대리자를 통한 원격증명 검증 및 보안 연결 성립 방법

이경룡*, 조영필**, 유준승*, 백윤흥*
*서울대학교 전기정보공학부
**한양대학교 컴퓨터소프트웨어학부
반도체공동연구소
krlee@sor.snu.ac.kr, jsyou@sor.snu.ac.kr, ypcho@hanyang.ac.kr, ypaek@snu.ac.kr

Method for Delegating Remote Attestation Verification and Establishing a Secure Channel

Kyeong-Ryong Lee*, Yeong-Pil Cho**, Jun-seung You*, Yun-Heung Paek*
*Dept. of Electrical and Computer Engineering, Seoul National University
**Dept. of Computer Science, Hanyang University
Semiconductor Research Center (ISRC), Seoul National University

요 약

Trusted Execution Environment(TEE) is an execution environment provided by CPU hardware to gain guarantee that the execution context is as expected by the execution requester. Remote attestation of the execution context naturally arises from the concept of TEEs. Many implementations of TEEs use cryptographic remote attestation methods. Though the implementation of attestation may be simple, the implementation of verification may be very complex and heavy. By using a server delegating the verification process of attestation information, one may produce lightweight binaries that may verify peers and establish a secure channel with verified peers.

1. Introduction

Recently, due to massive needs of computing power for many applications such as artificial intelligence, the need for cloud computing has increased more than ever. Furthermore, cloud computing customers now want to verify whether the execution of applications was as expected and ensure confidentiality and integrity of data inside the cloud application.

When running an application within a trusted execution environment, many trusted execution environment implementations provide a remote attestation method for the application to demonstrate that the behavior of execution was as expected by a remote entity[1]. This methodology provides a way to prove to remote entities that the desired content has been executed without malicious interventions from third parties within the trusted execution environment. Furthermore, remote attestation usually comes in pair with secure channel establishment, in order to guarantee secure communication with the verified application. In trusted execution environments such as Intel SGX, verifying the information provided by the application inside the trusted execution is very complex[2], and the code size and complexity of the implementation to verify the received information is very large, which can lead to inefficiency when multiple applications running in the cloud need to verify and establish a secure channel with each other. This paper introduces how to optimize this inefficiency and generate a lightweight application which can establish a secure TLS channel with a verified application inside a trusted execution environment by utilizing a remote attestation verification server ran by the owner of the application which performs the actual verification.

2. Background

2.1 Remote Attestation

Trusted Execution Environment(TEE) is a special execution environment provided by the CPU hardware in order to guarantee several security properties such as confidentiality, integrity. The security properties of applications running inside the trusted execution environment are still guaranteed even if the operating system is compromised by an attacker. TEEs are provided by several CPU vendors such as Intel Software Guard Extensions(Intel SGX), AMD Secure Processor, ARM TrustZone.

The problem of verifying whether the execution of applications inside the TEE naturally emerges since the trusted execution environment is targeted for use by customers who do not own the entire platform. This is because loading the executable inside the trusted execution environment is performed by the platform owner rather than by the customer. In cryptography based remote attestation,

the application inside the TEE encrypts and transmits information about the execution environment and execution context to a remote entity. The remote entity verifies the execution of application inside the trusted execution environment by decrypting and verifying the information.

2.2 Transport Layer Security

Transport Layer Security(TLS) is a method to establish a secure channel over an unsecure channel for communication[3]. It usually utilizes certificates to prove the identity of the entity at the other endpoint of the communication channel.

By using TLS, the followings are guaranteed.

a. A secure channel with a shared symmetric key is established.

b. Through the channel, data confidentiality and data integrity is guaranteed.

c. The entity at the other endpoint of the channel is the owner of the certificate provided through the TLS session.

The third guarantee is very important when identifying the entity at the other side of the channel. In this paper, we will use this fact to guarantee the correctness of the method.

3. Assumptions on Verification Delegation Method

In order to delegate verification of remote attestation attester data, we assume the following.

(a) An authority exists such that it can verify or delegate the verification of remote attestation data.
(b) The challenger and verifier has the ability to attest itself to the authority using remote attestation.
(c) The challenger and verifier has the ability to use transport layer security.

Assumption (a) can be satisfied by the owner of the applications by running a server that verifies the information transmitted by applications inside TEEs. Assumption (b) may be available by implementing the remote attestation method that the CPU vendor provided. Assumption (c) is used in order to establish a secure TLS channel with the verified entity.

