Fidelius: A Blockchain and SGX Based Data Collaboration System

Abstract

We study data collaboration, where a data analyzer demand the result of an analysis program with specific data as input, which is owned by a data provider. To keep privacy and sustain data value, raw data not being revealed is required. "Data not out of domain"— to let data provider execute analysis program and return the result to data analyzer— is a smart solution. To realize, the main challenge turns to be the reliability of analysis result provided by data provider. To address this issue, we give a data collaboration system based on Intel SGX and blockchain, called Fidelius, such that trusted execution environment of analysis program is guaranteed. Compared to previous solution, our system does not rely on Intel's remote attestation for data transmission between trusted environments of different platforms, which requires WAN environment to interact with Intel server. Experimental results show that ...

1 Introduction

In the era of big data, the ability of human beings to obtain, manage and use data has been improved unprecedentedly, and the value of data has been widely concerned. Compared with the traditional factors of production, data has a very strong network effect. When a variety of data from different dimensions and different sources are combined, the inherent social and economic value can reach the effect of $1 + 1 > 2$. For example, combination of medical data and location information can promptly control the scale of infectious diseases.

To facilitate the realization of data value appreciation, data collaboration is proposed, which means cross-domain data processing with multiple participation. Usually, there are two roles in data collaboration: data provider and data user. The former provides the data and the later uses the data, for purposes range from practical statistics to theoretical researches. More often, data trading platforms, working as an intermediate between data providers and data users, have proliferated in recent years, including ...

A key issue for launching data collaboration is privacy protection, for the reason that raw data usually contains personal/corporate sensitive information that is illegal or reluctant to be exposed. Whereas in reality it is hard to avoid privacy leakage or loss of data value: though variants solutions [1] are proposed for ensuring data safety during transmission, there is no guarantee that data user or intermediate resells purchased raw data, without original data provider's knowledge and approval.

Follow from a series of literature, we aim to fundamentally eliminate the possibility of reselling, in accordance with the fact that for many data user, instead of raw data, their demands are data-driven decision-making based on data analysis. So the core idea is to transfer the role of data processing from data user to data provider, who only provide the analysis result to data user. Compare to raw data, the analysis result is usually task-specific and thus contains less private information, where the effect of being leakage is negligible[1].

However, trust issues arise when data analyzing task is transferred from data user to date provider. Typically:

- The correctness of analysis result: the result that data provider provides indeed comes from analysis program, which is not altered by data provider.
- The consistency of raw data: the input of analysis program is indeed the raw data that data provider claims.

1.1 Background

The ability to address trust issues is the key to launch data cooperation. Past works, typically exampled with [], make the use of two main techniques, blockchain and SGX, the trusted execution environment from Intel, which achieves two basic aims that a SGX program, called an enclave, i) can not be altered and ii) the intermediate variable can not be accessed. However the main drawback of past solutions is the

[1] We will introduce static checking in later sections to ensure that analysis program does not directly or indirectly expose raw data.

requirement of Intel's remote attestation, which is functionally contained in SGX for data transmission between cross-platform enclaves. The process of remote attestation needs an interaction to Intel official server, which means that both data provider and data user must be in WAN environment for a successful data collaboration. In contrast, for highly secure and confidential government agencies and enterprises, their production environment can only provide LAN connection. This limitation prompts a new solution in this paper for data transmission between cross-platform enclaves independent of WAN environment.

1.2 Our Contribution

Our contribution is from two aspects: first we provide a complete solution for data collaboration to address the trust issue. The core idea is to pack analysis program into an enclave, so nobody has the ability to alter its code. A piece of code verifying the hash of input data is inserted at the start of analysis program to check the consistency of raw data; another piece of code signing analysis result by data user's private key is inserted at the end of analysis program to ensure the correctness of analysis result, whereafter the signature is checked by blockchain's smart contract. Blockchain works as a trusted third party in the while processing for transfer messages, e.g., the enclave binaries, and store information, e.g. the hashes, public keys.

Second we provide a new method for realizing data transmission between cross-platform enclaves, i.e. remote attestation of SGX, independent of WAN environment, via blockchain and simple cryptography tools. The core idea is to establish an open-source enclave, which generates an asymmetric key pair through SGX's own functionality. It is guaranteed that the private key is only accessible by the open-source enclave, while corresponding public key is authorized by Fidelius admin and stored on blockchain. Therefore, one can trustfully encrypt private data by the public key and then send it to the open-source enclave.

