Scientific Research Publishing

Journal of Computer and Communications, 2021, 9, 103-120
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

AudiTEE: Efficient, General-Purpose and Privacy-Preserving Audit for Distributed Ledgers

Zhufeng Ye, Zhenghao Wu, Xianglan Tian

College of Cyber Security, Jinan University, Guangzhou, China
Email: allenye1996@foxmail.com

How to cite this paper: Ye, Z.F., Wu, Z.H. and Tian, X.L. (2021) AudiTEE: Efficient, General-Purpose and Privacy-Preserving Audit for Distributed Ledgers. *Journal of Computer and Communications*, **9**, 103-120.
https://doi.org/10.4236/jcc.2021.98007

Received: July 30, 2021
Accepted: August 28, 2021
Published: August 31, 2021

Abstract

Privacy-preservation and effective auditing are two desirable but challenging requirements on distributed ledgers. To meet the requirements, this paper presents an auditing scheme, called as AudiTEE, which can audit a distributed ledger in a generic, efficient, and privacy-preserving manner. AudiTEE leverages Trusted Execution Environment (TEE) to generate confidential but auditable transactions and realize arbitrary, efficient and confidential audit on them. Unfortunately, TEE suffers from some inherent barriers and is itself not a complete solution for fast audit. To tackle these challenges, AudiTEE takes advantage of KAMT (K-anonymity Authentication Based on Merkle Tree) protocol for efficient management on *account* and user-defined anonymous transactions. Further, to achieve a complete and fast audit with *unlinkability*, TEE doesn't process through all but only a comparatively small part of transactions according to a special *ktag* attached on each transaction to ensure that a user cannot hide transactions from auditor even when auditor is blind with who is involved in each transaction on the ledger. Apart from the above, AudiTEE allows flexible control on user behaviors. We implement a concrete instance of AudiTEE under a bank setting and demonstrate the scalability with all its core functionalities.

Keywords

Distributed Ledgers, Trusted Execution Environments, Auditing, Privacy-Preservation, Authentication

1. Introduction

A distributed ledger [1] [2] is a general distributed database that exists across

several locations or among multiple participants. As a result, it is decentralized to mitigate the need for a central authority or intermediary to process, validate or authenticate transactions. Its records are stored one after the other in a continuous ledger, but they can only be added when the participants reach some consensus.

Audit [3] [4] plays an indispensable role in deterring ledger malpractice from time to time[1]. In the financial sector, for example, auditors verify the correctness of the company statement with transaction records from banks [5]. In September 2020, the transaction regulators confirmed Luckin's falsification of financial and operational figures [6], removed Luckin from NASDAQ list. Audit is also required in many sectors such as government and charity organization [7] [8].

Due to their promising immutable data and public verifiability merits demonstrated in the auditing process, distributed ledgers have become a brand new sword in the practice of audit by pioneers such as the "Big Four" (Deloitte, PwC, EY and KPMG) [9]. In the schemes of the auditing process of distributed ledger, each ledger user is assigned with a unique ID, and can go to service providers, such as banks or brokers, to open its service *account*. Service providers will then perform operations (e.g., payment or exchanging) according to the request of a valid *account*. A regulator can impose *account*-based policy on user and an auditor can audit the service records to hold any misbehaved individual to account.

Although the auditing on distributed ledgers brings many advantages, it triggers concerns on several issues. 1) *Privacy leakage*: As auditing could be extremely challenging when an auditor is blind with which transactions belong to a user, the auditor is usually authorized to know all the transactions of the user, *i.e.*, the auditor requires to know the linkability of transactions [10] [11] [12] [13]. This requirement significantly threatens user's privacy on transaction time and payment frequency. Even worse, auditors may need excessive power to decrypt confidential transactions sometimes [10] [11] [12]. 2) *Limited functionalities*: To preserve the privacy of transactions, zero-knowledge proof is usually adopted to realize confidential audit in the schemes [13] [14] [15] [16]. However, the schemes can merely conform to some specific tasks, such as payment limitation and correct taxation. 3) *Inefficiency*: An audit based on zero-knowledge proof usually costs much time for the cooperation between the auditor and the user. Besides, zkLedger [16] enables a single audit to involve all irrelevant transactions in a time period for preserving *unlinkability*. Thus, they are impractical for large-scale audit on massive transactions and users.