4. Erroneous Verification Delegation Method and its weaknesses

One may propose the following procedures for delegating the verification to a third entity that is a trusted authority.

(1) The challenger successfully attests itself to the authority using remote attestation.
(2) The authority generates a random token to the challenger and stores the token in its memory.
(3) The challenger and verifier establishes a TLS connection.
(4) The challenger passes the token to the verifier.
(5) The verifier checks if the token is valid by checking through the authority.
(6) If successful, the verifier sends an OK message to the challenger.
(7) The challenger and verifier exchanges data assuming a secure channel is established.

However, we claim that these methods are both vulnerable to man-in-the-middle attacks by the following scenarios. In case of erroneous method 1, the following scenario works.

(1) The challenger successfully attests itself to the authority using remote attestation.
(2) The authority generates a random token to the challenger and stores the token in its memory.
(3) The man-in-the-middle attacker establishes a TLS connection with the challenger and the attester.
(4) The challenger passes the token to the attacker, where its intention was to pass it to the verifier.
(5) The attacker passes the token to the verifier, pretending that the attacker is the challenger.
(6) The verifier checks if the token is valid by checking through the authority, and receives an OK message because it was a legitimate token generated to the legitimate challenger.
(7) The verifier sends an OK message to the attacker, where its intention was to pass it to the challenger.
(8) The attacker sends the OK message to the challenger, pretending that the attacker is the verifier.
(9) The challenger and verifier passes the data through the channel, where their intention is to pass the data directly and confidentially to each other, but is actually passed to the attacker. The attacker reads the data and again passes the data to the intended original destination as if it does not exist. In this case, data confidentiality is broken.

In this case, a man-in-the-middle may hijack information flowing through both directions, and therefore the confidentiality guarantee is obviously broken. Furthermore, since the man-in-the-middle fully mediates transmission of information, it may even inject or omit some information transmitted between the verified entities. In this case, the integrity guarantee is also broken.

5. Verification Delegation Method

We propose the following procedures for delegating the verification to a third entity that is a trusted authority.

(1) The challenger successfully attests itself to the authority using remote attestation.
(2) The challenger generates a TLS certificate. A self-signed certificate is allowed.
(3) The challenger registers the generated certificate to the authority as a verified certificate.
(4) The challenger and verifier establishes a TLS connection, where the challenger uses the certificate to establish the TLS connection.
(5) The verifier checks if the certificate is valid by checking through the authority.
(6) If successful, the verifier sends an OK message to the challenger.
(7) The challenger and verifier exchanges data assuming a secure channel is established.

The following method correctness is guaranteed due to the following. Since the identity of the authority can be verified through pure TLS features, the challenger and verifier may trust information from the authority. After step (1), the authority successfully verified the challenger, so the verification result must be securely passed to the verifier. After step (3), the challenger may attest its identity through the certificate generated by itself. Although it is self-signed, the certificate still guarantees that the certificate owner is the challenger. This guarantee is sufficient to verify the challenger. At step (5), the verifier can check if the certificate owner is the expected entity by querying to the authority. Since the authority is trusted, and the authority knows if the certificate is legitimate, the verifier trusts the verification information by the authority. Therefore, after step (5), the verifier has successfully verified information about the challenger, and the secure TLS channel with the challenger has already been established at step (4).

The method can be performed symmetrically, meaning that the two entities can safely verify the other entity by acting like both a challenger and verifier. This can be done by the following steps. Each entity performs steps (1), (2), (3) assuming itself is the challenger. Then, at step (4), both entities use its certificate to establish a TLS channel. Then, both entities perform steps (5), (6) using each other’s certificate.

6. Conclusion

In this paper, we have introduced a method to generate a lightweight application that can be ran inside a trusted execution environment without including any specific implementations related to verifying remote attestation information of others using a known trusted authority, ran by the application owner.

7. Acknowledgement

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2020-0-00325, Traceability Assuarance Technology Development for Full Lifecycle Data Safety of Cloud Edge), the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (NRF-2020R1A2B5B03095204), the BK21 FOUR program of the Education and Research Program for Future ICT Pioneers, Seoul National University in 2021, and Inter-University Semiconductor Research Center (ISRC).

8. References

[1] Sfyrakis, Ioannis, and Thomas Gross. "A survey on hardware approaches for remote attestation in network infrastructures." arXiv preprint arXiv:2005.12453, 2020.
[2] Intel Corporation, Intel® Software Guard Extensions SDK Developer Reference for Linux OS, 2016.
[3] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.