The combination of these two contributions gives a complete solution of data collaboration independent of WAN environment. Meanwhile each of the contribution can be separately used as subroutine. For example, our data collaboration system can be directly used when WAN environment are allowed and Intel's servers are trusted; our replacement for remote attestation can be used in situations other than data collaboration when WAN environment are not allowed.

The roadmap of this paper is as follows: in section 2 we introduce preliminaries for basic technique and notations. In section 3 we introduce the basic framework of data collaboration system. In section 4 we introduce the implementation for cross-platform data transmission between enclaves. In section 5 we introduce the complete Fidelius system by combining all together. Experiment results and conclusion are shown in section 6 and 7.

2 Preliminary

2.1 Basic Techniques

2.1.1 SGX and Enclave

SGX (Software Guard Extensions), the trusted execution environment, protects the execution of programs at hardware level. It classifies a section of protect memory, called trusted environment, while all outside environment is regarded as untrusted. The CPU circuit (SGX compatible) ensures that every access to the protected memory from untrusted environment are prohibited. A program (more precisely, dynamic library) deployed in SGX trusted environment is called an enclave. An enclave can not interact with untrusted environment unless specific methods are declared (called Ocall or Ecall method).

To deploy an enclave, besides hardware compatibility and program codes, the following files are necessary:

- A signature to program codes' hash, where the private key used for signing is a RSA-3072 key and corresponding public key should be in Intel's whitelist.
- A license issued by Intel, to authorize that an RSA-3072 public key is in whitelist. An enclave developer should first send the RSA-3072 public key to Intel, applying to join whitelist. After that Intel adds the public key to whitelist and return the license (bind with the public key) to developer.

So an enclave installation package includes the program, its hash, the signature to the hash, the RSA-3072 public key and the license. Note that the RSA-3072 private key are not included. When the package is ready, SGX checks whether the program hash and its signature are valid (via RSA-3072 public key) and whether the RSA-3072 public key is in the whitelist (via the license). It is notable that this checking process does not require WAN environment.

For user experience, Fidelius admin offers uniform installation package, using Fidelius official RSA key which is in Intel's whitelist. So users do not need to send applications to Intel. Moreover, Fidelius admin offers interfaces that statically check the input programs and sign their hashes using the official RSA-3072 private key, with the private key keeping unrevealed, via open source enclave. So users do not need to interact with Fidelius admin to establish a new collaboration task either.

2.1.2 Key Methods In SGX

In the following we introduce key methods and operations contained in an enclave.

- *sgx_get_key*(): return a symmetric key based on the enclave's hash, the CPU's model and other configuration files, which is unique to each enclave. The symmetric

key is inaccessible outside the enclave unless exporting methods are declared.

- *sgx_ecc*256*_create_key_pair*() returns an ECC-256 asymmetric key pair, using SGX's self-contained random number generator. The asymmetric key pair can be reused by the seal operation, see below.

- To save private messages from an enclave to local storage, we need the seal operation, that uses the enclave's symmetric key to encrypt the message and saves it locally. It is ensured that the user is inaccessible to the plaintext of the sealed message since the symmetric key is unknown. To unseal the message, user sends it to the enclave throught specifically declared (ecall) method and then the enclave decrypts it using the enclave's symmetric key.

SGX supports two kinds of communications between two enclaves, local attestation and remote attestation. The former is inter-platform and the later is cross-platform, but the remote attestation requires WAN environment and interactions with Intel official server. So we replace it with our new solution. To execute local attestation between two enclaves, each enclave can selectively confirm the hash value or the signer of the interactive enclave.

2.1.3 Blockchain

Blockchain, more precisely permissioned blockchain, used in Fidelius can be regarded as a trusted and no-failure third party for data transmission, data storage and payment intermediation, as traditional mediator suffers the risk of single point failure, information being tampered, and the high cost of building P2P private lines. Permissioned blockchain is allowed to be deployed in LAN environment so each user can get access to on-chain information without the WAN requirement.

Smart contract of blockchain is an on-chain program where its code and execution process are totally transparent. Fidelius uses smart contract verifying the signature to analysis result given by data provider, which means that everyone can trusted the success of a data collaboration as long as the verification is passed.

2.2 Roles

We then introduce the roles and their tasks in data collaboration process.

- Data Provider (DP), who is the only owner of raw data. At the beginning, DP publishes the metadata on blockchain, including the hash and necessary introductions of raw data. After DP receives a requirement of data analysis task, it is DP's obligation to run analysis enclave and return analysis result to blockchain. The correctness of the metadata are endorsed by DP's credit, for example the number of successful collaborations in history.