Trusted Execution Environment (TEE) can enhance the confidentiality and integrity of computations [17]-[22]. Particularly, hybridized TEE-blockchain systems demonstrate the potential of TEE in the distributed ledger. For example, Proof of Luck [23] leverages TEE's random number generator to mimic the

[1]Originally defined in finance as independent examination of financial information of any entity, the concept of audit has now expanded to encompass so many areas in the public and corporate sector. It generally refers to inspection or examination, of a process or quality system, to ensure compliance to requirements.

original Proof of Work consensus in a low-latency, egalitarian and energy-saving way. Teechain [24] seeks to deal with the scalability of the notable Bitcoin blockchain [25] under the aegis of TEE. Notwithstanding, they don't focus on auditability. [26] provides a TEE-based method for efficient and privacy-protected policy compliance on distributed ledger. However, audit merely takes place in transaction verification. Regretfully, TEE-based authentication suffers from inherent obstacles: 1) TEE is generally constrained in trusted memory. For example, Intel SGX only has a trusted memory of 128 MB. Therefore, all *accounts* information cannot be put into the enclave for anonymous user authentication. 2) TEE in general lacks trusted time source. It cannot reliably determine the state of a user is fresh or stale. This can be misused by a malicious user to create transactions with invalid *account*. 3) Trusted memory usually has worse performance than untrusted memory [27] [28], thus slowing down the authentication process.

With regard to auditing, arbitrary audit function can be performed in plaintext speedily and the plaintext is only known to the enclave itself. However, TEE itself is not an efficient solution for *unlinkable* audit. When an auditor cannot determine who is involved in a transaction, a complete[2] audit has to be performed with all transactions in a time period as a price to ensure no transaction belonging to its auditee is left out. The exclusion of unrelated transactions will happen in the enclave to render a correct audit result. This is, nonetheless, highly impractical towards large scale audit.

The present scheme AudiTEE aims to provide efficient and privacy-preserving auditing on distributed ledgers. Specifically, it first creates an intermediary-based ledger, e.g., bank-intermediated ledger in the financial community [12] [29] [30], which enables a set of intermediary parties, namely, service providers, to post transactions of their users on the distributed ledger. Secondly, it employs an effective authentication component based on Trusted Execution Environment (TEE) which makes sure service providers only serve authorized users in a privacy-preserving way. Thirdly, auditable but confidential transactions and efficient, general-purpose and privacy-preserving audit are performed inside TEE to ensure confidentiality.

AudiTEE adopts KAMT (K-anonymity Authentication Based on Merkle Tree) protocol as a key component to enable TEE in efficient and anonymous user authentication and high-speed privacy-concealed audit: 1) To engage in user authentication, TEE simply stores a single short hash string rather than all *accounts* information. This short string helps TEE to identify that an *account* is currently valid. Meanwhile, the anonymous authentication for a user only relies on the verification of Merkle path, which is much faster than signature-based method. 2) Our special *ktag* produced allows any complete audit to be performed without the loss of *unlinkability* while much more practical and efficient than processing through all transactions, like zkLedger [16], as only a comparatively small part of transactions on the ledger needs to get involved. This prop-

[2]A complete audit involves all transactions of a user in a specific time period.

erty guarantees user privacy on when a transaction occurs or the frequency of transaction while no relative transaction will be left out for an audit. Meanwhile, an audit can be performed immediately when permitted by a trusted authority and does not involve the cooperation with the auditee. Thus, supported by KAMT protocol, AudiTEE realizes complicated, confidential but fast audit compared to others.

Our contributions can be summarized as follows:

- *AudiTEE framework.* We propose AudiTEE to support efficient, general-purpose and privacy-preserving audit on confidential distributed ledger.
- *KAMT protocol.* We introduce this special building block to provide TEE-empowered solution for efficient identity management, user-defined anonymous authentication and enable fast and complete audit for AudiTEE.
- *Implementation and Experiments.* We have an instantiation of AudiTEE under bank setting and demonstrate its gratifying performance.

