary web application to generate and review the proof file. Users can successfully generate a proof file by entering a legal url and open the requested webpage. Users can also verify a proof file in the review webpage.

- **Rui Zhang** For the coding part, I am responsible for writing code for signning-server. That is, listening to every authentication request from the Auditor. Then using socket connection to receive data from it, signing data with the server's private key. And replying back an encrypted message to the Auditor. Besides, I deployed the Auditor on AWS instance, in case enabling the Auditee and the Auditor to communicate with each other in public IP address. Also, I was responsible for building the backend of our web application using flask to generate and review the proof file. Thus, by inputting valid URLs, users could get generated proof files and stored in their local machine. They could also verify the proof files whenever he/she wants.

- **Jiayi Zhang** For the coding part, I'm responsible for implementing the notarize part of Auditee and the interface to call the notarize and review function with the lightweight framework Flask. The notarize function will complete the multiple interactions between Auditor and Auditee, Auditor and server to exchange session key and sign hash content, etc. The proof file for a certain URL will be produced. After finishing the implementation of the TLSNotary, I also take part in conducting experiments. Experiments from three aspects are conducted: functional, latency and security. I wrote multiple scripts, got the result data and analyzed the cause of latency and tradeoff between the latency and security accordingly.

6 Future Work

Currently, our algorithm improvement and implementations are based on TLS 1.0 and 1.1[14]. And our system is not compatible with TLS version $\geq$ 1.2[14]. We might modify and extend our modules' compatibility among higher versions of TLS protocols[14].

Malicious might collude with the server which hosts the Auditor. Because right now, the system administrator at Auditor has wide access permission, which might threaten the safety in the whole workflow. In the future, we could research to improve that. For example, we can change the password atomically after every booting up of our Auditor service.

Right now, our system can only support Server sites compatible with RSA asymmetric encryption algorithms[7] as well as MD5 and SHA1 hash algorithms, because the core of our Auditor lies in the multiplicative homomorphic property of RSA encryption[7]. In the future, we could work on other algorithms which could support more available Server sites.

References

[1] Rescorla, E. (2000). Http over TLS.

[2] Danezis, G. (2009). Traffic Analysis of the HTTP Protocol over TLS.

[3] Saito, T., Sekiguchi, K., & Hatsugai, R. (2008, September). Authentication Binding between TLS and HTTP. In International Conference on Network-Based Information Systems (pp. 252-262). Springer, Berlin, Heidelberg.

[4] Callegati, Franco, Walter Cerroni, and Marco Ramilli. "Man-in-the-Middle Attack to the HTTPS Protocol." IEEE Security & Privacy 7.1 (2009): 78-81.

[5] Szalachowski, P. (2018). Blockchain-based tls notary service. arXiv preprint arXiv:1804.00875.

[6] TLSnotary – a mechanism for independently audited https sessions. https://tlsnotary.org/TLSNotary.pdf, 10 Sept. 2014.

[7] Kaliski, B. (1998). PKCS# 1: RSA encryption version 1.5. RFC 2313, March.

[8] Morris, Liam. "Analysis of partially and fully homomorphic encryption." Rochester Institute of Technology (2013): 1-5.

[9] Ruehle, Michael. "Hash-based pseudo-random number generator." U.S. Patent Application No. 09/963,857.

[10] Tycksen Jr, Frank A., and Charles W. Jennings. "Digital Certificate." U.S. Patent No. 6,189,097. 13 Feb. 2001.

[11] Bleichenbacher, Daniel, Marc Joye, and Jean-Jacques Quisquater. "A new and optimal chosen-message attack on RSA-type cryptosystems." International Conference on Information and Communications Security. Springer, Berlin, Heidelberg, 1997.

[12] Christidis, Konstantinos, and Michael Devetsikiotis. "Blockchains and smart contracts for the internet of things." Ieee Access 4 (2016): 2292-2303.

[13] Day, John D., and Hubert Zimmermann. "The OSI reference model." Proceedings of the IEEE 71.12 (1983): 1334-1340.

[14] Dierks, T., & Allen, C. (1999). The TLS protocol version 1.0.