Computers, Materials & Continua
DOI: 10.32604/cmc.2022.030995
Article

Tech Science Press

TrustControl: Trusted Private Data Usage Control Based on Security Enhanced TrustZone

Hong Lei[1,2,3], Jun Li[1,*], Suozai Li[4], Ming Huang[4], Jieren Cheng[5], Yirui Bai[1], Xinman Luo[1] and Chao Liu[6]

[1]School of Cyberspace Security (School of Cryptology), Hainan University, Haikou, 570228, China
[2]SSC Holding Company Ltd., Chengmai, 571924, China
[3]Oxford-Hainan Blockchain Research Institute, Chengmai, 571924, China
[4]China Electronics Corporation Hainan Joint Innovation Research Institute Co. Ltd, Chengmai, 571924, China
[5]School of Computer Science and Technology, Hainan University, Haikou, 570228, China
[6]The Blockhouse Technology Limited, Oxford, OX2 6XJ, United Kingdom
*Corresponding Author: Jun Li. Email: junli_1632021@163.com
Received: 07 April 2022; Accepted: 07 June 2022

Abstract: The past decade has seen the rapid development of data in many areas. Data has enormous commercial potential as a new strategic resource that may efficiently boost technical growth and service innovation. However, individuals are becoming increasingly concerned about data misuse and leaks. To address these issues, in this paper, we propose TrustControl, a trusted data usage control system to control, process, and protect data usage without revealing privacy. A trusted execution environment (TEE) is exploited to process confidential user data. First of all, we design a secure and reliable remote attestation mechanism for ARM TrustZone, which can verify the security of the TEE platform and function code, thus guaranteeing data processing security. Secondly, to address the security problem that the raw data may be misused, we design a remote dynamic code injection method to regulate that data can only be processed for the expected purpose. Our solution focuses on protecting the sensitive data of the data owner and the function code of the data user to prevent data misuse and leakage. Furthermore, we implement the prototype system of TrustControl on TrustZone-enabled hardware. Real-world experiment results demonstrate that the proposed TrustControl is secure and the performance overhead of introducing our prototype system is very low.

Keywords: TrustZone; data usage control; privacy; security

1 Introduction

In recent years, the development of digitization has produced a large amount of data. Data is continuously produced, collected, shared, analyzed, and traded. However, these rapid changes are having a serious effect on data usage and have attracted huge attention. Once the data owner's data

is processed as a valuable data resource, it needs to be ensured that the data is not leaked, misused, freely distributed, etc. Otherwise, it is difficult for the data owner to provide sensitive data to the data user, and therefore the data user cannot obtain high-quality services. Given the growing importance of data and the reliance on private data, a flexible and configurable data usage control is needed. Data usage control aims to process user private data expectedly to ensure the authenticity of data processing results and reduce the risk of data leakages and misuses. Therefore, the ability to use user-defined and validated function code for data processing is essential in data usage control. The research community has done a lot of work to protect sensitive data security and privacy. In terms of addressing data usage conditions, there are data anonymization [1–3] and various data access control methods which include access control in internet of things [4,5], access control in cloud environments [6–8]. To solve the purpose of data usage, many privacy-protection system architectures can be found between decentralized data privacy with blockchain [9–11], data use policy [12], privacy property [13], and data usage control [14]. In the existing data usage control methods, once the data owner authorizes the user to obtain access to the data, the raw data is directly delivered to the user. However, under normal circumstances, the data owner has almost no authority to control how the data user will use the data. Therefore, the existing data usage control methods will expose the raw data of the data owner and cannot effectively guarantee the security and privacy of data [15]. Fortunately, as emerging technologies, hardware-enforced trusted execution environment (TEE) techniques are attracting academic attention in terms of improving the reliability, privacy, and security of data and code. There have also been studies focusing on secure data processing by levering TEE. For example, reference [16] mainly uses TEE to protect the data collection process. Reference [17] proposed a new blockchain-based data trading ecosystem. In this ecosystem, TEE is used for securing data processing. Reference [18] uses TEE to implement a trusted off-chain contract execution engine. However, in these studies, risk refers to the authenticity of the data processing code and the trustworthiness of the results. TEE is considered secure by default and is extended to believe that the function code that processes user data in TEE also performs as expected by the data user. Such solutions may face the challenge that the data processing code in TEE is not reliable or does not process data in the way that the data user expects. Hence, the mechanism to perform a specific data process operation should be verified before usage, in case of inaccuracy of data processing results. The hardware supports for TEE is one of the more practical solutions for data privacy protection. TrustZone has been recognized as a reliable data security protection and processing technology on the ARM platform [19,20]. It implements a system-level isolation environment by creating the TEE for security-sensitive code and data protection, thereby protecting software from untrusted rich execution environments (REE) [21]. Popular trusted execution environments also include Intel software guard extensions (SGX), but so far, remote attestation has only been used with Intel SGX [22,23]. One of the goals of this paper has therefore been to establish the remote attestation mechanism on ARM TrustZone.

In this paper, we leverage ARM TtustZone as the TEE technology to implement the prototype of TrustControl. TrustControl exploits the TEE to load, process, and protect sensitive code and data by providing secure remote attestation, remote dynamic code injection, and secure storage. We design the remote attestation mechanism to solve the lack of remote attestation for the ARM platform, allowing the ARM platform to verify the authenticity and reliability of the platform and application program. In particular, this paper also designs a remote dynamic injection method of trusted code to perform security-critical data processing to ensure that data is processed with expected behavior. The experiment results show that the system meets the goal of user-defined data usage control at a reasonable cost. Our future research roadmap will focus on exploring ways to enhance the performance

of ARM TrustZone with the intention of applying it to the server side to conduct more in-depth privacy protection research.