Outline of this work. The rest of this paper proceeds as follows. In Section 2, we propose AudiTEE scheme including the system model, security model and modules. In Section 3, we discuss the performance of AudiTEE. We instantiate AudiTEE under a banking scenario to examine its performance in Section 4. The paper is concluded in Section 5.

2. The Proposed Auditing Scheme

AudiTEE provides efficient, general-purpose and privacy-preserving audit by integrating TEE-enabled hosts into distributed ledgers. It can achieve the following goals: 1) Privacy-preservation. The transactions of users are confidential and *unlinkable*. 2) Correctness: Only valid transactions can be produced. 3) Completeness: An audit cannot perform at the price of *linkability* although all relative transactions are checked. The notations are listed in Table 1.

Table 1. Notations.

Variable	Definition
Merkle Verify (proof, v, MR)	Function to verify whether *proof* is the existential proof of leaf *v* with Merkle root *MR*.
ktagVerify (proof, ktag)	Function to verify whether *ktag* conforms to existential *proof*.
(epk, esk)	Key pair for encryption use.
(spk, ssk)	Key pair for signing use.
K	Symmetric key *K*.
$sign_{ssk}$ *(m)*	To sign message *m* with *ssk*.
enc_{epk} *(m)*	To encrypt message *m* with *epk*.
enc_K *(m)*	To encrypt message *m* with *K*.
rand (λ)	To generate a randomness of length *λ*.
H (m)	Collision-resistant hash function with input *m*.

2.1. System Model

As shown in Figure 1, AudiTEE consists of four parties: regulator, service providers, users and auditors:

Regulator R is the party to supervise the distributed ledger. It is in charge of 1) managing *accounts* for users and service providers; 2) enforcing policy on user behaviors; 3) judging the lawfulness of an auditing request from an auditor; and 4) helping the authorized auditor to get the result.

TEE-empowered Service Provider P_i enrolls a user and receives his service requests through TEE, verifies the requests, and forms confidential and authenticated transactions inside a TEE.

User U is at least enrolled by one P_i. He performs actions on the ledger through the TEEs of service providers as an intermediary and follows certain policy. His behaviors or transactions can be audited by an auditor under the permission of R.

Auditor A is to inspect the transactions of users. Under the authorization of R, an auditing process takes place in the TEE of some P_i.

AudiTEE consists of 3 phases. In phase I, each user U registers with some provider P_i for an *account* under the confirmation of the regulator R. In phase II, user U anonymously authenticates himself to the TEE of P_i with KAMT protocol and then posts confidential transactions on the distributed ledger. In phase III, auditor A sends the auditing request to R. As long as the auditing request is approved, auditor A will carry on auditing process on U's transactions inside the TEE of some P_i.

