Tsinghua Science and Technology

Volume 22 | Issue 2 Article 9

2017

A Private User Data Protection Mechanism in TrustZone Architecture Based on Identity Authentication

Bo Zhao
School of Computer, Wuhan University, Wuhan 430072, China.

Yu Xiao
School of Computer, Wuhan University, Wuhan 430072, China.

Yuqing Huang
School of Computer, Wuhan University, Wuhan 430072, China.

Xiaoyu Cui
School of Computer, Wuhan University, Wuhan 430072, China.

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 09/09 pp218–225
Volume 22, Number 2, April 2017

A Private User Data Protection Mechanism in TrustZone Architecture Based on Identity Authentication

Bo Zhao, Yu Xiao*, Yuqing Huang, and Xiaoyu Cui

Abstract: In TrustZone architecture, the Trusted Application (TA) in the secure world does not certify the identity of Client Applications (CA) in the normal world that request data access, which represents a user data leakage risk. This paper proposes a private user data protection mechanism in TrustZone to avoid such risks. We add corresponding modules to both the secure world and the normal world and authenticate the identity of CA to prevent illegal access to private user data. Then we analyze the system security, and perform validity and performance tests. The results show that this method can perform effective identity recognition and control of CA to protect the security of private user data. After adding authentication modules, the data operation time of system increases by about 0.16 s, an acceptable price to pay for the improved security.

Key words: embedded system; TrustZone; Trusted Application (TA); identity authentication; private data protection

1 Introduction

As the information processing capacity of mobile terminals rises, more and more data handling tasks are transferred to mobile devices. But the complexity and openness of terminal systems make them targets of malware, such as virus and Trojans, and users' private data is vulnerable to theft and tampering. To address this challenge, ARM proposes a software and hardware architecture, TrustZone, based on secure isolation. It divides the software and hardware resources of the system into two parts, each belongs to one "world": a secure world and a normal world[1]. The system controls resource access authority via bus, so that normal-world processes can only access normal-world resources while secure-world processes can access all resources[2]. The secure world is a trusted execution environment. When processing operations include users' private information and involve security services, this arrangement prevents normal-world access to avoid private-data attacks at run time[3].

However, data protection includes not only dynamic security at run time, but also static security in storage. After processing, the data is stored in a normal-world memory area. The data is far from secure if malicious activities such as data theft and damage in static storage cannot be resisted. Static-data security includes storage-media security, storage-environment security, virus defense, and storage-access security[4]. The first two are hardware issues, while malicious activities using viruses include data theft and data tampering. Aiming at this problem, many researchers have proposed feasible defense schemes base on embedded TrustZone architecture. Wei[5] realized a trusted operating system, T-OS, in TrustZone. They added an encryption module based on various encryption algorithms, and stored the encrypted data in a file system realized in a security chip. Zhao et al.[6] produced a root of trust based on SRAM Physical Unclonable Functions (PUFs) without adding security hardware. They generated a security key to encrypt data using this root of trust, and stored the encrypted data in the system hardware. Hein et al.[7] realized a security block device that depends on there being a trusted root key in the system. Combined with Merkle-Tree and

• Bo Zhao, Yu Xiao, Yuqing Huang, and Xiaoyu Cui are with School of Computer, Wuhan University, Wuhan 430072, China. E-mail: zhaobowhu@163.com; 597260477@qq.com; huang_yuq@163.com; 416658309@qq.com.
* To whom correspondence should be addressed.
Manuscript received: 2016-09-29; accepted: 2016-12-21

authenticated encryption, this method can prevent data tampering, redirection attacks, and reply attacks.

Existing research focuses on the data encryption and storage in TrustZone, but does not consider the data-access security problem. According to the TEE Internal Core API Specification[8], the manipulation of data is controlled by the Trusted Application (TA) in the secure world. Each TA owns a storage area in memory to store the data it creates. This area can only be accessed by its TA, thus avoiding data leakage among different TAs. Nonetheless, a TA does not verify the identity of data-access requesters; a malicious attacker could forge its identity and get users' private data stored by other applications via normal data-access processes.