- Data Analyzer (DA, the same role as data user in introduction section), the one requires analysis result of analysis program whose input is the designated raw data. DA selects the raw data he/she wants by searching the metadata on blockchain and then starts a data collaboration. It is DA's obligation to write analysis program in a enclave format, which must include verifying the raw data's hash, signing the results and other methods necessary for data transmission.

  DA and DP are collectively called users.

- Blockchain, works as a trusted and no-failure third party, for data transmission and storage. In particular, smart contract verifies the correctness of the signature of analysis result and mediates the payment of data collaboration.

- Fidelius admin (FA), works as a trusted party used in setup process. FA's job is to install open-source enclaves for a new user, and authorize public keys generated by open-source enclaves. Note that the reliance of FA's honesty is one-time — when setup process is done users no longer need FA's participation.

2.3 Environment and Assumption

The only party that each user can get access to is blockchain, which is assumed to be trusted and no-failure. The interaction to Intel server is not allowed during the whole process for all users.

FA is assumed to be trusted during setup process. The installation package of open-source enclave is assume to be complete and generated public keys are correctly sent to blockchain and authorized by FA (for detail see following sections).

SGX's hardware functionality is assumed to be complete as Intel claimed, i.e. the code in an enclave can not be altered and the value of inside variables are inaccessible (by directly getting access to the memory). It is notable that the reliance to Intel is also one-time as we do not need to interact with Intel server as long as setup process is done, in contrast to previous works using remote attestation which requires Intel server to be honest and work normally during each data collaboration task.

Users are allowed to be dishonest, which means that all users will attempt to obtain private keys or raw data. In particular, DA and DP can collude to improve DP's credit, for example, DA can give the key that signs analysis result to DP, so that DP can arbitrarily generate outputs that pass the verification of smart contract, without the execution of analysis program. To address this issue, our solution is to let DA can not get access to the private key for signing the result either.

It is notable that, in this paper the measure criterion of a success data collaboration is the correct execution of analysis program given by DA. So the collusion that DA provides very simple analysis program to let DP easily improve credit is allowed. In section 5 we introduce static checking to suppress this collusion.

Finally, we assume the existence of collation-resistant hash function and secure signature and encryption scheme. In particular, our signatures are associated with *nonce* so that a vaild signature (m, sig_m) can not be given by an adversary that does not have the private key (historical signatures are regarded as invalid).

3 Basic Framework

In this section we briefly introduce our data collaboration framework. For simplicity, we assume remote attestation is allowed just in this section. We also assume that DA can apply a signature to the hash of the enclave that contains analysis program.

The notations are as follows:

- *raw_data*, the raw data of the data collaboration, only owned by DP.
- $hash_{raw\_data}$, the hash value of raw data, on-chain information as a part of metadata given by DP and assumed to be correct.
- $H()$, the collation-resistant hash function.
- *analysis_program*, the program for computing the analysis result, in which *raw_data* is the input and *analysis_result* records the output.
- sk_{result}, the secret key to sign analysis result, only owned by DA.
- $sign(sk, msg)$, the signature function, using secret key *sk* to sign *msg*.
- pk_{result}, the public key to verify analysis results, on-chain information.
- $verify(msg, sign_{msg}, pk)$, using public key *pk* to check if $sign_{msg}$ is the valid signature of *msg*.

Analysis enclave is constructed as follows in Algorithm 1. It is notable that DP can not directly hardcode sk_{result} into analysis enclave, since the code of an enclave is not secret. The constant $hash_{raw\_data}$ is hardcoded.

The smart contract that checking the output of DP works as follows in Algorithm 2.

To sum up,

- DA sends purchase token to verification contract. DA then writes *analysis_enclave*, and applies the signature of its hash and the license from FA; then packs them together to get installation package of *analysis_enclave* and sent its binary to blockchain.
- DP downloads the *analysis_enclave* from blockchain and deploys it locally.
- DA puts the sk_{result} into a new enclave and send it to DP side *analysis_enclave* via remote attestation. The new enclave should check the hash of interactive enclave (equals to $H(analysis\_enclave)$).
- DP executes *analysis_enclave* and sent the output to blockchain.
- Verification contract is executed to verify the output provided by DP. If returned result is true, then data collaboration is success and the smart contract transmits the frozen token (given by DA) to DP, otherwise data collaboration fails and the smart contract returns the frozen token to DA.