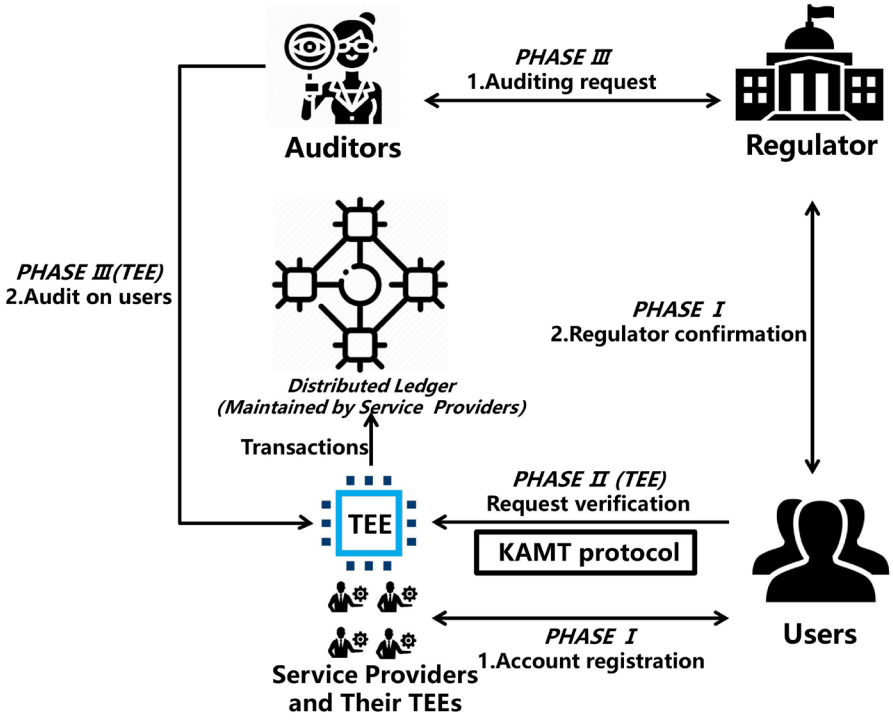

Figure 1. AudiTEE system model. Authentication and audit are performed inside the TEEs of service providers.

2.2. Security Model

We now elaborate on assumptions of AudiTEE:

Regulator is semi-honest. It honestly follows the AudiTEE protocols and doesn't misuse the power of audit, but is curious about who is making a transaction.

Service Provider is considered to be dishonest if it: 1) helps malicious users to misbehave; 2) breaks the privacy of transactions.

User is assumed to be untrustworthy if he behaves in the following ways: 1) lies about his transactions during auditing; 2) makes a transaction against policy; 3) makes a transaction from an invalid *account*.

Auditor is semi-honest. She follows the AudiTEE protocols but is curious about user privacy during auditing.

AudiTEE doesn't protect against an adversary who performs network traffic analysis. For example, if U sends a message to the TEE of P_i, it's reasonable to assume that a new transaction involves U.

2.3. AudiTEE Protocol

According to Figure 1, AudiTEE comprises of three modules, 1) initialization (phase I): initializes Merkle tree for user information; 2) authenticated transaction generation (phase II): generates the transaction and 3) privacy-preserving auditing (phase III): performs auditing process.

Initialization (Phase I). In the beginning, user U registers with some provider P_i under the confirmation of R as shown in Figure 2. Specifically, user U produces his asymmetric key pair (epk_U, esk_U), and requests for registration to some provider P_i. Afterwards, P_i signs epk_U and a timestamp ts with its signing key ssk_{Pi} as σ_{Pi}. U sends a tuple $< spk_{Pi}, epk_U, ts, \sigma_{Pi} >$ to R for confirmation.

R defines a user policy *usrSpecif* on user U, and forms user information $info = H(epk_U, usrSpecif, r)$, where r is a nonce, and $H(\cdot)$ is a one-way hash function. *info* will be included into a publicly accessible Merkle tree maintained by the regulator R as Figure 3. R then returns to the user U a tuple $< epk_U, usrSpecif, r, index >$, where *index* is the position of leaf node for *info*. The Merkle tree will be re-produced by the regulator when there are some events including user join, user leave or *info* update.

Authenticated transaction generation (Phase II). To post a verifiable transaction on the distributed ledger, U needs to authenticate himself through TEE in an anonymous and efficient manner using KAMT protocol.

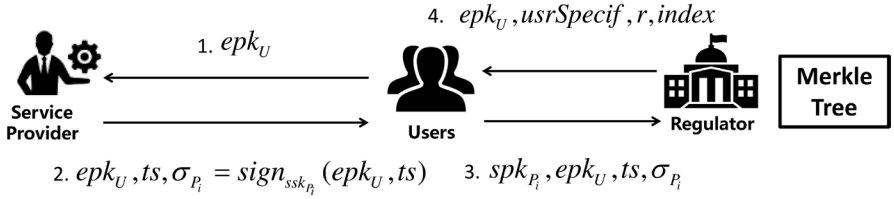

Figure 2. Registration process.

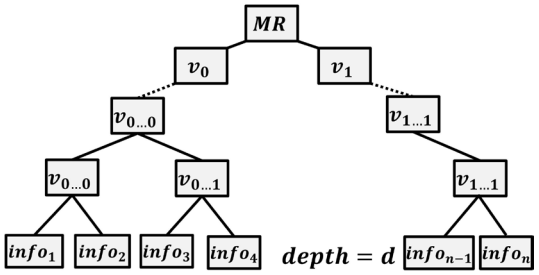

Figure 3. Identity management with Merkle tree. Any leaf node is a value *info* for a user or 0, and any non-leaf node is $v_y = H(v_{y0}//v_{y1})$ for two children nodes v_{y0} and v_{y1}.