2 Background

TEE is widely used and will likely become even more pervasive, considering TEE can be found on low-cost ARM processors [24]. TrustZone technology is composed of hardware security extension of the ARM processor, among which hardware security extension has great advantages and has low influence on core power consumption and performance [25]. The CPU has its own separate physical address space when executing programs in TEE and REE. The REE software only has access to its own physical address space, while the TEE software has access to both REE and TEE physical address spaces. TrustZone technology divides system resources into normal world (NW) and secure world (SW). The SW has higher access rights than the NW, so it can access the resources of the NW more flexibly, while the access rights of the NW to the SW are strictly limited. In this way, the memory space of the TEE cannot be viewed or tampered with, thereby improving security [26]. ARM TrustZone technology introduces a special monitor mode mechanism [27]. The monitor mode is responsible for the state switching of the processor, including the security state and the non-safe state.

The extension of the TrustZone hardware architecture embeds security in the processor, implements a trusted operating system (TOS). TOS has independent exception handling, interrupt handling, scheduling, application, process, thread, and driver [26]. The TrustZone-based system realizes the trust environment of the entire TEE, including multiple trusted applications (TAs) [28,29]. TA refers to a trusted application running in SW, uses the services provided by the TEE kernel to access system resources. The role of TA is to provide different services for standard user programs or other TAs. By implementing various TAs with different functions in TEE, TEE systems are becoming more and more usable and practical. As a trusted hardware platform, ARM TrustZone is widely recognized by the industry for its security architecture and many open sources projects.

3 Overview of TrustControl

3.1 Threat Model and Assumptions

We assume that the hardware which is equipped with ARM TrustZone is properly implemented, securely manufactured, and against malicious attacks from the operating system. We assume that the operating system and the user space in the NW are not trusted. However, we assume that TEE components, including the bootloader, firmware, secure monitor, and TOS, are trustworthy and implemented according to the GlobalPlatform (GP) TEE specification. We assume the attestation server will perform the required service correctly and is always available. We also assume that all data operations required by data users and data owners are trustworthy, and the data owners do not provide meaningless or false data on purpose. We also assume that TA sealed data is secure, which means that no one else other than TA can decrypt the sealed data. We recognize that previous work has demonstrated that the confidentiality of ARM TrustZone can be compromised via side-channel attacks [30,31] preventing such attacks is an important but ongoing task. In light of this threat, the mitigations which continue to be released can effectively mitigate the risk of TEE [32].

3.2 TrustControl Overview

We implement a prototype system of TrustControl, which uses ARM TrustZone hardware security extensions to perform remote attestation and data usage control in the SW. The system architecture

of TrustControl is shown in Fig. 1. The application calls the code isolated in the SW through the GP TEE Client application programming interface (API) and the GP TEE Internal API [33]. TrustControl consists of four main functional components: data owner, data user, attestation server, and trusted computing execution. The following four components are described in detail.

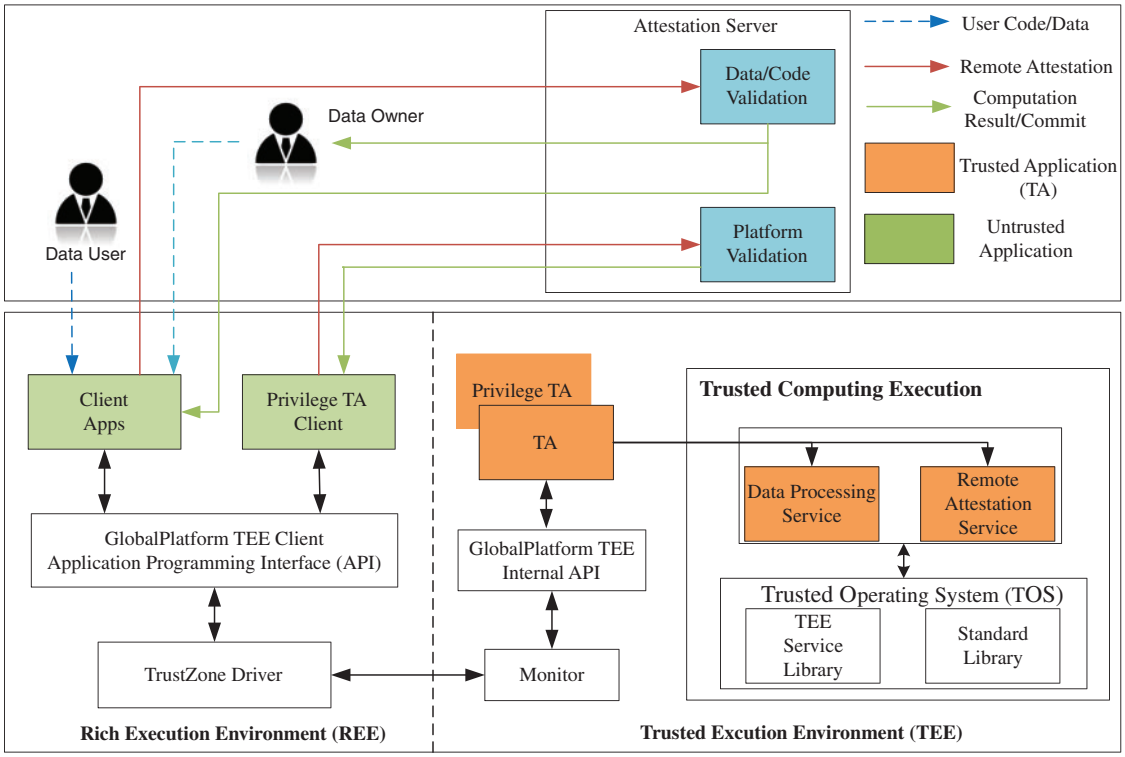

Figure 1: An overview of TrustControl

Data Owner (DO). The DO refers to the person who owns the data. And DO will only provide the raw data based on the remote attestation result, to ensure that the raw data can be processed expectedly.

Data User (DU). The DU refers to the person who provides the data processing function code and gets the data processing results.

Attestation Server. The attestation server aims to verify the legitimacy of remote attestation data, the security and reliability of the platform, application programs, and function codes. Now we assume that the attestation server is sufficiently secure.