To solve this problem, this paper proposes a private user data protection mechanism in TrustZone architecture based on identity authentication. We add an identity verification module to a multi-user TrustZone environment, and establish the correspondence between a data object and its owner to verify the identity of each data access requester. This mechanism guarantees the security of data access, and enhances the protection of users' private data.

2 Threat Model

In TrustZone, different Client Applications (CA) in the normal world make data processing requests, and the commands are sent to the underlying driver via a client API. Then they are sent to a TA in the secure world through an SMC instruction call. The TA sends the command to the Internal API, and then calls kernel functions to perform data processing operations.

There is no secure permanent storage area in TrustZone. The system allocates an independent storage area, TA Space, in storage devices of the normal world, such as flash memory, for each TA to store data objects it creates. When creating a data object, a TA allocates an ID for it, and generates a directory in its own TA Space, named for the Object ID. Then it encrypts the object and stores the encrypted data in the new directory in a block. When a CA requests access to a data object, the kernel reads the encrypted data from the normal world into the secure world via its Object ID, and returns the decrypted data to the CA. The overall framework is shown in Fig. 1.

This mechanism guarantees storage security and runtime data security. But the TA does not check the identity of the CA requesting data access, and only

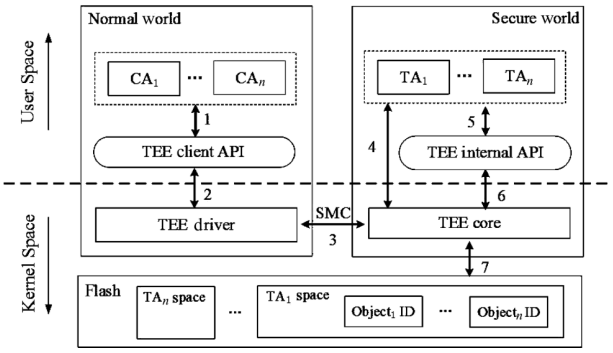

Fig. 1 TrustZone framework.

checks whether the Object ID provided by the CA is legal. There is no security protection scheme for the flash memory that stores objects, and its contents can be accessed by anyone. Thus, an attacker could read the directory name, namely, the Object ID, which stores the object and the request for access to the object to get the private user data illegally. The threat model is shown in Fig. 2.

CA_1 requests for creating new data $Object_1$, then TA_1 allocates $Object_1$ ID and sends the request to the kernel. The kernel creates a directory named after $Object_1$ ID in TA_1 Space of flash and stores the encrypted object in block under the directory. Meanwhile, the attacker CA_2 looks up for the new directory in flash to get the $Object_1$ ID and sends request for accessing $Object_1$ to TA_1. TA_1 checks that the $Object_1$ ID is in TA_1 Space and then loads the encrypted data of $Object_1$ into secure world, decrypts it and then returns the $Object_1$ to CA_2. TA_1 does not check whether $Object_1$ is created by CA_2 or not and thus resulting in the object leakage and the private data of CA_1 is under heavy security threat.

Following the threat model, we performed the attack experiment on a Hikey board. We created two CAs

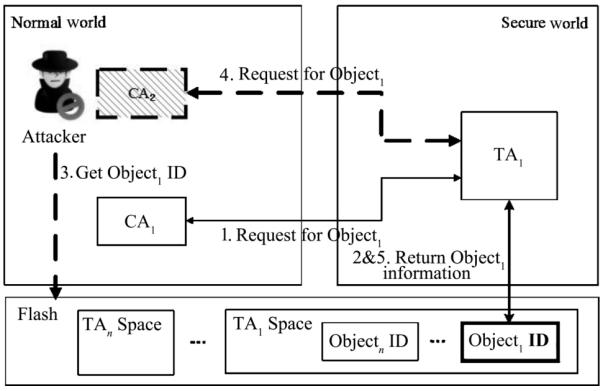

Fig. 2 Threat model of data access in TrustZone.

(CA_1 and CA_2). CA_1 created $Object_1$ whose $Object_{ID}$ was 0000 and content was "this_is_object_access_test". Then CA_1 and CA_2 both requested access to $Object_1$ via $Object_{ID}$ 0000. The results are shown in Figs. 3 and 4.