Due to the Merkle tree update, *U* needs to fetch the latest *proof* of his identity. Furthermore, in order to hide his identity in the retrieval, rather than asking for a specific *proof*, this can be done by blinding part of the original *proof* with self-defined 2^m leaves under a specific node amid the path from *info* to the root. Technically, we call the index of this specific node *ktag*, the subroot index of a subtree. For example, supposing $d = 10$ (the tree is with 2^9 leaves) and $m = 6$, the retrieval of the Merkle *proof* of $v_{011101011}$ (a specific *info*) includes all $v_{011x_4\ldots x_9}$, where $x_i \in \{0, 1\}$ and $|011x_4\ldots x_9| = 9$, and (v_{010}, v_{00}, v_1). User can choose a desirable 2^m including enough non-empty *info* and easily recover its complete *proof* $(v_{011101011}, v_{011101010}, v_{01110100}, v_{0111011}, v_{011100}, v_{01111}, v_{0110}, v_{010}, v_{00}, v_1)$. The *ktag* for *U* is 011, indicating *U* is some user under v_{011}. The design of KAMT protocol ensures *ktag* is bound with an existential *proof* and cannot be forged.

P_i runs a TEE program which receives the Merkle tree root *MR*. Meanwhile, *U* sends the tuple $< epk_U, usrSpecif, r, proof >$ to TEE. By checking the recovered root value and retrieved root value *MR*, TEE determines whether the user is genuine or not.

Now we integrate KAMT protocol to the generation of authenticated transaction. We illustrate the whole process in **Figure 4**. Any genuine user *U* authenticated sends request *txReq* to TEE_{Pi}, where *txReq* including a pack of data that's required for *service-specific logic*, such as *epk* for c_p, *ktag* and other *service-dependent* data. For the validated request, TEE_{Pi} will forms an authenticated transaction *TX* as described in Algorithm 1 in **Figure 5**. **Table 2** is the authenticated transaction, where c_p refers to secret information and *ktag* denotes an anonymous participant inside a group of *accounts*. For simplicity, c_p is encrypted under one *epk* and there is only one *ktag* for a single anonymous user in a transaction. The transaction *TX* is broadcast to the distributed ledger network for further validation such as double-spending checking.

*Privacy-preserving audit (Phase III).*For any user in the distributed ledger, AudiTEE can perform efficient, general-purpose and confidential auditing and will not leak any extra information.

Algorithm 2 in **Figure 6** illustrates the auditing process. Auditor *A* first sends a specific audit request on the user *U* to the regulator *R*, which is presented as code AudProg. If the request is legal, *R* informs the auditor *A* all the leaves belonging to *U*. Then *A* posits the transactions' $\boldsymbol{c_a}$, a set of c_a with *ktag* involving

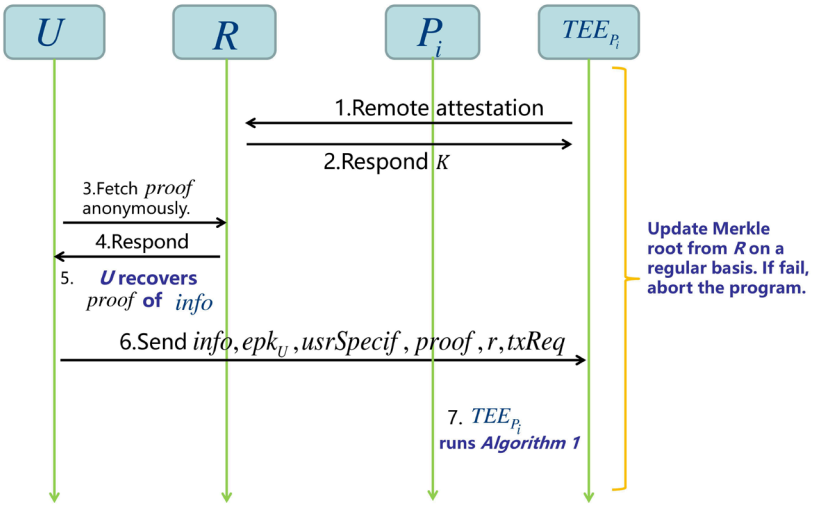

Figure 4. Generation of authenticated transaction.