Trusted Computing Execution. Trusted computing execution provides data processing and remote attestation services. The data processing service is responsible for the loading, execution, and unloading of function code, etc. Remote attestation service responsible for responding to the request, generating attestation data and certificates required for remote attestation. By using the function code of the DO to perform data operations in the TEE, the correctness of the data processing results is guaranteed.

Generally speaking, the main workflow of TrustControl is as follows. Firstly, DUs and DOs perform remote attestation to TrustControl and proceed to the next step after successful attestation. Secondly, DUs deploy data processing code to TrustControl, TrustControl receives the code and

validates it, and loads it into the TEE for execution after successful validation. Thirdly, DOs send data to TrustControl, and TrustControl loads the data into TEE after successful authentication, which is processed by secure TA. TrustControl is used to perform remote attestation, remote dynamic code trusted injection, and secure data processing.

4 System Design

4.1 Remote Attestation Design

Remote attestation provides a highly secure and reliable mechanism that can verify the security and reliability of platform and applications. The remote attestation mechanism can request and import attestation keys and perform remote attestation through the system calls. To this end, we implemented a privilege TA to request and import attestation keys, and prohibit other TA to perform these operations. Privilege TA refers to the remote attestation executor, responsible for applying for the certificate of the attestation key, interacting with the TEE, attestation server, executing requests, and importing remote attestation certificates.

We assume $AK = attestation\,key$, which contains the secret key pairs for attestation. The privilege TA calls AKRequest method to request open-source portable trusted execution environment operating system (OP-TEE OS) to generate AK. Specifically, the privilege TA will call the method in the NW to request the AK. The remote attestation service will check whether the AK exists after receiving the request in the SW. If the check result is true, the AK will be returned, otherwise, the remote attestation service will use the RSA algorithm provided by the crypto service to generate the AK. After that, the AK will be stored in the SW.

$$AK = \{pubAK, privAK\} \tag{1}$$

$$sigAK = Sign(pubAK) \tag{2}$$

In Eq. (1), pubAK represents the public key of attestation key, while privAK represents the secret key, which is only known by TEE itself and will not be disclosed to REE. After the AK is generated, OP-TEE OS signs the pubAK with the TEE private key to get sigAK, see Eq. (2). The sigAK is used to prove that AK was generated in the TEE. Then send the sigAK and pubAK to the privilege TA.

As shown in Eq. (3), we assume $AD = attestation\,data$, which contains pubAK and sigAK, the privilege TA sends AD to the attestation server.

$$AD = \{pubAK, sigAK\} \tag{3}$$

If the attestation server can verify sigAK, it will issue a certificate certAK for the pubAK and send the certAK to the privilege TA. Then OP-TEE OS will save the certAK. When the verification is passed, the privilege TA will call AKSeal to seal the AK, so that after the TEE platform restarts, we don't need to generate a new AK again. The sealing operation will be completed in TEE.

To complete the TEE platform initialization for the remote attestation process, the following remote attestation methods were designed.

Privilege_TA init: initialize the TEE platform.

Privilege_TA import_ak: import AK from the sealed key.

Privilege_TA connect: connect to the attestation server, issue a certificate to pubAK, and send the certAK to TEE.

Privilege_TA export_cert: export the certAK in TEE.

We now sketch the process of TEE platform initialization in Fig. 2.

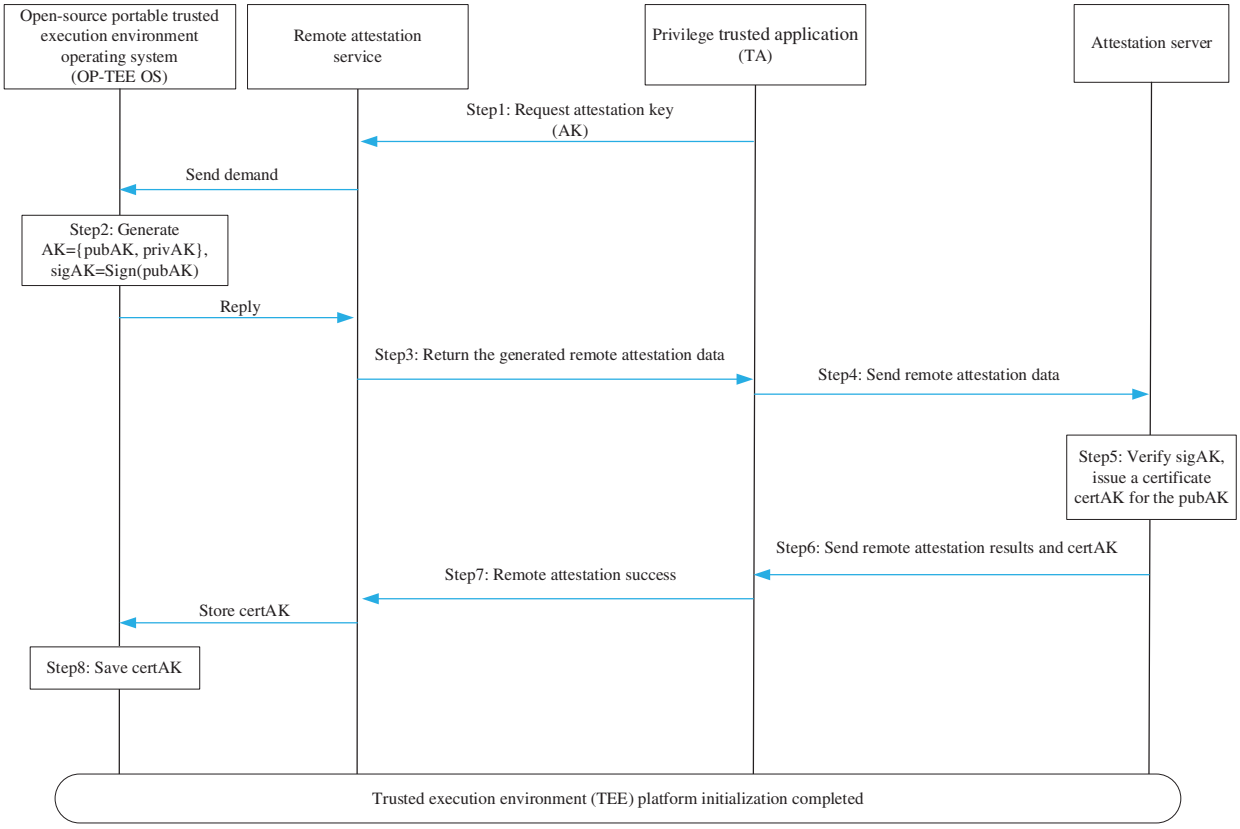

Figure 2: The process of TEE platform initialization

Step 1: Privilege TA performs remote attestation and requests OP-TEE OS to generate attestation data;