The results show that both CAs were able to obtain the content of $Object_1$—indicating that the TA does not recognize whether the requesting CA is the creator of the requested Object, and therefore returns the content to all requesters. Therefore, the users' private data is at risk.

3 User-Data Protection Mechanism Based on Identity Authentication

3.1 Identity authentication framework

To protect users' private data, we should authenticate the identity of any CA requesting data access. The authentication process involves interaction among a CA, an Identity Obtaining Daemon, a TA, an Identity Authentication Module, and the access and modification of a Data Authentication Table. The framework is shown in Fig. 5. The modules added to the original TrustZone include the following:

(1) Identity-Obtaining Daemon (IOD): A process to obtain the identity of a CA; it runs in the normal world. We use technologies such as software protection to ensure its security.

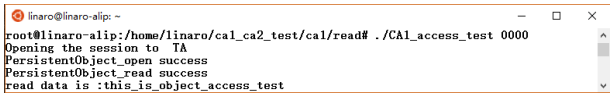

Fig. 3 CA_1 requests access to $Object_1$.

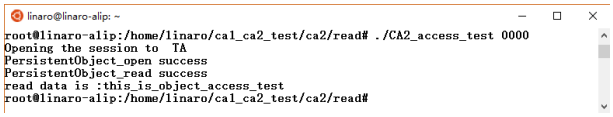

Fig. 4 CA_2 requests access to $Object_1$.

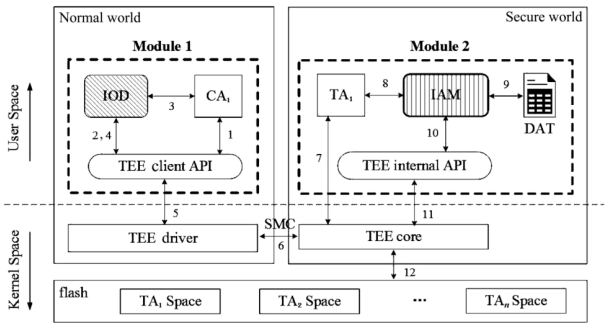

Fig. 5 Identity authentication framework.

(2) Identity Authentication Module (IAM): The core of the CA identity authentication process; it runs in the secure world, and performs access, search, and modify operations on the Data Authentication Table. Its security is guaranteed by the isolated execution environment provided by TrustZone.

(3) Data Authentication Table (DAT): A table that stores the correspondence of the CA, the TA and the data object. It is the key to CA identity authentication.

3.2 CA identity authentication design

CA identity authentication has two steps: CA identity obtainment in the normal world and CA identity authentication in the secure world. Here is a detailed description of these two steps.

3.2.1 Identity obtainment

We must obtain the CA identity before authentication. The normal world is not secure, and there may be attackers pretending to be normal CAs. Thus, the identity is not provided by the CA, but by the newly added IOD.

The framework of the CA identity obtainment module is shown in Module 1 of Fig. 5. When the request command of a CA is transmitted to Client API, Client API determines the command type. If it is related to a data operation, the IOD begins to obtain the CA identity and send it to the secure world. The identity is authenticated in the secure world; the authentication method is explained in Section 3.2.3.

The most important part of the process is getting the identity of CA. CA is a normal process running in the operating system, and has a corresponding image in memory. The image exists in memory before the program ends, and contains information related to the process. Parts of the image, such as the data segment, change dynamically while the program runs; other parts, such as code segments, remain the same during the whole running period. The invariant parts could be used to identify the uniqueness of the process.

In the Linux kernel, the system creates a PID hash table via pidhash_init() during the initiating procedure, and allocates a unique id for each process[9]. With the help of the PID hash table, we could find the process descriptor of a process via its id ($process_{ID}$). The process descriptor data structure task_struct includes a memory descriptor field named mm_struct that describes the data storage of a process. The detailed structure is shown in Fig. 6.