Algorithm 1 TEE Verification **(inside TEE)**

Input: $info, epk_U, usrSpecif, proof, r, txReq, MR$
Output: TX

1: TEE_{P_i} checks:
$info = H(epk_U, usrSpecif, r)$
$true \leftarrow merkleVerify(proof, info, MR)$
/* MR is obtained from R. */
$true \leftarrow ktagVerify(proof, ktag)$
2: TEE_{P_i} performs *service-specific logic* on $txReq$ and ensures compliance to policy defined in $usrSpecif$.
3: According to *service-specific logic*, TEE_{P_i} extracts:
auditable data and encrypts it with K to form c_a
private data and encrypts it with specified epk in $txReq$ to form c_p
public accessible data to form pub
4: set $outp := (c_a, c_p, pub, ktag)$
5: set $\sigma_{TEE} := sign_{ssk_{TEE}}(prog, outp)$
6: Broadcast $TX := (c_a, c_p, pub, ktag, \sigma_{TEE})$.
7: TEE_{P_i} waits for a new request and then goes to line 1.

Figure 5. Algorithm 1 for TEE verification.

Table 2. Transaction in AudiTEE.

$TX = (c_a, c_p, pub, ktag, \sigma_{TEE})$	
c_a	Auditable contents encrypted under the symmetric key K of R.
c_p	Secret information for specific user.
pub	Publicly accessible information.
$ktag$	Tag to indicate anonymity set of *accounts*.
σ_{TEE}	Proof of the correct formation of a transaction in TEE.

leaves of U (U is under a subtree with subroot index of $ktag$). All $\boldsymbol{c_a}$ will be sent into the TEE of P_i and decrypted with K. The plaintext of c_a should generally include the participant for TEE to exclude any transaction irrelevant to U. TEE

Algorithm 2 TEE Audit **(inside TEE)**

Input: c_a, epk_A
Output: $outp, \sigma_{TEE}$
1: Receive K from R.
2: Decrypt c_a with K to get $p_1, p_2, ..., p_n$.
3: Initiate *result* to store audit result.
4: set $i := 0$
5: **if** p_i includes U **then**
6: Perform audit of **AudProg** on p_i and update *result*.
7: **end if**
8: set $i := i + 1$
9: **if** $i! = n$ **then**
10: Goto line 5.
11: **end if**
12: set $outp := (enc_{epk_A}(result), H(epk_A, c_a))$.
13: set $\sigma_{TEE} := sign_{ssk_{TEE}}(prog, outp)$.
14: Send $(outp, \sigma_{TEE})$ to A.

Figure 6. Algorithm 2 for TEE audit.

performs the operation defined in AudProg only on transactions of U. After the process in TEE, A checks σ_{TEE} to ensure the integrity of c_a and the belonging of result by $H(epk_A, c_a)$ and decrypts $enc_{epkA}(result)$ for the result.

3. Performance Analysis of AudiTEE

3.1. Trade-off in Anonymity

Although AudiTEE ensures user anonymity flexibly, side effect needs to be taken into consideration. Firstly, a transaction with higher anonymity will suffer from extra transaction delay due to network communication and recovery of Merkle *proof*. Meanwhile, stronger anonymity asks for more disk space to store the retrieved data before recovering the complete Merkle *proof*. What's more, extra burden may take place depending on a concrete scenario. For example, in our experimentation of a banking scenario in Section 4, the tracing of *UTXO* can be unwieldy when we choose a substantially safe *ktag* because such *ktag* is more likely to collide with other user's choice. All the above concerns may push normal user, e.g., mobile user, to choose a proper privacy-protecting level rather than the highest allowed and as a result, the process of audit can be faster. Thus, there is a trade-off between anonymity and audit efficiency.

