Security Access Solution of Cloud Services for Trusted Mobile Terminals Based on TrustZone

Hui Xia[1] and Weiji Yang[2]
(Corresponding author: Weiji Yang)
Shenyang Normal University, Shenyang 110034, China[1]
Zhejiang Chinese Medical University, HangZhou 310000 , China[2]
(Email: yangweiji@163.com)
(Received Oct. 15, 2018; Revised and Accepted May 17, 2019; First Online Sept. 21, 2019)

Abstract

Trusted cloud architecture provides secure and trustworthy execution environment for cloud computing users, which protects the private data's computing and storage security. However, with the rapid development of mobile cloud computing, there is currently still no secure solution for mobile terminals accessing trusted cloud architecture. Aiming at the above issues, a secure access scheme of cloud services for trusted mobile terminals is proposed. The program fully considers the background of mobile cloud computing applications, uses ARM TrustZone hardware-based isolation technology to build a trusted mobile terminal that could protect cloud service customers and security-sensitive operations on the terminal from malicious attacks. Physical unclonable function (PUF), the key and sensitive data management mechanism is put forward. The secure access protocol is designed based on the trusted mobile terminal and by employing trusted computing technology. The protocol is compatible with trusted cloud architecture and establishes end-to-end authenticated channel between cloud server and the mobile client. Six security properties of the scheme are analyzed and a scenario-based mobile cloud storage example is presented. Finally a prototype system is implement. Experimental results show that the proposed scheme has good expandability and secure controllability. Moreover, the scheme achieves small TCB((trusted computing base) for mobile terminal and high operating efficiency for cloud users.

Keywords: Mobile Cloud Computing; PUF; Secure Access;Trusted Computing; TrustZone

1 Introduction

With the rapid development of cloud computing technology, mobile terminal equipment, international mobile communication technology and mobile internet applications, the concept of mobile cloud computing (MCC) are gradually affecting people's daily life. International Mobile Cloud Computing Forum [4] and Intel Aepona [3] give the relevant definition, mobile cloud computing is a comprehensive technology for mobile terminal devices to outsource data processing and data storage to resource-rich computing platforms through mobile cloud applications. Mobile cloud computing can be effective reduce the cost of computing resources, storage resources and electricity and can enhance the usability of complex applications in mobile terminals [2]. Mobile cloud computing presents software as a service's (referred to as SaaS) level [10] to the users, users use mobile devices to thin client software or web browser through a wireless network to access remote cloud services. In recent years, mobile cloud computing promotion areas include cloud office, cloud mail, cloud storage, cloud payment, cloud games and cloud video. Companies have also launched corresponding support technologies and products for mobile cloud computing, including Apple's iCloud, Google's Cloud Console, Microsoft's OneDrive and Amazon's App-Strea,which have greatly improved the convenience of mobile users to experience cloud services. The main contributions of this paper are as follows:

- For the mobile cloud computing scenario, a method based on TrustZone technology to build a trusted mobile terminal is proposed to ensure the security and reliability of the cloud service client program and related sensitive operations in the mobile terminal.
- A key and sensitive data management mechanism based on PUF technology is proposed. This mechanism cooperates with TrustZone technology to provide a trusted root function for trusted mobile terminals and cloud service security access.
- A trusted mobile terminal cloud service security access protocol is proposed, which uses trusted computing technology to establish end-to-end bi-directional authentication channel between the cloud server and the mobile client to protect user data and cloud service access requests's authentication, confidentiality

and integrity in the access' process. The protocol is compatible with the trusted cloud architecture.

Section 1 of this paper discusses the work in this area. Related work and some research achievement about the topic is discussed in Section 2. Section 3 presents preliminary knowledge about key and sensitive data management for trusted mobile terminal cloud.Section 4 expatiates on the design of secure access scheme for trusted mobile terminal cloud: trusted mobile terminal architecture, and cloud service security.Example and Scheme evaluation based on the system is given in Sections 5 & 6. Section 7 summarizes the full text and looks forward to future research.

2 Related Work

In order to solve the pro blem of information security from the underlying computer hardware, the concept of trusted computing has been proposed and popularized in scientific research and industry. Trusted Computing Group (TCG) has launched a TPM (Trusted Platform Module) security solution for x86 hardware platform. Trusted platform module(referred to as TPM), the TCM's main TPM1.2 specification [16], was revised several times in 2009 to receive the ISO in 2009. In 2013, TCG officially released a new security solution TPM2.0 standard [17]. In China, the National Cryptology Authority in 2007 proposed a trusted cryptography module (TCM) [19] with independent intellectual property rights and related interface specifications. As a basic security technology, an important application scenario of trusted computing is to construct a trusted virtualization platform. The virtual trusted platform module(vTPM [6]) can be used to protect the security of virtual machine monitor. TrustVisor [14] provides trusted services for isolated code by creating a virtual TPM instance . Because of the advantages of security isolation, security intervention and data protection, infrastructure virtualization technology is widely used in cloud computing architecture, and it is a hot research area in recent years to build trusted cloud computing environment by trusted virtualization technology. Literature [15] outlines the concept of trusted cloud computing platform (TCCP), which provides a closed operating environment for user virtual machines by extending the functionality of the trusted platform to the cloud infrastructure, thus user data confidentiality and integrity can be effectively protected.Wuhan University Professor Zhao Bo et al. [23] summarizes the trusted cloud computing environment to build the technical methods and challenges. TrustCloud [1] designed a framework for trust building and security auditing for cloud computing. Cloud Terminal [9] uses a trusted authentication method to outsource the data-processing security of user-sensitive applications to the cloud service provider, where the user's local host only displays the interface. CloudProxy [8] uses trusted computing technology to establish an end-to-end trusted connection between the cloud host and the user's host to protect the security of user data during transmission and cloud operations. However, the above-mentioned building methods of trusted cloud computing environment are designed for x86 hardware platform. How to access the trusted cloud environment safely and effectively by mobile terminal equipment is still a problem to be solved.

