Side Channel Resistant Crypto Library for TEE

Sun Yongkang
11911409

Wang Chenyu
11911104

I. Abstract

In order to protect the running environment of security-sensitive programs in computing devices, researchers proposed TEE technology, which provides a safe running environment for security-sensitive programs isolated from general computing environment by isolating hardware and software. Side-channel attacks based on information obtained from the physical implementation of a crypto system. For example, time information, power consumption, electromagnetic leakage or even sound can provide additional sources of information that can be used to further hack the system. The TEE architecture only provides an isolation mechanism but can not resist this type of emerging software side-channel attacks. In this project we purpose a side channel attack resistible crypto library by prune and modify an existed open source one. Our library will contain most commonly used crypto algorithm used in developing an OS (ECC, AES, etc.).

II. Motivation and Introduction

A. Research Purpose

As a part of the TEE-OS group, our final purpose is to build a complete secured operating system in *TEE* (Trusted Execution Environment). We break it into several part. And our group is responsible for creating a *side channel resistant crypto library* with *RUST* programming language, used in our final TEE-OS.

B. Research Significance

a) Why need an OS in TEE: Firstly, "*trusted execution environment (TEE)* is a secure area of a main processor. It guarantees code and data loaded inside to be protected with respect to confidentiality and integrity. A TEE as an isolated execution environment provides security features such as isolated execution, integrity of applications executing with the TEE, along with confidentiality of their assets." according to wikipedia (1). As a conclusion, TEE can promise us:

- *Data Confidentiality*
- *Data Integrity*
- *Code Integrity*

Secondly, in order to run applications in TEE, we have several ways as shown in figure 1. It is a trade off between security and usability. Beside these three methods, we plan to implement a mini-OS inside TEE. Through this way, we can have a balance between security and usability.

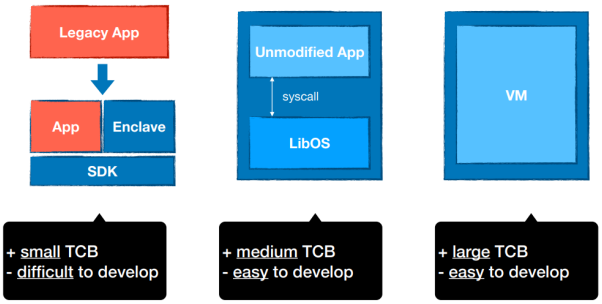

Fig. 1: TEE Development Model

b) Why Rust: Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety. *Rust* can also ensure memory safety through features like Ownership and Borrowing.

c) Why OS need a crypto library: Cryptography can be used for the following purposes:

- Confidentiality: Prevents the user's identity or data from being read.
- Data integrity: Preventing data from being changed.
- Authentication: Ensuring that data is sent from a particular party.

For example, an OS in TEE need to encrypt it's data in cache to store it in memory, and also some TEE have the ability to do authentication remotely which need asymmetric cryptographic algorithm like *RSA*.

d) Why need side channel attack resistance: Many widely used encryption algorithms have been hacked through side channel attack (SCA). For example:

- Osvik, Shamir, Tromer, 2006: Recover AES-256 secret key of Linux's dmcrypt in just 65 ms, according to Osvik, Shamir and Tromer, 2006 (2)

- AlFardan, Paterson, 2013: “Lucky13” recovers plaintext of CBC-mode encryption in pretty much all TLS implementations, accoeding to AlFardan and Paterson, 2013(3)
- Yarom, Falkner, 2014: Attack against RSA-2048 in GnuPG 1.4.13: “On average, the attack is able to recover 96.7% of the bits of the secret key by observing a single signature or decryption round.”, accoeding to Yarom and Falkner, 2014(4)
- Benger, van de Pol, Smart, Yarom, 2014: “reasonable level of success in recovering the secret key” for OpenSSL ECDSA using secp256k1 “with as little as 200 signatures”, accoeding to Benger, van de Pol, Smart and Yarom, 2014 (5)

Also, although TEE is well secured, it can do nothing to avoid SCA unless programmer modifying it in the source code level.

III. A PROPOSED APPROACH

A. *research method*

Our research is based on three steps:

a) First Step: To begin our research, we have to choose and learn to use a current existing TEE, in order to test our crypto algorithms inside it.