Step 2: Remote attestation service process the command request sent by the privilege TA in the NW and generate AK, sigAK;

Step 3: After the TEE generates the attestation data, remote attestation service sends AD to the privilege TA;

Step 4: Privilege TA sends AD to the attestation server;

Step 5: The attestation server verifies AD. The verification results include trusted and untrusted; if the attestation server verifies that sigAK is credible, it will issue a certificate for the pubAK, certAK;

Step 6: The attestation server sends the remote attestation results and the certAK to the privilege TA;

Step 7: Privilege TA executes the import certificate command and saves the certAK in TEE;

Step 8: TEE processes the commands and saves the certAK issued by the attestation server.

After TEE platform initialization, the client can verify the authenticity of the TA. First, the TA generates a pair of public and private keys for creating a secure channel, signs the TA's public key and

measurements with privAK, and then the TA sends the message to the client. The qualified Message is shown in Eq. (4). Then, the client requests the attestation server to get the root certificate, verifies certAK, extracts the public key to verify the signed data after successful verification, and finally verifies the TA's measurement, and gets the data after successful verification. Note that the data can be the public key or other data. When the client gets the public key, it will first generate a secrete key K, then encrypt K with the public key and send the encryption result to TA, so that both parties negotiate the encryption secret key and subsequently encrypt the data by symmetric encryption. One advantage of this is to avoid the performance degradation problem caused by using asymmetric encryption.

$$Message = \{data||Signature||certAK\} \tag{4}$$

We provide an interactive process between the OP-TEE OS, TA, client, and the attestation server execution workflow, providing further step details on Fig. 3.

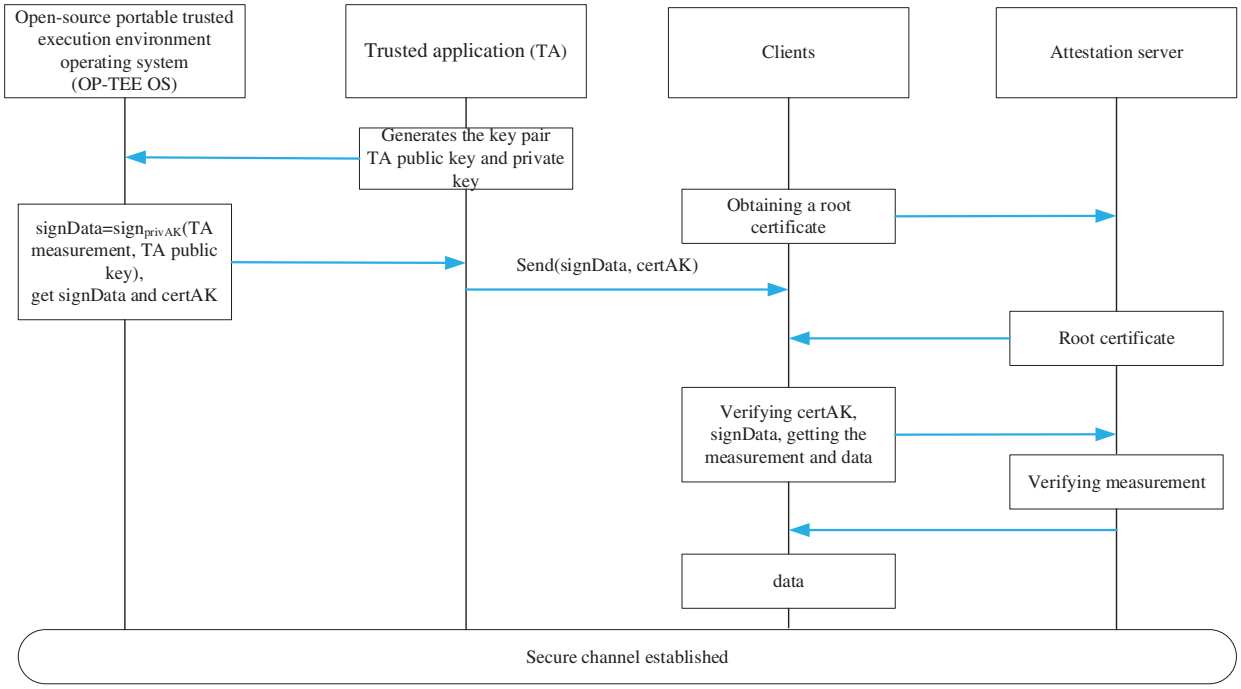

Figure 3: The process of performing remote attestation

4.2 Remote Dynamic Code Injection

Remote dynamic code injection provides a secure method of code injection that dynamically injects legitimate code remotely. At the same time, remote dynamic code injection provides two modes: temporary code injection and long-term code injection, thus providing different storage methods for codes with different needs. In this way, we can easily add and modify the function code to improve the authenticity and reliability of data processing results.

The remote dynamic code injection process is concluded as Algorithm 1.