In the rapid development of mobile cloud computing today, mobile cloud computing security has attracted more and more people's attention. Related research [7,13] pointed out that: the mobile user's private data in the mobile terminal, cloud host and communication channel on the confidentiality and integrity, are the key to mobile cloud computing security. Trusted virtualization technology and cloud architecture can protect user's data in the cloud host, but lack of suppling and supporting the mobile terminal and mobile network communications in the protection of user data and aim at cloud environment for the design of trusted solutions. Literature [20] gives a trusted security isolation method based on mobile operating system access control strategy, which is based on the premise of mobile operating system security. However, the successful use of Android system vulnerabilities in the implementation of the attack is endless, the operating system itself does not provide high-intensity security. For the research of trusted mobile terminals, the Mobile Trusted Module (MTM) specification [18] has been released for mobile terminals, but it is not promoted in the mobile industry due to the need to rely on additional hardware modules, and the specification has not been promoted in the mobile industry.

3 Problem Statement and Preliminaries

3.1 Root Key Seed Extraction

In this paper, the literature [24] proposed SRAM (Static Random Access Memory) PUF technique to extract the root key seed S, S is a piece of unique bit string randomly selected by the mobile terminal manufacturer M in the production process of the device, M uses the physical characteristics of the SRAM-specific area in the mobile terminal T to store S, therein. S is only reproduce from the SRAM PUF component every time T is normally powered up and is safely cached by the key manager in the SW (Secure World)). S confidentiality is strictly protected by TrustZone.

3.2 Key Derivation

In the mobile terminal SW, KDF (key derivation function) is the key manager of the trusted service and has a key generation function, which is a deterministic mapping: $\widetilde{S} \times \widetilde{P} \longrightarrow \widetilde{K}$, where $\widetilde{S}$ is the key seed space, $\widetilde{P}$ is a set of string parameters for declaring the usage of the key, and $\widetilde{K}$ is the space for generating the keys, using the KDF and the root key seed S, a public private

key pairs (dpk_T, dsk_T) that uniquely identify the identity of the mobile terminal can be generated, the generation method is:

$$(dpk_T, dsk_T) \longleftarrow KDF_s(\text{"}identity\text{"}).$$

Similarly, srk (Storage Root Key) can be generated in the form $srk \leftarrow KDFs(\text{"}storage\_root\text{"})$. srk is used to further generate a storage key to store and protect the actual sensitive data, and this set of storage key can enhance the isolation and security. It is worth emphasizing that the private keys of all storage keys and device keys generated here never leave the SW and are not stored on the nonvolatile memory of the mobile device. If needed, they will be used in the same way as the KDF way refactoring, which can reduce the risk of key loss.

3.3 Sensitive Data Management

A variety of storage keys derived from srk encapsulate and store the public key apk of the application service provider A and the key package $(ID, k^{Enc}, k^{MAC}, n_i)$ is required for the cloud service session. The key data's specific meaning and usages will be described in detail in Section 4.1. The encapsulation operation is performed in the data processor of the SW trusted service. The data processor implements the data encapsulation function $Data\_Seal()$. The encapsulated data block can be stored in the public nonvolatile memory of the device. In this paper, $MAC_k(m)$ represent the calculation of the message authentication code for the data m by using the key k; $Enc_k(m)$ means that the data m is encrypted with the key k, and the symmetric and asymmetric encryption can be expressed according to the type of k; $Sign_k(m)$ represents the signature operation; $\|$ indicates the connection of the data. The following are the specific encapsulation methods:

- For the public key apk, the encapsulation only needs to protect the integrity of the public key to prevent mobile applications are malicious tampering caused public key damage, the steps are as follows:

$$\begin{aligned} mk_{apk} &\longleftarrow KDF_{srk}(\text{"}storage\_key\text{"}, \text{"}MAC\text{"}, apk), \\ blob_{apk} &\longleftarrow Data\_Seal(\text{"}MAC\text{"}, mk_{apk}, apk); \end{aligned}$$

  where KDF_{srk} means storage_root from key derivation function, among them,

$$blob_{apk} = apk \| MAC_{mk_{apk}}(apk).$$

- For the key package $(ID, k^{Enc}, k^{MAC}, n_i)$, when packing,we need to protect their confidentiality and integrity to prevent the rival's theft or tampering, the steps are as follows:

$$\begin{aligned} (sk_{ID}, mk_{ID}) &\longleftarrow KDF_{srk}(\text{"}storage\_key\text{"}, \\ &\qquad \text{"}Enc + MAC\text{"}, ID), \\ blob_{ID} &\longleftarrow Data\_Seal(\text{"}Enc + MAC\text{"}, sk_{ID}, \\ &\qquad mk_{ID}, (ID, k^{enc}, k^{MAC}, n_i)); \end{aligned}$$

$$\begin{aligned} blob_{ID} = \; & Enc_{sk_{ID}}(ID, k^{Enc}, k^{MAC}, n_i) \| \\ & MAC_{mk_{ID}}(Enc_{sk_{ID}}(ID, k^{Enc}, \\ & k^{MAC}, n_i)). \end{aligned}$$

With the storage key refactored in the key manager, the data processor can call the Data_Unseal() function to recover and validate the sensitive data from the corresponding data block.

4 Design of Secure Access Scheme for Trusted Mobile Terminal Cloud Service

4.1 Trusted Mobile Terminal Architecture

With TrustZone and PUF technology, we designed a trusted mobile terminal architecture for cloud computing scenarios. On the basis of the existing mobile terminal hardware architecture, our trusted terminal program is based on software design and implementation as a focus, targeting low cost, flexibility and scalability. Figure 1 shows the proposed trusted mobile terminal architecture and the interaction between the various components of the details in this paper.