As a mature and most widely used choice, we choose *Intel SGX*. ”Intel’s Software Guard Extensions (SGX) is a set of extensions to the Intel architecture that aims to provide integrity and confidentiality guarantees to security sensitive computation performed on a computer where all the privileged software (kernel, hypervisor and etc.) is potentially malicious.” according to an article of general description of SGX (6). Below in figure 2 is the basic model of SGX. Further description of SGX can be found in the article (6).

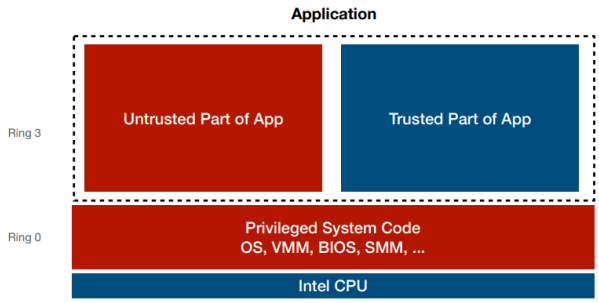

Fig. 2: SGX

Further, We choose to use a SGX development tool, named *TEACLAVE*, which is build up with three layer:

- At the bottom is the Intel SGX SDK implemented using C/C++ and assembly.
- The middle layer is Rust’s FFI (Foreign Function Interfaces) to C/C++.
- At the top is the Teaclave SGX SDK.

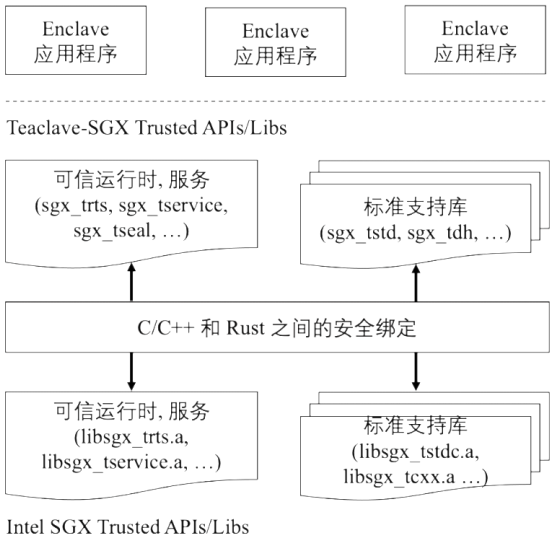

Fig. 3: teaclave

With using these tool, we could decelop based on only the topmost Teaclave SGX SDK. Further description can be found in TEACLAVE’s website (7)

b) Second Step: Read and learn current existing Rust crypto library (8), which is a open source project contains most of the cryptographic algorithms written in pure Rust.

This library contains algorithms like *AEADs*, *hashes*, *RSA* and etc. We will prune some of these algorithms and test to use it in Intel-SGX.

c) Third Step: Our goal is to modify the existing crypto algorithms in order to make it side channel attacks resist. So, our third step is to apply different kind of side channel attack to the original code. And find the weak point of the code against our attack.

A list of attack can be conducted to RSA, DSA, and Diffie-Hellman Key Exchange is concluded by an article by ”Bundesamt für Sicherheit in der Informationstechnik”, 2013 (9), which gives an overview of relevant literature about side-channel attacks on implementations of either integer factorization cryptography or discrete logarithm cryptography.

More attacks about ECC can be found in an article by ”Federal Office for Information Security”, 2016 (10). This document provides a guideline for security evaluators to test implementations of elliptic-curve cryptography over Fp for resistance against side-channel attacks

with high attack potential according to version 3.1 of the Common Criteria (CC).

d) Fourth Step: After applying attacks to original code in the library, we have to modify the code against these attacks based on the data obtained. The above two articles not only contains different attacks but also includes methods to against each attack. So, our fourth step is to study the article and modify our code according to it.

After modified, we plan to attack it again to check the effectiveness of our defend.

B. research content

Our topic is mostly based on research. So, most of our research contents is about to read articles and recurrence it. Our basic research content is to learn and prune an open sources crypto library. Then test it's usability inside TEE. Our core research content is to investigate current side channel attacks, and ways to defend them. Then apply it to our crypto library.