The address space of a process can be accessed

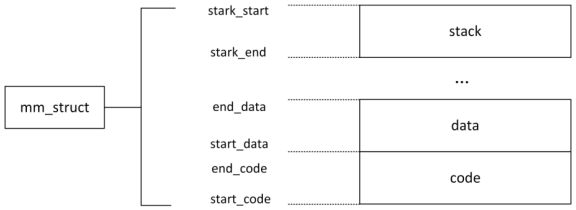

Fig. 6 Data storage structure of a process in memory.

by mm_struct[10]. In that space, [start_code, end_code) indicates the address of the code segment, and we can obtain its value. The CA identity is based on this value. Here is how this process works:

Step 1 IOD receives the $\text{process}_{\text{ID}}$ sent by Client API, and finds its process descriptor task_struct in the process list.

Step 2 IOD jumps to the mm_struct via task_struct of the process, and obtains the basic addresses of the process, such as for a code segment, a data segment, or a parameter segment. The addresses are linear.

Step 3 IOD transforms the linear address into a physical address and gets the content. We choose the code segment, which remains unchanged during the running period, to compute the CA identity.

Step 4 IOD computes the hash value of the code and signs the value and timestamp with a private key. Then it connects the signature and hash value as the CA identity CA_{IDE}, and sends it to Client API.

3.2.2 Identity authentication

In the secure world, we need to modify some components to authenticate the CA identity. TA receives the SMC command from the normal world, while Internal API calls the kernel to activate real functions. Therefore, we added an Identity Authentication Module (IAD) between the two components to effect identity authentication. The framework of the CA identity authentication module is shown in Module 2 of Fig. 5.

IAM runs in the secure world; its security is guaranteed by the trusted execution environment isolated by TrustZone. This module performs two functions: one is to receive requests from the normal world delivered by TA, and check the validity of the timestamp and hash value with CA_{IDE}; the other is to check whether the requested object is owned by the requesting CA by looking it up in the DAT.

A key component in the process is the DAT. Below, we describe its design, maintenance, and security guarantee.

(1) Design of the DAT

The DAT must maintain the correspondence of the CA and the object to perform the CA authentication. Therefore, the CA hash value and the object ID must be stored in the table. TrustZone provides independent storage space for each TA to store the objects created by it. Thus, the DAT must store the TA ID to make it possible to locate the TA when searching the table. After locating the TA and the object, the process must check whether the CA hash value in the table is the same as the requesting CA hash value to decide whether to allow access to the requested object. At the same time, a flag is set in the table whose function is described below. The design of the DAT is shown in Table 1.

(2) Maintenance of the DAT

The DAT is stored in the normal world, and loaded into the secure world when system boots. Its contents are updated in real time once there is data operation in the secure world. Alterations should be synchronized with normal-world storage. According to the object operation functions in the TEE Internal Core API Specification, the following processes involve modification of the DAT:

(a) Object creation. When creating a new object, we create a new entry in the DAT and store the related TA_{ID}, $\text{Object}_{\text{ID}}$, and CA_{hash}, and set the flag to 1. The request is sent to the Internal API. If the creation succeeds, the flag is set to 0 to indicate the validity of the entry; else, the entry is deleted.

(b) Object renaming. When there is an object renaming request, IAM checks if the $\text{Object}_{\text{ID}}$ matches the requesting CA_{hash}. If it matches, the request is passed to the Internal API. If the request completes successfully, IAM modifies the $\text{Object}_{\text{ID}}$ in the table to the new ID.

(c) Object deletion. When there is an object deletion request, the IAM authenticates the identity of the requesting CA. If the requested object belongs to the

Table 1 Design of Data Authentication Table.

Trusted application ID	Requested object ID	Client application ID	Flag
TA_{ID1}	$\text{Object}_{\text{ID1}}$	CA_{hash1}	Flag_1
	$\text{Object}_{\text{ID2}}$	CA_{hash1}	Flag_2
	$\text{Object}_{\text{ID3}}$	CA_{hash2}	Flag_3
	...	...	...
TA_{ID2}	$\text{Object}_{\text{ID4}}$	CA_{hash3}	Flag_4
	...	...	...
...	...	...	...