Using the method given in literature [21], it is possible to construct TEE (Trusted Execution Environment) safely and efficiently in the TrustZone SW, The TEE implemented in the SW is physically isolated from the universal mobile system environment implemented in the NW (Normal World), running a custom TEE OS in OS to executing security-sensitive program code. There is a Universal Mobile OS running on the NW, which can be an Android or iOS system capable of performing regular mobile applications, and the functions of each component described in details below.

1) Trusted agent: The trusted agent interacts directly with the mobile application in the NW. The component receives a trusted service request from the mobile application, assembles the command for calling the trusted service component in the SW (Secure World) according to the request type, and prepares for the substantial security operation in the SW. The component contains the following two subcomponents:

   - Software stack: Mobile applications to provide high-level trusted service interface, responsible for resolving the application request data, and return the results of service response;
   - Command caller: Assembling the trusted service invocation command, interacting with the trusted service component in the SW, and transmitting the command through the canonical Global Platform TEE client API [5], requesting the NW to switch to the SW by means of

the NW underlying driver and wait for the data to return.

2) Trusted service: Trusted service component is a core component of Trusted Mobile Terminal, which not only realizes Trusted Computing related functions, such as Trust Root Rendering, Key and Sensitive Data Management and Trusted Environment Authentication, but also implements the security crediting protocol execution in mobile terminal Logic. The code execution of this component is protected by the TrustZone quarantine mechanism and consists of the following five subcomponents:

   - API functions: Receive trusted service requests from trusted agents in the NW, parse command data, pass operational instructions to the logic engine, and wait for the results to return to the trusted agent;
   - Key manager: Use the root key seed extracted from the SRAM PUF to produce a variety of cryptographic keys and provide the key to the data processor for user;
   - Data handlers: In order to prevent adversaries from forging security parameters (usually user names and passwords), the data processor receives only the parameter input from the mobile application trusted cell in the SW and passes the parameters to the logic engine. In addition, the subcomponent is also responsible for the encapsulation and de-encapsulation of sensitive data, encapsulated data can be stored in the general non-volatile memory of the mobile device;
   - Crypto library: It provides cryptographic algorithm support for key manager, data processor and logic engine, which implements symmetric and asymmetric encryption,decryption and signature verification algorithms and a variety of message digest algorithms.
   - Logic engine: Obtains the necessary parameter input from other sub-components. According to the designed security access protocol logic, it performs the security-sensitive trusted service operation of the mobile terminal and outputs the execution result. In addition, the subcomponent implements the load measurement and start-up control of the application process in SW.

3) Mobile application (App) and App trustlet: When the mobile user wants to access the cloud services at C, whether it is browser or client mode, you need to start the appropriate mobile applications. The mobile application provided by the application service provider A comprises two parts: a mobile application running in the NW and a mobile application trustlet running in the SW. App only provides the user with a graphical user interface (GUI) and basic non-security-sensitive functions, App trustlet is responsible for collecting and pre-processing sensitive data information needed to access the cloud service and presenting it to the trusted service for operation of the secure access protocol. When the App needs to access the cloud service through the implementation of secure access protocol, it calls trusted agent software stack to make trusted service requests. After TrustZone uses the system interrupt to complete the NW to SW switchover, the Trusted Service will load the startup App trustlet whose code integrity is measured by the logical engine of the trusted service. Based on our previous study work [22], once the App trustlet is found to have been tampered with by an adversary using the whitelist mechanism in the SW, it can be disabled. When the App trustlet is properly started, the user can send cloud service user name and password and other sensitive data information into App trustlet in the security mode, and then hand over to the trusted service for processing. The App implements the communication with the App trustlet through the inter-domain communication mechanism [12] provided by TrustZone, where the mobile application design conforms to the current TrustZone's normal application mode.

4) Components in the kernel: In SW's TEE operating system kernel, there is a drive component SW-Driver; in the NW mobile operating system kernel, there is a drive component NW-Driver. The above two driver components are used to handle the request and response commands of the two world switching in the TrustZone, which contains the communication data of the two. As an implementation of the security monitor defined by TrustZone, the monitor is located in the system kernel of the SW, which controls the underlying hardware to perform the specific actions of the TrustZone world switch. In addition to these special components, the OS kernel implementation in the NW has a variety of generic hardware's driver, including network communication drivers, which is relied by the data communication between the trusted mobile terminal and the cloud service.

5) Components in the hardware: Trusted mobile terminal hardware support ARM TrustZone extension technology, by the protection of the technology, the SRAM PUF physical components located in the hardware can only be accessed by the SW, and the software algorithm of the PUF is implemented by the key manager of the trusted service.

4.2 Cloud Service Security Access Protocol

The interactive participant entities of the cloud security access protocol are T, A, and C, and an overview of the protocol implementation is shown in Figure 2 under normal circumstances. In a certain period of time, when T