DOs and DUs send registration requests, and then the TEE platform generates registration data. The registration request from the DO includes preset user information, such as account number and password, the attribute information of the original data, such as whether the data is public or

semi-public, and the attribute information of the DO, such as enterprise and individual. The DU registration request includes predefined code information of the DU, such as whether the injected code is temporarily stored or permanently stored. After the initial registration, the TEE platform will distribute secret key pairs for DOs and DUs. And the registration result includes a predefined user identifier and a monotonic counter. The monotonic counter specifically refers to the logical mechanism provided for TEE technology to prevent data from being replayed. In addition, the monotonic counter identifies the number of times the data has been used, for example, if the data has been used 3 times, the monotonic counter is 3 to prevent the data from being used secretly. The TEE platform receives the function code injection request sent by the DU. Then it verifies the security of DU based on the registered data (line 3 in Algorithm 1). Note that the TEE platform receives data and function code only if the DU and DO pass the security verification. The TEE platform generates authentication data based on the remote function code and sends the authentication data to the remote attestation server (lines 4–6 in Algorithm 1). The attestation server verifies the authentication data and returns the result. The DO receives the attestation result and verifies its authenticity (lines 12–13 in Algorithm 1). The DO provides the raw data only based on the true attestation result (line 14 in Algorithm 1), thus ensuring that the raw data can be processed with the expected behavior. The TEE platform receives the raw data after verifying the security of the DO (lines 18–19 in Algorithm 1).

Remote dynamic code injection provides three important methods, namely install_TA, execution_TA, and uninstall_TA, which are presented as install, execute, and uninstall trusted dynamic function code. When TEE platform receives a specific computation task from the DU, it will load the corresponding TA for that task. Then TEE platform will remotely verify the authenticity and integrity of TA. The immediate step before the DO sends the raw data is to remotely verify the function code. Once the function code is successfully verified, the DO sends the raw data to the TEE platform, both the TEE platform and DO can then extend their trust to the DU, knowing that the raw data is securely processed by the trusted function code in the TEE platform until termination, while the computation results will be genuine and reliable. This approach guarantees that the raw data is used for the intended behavior. The TEE platform calls the install_TA method, loads the verified function code to process the raw data, then sends the trusted data processing result to DU (lines 24–26 in Algorithm 1). After the data is processed, we can call the uninstall_TA method to uninstall the function code (lines 30–31 in Algorithm 1). To prevent raw data leakage, the TEE platform also deletes the locally stored raw data after the data processing is finished. The trusted function code is called to process the raw data, which can prevent the DU from obtaining or reusing the raw data, thus effectively improving the security and privacy of the raw data.

Algorithm 1: Algorithm for remote dynamic code injection

Input: remote dynamic code, raw data
Output: data processing result Res

```
1   procedure remoteDynamicCodeInjection
2       if requestReceived == true then
3           if DUAuthentication == true then
4               receive dynamic code
5               generate authentication data
6               send authentication data to remote attestation server
7           else
8                deny
9           end if
```

(Continued)

```
10      end if
11      wait() // wait for remote attestation result
12      if DOReceivedResult == true then
13        if verifyResult == true then
14          send raw data to TEE platform
15        end if
16      end if
17      if requestReceived == true then
18        if DOAuthentication == true then
19          receive raw data
20        else
21          deny
22        end if
23      end if
24      if install_TA() == true then
25        Res = execution_TA()
26        return Res
27      else
28        report install error
29      end if
30      if uninstall_TA() != true then
31        report uninstall error
32      end if
33    end procedure
```

5 System Implementation and Evaluation

5.1 Experiment Setup

One of the most well-known open-sourced TEE projects for ARM platforms is OP-TEE, an open-source Linux-based TEE platform compatible with ARM TrustZone [34]. Many researchers have utilized OP-TEE to evaluate their TEE prototype [35,36]. We provide an implementation of TrustControl based on the OP-TEE project. And TrustControl applications were deployed in a TrustZone-enabled development board based on the ARM aarch64. The main memory of the development board is an 8GB DRAM. The TrustControl components include attestation server service, client apps, secure TA, and data processing service. They were written in C based on OP-TEE v3.8 and run on top of Ubuntu 18.04 LTS. The total software lines of code (SLOC) for TrustControl components are shown in Tab. 1.

Table 1: Component's LOC

Component	Lines of code
Remote attestation service	365
Data processing service	418
Client apps	1389

(Continued)

Table 1: Continued

Component	Lines of code
Privilege TA client	546
Attestation server	405
Total	3123

5.2 Remote Attestation Overhead

To improve the reliability of the remote attestation, the method which was Privilege_TA init, Privilege_TA import_ak, Privilege_TA connect, and Privilege_TA export_cert was each tested 100 times, respectively. The comparison of the four methods' average execution time is shown in Fig. 4. It is apparent from Fig. 4 that the overhead of most remote attestation methods is very short. Among the four methods, the numerical difference can be introduced because of the different functions of code. The import_ak, connect, and export_cert method average runtime is only 0.19, 0.16, and 0.018 s, respectively. By contrast, there is a significant difference between the init method and the other three methods. The average run time of the init method ranged from 2.8 to 4.8 s, indicating that it takes a significant amount of time to obtain the remote attestation data for the first time. When the device finishes running the init method or restarts, the method is only executed for 0.6 s, which is fast. We remark that efficient remote attestation is very important.