CA, the request is sent to the Internal API. If the object is successfully deleted, the related entry in the IAM is deleted.

3.2.3 Identity authentication process

After adding identity authentication modules to TrustZone, we can determine the identity of a CA requesting data access, and ensure that only the CA that created the object can access it. The process, from a CA requesting data access, to the secure world returning a result, is shown in Fig. 7.

Step 1 CA → Client API: REQ($\text{Object}_{\text{ID}}$). CA sends data operation REQ that includes requested $\text{Object}_{\text{ID}}$ to TEE Client API. Client API determines the operation type and if it is data related, execute Client API → IOD: $\text{process}_{\text{ID}}$, or execute Client API → TA: REQ($\text{Object}_{\text{ID}}$).

Step 2 IOD finds the code segment content of CA according to $\text{process}_{\text{ID}}$, and gets its hash value CA_{hash}. CA_{IDE}=CA_{hash}+Sign(CA_{hash}+timestamp). Then IOD signs the CA_{hash} and timestamp with a private key, and connects the signature and CA_{hash} as CA_{IDE}, and returns it to TEE Client API.

Step 3 Client API → TA: REQ($\text{Object}_{\text{ID}}$, CA_{IDE}). Client API packages the CA_{IDE} and data operation request REQ and sends them to TA in the secure world via SMC.

Step 4 TA → IAM: REQ($\text{Object}_{\text{ID}}$, CA_{IDE}, TA_{ID}). TA receives the data from the normal world and sends it to IAM together with its own identity TA_{ID}. IAM separates the CA_{hash} and the signature, and verifies the signature to get timestamp and CA'_{hash}. Then IAM checks the validity of timestamp, and then checks if CA'_{hash} is equal to CA_{hash}. If all the checks are successful, it indicates that the data came from the rightful IOD and is credible.

Step 5 Check if($\text{Object}_{\text{ID}} \in \text{CA}_{\text{hash}}$). IAM searches the DAT and gets the CA_{hash} of the requested Object. Then IAM checks if CA_{hash} is the same as the requester CA_{hash}. If it is not, this shows that the object was not created by the CA, and the access request is then denied.

Step 6 IAM sends $\text{Object}_{\text{ID}}$ and TA_{ID} to the Internal API. The API calls the kernel to the requested data operation. Then IAM modifies the DAT according to the returned result, and sends the result to TA. Finally, the result is delivered to the CA.

3.3 Security analysis

To analyze the security of our approach, we focus on the newly added modules: IOD, IAM, and DAT. Since TrustZone architecture provides an isolated environment and guarantees the credibility of the secure world, we consider the IAM, which runs in secure world, secure, and offer a detailed security analysis of the other two modules.

3.3.1 IOD

The IOD produces asymmetrical keys, and uses the private key to sign data while delivering the public key to the secure world for verification. It is the subject of the identity obtainment process, and owns a private key, so it is of great importance to guarantee its security. In the current study, technologies such as shell adding[11], process isolation[12], and software protection[13] could prevent monitoring and tampering by attackers. There have already been some effective implementation schemes. However, most of them involve complex memory protection or process isolation, and are not the key points of this paper. Thus, we consider the IOD to

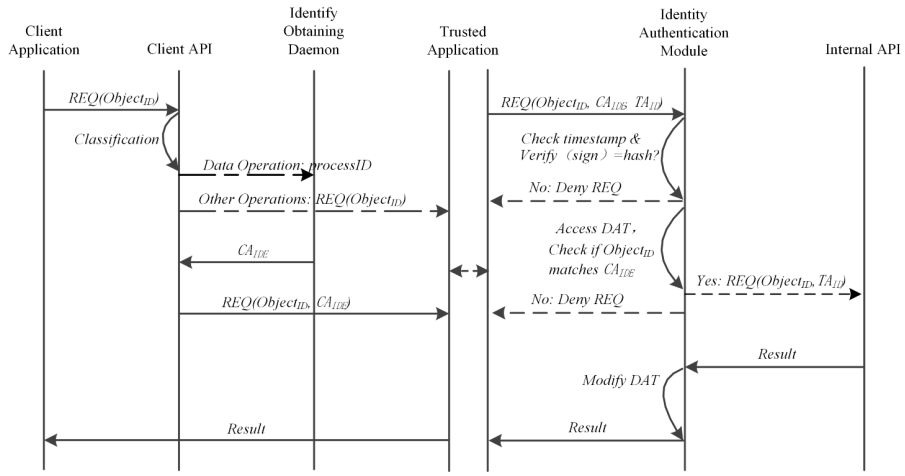

Fig. 7 Process of CA identity authentication.