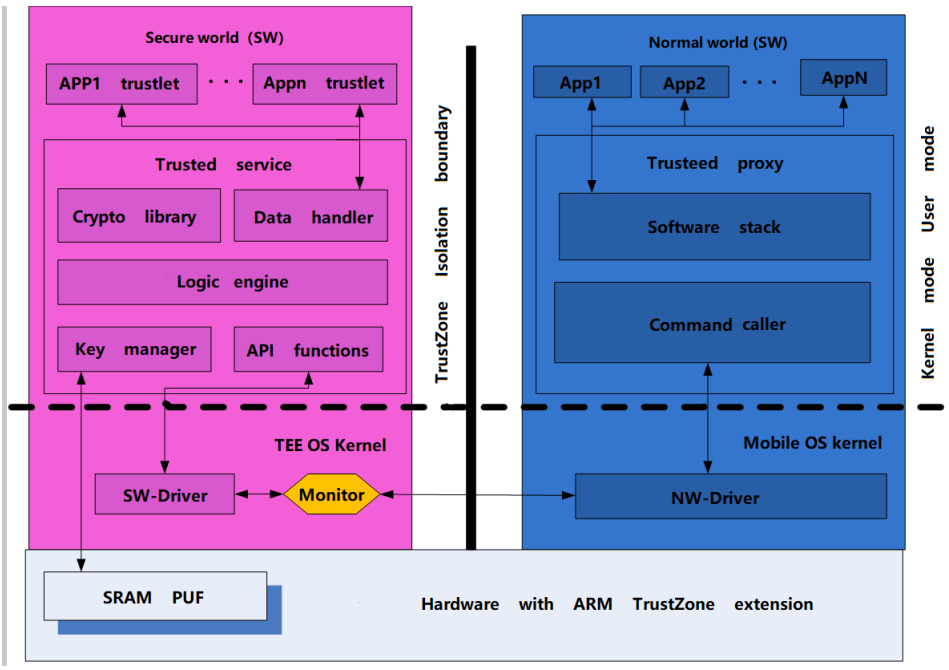

Figure 1: Architecture of trusted mobile terminal for cloud computing

first accesses a cloud service located at C, it first sends an authorization request for access service to A; after charging and legitimacy authentication, A generates a session key package and issues it to T; at the same time, A will be certified to the user data sent to C through the security channel, here, we do not distinguish the user management host in C and service operations host; after obtaining the authorized session key package, T sends the service access request to C by using the relevant key and the authentication information; after C verified the request, the verification results will return to T; then complete access authentication, T and C start a normal cloud service interaction. The secure access protocol we provide can be interfaced with the trusted cloud architecture to achieve bidirectional authentication between the T and C secure execution environments. This paper assumes that C adopts the trusted cloud architecture proposed in literature [8], when returning the T service access request verification result, C will attach the integrity metric of the cloud service program from cloud host.

The secure access protocol with the trusted mobile terminal as the core consists of 4 parts, authorization application, access request, authentication response and authorization revocation. Among them, the authorization application is only executed in three cases: (1) The first time users use T to request access to cloud services; (2) The last authorization application has expired; (3) Due to network error or malicious attack, authorization was revoked. After successfully executing the authorization request, the user can use T to request access to the cloud service several times within a certain period of time. The access request and authentication response of the protocol can be executed multiple times.

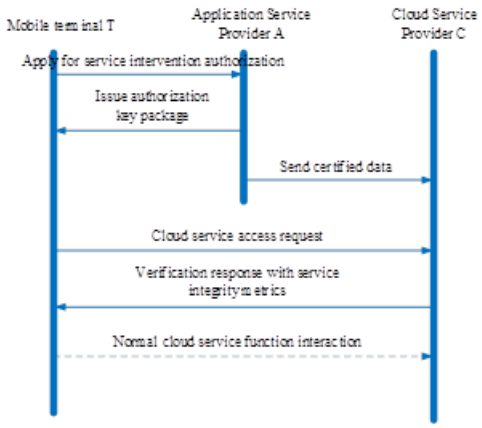

Figure 2: An overview of secure access protocol under normal condition

4.2.1 Application for Authorization

In this part of the agreement, the user using T sends a cloud service access authorization request to A, and A verifies the relevant parameters in the application. After determining the legality of the T and its users, generate a secret for the future conversation between T and C. The key package is sent to both parties, as follows:

1) The user operates the App in NW to request access to the cloud service, TrustZone switches to SW, and T calls KDF to generate the message integrity protection key mkauth, which is used to protect the integrity of data communication when A sends a session key package to T. The key generation method is as follows:

$$mk_{auth} \longleftarrow KDF_s(\text{"}session\_key\text{"}, \text{"}MAC\text{"}, r),$$

among them, r is the key generated by the key manager for generating different mk_{auth}.

2) When T loads the launch App trustlet in the SW, it performs integrity metrics on the loaded code, and uses the hash function to get the metric $\mu(app)$. Based on our whitelist mechanism [12], we can find the tampering of the App trustlet, if tampered, the agreement will terminate execution.

3) The user input user name user and password pswd of the login cloud service to the SW's App trustlet, and the App trustlet calculates the password hash value $H(pswd)$, and sends it to the data processor of the trusted service along with the username, and App trustlet will be shut down by the trusted service.

4) The logic engine of the trusted service in SW calls the authorization application API: $Apply()$, generates the authorization request message m_apply: $m\_apply \longleftarrow Apply(Cert_T, dsk_T, blob_{apk}, mk_{auth}, \mu(app), user, H(pswd))$. The API specifically performs the following operations:

   a. Call the key manager to reconstruct the device key private key dsk_T;

   b. Call the key manager to reconstruct the apk storage protection key, call the data processor to unblock $blob_{apk}$, then get the correct apk;

   c. Call the signature function Generate the signature:

$$w := Sign_{dsk_T}(mk_{auth}, u(app), user, H(pswd));$$

   d. Call the encryption function to generate the end of the communication message:

$$m\_apply :== Enc_{apk}(Cert_T, mk_{auth}, u(app), user, H(pswd), w).$$

   Here, apk is derived from A, in fact, A generates a pair of public-private key pairs (apk, ask) for each application issued, apk can be extracted by T for authentication communication with A when the application is installed; in addition, apk can uniquely identify an application.