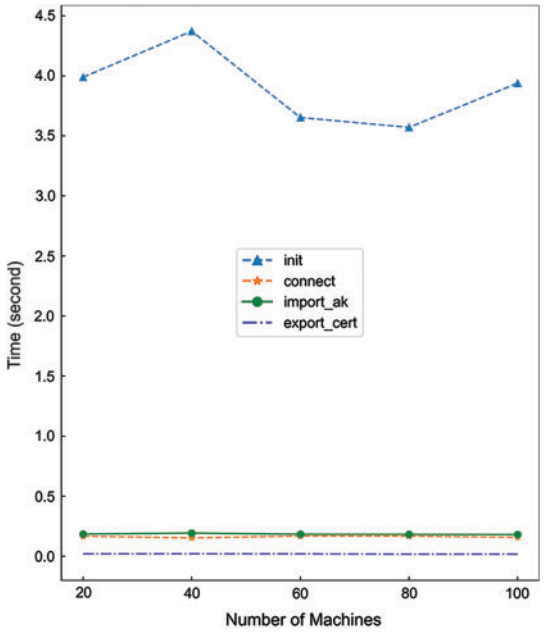

Figure 4: Overhead of executing different methods

5.3 Computational Cost of Crypto Service

Due to the introduction of data encryption and decryption in TrustControl, so we evaluate and compare the encryption and decryption efficiency in crypto service between SHA1, SHA256, AES128, and AES256. Considering that users may deliver data of different sizes, we focus on the performance

overhead to compute different data sizes which are from 10 to 40 MB for SHA1 and SHA256, 10 to 300 MB for AES128 and AES256. We tested the encryption and decryption overhead using two different ways on a real experiment board: software encryption and ARM instruction encryption. The comparison result of encryption and decryption between software encryption and ARM instruction encryption is presented in Tab. 2. We repeated each test 100 times to reduce the test errors and report the average result. As shown in Tab. 2, SHA1 and SHA256 compute the message digest on 10, 20, and 40 MB, respectively. We can observe that as the encrypted data size grows from 10 to 40 MB, the encryption time for SHA1 and SHA256 increases regardless of whether software or ARM instruction encryption is used, but the SHA1 (ARM instruction) and SHA256 (ARM instruction) encryption overheads are very low compared to software encryption. For example, using the SHA256 algorithm to encrypt 40 MB of data, software encryption takes about 10 times as long as ARM instruction encryption. For AES encryption and decryption, we can see that there is a significantly different between software encryption and ARM instruction encryption. As the data increases, the time spent on software encryption and decryption increase rapidly. However, AES (ARM instruction) will greatly increase the encryption and decryption speed, which is especially obvious when encrypting and decrypting large amounts of data. For example, when encrypting 100 MB of data, AES256 (ARM instruction) takes only 0.099 s, and the time to decrypt the data is also similar. The experimental results show that the encryption overhead using software encryption is 4 to 9 times higher than using ARM instruction encryption when the encrypted data is small, and 18 to 27 times higher when the encrypted data is large, demonstrating the excellent encryption and decryption efficiency of ARM instruction encryption. In this experiment, encrypting the data which is 40 MB with SHA256 (ARM Instruction) and 300 MB with AES256 (ARM Instruction) only needs 0.056 and 0.208 s, which has a tiny impact on the whole crypto service. From these findings, we can infer that when the encrypted data size grows, continuing to use software encryption imposes significant encryption and decryption overhead, while using ARM instruction encryption a less computational cost.

Table 2: Computational cost of crypto service

Algorithm type	Operation	File size (MB)	Secure world time (second)
SHA1	Message Digest	10	0.139
SHA1	Message Digest	20	0.259
SHA1	Message Digest	40	0.501
SHA1(ARM Instruction)	Message Digest	10	0.028
SHA1(ARM Instruction)	Message Digest	20	0.038
SHA1(ARM Instruction)	Message Digest	40	0.058
SHA256	Message Digest	10	0.153
SHA256	Message Digest	20	0.287
SHA256	Message Digest	40	0.555
SHA256(ARM Instruction)	Message Digest	10	0.028

(Continued)

Table 2: Continued

Algorithm type	Operation	File size (MB)	Secure world time (second)
SHA256(ARM Instruction)	Message Digest	20	0.037
SHA256(ARM Instruction)	Message Digest	40	0.056
AES128	Encryption/Decryption	10	0.206/0.205
AES128	Encryption/Decryption	100	1.650/1.645
AES128	Encryption/Decryption	300	4.870/4.847
AES128(ARM Instruction)	Encryption/Decryption	10	0.050/0.051
AES128(ARM Instruction)	Encryption/Decryption	100	0.090/0.089
AES128(ARM Instruction)	Encryption/Decryption	300	0.178/0.178
AES256	Encryption/Decryption	10	0.223/0.225
AES256	Encryption/Decryption	100	1.835/1.845
AES256	Encryption/Decryption	300	5.407/5.450
AES256(ARM Instruction)	Encryption/Decryption	10	0.051/0.051
AES256(ARM Instruction)	Encryption/Decryption	100	0.099/0.098
AES256(ARM Instruction)	Encryption/Decryption	300	0.208/0.208

5.4 Remote Dynamic Code Injection and Runtime

Here we use a development board that supports ARM TrustZone technology as the TEE platform, and use a personal computer (PC) as the attestation server, the manipulation device for DU and DO, and the TEE platform console, as shown in Fig. 5.

The remote dynamic injection and runtime of the function code were evaluated through a series of experiments. The install_TA, execution_TA, and uninstall_TA methods were run 100 times and the average value was recorded. Due to the different DUs may have different processing purposes for raw data, in this experiment, we choose a simple remote dynamic code injection example. When the TEE platform receives function code from the DU, it invokes the pseudo TA to perform operations on the function code, including installation, execution, and uninstallation. When the function code is installed, the raw data can be processed by means of API calls. When the pseudo TA calls the function code for installation and uninstallation, the installation time and uninstallation time are shown in the first and third columns of Fig. 6, respectively. The second histogram in Fig. 6 shows the time taken to execute the function code. The results show that the installation time for the function code is 0.294 s and the uninstallation time is 0.194 s. In addition, both the installation and uninstallation times are low and well acceptable. Thus, we can conclude that the introduction of the remote dynamic code injection method has little impact on the efficiency of processing data.