This is the process sketch of Fidelius' data collaboration. According to our assumptions, if the signature of analysis result is valid, it is ensured that analysis result is DA's desired one.

Algorithm 1: Analysis Enclave

Input: *raw_data*
if $H(raw\_data) \neq hash_{raw\_data}$ **then**
  throw: "Invalid raw data";
Run *analysis_program* to get *analysis_result*;
Receive sk_{result} via remote attestation;
$sign_{result} \leftarrow sign(sk_{result}, analysis\_result)$;
return $(analysis\_result, sign_{result})$ *and sent it to blockchain*;

Algorithm 2: Verification Contract

Input: $analysis\_result, sign_{result}, pk_{result}$
if $verify(analysis\_result, sign_{result}, pk_{result})$ **then**
  return *True*;
else
  return *False*;

4 Replacement of Remote Attestation

In this section we introduce our new method that realizes data transmission between cross-platform enclaves. Specifically, we consider the situation of data collaboration in last section where DA wants to send a private key (say, sk_{result}) into DP's enclave (say, *analysis_enclave*). To be more general, we let DA can not get access to the private key either.

The key technique we use is open-source enclave whose codes and hashes are completely public. During its setup process, it calls *sgx_ecc*256*_create_key_pair*() to generate an ECC-256 key pair in it, of which the public key is sent to blockchain and authorized by FA[2], and the private key is sealed locally. Since everyone can check that open-source enclaves do not contain malicious methods, it is safe to use authorized public keys to encrypt messages. We assume the correctness of the on-chain public keys and its authorization by trusting FA during setup process.

The notations are as follows:

- *oenclave_a*, the open-source enclave that generated the private key to be sent, deployed on DA's platform.
- *oenclave_p*, the open-source enclave that temporarily receives the private key, deployed on DP's platform.
- *analysis_enclave*, the enclave that finally receives the private key (for signing analysis result in data collaboration). We assume that *enclave_analysis* does not reveal the value of the private key (It is guaranteed by Fidelius' static analysis. See next section).
- *skp*, *ska*, the secret keys generate by *oenclave_p*, *oenclave_a* respectively. In setup process, they are generated by calling *sgx_ecc*256*_create_key_pair*(), and sealed locally. In the following we simply write the code "unseal" when they are used.

  Moreover, *ska* is the private key to be sent.
- *pkp*, *pka*, the public keys generate by *oenclave_p*, *oenclave_a* respectively, where they and their authorization are on blockchain.

It is notable that the codes and the hashes of *oenclave_p* and *oenclave_a* are independent of users and tasks.

Open-source enclave *oenclave_a* works as follows in Algorithm 3.

Algorithm 3: *oenclave_a*

Input: *pkp* and its authorization
Unseal *ska*;
if *verification of pkp's authorization fails* **then**
  return *"Invalid public key"*;
$encp\_ska \leftarrow Encrypt(pkp, ska)$;
return *encp_ska and sent it to blockchain*;

Open-source enclave *oenclave_p* works as follows in Algorithm 4.

[2]The authorization process is that FA uses Fidelius official private key to sign the content. Fidelius official public key is officially public so that everyone can verify the authorization. Only public keys generated from a open-source enclave can be authorized.

Algorithm 4: *oenclave_p*

Input: *encp_ska*
Unseal *skp*;
$ska \leftarrow Decrypt(skp, encp\_ska)$;
Establish local attestation to *analysis_enclave*:
begin
  Confirm objective signer: FA;
  Send *ska* to *analysis_enclave*;

The confirmation of the signer ensures that DP can not import *ska* to a malicious enclave that signed by his/herself. DP can import *ska* to historical enclaves signed by FA but i) the signature of analysis result will be invalid (*nonce* is introduced) so the checking by smart contract can not pass and ii) all enclaves signed by FA are guaranteed to contain no malicious method that reveals private keys. So DP has no incentive to import *ska* to other enclaves other than *analysis_enclave*.

Finally, *analysis_enclave* establishes local attestation to *oenclave_p* to receive *ska*, confirming the objective hash to be the hash of *oenclave_p* (hardcode).

So our replacement of local attestation realizes the cross-platform private data transmission.

5 Fidelius Data Collaboration System

In this section we introduce Fidelius, our final system for data collaboration. Compared to the brief one in section 3, it makes the following improvement:

- The replacement of remote attestation is compatible to the network environment assumption in this paper.
- The collation of DA and DP to click farming are prevented.
- The interface to sign analysis enclave hash is implemented, so that DA does not need to apply to Fidelius server for each data collaboration.
- Static analysis is introduced to prevent analysis program from revealing raw data and private keys.