5) T switches from SW to NW, sends m_apply to A, A decrypts the message with its own private key ask, uses the public key M certificate $Cert_T$ issued by authority, and obtains the device key public key dpk_T of T, verifying the signature of the relevant data, extracting valid data tuples of the authorization request message:

$$(mk_{auth}, \mu(app), user, H(pswd)).$$

6) A verifies the integrity measure $\mu(app)$ of the App trustlet in T according to their published application code, and uses the user and $H(pswd)$ verify the legitimacy of the user account, you can check the balance of the account: if the relevant authentication fails, returns the message and reason why the application of T failed; if all authentication passes, A generates a session key packet tuple $(ID, k^{Enc}, k^{MAC}, n_0)$ for T and C, where ID uniquely identifies the key package; k^{Enc} is used to protect the confidentiality of the conversation; k^{MAC} is used to protect the integrity of the session; n_0 is a randomly selected nonce value and to prevent replay attacks. T and C access once correct service connection, each of the value will plus 1, so the $(i+1)^{th}$ connection get n_i;

7) A is the authorized session key package for T, and A will use the dpk_T encryption to generate σ after signing the session cipher package:

$$\sigma := Enc_{dpk_T}(apk, (ID, k^{Enc}, k^{MAC}, n_0), Sign_{ask}(ID, k^{Enc}, k^{MAC}, n_0)).$$

Among them, apk is used to identify the application corresponding to the encrypted data, and A generates an authorization response message in the following manner:

$$m\_reply := \sigma \| MAC_{mk_{auth}}(\sigma).$$

8) A sends m_reply to T and sends (ID, k^{Enc}, k^{MAC}, n_0) along with user and $\mu(app)$ to C through the security channel. If C finds the same user's previous session in the database package through user, the old key package is deleted. The lifetime of the session key package can be set to a different length depending on the security sensitivity of the cloud service, which can be 1 day, 7 days, or 30 days. The validity period is recorded at C, and if the expiration date is exceeded, the session key package will automatically invalidated, and the expired session key package is periodically cleaned and deleted by C;

9) After T receives m_reply, it switches to SW, and the trusted service parses m_reply. After verifying

the message integrity and signature correctness, the extracted session key package $(ID, k^{Enc}, k^{MAC}, n_0)$ is encapsulated as a $blob_{ID}$ and stored in a mobile device.

4.2.2 Access Request

In this part of the protocol, T uses the session key package to send a cloud service access request to C. The specific steps are as follows:

1) T reloads start App trustlet, and measures the loaded the integrity of the code once again, the use of hash function to obtain the metric $\mu'(app)$, you can use the white list mechanism to check again whether the App trustlet is tampered;

2) The logical engine of the trusted service in SW calls the cloud service access request API: $Request()$ generates $m\_request$:

$$m\_request \Longleftarrow Request(blob_{ID}, \mu'(app)).$$

The API specifically performs the following operations:

1) Call the key manager to reconstruct the storage protection key of the session key package, and call the data processor to unlock the $blob_{ID}$ to get the correct session key packet tuple $(ID, k^{Enc}, k^{MAC}, n_i)$;

2) Generate cloud service access request communication message $m\_request$:

$$\begin{aligned} m\_request \quad := \quad & ID \| Enc_{k^{Enc}}(\text{"}request\text{"}, n_i, \mu'(app)) \| \\ & MAC_{k^{MAC}}(ID, Enc_{k^{Enc}}(\text{"}request\text{"}, \\ & n_i, \mu'(app))). \end{aligned}$$

Among them, ID is used to tell C which session key packet to use to decrypt and verify the message; the request is a command parameter that identifies the execution of the cloud service at the request C. Here, the command to apply for access to the cloud service is indicated.

3) T switches to NW and sends $m\_request$ to C.

4.2.3 Verify the Response

In this part of the protocol, C uses the session key package to parse the access request from T, and returns the verification result and the integrity measure of the cloud service executive to T, and the concrete steps are as follows:

1) After receiving the $m\_request$, C finds the corresponding session key package in the database according to ID, to check whether the session key package is still valid, expires or revokes, or fails to find the key package. In case, C sends a response marked as verification failure to T, and T will re-execute the authorization application agreement.

2) C uses the legitimate key package parsing $m\_request$, and matches the value of n_i to the current nonce value of the session key package record in the database, if they are not the same, then return the validation failure response to T, T will re-execute the authorization request protocol.

3) C matches the $\mu'(app)$ in $m\_request$ with the original $\mu(app)$ of the corresponding session key package in the database, if not the same, that App Trustlet in T is likely to have been tampered, C will reject the access request of T, and the response to the authentication failure is returned.

4) After the above 3-step verification, C uses the security method in the trusted cloud architecture to generate an integrity metric $\mu(csp)$ of the program running the T-requested cloud service in the virtual machine. In some specific application scenarios , the metric may be derived from a hash metric for the entire virtual machine image, for $\mu(csp)$ certification methods can refer to the specific agreement of the cloud architecture.

5) C generates a communication message $m\_response$ for the verification response:

$$\begin{aligned} & m\_response \\ := \quad & ID \| Enc_{k^{Enc}}(\text{"}response\text{"}, passed\text{"}, n_i, apk, \\ & \mu(csp)) \| MAC_{k^{MAC}}(ID, Enc_{k^{Enc}}(\text{"}response\text{"}, \\ & \text{"}passed\text{"}, n_i, apk, \mu(csp))). \end{aligned}$$

6) After C sends $m\_response$ to T, it updates the nonce value: $n_{i+1} = n_i + 1$, and sets the nonce limit value of the access service to be $limit\_n = n_i + j$, j represents the T access to the cloud service after the verification. If $n_{i+x} > limit\_n$ of C at the x^{th} access, T needs to re-execute the access request protocol to let C update and set $limit\_n$;

7) After receiving $m\_response$, T switches to SW to parse and verify it. Meanwhile, it updates the nonce value of the itselves' session key package: $n_{i+1} = n_i + 1$. The security method with trusted cloud architecture can verify the integrity of the cloud service program through $\mu(csp)$. After the verification is passed, the App trustlet transmits command parameters to the trusted service according to the specific function of the user requesting the cloud service. The trusted service uses the session key package to assemble the cloud service operation command, and communicates with C to complete the specific Cloud service features.