3.2. Quick Bootuping

Algorithm 1, involving the encryption under regulator's key K, entails a bootup to recover K. Thus, a cooperation between P_i and R is demanded. In practice, thousands of TEEs may be run by a single P_i, requiring thousands of bootup cooperation. Also, unexpected breakdown of certain TEE will require a reboot. All these can pose a substantial burden to the whole system. We propose a knack to expedite the bootup process. At the onset, a TEE TEE_1 performs its bootup as usual. Ever since then, any new TEE, such as TEE_2, to join in can simply authen-

ticate itself to a random TEE which has already booted up, e.g., TEE_1. TEE_2 will prove that it's running in an enclave with homogeneous program with TEE_1 by *remote attestation*. If successful, TEE_1 will directly share K with TEE_2. This protocol doesn't require further participation of R and can be quite effective when a cluster of TEEs are managed by a single party, like a bank.

3.3. Optimization of TEE

An optimization makes use of an enclave-generated ssk_T to sign any output and epk_T to encrypt input. So only one *remote attestation* is required for the generation of (epk_T, spk_T) and the protocol can then be run repeatedly without attestation. Meanwhile, U can directly send a message to TEE without key exchange for symmetric key.

3.4. Secret-Sharing on Regulator's Secret

To enhance robustness of AudiTEE, any secret-sharing scheme [31] [32] [33] can be adopted to distribute the regulator's secret K into pieces for several regulators. Thus, a single party cannot decrypt user secret furtively and an audit needs to be granted by a majority of regulators to be effective.

3.5. Privacy-Preserving

Theorem 1. Assume that the *remote attestation* scheme of TEE and the digital signature used for TEE-generated *(spk, ssk)* are existentially unforgeable under chosen message attacks (EU-CMA), that TEE guarantees confidentiality and integrity and that the anonymity set for R is properly chosen, user can enjoy self-defined anonymity.

Proof. As the transaction is formed inside TEE, the private contents of a transaction can be protected. Meanwhile, supposing a user will post a transaction immediately after fetching its existential *proof*, the user authentication protocol only tells that a transaction is made by one of the chosen leaf nodes under the subtree of *ktag*.

3.6. Correctness and Completeness

Theorem 2. Assume that the *remote attestation* scheme of TEE and the digital signature used for TEE-generated *(spk, ssk)* are existentially unforgeable under chosen message attacks (EU-CMA), a user can form an authenticated transaction only when he owns a valid *account* and conforms to certain policy and auditor enjoys a complete audit on any user.

Proof. To break the correctness of AudiTEE, a malicious user has to generate $(epk_U, usrSpecif, r)$ not existing in the Merkle tree maintained by R but with an existential *proof* successfully leading to the root of the tree, *MR*.

The correct attachment of *ktag* helps auditor to target a set of transactions including but larger than those for its auditee. The design of Merkle tree ensures that the chosen *ktag* is bound with an existential *proof*.

As the hash function $H(\cdot)$ is one-way, the forgery of *proof* and *ktag* is computationally infeasible.

4. Experiment and Result

4.1. Experiment Design

We evaluate the performance of AudiTEE under a bank application field, including parties of users, commercial banks, auditors and central bank. Several commercial banks, as the role of service providers will enroll users and enable their access to the payment system built up with distributed ledger under the confirmation of central bank. Bank account is in the form $(epk_U, bank_U, ktag)$ to receive payment and $bank_U$ refers to an identifier for the bank enrolled U. Central bank plays the role of regulator to take care of valid *accounts* in a Merkle tree and audit request needs to be granted by it to be effective. For simplicity, the order of transaction will be decided by the central bank with its signature. Table 3 shows the concrete parameters of transaction. As explained in Algorithm 3 in Figure 7, each bank will receive and process payment request from user (including an identity *proof* fetched from central bank and other *service-dependent* data) via its TEE, TEE_{Bi}, and a half done transaction, TX', will be directly sent to the central bank from the enclave. The central bank will first verify the signature of TEE, σ_T, is authentic, the *UTXO* has not been spent and then include the hash of previous transaction, h, to complete the new transaction. Finally, it signs the new transaction to produce σ_{CB}. The settled transaction, TX, will be sent to each bank for another check on order correctness, *UTXO* freshness, signature of both TEE and central bank and stored locally.