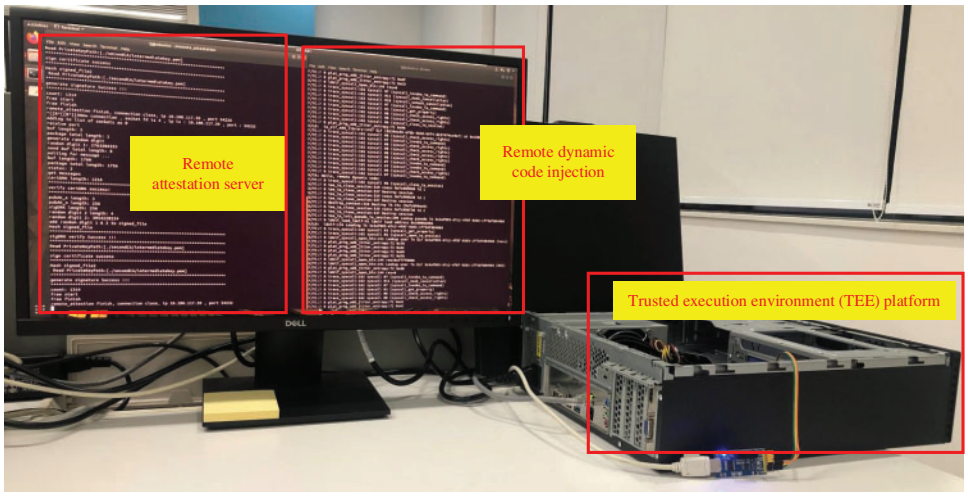

Figure 5: Implementation of TrustControl on PC and a development board

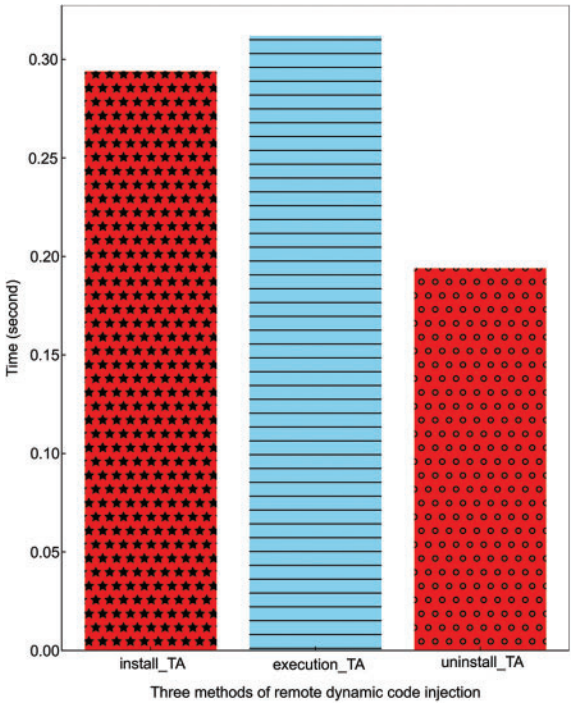

Figure 6: Remote dynamic code injection and runtime

6 Conclusion

The privacy and control of data are being increasingly threatened as huge amounts of data are collected, analyzed, exchanged, and stored in various ways. TrustZone is a widely available system-level security solution that can provide a trusted execution environment to protect the security and privacy of data and code. In this paper, we propose TrustControl, a user-centric prototype for secure

data-trusted computing. We design the remote attestation mechanism to validate the security of the TEE platform and remote function codes to achieve secure data processing on the TEE platform. To ensure that the raw data of the data owner is processed under the expected behavior, we implement remote dynamic code injection to provide installation, execution, and uninstall features. This study has shown that TrustControl can be used to implement a variety of secure applications that strengthen data usage control. Our solution provides a higher level of security for user privacy and the evaluation results show that TrustControl can work efficiently with acceptable performance.

Funding Statement: This work was supported by the National Key R&D Program of China (No. 2021YFB2700601), Research Project of Hainan University (No. HD-KYH-2021240), Finance Science and Technology Project of Hainan Province (No. ZDKJ2020009 and ZDKJ2020012), National Natural Science Foundation of China (No. 62163011, 62162022 and 62162024), and Key Projects in Hainan Province (No. ZDYF2021GXJS003 and ZDYF2020040).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] S. Abd Razak, N. H. Mohd Nazari and A. Al-Dhaqm, "Data anonymization using pseudonym system to preserve data privacy," *IEEE Access*, vol. 8, pp. 43256–43264, 2020.

[2] M. Yamaç, M. Ahishali, N. Passalis, J. Raitoharju, B. Sankur *et al.,* "Multi-level reversible data anonymization via compressive sensing and data hiding," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1014–1028, 2021.

[3] R. Bild, K. A. Kuhn and F. Prasser, "SafePub: A truthful data anonymization algorithm with strong privacy guarantees," *Proc. on Privacy Enhancing Technologies*, vol. 2018, no. 1, pp. 67–87, 2018.

[4] S. Qi, Y. Lu, W. Wei and X. Chen, "Efficient data access control with fine-grained data protection in cloud-assisted IIoT," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2886–2899, 2021.

[5] I. Ullah, H. Zahid and M. A. Khan, "An access control scheme using heterogeneous signcryption for iot environments," *Computers, Materials & Continua*, vol. 70, no. 3, pp. 4307–4321, 2022.

[6] S. Yu, C. Wang, K. Ren and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. 2010 Proc. IEEE Int. Conf. on Computer Communications*, San Diego, CA, USA, pp. 1–9, 2010.

[7] V. Rajkumar, M. Prakash and V. Vennila, "Secure data sharing with confidentiality, integrity and access control in cloud environment," *Computer Systems Science and Engineering*, vol. 40, no. 2, pp. 779–793, 2022.

[8] N. R. R. Paul and D. P. Raj, "Enhanced trust based access control for multi-cloud environment," *Computers, Materials & Continua*, vol. 69, no. 3, pp. 3079–3093, 2021.

[9] G. Zyskind, O. Nathan and A. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *Proc. 2015 IEEE Security and Privacy Workshops*, San Jose, CA, USA, pp. 180–184, 2015.

[10] G. Zyskind, O. Nathan and A. Pentland, "Enigma: Decentralized computation platform with guaranteed privacy," 2015. [Online]. Available: https://arxiv.org/abs/1506.03471.