5 Mobile Cloud Storage Application Examples

Based on the proposed secure access solution for mobile terminal cloud services, we designed a mobile cloud stor-

age application instance MCFile. MCFile security objectives including: (1) protecting the confidentiality and integrity of mobile terminal users sending and receiving files to the cloud server; (2)The cloud server can enforce mandatory security authentication and access control policies for mobile terminal users' access requests and file access. The cloud storage service operation commands implemented by MCFile can be: create files (create). Delete files, write files, read files, addrights, and removerights. Assuming that there are two users whose user names are user1 and user2, user1 stores the file named Fuser1 in the cloud. Then, after the user1 successfully performs the authentication response of the cloud service security access protocol using the mobile terminal T, he can generate the following cloud service request command in the SW to read the cloud file Fuser1 :

$$ID \| Enc_{k^{Enc}}(\text{"}read\text{"}, F_{user\ 1}, n_{i+1}) \| MAC_{k^{MAC}}(ID, Enc_{k^{Enc}}(\text{"}read\text{"}, F_{user\ 1}, n_{i+1})).$$

After C received the command, find the associated session key package and user name of user1 according to the ID. After parsing the request command and verifying the legitimacy of the command, C checks the user1's permission for the file Fuser1. If it is judged as readable, the following service response will be returned: where File (Fuser1) represents the file entity specified by Fuser1, the use of data block technology can achieve large volume of the file network encryption transmission. If user1 wants to share the file Fuser1 with user2, it can add the read permission of Fuser1 to user2. The corresponding cloud service request command is as follows:

$$ID \| Enc_{k^{Enc}}(\text{"}addright\text{"}, user2, \text{"}read\text{"}, F_{user1}, n_{i+1}) \| MAC_{k^{MAC}}(ID, Enc_{k^{Enc}}(\text{"}addright\text{"}, user2, \text{"}read\text{"}, F_{user1}, n_{i+1})).$$

After receiving the column verification, C adds a user2 readable entry in the permission list of file Fuser1 , but at this time the owner of Fuser1 is still user1, and user1 can send a command to cancel the read permission of user2 to Fuser1 . In addition, when the symbol * is used in the above request command instead of user2, user1 assigns the readable authority of Fuser1 to all legitimate users, that is, the public sharing of files is realized. The other functions of MCFile other cloud storage service functions can be implemented by this method. Analogy, no more description here.

6 Assessment

We simulated and realized the mobile terminal T, the application service provider A and the cloud service provider C respectively. For the simulation and realization of mobile terminal equipment, we used the embedded development board Zynq-7000 AP Soc Evaluation Kit. The board supports the TrustZone security extension with an ARM Cortex-A9 MPCore processor, 1GB of DDR3 memory, and OCM (on-chip memory) module with 256KB SRAM.

For the application service provider's simulation implementation, we used a Dell OptiPlex 990 desktop computer with a 3.3GHz Intel i3-2120 dual-core processor and 4GB of memory, running the Ubuntu10.04 operating system with kernel version Linux 2.6.32. For cloud service provider simulation implementation, we used a Lenovo ThinkCentre M8500t desktop computer, equipped with 3.4GHz Intel i7-4770 quad-core processor and 8GB of memory, the operating system is the same as the former.

6.1 Code Amount and Trusted Computing Based

In the program prototype system, the realization of the components of the C code's approximate lines number (lines of code, referred to as LoC) in Table 1. The trusted computing base (TCB) of a device is a collection of software, hardware, and firmware required to achieve device security. The smaller the scale is, the more difficult to be attacked by rivals, and the security is relatively easy to be guaranteed. In this scheme, the TCB of the trusted mobile terminal contains only the mobile device hardware and the software running in the SW. According to the literature [11], a certain type of SW security OS currently in the mobile commercial market has 6000 LoC. If this type of OS is used, plus the trusted service and App trustlet that we implement, the TCB software part of the scheme is only 9100 LoC. This scale is relatively small, and the controllability of system security is relatively high.

6.2 Performance Evaluation

Using the prototype system, we experimented the related operations required by the mobile terminal T to perform the solution in this paper. The program include encapsulation and unblocking sensitive data and communication interactions in the process of generating and resolving authorization requests, access requests, and requests for cloud services, among them, cloud service request and response messages do not consider specific cloud service commands. The operating time cost statistics take the average of 100 runs, and the experimental results are shown in Table 2.

6.3 Performance Evaluation of Server-side Program

Using the prototype system, we experimented with the related operations required by the application service provider A and the cloud service provider C in the implementation of this program. In the scheme, A is responsible for receiving and resolving the authorization application messages sent by the mobile terminal and verifying it to generate an authorization response message. In this experiment, we take this process as a response. First,

Table 1: Code size and TCB implemented components

Entity	Components	Loc	TCB
Mobile Terminal T	Trusted service	2300	V
	Trusted proxy	1500	X
	App trustlet	800	V
	App	500	-
Application Service Provider A	Authorization procedure	5600	-
Cloud Service Provider C	Access authentication procedure	6300	-