be secure, and do not give a detailed description of its protection method.

3.3.2 DAT

TrustZone does not provide secure storage. To ensure security, private data is usually encrypted before storage. This paper adopts the widely used embedded Multi-Media Controller (eMMC) in embedded devices to guarantee the security of the DAT. eMMC refers to a package consisting of both flash memory and a flash-memory controller, integrated on the same silicon die[14]. It offers a Replay-Protected Memory Block (RPMB) partition that requires an authentication key for access[15]. We refer to Ref. [6], and produce a root of trust based on SRAM PUFs; we use it as the authentication key. We also refer to some new encryption schemes to increase the security and efficiency of the encryption process[16–19]. The DAT is stored in an RPMB partition. Since an attacker cannot get the authentication key, its static storage security is guaranteed. When the system boots, the DAT is loaded into the secure world, where its dynamic running security is guaranteed.

4 Experiment and Analysis

We built the private user data protection mechanism based on identity authentication in a Hikey board. First, we created the experimental environment. Then we performed a validity test. Finally, we analyzed the system performance after adding our new modules.

4.1 Experimental environment

In the experiment, we cross-compiled an executable program for a Hikey board in the Ubuntu 14.04.3 system.

The Hikey board is based on a HiSilicon Kirin 620 64-bit eight-core 1.2 GHz ARM Cortex-A53 CPU, with 2 GB LPDDR3 SDRAM and 8 GB eMMC Flash storage. The secure world ran in a trusted OS, the open source OP-TEE[20], and the normal world ran in a Debian system.

4.2 Validity analysis

According to the threat model proposed in Section 2, the validity test verifies whether a CA could access an object that was not created by it. Attacker CA_2 gets the $Object_1$ ID, and sends a request to access $Object_1$. IOD gets the hash value of the requesting CA (CA_{hash2}), and sends it to the secure world. IAM in the secure world searches the DAT, and gets the creator CA hash value of $Object_1$ (CA_{hash1}). The two hash values are different, so IAM denies the data request of CA_2.

We did the contrasting experiment on a Hikey board to verify the effect of the proposed mechanism. Similar to the experiment in the threat model, we created two CAs (CA_1 and CA_2). CA_1 created $Object_1$ whose $Object_{ID}$ was 0000. Then CA_1 and CA_2 both requested access to $Object_1$. The results are shown in Figs. 8 and 9.

In Fig. 8, the upper and lower parts show the output of the normal world and the secure world of TrustZone, respectively. CA_1, the creator of $Object_1$, requested access to it via its $Object_{ID}$ 0000. The secure world verified the signature and timestamp, and searched the DAT. It checked that the CA_{hash} of $Object_1$ is CA_1, and approved the request. Finally, the secure world returned the content of $Object_1$ to CA_1. In Fig. 9, CA_2 made the same request, but $hash_{CA2}$ did not match the hash value in the DAT, so the secure world denied the object request.

The result shows that only the creator CA of the Object could access it; this indicates the mechanism can determine the identity of a requester, and thus control access to an object. This ensures that only the owner of data can access it, and thereby protects users' private data.