[11] T. Feng, H. Pei, R. Ma, Y. Tian and X. Feng, "Blockchain data privacy access control based on searchable attribute encryption," *Computers, Materials & Continua*, vol. 66, no. 1, pp. 871–890, 2021.

[12] E. Elnikety, A. Mehta, A. Vahldiek-Oberwagner, D. Garg and P. Druschel, "Thoth: Comprehensive policy compliance in data retrieval systems," in *Proc. the 25th USENIX Conf. on Security Symp.*, Austin, TX, USA, pp. 637–654, 2016.

[13] A. Datta, M. Fredrikson, G. Ko, P. Mardziel and S. Sen, "Use privacy in data-driven systems: Theory and experiments with machine learnt programs," in *Proc. the 2017 ACM Special Interest Group on Security, Audit and Control Conf.*, Dallas, TX, USA, pp. 1193–1210, 2017.

[14] Y. Xiao, N. Zhang, W. Lou and Y. Hou, "PrivacyGuard: Enforcing private data usage control with blockchain and attested off-chain contract execution," in *Proc. the 25th European Symp. on Research in Computer Security*, Guildford, United Kingdom, pp. 610–629, 2020.
[15] GlobalPlatform, "TEE internal API specification v1.3," 2021. [Online]. Available: https://globalplatform.org/specs-library/tee-internal-core-api-specification/.
[16] T. Zheng, Y. Luo, T. Zhou and Z. Cai, "Towards differential access control and privacy-preserving for secure media data sharing in the cloud," *Computers, & Security*, vol. 113, no. 1, pp. 102553, 2022.
[17] F. Zhang, E. Cecchetti, K. Croman, A. Juels and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proc. the 2016 ACM Special Interest Group on Security, Audit and Control Conf. on Computer and Communications Security (CCS)*, Vienna, Austria, pp. 270–282, 2016.
[18] W. Dai, C. Dai, K. -K. Choo, C. Cui, D. Zou *et al.,* "SDTE: A secure blockchain-based data trading ecosystem," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 725–737, 2020.
[19] F. Brasser, D. Gens, P. Jauernig, A. R. Sadeghi and E. Stapf, "Sanctuary: Arming trustzone with user-space enclaves," in *Proc. 2019 Network and Distributed System Security Symp.*, San Diego, California, 2019.
[20] H. Sun, K. Sun, Y. W. Wang, J. W. Jing and H. N. Wang, "TrustICE: Hardware-assisted isolated computing environments on mobile devices," in *Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, Rio de Janeiro, Brazil, pp. 367–378, 2015.
[21] M. Sabt, M. Achemlal and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *Proc. 2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland, pp. 57–64, 2015.
[22] S. Johnson, V. Scarlata, C. Rozas, E. Brickell and F. Mckeen, "Intel® software guard extensions: EPID provisioning and attestation services," 2016. [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/671370?explicitVersion=true&wapkw=EPIDprovisioningandattestationservices.
[23] V. Scarlata, S. Johnson, J. Beaney and P. Zmijewski, "Supporting third party attestation for Intel® SGX with Intel® data center attestation primitives," 2018. [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/intel-sgx-support-for-third-party-attestation-801017.pdf.
[24] R. Coombs, "Adapting mobile security architecture for IoT," 2018. [Online]. Available: https://community.arm.com/arm-community-blogs/b/embedded-blog/posts/adapting-mobile-security-architecture-for-iot.
[25] X. Zheng, L. Yang, J. Ma, G. Shi and D. Meng, "TrustPAY: Trusted mobile payment on security enhanced ARM TrustZone platforms," in *Proc. 2016 IEEE Symp. on Computers and Communication (ISCC)*, Messina, Italy, pp. 456–462, 2016.
[26] P. Wilson, A. Frey, T. Mihm, D. Kershaw and T. Alves, "Implementing embedded security on dual-virtual-CPU systems," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 582–591, 2007.
[27] T. Alves and D. Felton, "TrustZone: Integrated hardware and software security enabling trusted computing in embedded system," *Government Information Quarterly*, vol. 3, no. 4, pp. 18–24, 2005.
[28] N. Zhang, K. Sun, W. Lou and Y. T. Hou, "CaSE: Cache-assisted secure execution on ARM processors," in *Proc. 2016 IEEE Symp. on Security and Privacy (SP)*, San Jose, CA, USA, pp. 72–90, 2016.
[29] ARM Limited, "TrustZone technology for the ARMv8-M architecture version 2.0," 2017. [Online]. Available: https://developer.arm.com/documentation/100690/0200/ARM-TrustZone-technology.
[30] R. Keegan, "Hardware-backed heist: Extracting ECDSA keys from qualcomm's TrustZone," in *Proc. the 2019 ACM Special Interest Group on Security, Audit and Control Conf. on Computer and Communications Security*, London, United Kingdom, pp. 181–194, 2019.
[31] P. F. Qiu, D. S. Wang, Y. G. Lyu and G. Qu, "VoltJockey: Breaching TrustZone by software-controlled voltage manipulation over multi-core frequencies," in *Proc. the 2019 ACM Special Interest Group on Security, Audit and Control Conf. on Computer and Communications Security*, London, United Kingdom, pp. 195–209, 2019.
[32] "ARM Speculation Barrier," 2020. [Online]. Available: https://github.com/ARM-software/speculation-barrier.

[33] GlobalPlatform, "GlobalPlatform device technology TEE client API specification v1.0," 2010. [Online]. Available: https://globalplatform.org/specs-library/tee-client-api-specification/.
[34] Linaro, "OPTEE secure OS," 2020. [Online]. Available: https://github.com/OP-TEE/optee_os.
[35] S. Y. Wan, M. S. Sun, K. Sun, N. Zhang and X. He, "RusTEE: Developing memory-safe ARM TrustZone applications," in *Proc. Annual Computer Security Applications Conf.*, Austin, USA, pp. 442–453, 2020.
[36] Linaro, "OPTEE device," 2020. [Online]. Available: https://optee.readthedocs.io/en/latest/building/index.html.