Table 2: Time overheads of the operations on mobile terminal

Operatations	Time Consumption (ms)
Encapsulation apk	0.030
Unblock apk	0.020
Encapsulation session key package	0.081
Unblock session key package	0.093
Generate m_apply	129.906
Resolve m reply	128.738
Generate m request	0.117
Resolve m response	0.110
Generate cloud service access request	0.152
Resolve Cloud Service Authentication Response	0.150

we experimented with the time cost of A single thread to complete a response, taking the average of 100 runs independently. The experimental results are single-threaded single time-consuming 13.225ms. Then, we experimented with the time required to complete a single response when using A thread pool concurrent execution with a large number of authorization requests. The experimental results are shown in Figure 3 . It can be seen from the graph that as the number of concurrent requests increases from 100 to 500 on the abscissa, the response time of a single request increases from about 400 ms to about 2300 ms on the ordinate. This substantial increase is due to the fact that A needs to be in a single response execute asymmetric encryption,decryption,signature and verify every time, these operations consume more system resources. Our experiments are based on a generic desktop computer, taking into account that A is usually implemented by several professional server clusters in practical application, and the optimized concurrency response will have a lot of room for improvement. In addition, the frequency of mobile users to implement authorization applications is not high compared to the mobile network delay. Moreover, server response delay about 2 000ms can not be accepted.

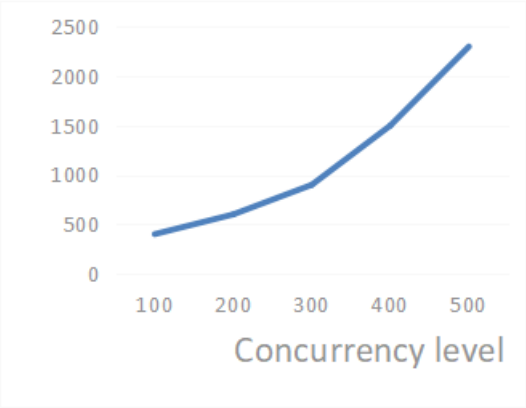

Figure 3: Authorization response latency in A

C is responsible for receiving and resolving the access request message sent by the mobile terminal in the scheme and validating it to generate a verification response message. In the experiment, we take this process as a response. Similarly,we first experimented with the execution of a single response of C single thread, and the ex-

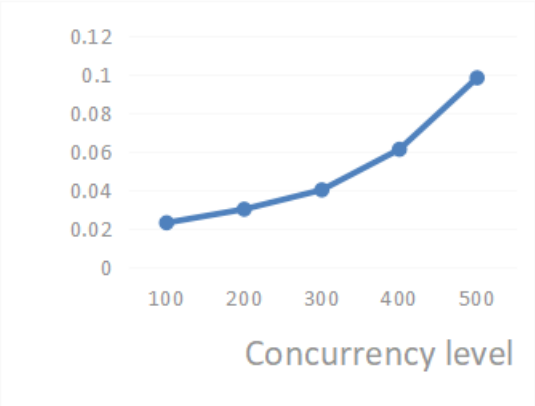

Figure 4: Authentication response latency in C

perimental results were 0.016 ms for single-thread single response. Then, we experimented with C in the case of concurrent execution to complete a single response time, the experimental results can be shown in Figure 4. It can be seen from the figure that as the number of concurrent requests increases, the response time of a single request increases from about 0.030ms to about 0.100ms, and the absolute value and the growth rate are not large, which is due to the operations that the C in the process of one response does not consume a lot of system resources. In addition to sending a verification response message to the mobile terminal, C will also interact with the mobile terminal in large numbers to complete the specific cloud service function response, which is basically consistent with the authentication response. Regardless of the specific cloud service operation, the response time overhead will be at the same level as the experimental results in Figure 4. The request response handled by C will be executed frequently in this scenario, which occupies a large proportion of the actual running interaction of the scheme. The low response delay reflected in the experiment indicates that the solution has good performance at the cloud service provider.

7 Discussion and Conclusions

7.1 Discussion

In many anonymous authentication systems, the direct anonymous attestation (DAA) protocol was officially released by TCG for anonymous proof based on TPM and has now been accepted as an ISO standard. The protocol can be modified to apply to the program in order to achieve anonymous authentication of the service provider to the user. In the previous work, we designed the DAA-TZ scheme for mobile terminals based on the DAA protocol. The scheme used TrustZone's good features to provide secure and efficient anonymous authentication services. The program system has been fully implemented and tested. Therefore, combining with DAA-TZ, it is possible to design and implement a trusted terminal cloud service secure access scheme with anonymous attributes.

7.2 Conclusion and Future Studies

This paper analyzes the related security issues of mobile terminal access cloud service for mobile cloud computing scene, and proposes a trusted mobile terminal cloud service security access scheme. The scheme uses TrustZone security extension technology to construct trusted mobile terminal architecture. Trusted mobile terminal uses SRAM PUF to obtain root key seed, and realized the security management mechanism of key and sensitive data.Secondly, based on the idea of trusted computing technology, the cloud service security access protocol is designed on the basis of trusted mobile terminal, and the protocol is compatible with trusted cloud computing architecture. The analysis and experimental results show that the security access scheme proposed in this paper can effectively realize the security authentication of the mobile terminal in the process of accessing the cloud service and protect the private data security of the mobile user in the cloud service. The program has better scalability and smaller mobile terminal TCB, its overall operation efficiency is higher, mobile users wait for the delay within the acceptable range. In the future work,we will do a formal analysis for the security access protocol which presented in the program , and give a more detailed proof of security.

Acknowledgments

This work is supported by Scientific Study Project for Institutes of Higher Learning, Ministry of Education, Liaoning Province (LQN201720), and Natural Science Foundation of LaioNing Province, China (20170540819). The authors gratefully acknowledge the anonymous reviewers for their valuable comments.