The notations are as follows:

- *oenclave_sign*, the open-source enclave that statically checks the analysis enclave and sign its hash, deployed on DA's platform.
- *pksign*, *sksign*, the ECC-256 key pair generated by *oenclave_sign*.
- *skRSA*, the RSA-3027 key that signs the hash of an enclave to deploy, own only by FA.
- *pkRSA*, the correspond public key above.

- $oenclave\_a, oenclave\_p, skp, ska, pkp, pka$ are the same as last section.

5.1 Setup Process

The setup process works as follows:

- Open-source enclaves *oenclave_a*, *oenclave_p* are installed in the same way as last section, where the private keys ska, skp are locally sealed and the public keys pka, pkp are sent to blockchain associated with their authorization.
- The setup process of *oenclave_sign* are introduced in Algorithm 5, where the ECC-256 key pair $sksign, pksign$ is generated. Then,
  - FA sends *encsign_skRSA*, encrypted *skRSA* by *pksign*, to blockchain.
  - *oenclave_sign* inputs *encsign_skRSA* and decrypts it via *sksign*
  - *oenclave_sign* seals decrypted *skRSA* locally.

It is equivalent that FA establish a remote attestation to *oenclave_sign* to send *skRSA*.

Algorithm 5: *oenclave_sign* (setup process)

Run $sgx\_ecc256\_create\_key\_pair()$ to get $sksign, pksign$, send *pksign* to blockchain;
Input: Receive *encsign_skRSA* from FA
$skRSA \leftarrow Decrypt(sksign, encsign\_skRSA)$;
Seal *skRSA* locally;

5.2 Constructions of Enclaves

When the setup process is done, users are ready to begin a data collaboration. Suppose DA has found the desired metadata and written the analysis program.

Enclaves *enclave_a* ,*enclave_p* are the same as described in previous section. The rest two enclaves are constructed as follows.

The *oenclave_sign* is embedded with Fidelius' static checking system, to check whether a given analysis program satisfies the following conditions

- The program does not reveal raw data above a certain degree, e.g. revealing the whole raw data is prohibited while revealing the average of raw data is allowed.
- The program does not reveal the private keys, e.g., the imported *ska* and DA's private key (if DA wants analysis result to be in an encrypted format).
- The program is not too simple for "click farming", e.g. the program must contain the input process of *raw_data*.

Algorithm 6: *analysis_enclave*(New)

Input: *raw_data*
if $H(raw\_data) \neq hash_{raw\_data}$ *(hardcode)* **then**
  return *"Invalid raw data"*;
Run *analysis_program* to get *analysis_result*;
Establish local attestation to *oenclave_p*;
begin
  Confirm objective hash: $hash_{oenclave\_p}$ (hardcode);
  Receive *ska* from *oenclave_p*;
$sign_{result} \leftarrow sign(ska, analysis\_result)$;
return $(analysis\_result, sign_{result})$ *and sent it to blockchain*;

We omit the detail of static checking system's implementation but assume its existence instead.

The *oenclave_sign* is as follows:

Algorithm 7: *oenclave_sign* (signing process)

Input: *analysis_enclave*(binary)
if *Static checking of analysis_enclave pass* **then**
  Unseal *skRSA* ;
  return $sign(skRSA, H(analysis\_enclave))$;

5.3 Data Collaboration Process

The process works as follows:

- DA writes the *analysis_enclave* and send it to *oenclave_sign* to get its signature.
- DA packs *analysis_enclave*, $H(analysis\_enclave)$, the signature, *pkRSA*, and the license together as installation package and sent it to blockchain.
- DA runs *oenclave_a* to send *encp_ska* to blockchain.
- DP receives *analysis_enclave* and locally deploys it.
- DP receives *encp_ska* and sends to *oenclave_p*.
- DP runs *analysis_enclave* and *oenclave_p* to get *analysis_result* and $sign_{result}$, and sends them to blockchain.
- Verification contract verifies outputs by DP and completes data collaboration as in section 3.

References

[1] Carl A. Waldspurger. Memory resource management in VMware ESX server. In *USENIX Symposium on Operating System Design and*

Implementation (OSDI), pages 181–194, 2002. https://www.usenix.org/legacy/event/osdi02/ tech/waldspurger/waldspurger.pdf.