4.3 Performance analysis

After validating the functionality of our approach, we

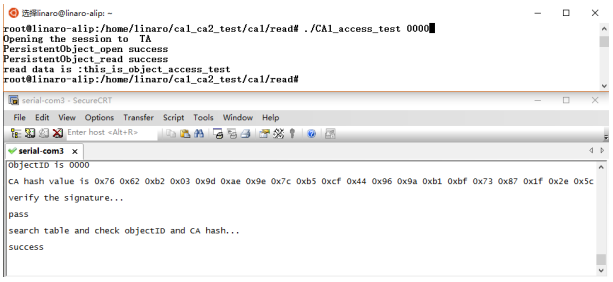

Fig. 8 CA_1 **requests access to** $Object_1$**.**

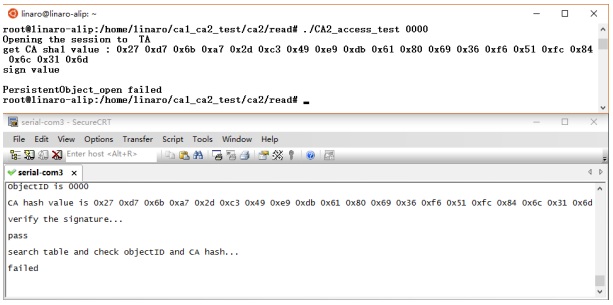

Fig. 9 CA_2 **requests access to** $Object_1$**.**

tested the performance of the data accessing operations after adding modules to TrustZone. Object creation involves the entire process: read the code segment content; compute a hash value and sign in the normal world; verify the signature; create an object, and modify the DAT in the secure world. We performed the object creation contrast test in the unmodified system and in the modified system. To eliminate the effects of other factors, we did not perform other data operations during the experiment. We added a time recording function at the start and end of the process, and printed out the time lag. The experiment was repeated 10 times; the results are shown in Fig. 10.

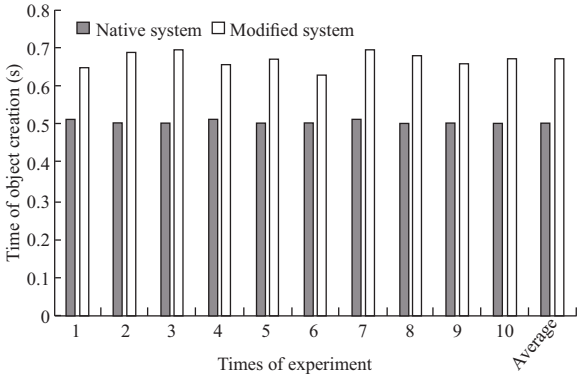

Fig. 10 Results of performance test.

Considering the original object creation time and added system security, the threshold of acceptable increased time is 35%. The result shows that after adding authentication modules to the system, the average object creation time increased by 0.159 67 s. We used 1024-bit RSA for signing; the time consumed by signing and verifying was 0.0421 s and 0.016 s, respectively. Other operations consumed 0.101 57 s. The overall time increase was about 30%, lower than the threshold. This indicates the mechanism is useful and practical.

5 Conclusion

This paper proposes an identity authentication mechanism based on the secure isolation environment provided by TrustZone to solve the problem that the data access requester CA is not verified in the secure world. The mechanism determines the identity of the data access requester, and thus guarantees data access security. We performed some experiments whose results show that this mechanism can prevent illegal data access in a system. The modules we added had little impact on data operation performance, and offer reference value to the TrustZone implementation. At present, the communication between the normal world and the secure world employs the default SMC command. We consider studying the communication protocol to ensure the security and effectiveness of communication in future work.

Acknowledgment

This work was supported by the National High-Tech Research and Development (863) Program (No. 2015AA016002), the National Key Basic Research Program of China (No. 2014CB340600), the National Natural Science Foundation of China (Nos. 61303024 and 61272452) and the Natural Science Foundation of Jiangsu Province (Nos. BK20130372).

References

[1] T. Alves and D. Felton, TrustZone: Integrated hardware and software security, *ARM White Paper*, vol. 3, no. 4, pp. 18–24, 2004.

[2] GlobalPlatform Inc., TEE system architecture v1.1, https://www.globalplatform.org/specificationsdevice.asp, Jan. 2017.

[3] GlobalPlatform Inc., The trusted execution environment: Delivering enhanced security at a lower cost to the mobile market, http://www.globalplatform.org/documents/GlobalPlatform_TEE_White_Paper_Feb2011.pdf, Feb. 2011.