References

[1] A. Alaqra, S. Fischer-Hübner, T. Groß, *et al.*, "Signatures for privacy, trust and accountability in the cloud: Applications and requirements," *Privacy and Identity Management. Time for a Revolution?*, vol. 476, pp. 79-96, 2016.

[2] S. Al-Janabi, I. Al-Shourbaji, M. Shojafar, *et al.*, "Mobile cloud computing: Challenges and future research directions," in *International Conference on Developments in Esystems Engineering*, 2017. DOI: 10.1109/DeSE.2017.21.

[3] A. Alzahrani, N. Alalwan, M. Sarrab, "Mobile cloud computing: Advantage, disadvantage and open challenge," in *Proceedings of the 7th Euro American Conference on Telematics and Information Systems*, 2014. DOI:10.1145/2590651.2590670.

[4] H. T. Dinh, C. Lee, D. Niyato, P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wireless Communications & Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[5] GlobalPlatform device technology, *TEE Client API Specification Version 1.0*, 2010. (http://globalplatform.org)

[6] S. Hosseinzadeh, S. Laurén, V. Leppänen, "Security in container-based virtualization through vTPM," in *IEEE/ACM International Conference on Utility & Cloud Computing*, 2017. DOI: 10.1145/2996890.3009903.

[7] Q. Jiang, J. Ma, F. Wei, "On the security of a privacy-aware authentication scheme for distributed mobile cloud computing services," *IEEE Systems Journal*, vol. 12, no. 2, pp. 1-4, 2016.

[8] M. John, R. Tom, S. Fred, *The Cloud-Proxy Tao for Trusted Vomputing*, Technical Report, No.UCB/EECS-2013-135, 2013. (http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-135.html)

[9] C. Lee, *Security Control Apparatus and Method for Cloud-based Virtual Desktop*, 2017. (https://patents.google.com/patent/US9674143B2/en)

[10] Y. Li, Z. Han, Z. Huang, *et al.*, "A remotely keyed file encryption scheme under mobile cloud computing," *Journal of Network & Computer Applications*, vol. 106, pp. 90-99, 2018.

[11] W. H. Li, H. B. Li, H. B. Chen, Y. B. Xia, "AdAttester: Secure online mobile advertisement attestation using TrustZone," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 75–88, 2015.

[12] J. Lind, I. Eyal, F. Kelbert, *et al.*, "Teechain: Scalable blockchain payments using trusted execution environments," *ArXiv*, 2017. (https://www.researchgate.net/publication/318528079_Teechain_Scalable_Blockchain_Payments_using_Trusted_Execution_Environments)

[13] M. B. Mollah, M. A. K. Azad, A. Vasilakos, "Security and privacy challenges in mobile cloud computing: Survey and way ahead," *Journal of Network and Computer Applications*, vol. 84, pp. 34-54, 2017.

[14] W. Pan, Y. Zhang, M. Yu, *et al.*, "Improving virtualization security by splitting hypervisor into smaller components," in *Data and Applications Security and Privacy XXVI*, pp. 298-313, 2012.

[15] N. Santos, K. P. Gummadi, R. Rodrigues, "Towards trusted cloud computing," *Proceedings of the Conference on Hot Topics in Cloud Computing*, 2009. (https://www.usenix.org/legacy/event/hotcloud09/tech/full_papers/santos.pdf)

[16] Trusted computing group, *TPM Main Specification, Version1.2, Revision 116*, 2011. (http://www.trustedcomputinggroup.org)

[17] Trusted computing group, *Trusted Platform Module Library, Family 2.0, Revision 01.16*, 2014. (http://www.trustedcomputinggroup.org)

[18] Trusted computing group, *TCG Mobile Trusted Module Specification, Version1.0, Revision 7.02*, 2010. (http://www.trustedcomputinggroup.org)

[19] Q. X. Wu, X. W. Yang, H. Zou, F. J. Yu, X. K. Ning, Z. Wang, "Technic specification of cryptography supporting platform for trusted computing," *China State Password Administration Committee*, 2007. (http://www.oscca.gov.cn)

[20] C. Wu, Y. J. Zhou, K. Patel, Z. K. Liang, X. X. Jiang, "AirBag: Boosting smartphone resistance to malware infection," in *Proceedings of Network and Distributed System Security Symp (NDSS'14)*, 2014. (https://pdfs.semanticscholar.org/4823/f6af261a88716980485638f2d06f94bbf2d4.pdf)

[21] B. Yang, D. G. Feng, Y. Qin, "A lightweight anonymous mobile shopping scheme based on DAA for trusted mobile platform," in *Proceedings of the IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 9–17, 2014.

[22] Y. J. Zhang, D. G. Feng, Y. Qin, B. Yang, "A TrustZone based trusted code execution with strong security requirements," *Journal of Computer Research and Development*, vol. 52, no. 10, pp. 2224–2238, 2015.

[23] B. Zhao, F. Yan, L. Q. Zhang, J. Wang, "Build trusted cloud computing environment," *Communications of China Computer Federation (CCF'12)*, vol. 8, no. 7, pp. 28–34, 2012.

[24] S. J. Zhao, Q. Y. Zhang, G. Y. Hu, Y. Qin, D. G. Feng, "Providing root of trust for ARM TrustZone using on-chip SRAM," in *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*, pp. 25-36, 2014.

Biography

Hui Xia received his B.S. and M.S. degree from Xidian University,China, in 2003, 2006,respectively. He is currently a associate professor with Shenyang Normal University. His research interests include cloud computing, cryptography and information security.

WeiJi Yang received his B.S. degree from Zhejiang Chinese Medical University , China, in 2005, M.S. degrees from Beijing University of Posts and Telecommunications, China, in 2009. He is currently a Research Associate with ZheJiang Chinese Medical University. His research interests include Information Security and Traditional Chinese Medicine Informatics.