Since learning and modify a new algorithms takes up great time, so our team focus on mainly two algorithms:

- RSA
- ECDSA

IV. Research conditions and possible problems

A. Research conditions

a) Team structure and division of the work: Our team is made up of two students, SUN Yongkang and WANG Chenyu. Our basic division of work is that SUN is responsible for the whole study of algorithm RSA and WNAG focus on ECDSA. Through this way, although we may both study most of the same things, we could help each other and move the whole project forward in discussion, because of the parallel part has a large proportion. Our team is instructed by our tutor ZHANG Yinqian and teacher SUN mingshen.

b) Resources we have: There are several resources we currently have, and will need to use during the research:

- Intel CPU with SGX function
- Platform for building SGX software: TEACLAVE
- An existing Crypto library: https://github.com/orgs/RustCrypto/repositories
- Some practical articles about side channel attacks and defence.

B. possible problems

Because of our topic is more "research" than "practical", so there exists following problem currently:

- 1. The learning content is heavy, and a lot of practical content needs to be built on the basis of learning. As a result, it's harder to measure efficiency and time commitment.
- 2. According some articles, side channel attack is very diverse. "Side-channel attacks consist either in passive attacks on implementations, i.e. disclosing secret data by analyzing physical observable during the computation, or active attacks, which perturb the computation to obtain information about the secret data. " according to article(9). The document we use shall be considered as a guideline rather than a checklist containing all possible requirements of a vulnerability assessment of a Target Of Evaluation (TOE). A TOE will have its own implementation of the employed cryptographic system and the evaluator is responsible for adapting and extending the side-channel attacks treated in this guideline according to article(9). As a result, it's hard for us to cover all SCA and unavailable for us to value its ability against SCA based on only a small part of these attacks.
- 3. Existing tools or suites used at different stages are not applicable. Because on each stage of our research, we found insufficient source of knowledge and no well-implemented test tools. So, we have to develop most of the tools we have to use by ourselves according to the articles.

C. innovation point

- No complete SCA-proof cryptography suite for TEE-OS is currently available on the market. So our work may not only be used in our TEE-OS.
- With the research continue, we may develop some tools or implement some algorithms to test if one's code is SCA-secured.

V. planned results

We planed to end with a SCA-secured crypto library, along with some testing methods or tools of SCA.

VI. Timeline

- Currently: We have learned some basic concepts of TEE and Rust programming language. Also we learned basic usage of Intel SGX, and successfully build up the environment our research needs.

- Until Week 8: Finish learning existing Rust crypto library. Prune and try to use this existing library inside TEE (Intel SGX).
- Until Week 10: Finish learning background knowledge of Side-Channel Attacks. Trying to find ways to perform a side channel attacks to these existed library.
- Until Week 12: Finish learning methods to resist side channel attacks, and try to modify the most commonly used two algorithms: RSA and ECDSA.
- Until Week 13: Test and Debug the ability of RSA and ECDSA resisting SAC.
- Until Week 15: Complete other algorithms. Test the performance and ability of our modified library.

REFERENCES

[1] trusted execution environment - wikipedia$_2$021 (2021).
URL https://en.wikipedia.org/wiki/Trusted_execution_environment

[2] E. Tromer, D. A. Osvik, A. Shamir, Efficient cache attacks on aes, and countermeasures, Journal of Cryptology 23 (1) (2009) 37–71. doi:10.1007/s00145-009-9049-y.

[3] N. J. AlFardan, K. G. Paterson, Lucky thirteen: Breaking the tls and dtls record protocols.
URL http://www.isg.rhul.ac.uk/tls/Lucky13.html#Team

[4] Yarom, Falkner, Flush + reload: a high resolution, low noise, l3 cache side-channel attack.
URL http://eprint.iacr.org/2013/448/

[5] Benger, van de Pol, Smart, Yarom, "ooh aah... just a little bit": A small amount of side channel can go a long way.
URL http://eprint.iacr.org/2014/161/

[6] V. Costan, S. Devadas, Intel sgx explained.
URL https://eprint.iacr.org/2016/086.pdf

[7] blog of tool teaclave (2021).
URL https://teaclave.apache.org/blog/2021-08-25-developing-sgx-application-with-teaclave-sgx-sdk/

[8] Rust crypto library (2021).
URL https://github.com/RustCrypto

[9] Minimum requirements for evaluating side-channel attack resistance of rsa, dsa and diffie-hellman key exchange implementations (2013).
URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_46_BSI_guidelines_SCA_RSA_V1_0_e_pdf.pdf?__blob=publicationFile&v=1

[10] Minimum requirements for evaluating side-channel attack resistance of elliptic curve implementations (2016).
URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_46_ECCGuide_e_pdf.pdf?__blob=publicationFile&v=1