[4] J Hughes, IEEE standards for encrypted storage, *Computer*, vol. 37, no. 11, pp. 110–112, 2004.

[5] L. Wei, Secure storage based on ARM TrustZone research and implement, master's dissertation, School of Information and Software Engineering, University of Electronic Science and Technology, Sichuan, China, 2015.

[6] S. Zhao, Q. Zhang, G. Hu, Y. Qin, and D. Feng, Providing root of trust for ARM TrustZone using on-chip SRAM, in *Proc. 4th Int. Workshop on Trustworthy Embedded Devices*, New York, NY, USA, 2014, pp. 25–36.

[7] D. Hein, J. Winter, and A. Fitzek, Secure block device–secure, flexible, and efficient data storage for ARM TrustZone systems, in *Proc. 2015 IEEE Trustcom/BigDataSE/ISPA*, New Yrok, NY, USA, 2015, pp. 222–229.

[8] GlobalPlatform Inc., TEE Internal Core API Specification v1.1.1, https://www.globalplatform.org/specificationsdevice.asp, June 2016.

[9] W. Mauerer, *Professional Linux Kernel Architecture*. John Wiley & Sons, 2010.

[10] Z. Liu and D. Feng, TPM-based dynamic integrity measurement architecture, (in Chinese), *Journal of Electronics and Information Technology*, vol. 32, no. 4, pp. 875–879, 2010.

[11] Y. Luo, Z. Wang, and X. Jia, Research of shell software technology based on Linux, (in Chinese), *International Electronic Elements*, vol. 20, no. 10, pp. 13–15, 2012.

[12] B. Zhao, Z. Xia, Y. An, and S. Xiang, Research and implementation of process isolation under virtualization environment, (in Chinese), *Journal of Huazhong University of Science and Technology: Natural Science Edition*, no. 11, pp. 74–79, 2014.

[13] Y. Yang, Research of software protection on Android platform, master's dissertation, College of Information Engineering, Beijing University of Posts and Telecommunications, Beijing, China, 2014.

[14] Datalight. What is eMMC, http://www.datalight.com/solutions/technologies/emmc/what-is-emmc, 2015.

[15] S. Thom, J. Cox, D. Linsley, M. Nystrom, H. Raj, D. Robinson, S. Saroiu, R. Spiger, and A. Wolman, TrustZone-based integrity measurements and verification using a software-based trusted platform module, US Patent US20160048678A1, Feb. 18, 2016.

[16] H. Wang, H. Zhang, and S. Tang, Key recovery on several matrix public-key encryption schemes, *IET Information Security*, vol. 10, no. 3, pp. 152–155, 2015.

[17] H. Wang, H. Zhang, Z. Wang, and M. Tang, Extended multivariate public key cryptosystems with secure encrytion function, *Science China Information Science*, vol. 54, no. 6, pp. 1161–1171, 2011.

[18] H. Wang, H. Zhang, Z. Xu, and H. Zhang, Multivariate public key encryption scheme based on error correcting codes, *China Communications*, vol. 8, no. 4, pp. 22–31, 2011.

[19] W. Wu, H. Zhang, H. Wang, S. Mao, J. Jia, and J. Liu, A public key cryptosystem based on data complexity under quantum environment, *Science China Information Science*, vol. 58, no. 11, pp. 44–54, 2015.

[20] STMicroelectronics and Linaro Security Working Group. OP-TEE, https://github.com/OP-TEE, 2017.

Bo Zhao received the PhD degree in information security from Wuhan University in 2006. He is currently a professor in Wuhan University and his research interests include information security and trusted computing.

Yu Xiao received the BS degree from Wuhan University in 2014 and is currently working toward the MS degree in Wuhan University. Her research interests include information security and trusted computing.

Yuqing Huang received the BS degree from Jilin University in 2012 and is currently working toward the MS degree in Wuhan University. His research interests include information security and trusted computing.

Xiaoyu Cui received the BS degree from Wuhan University in 2014 and is currently working toward the MS degree in Wuhan University. Her research interests include information security and trusted computing.