4.2. Implementation

We implemented the above application of AudiTEE in C. All of our experiment is run on desktop with Intel (R) Core (TM) i5-7500 3.40 GHz CPU (SGX supported) and 32 GB RAM on Ubuntu 18.04. We detail each component in the following: 1) Cryptographic tools: Our prototype uses *OpenSSL (Version 1.1.1)*

Table 3. Parameter setting (banking).

$TX = (c_a, c_p, pub, ktag, \sigma_T, h, \sigma_{CB})$	
c_a	The secret of payer, payee and amount under the encryption of K for audit. $c_a = enc_K(epk_{payer}, epk_{payee}, v)$.
c_p	A secret s encrypted by payee's epk to prove ownership of the new-formed $UTXO$. $c_p = enc_{epk_{payee}}(bank_{payee}, v, s)$.
pub	The $UTXO$ spent and a new-formed $UTXO$. $UTXO = H(epk_{payee}, bank_{payee}, v, ktag, s)$.
$ktag$	A tag to indicate an anonymity *account* set of payee.
σ_T	Proof of the correct formation of a transaction in TEE (see Section 3.3).
h	The hash of prior transaction (added by central bank for order decision).
σ_{CB}	Proof of central bank to confirm the transaction.

Algorithm 3 TEE Verification in Bank Field **(inside TEE)**

Input: $info, epk_{payer}, usrSpecif, proof, r, MR, ktag_{payer}, s_{spent}, bank_{payer}, UTXO_{spent}, K, epk_{payee}, bank_{payee}, ktag_{payee}, ssk_T$
Output: TX'

1: TEE_{B_i} checks:
$info = H(epk_{payer}, usrSpecif, r)$
$true \leftarrow merkleVerify(proof, info, MR)$
/* MR is obtained from central bank. */
$true \leftarrow ktagVerify(proof, ktag_{payer})$
$UTXO_{spent} = H(epk_{payer}, bank_{payer}, v, ktag_{payer}, s_{spent})$
/* To verify the ownership of the $UTXO$ spent. */
the payment conforms to policy defined in $usrSpecif$
2: set $c_a := enc_K(epk_{payer}, epk_{payee}, v)$
3: set $s := rand(\lambda)$
/* TEE generates secret s for payee to redeem the received amount. */
4: set $c_p := enc_{epk_{payee}}(bank_{payee}, v, s)$
5: set $ktag := ktag_{payee}$
6: set $UTXO := H(epk_{payee}, bank_{payee}, v, ktag, s)$
7: set $pub := (UTXO_{spent}, UTXO)$
8: set $outp := (c_a, c_p, pub, ktag)$
9: set $\sigma_T := sign_{ssk_T}(outp)$
10: set $TX' = (c_a, c_p, pub, ktag, \sigma_T)$
11: TEE_{B_i} sends TX' to central bank.
/* Confirmed by central bank and complemented with h and σ_{CB}, TX' is complete as TX. */
12: TEE_{B_i} waits for a new request and then goes to line 1.

Figure 7. Algorithm 3 for TEE verification in bank field.

library for our cryptographic blocks. We choose *SHA256* for our hash function, *AES_256_CBC* with 256-bit key and 128-bit initialization vector for symmetric encryption. The *PKCS #1 v1.5 padding* for asymmetric encryption and *PKCS #1 v2.0* for signing scheme are both with 1024-bit key. 2) Trusted Execution Environment: We use Intel SGX as our TEE component and the program is run with the support of *Occlum library OS* [34]. The server program is partitioned into trusted part, receiving, processing user request for a half done transaction (for central bank), and untrusted part, accepting and verifying final transaction from central bank and storing it locally. 3) Database and distributed ledger: *Redis* library is used as database for the storage of *UTXOs* for banks and central bank. It's also for the formation and maintenance of the Merkle tree for user management in the central bank side. Particularly, the *subscriber-publisher model* is adopted for the distribution of final transaction signed by central bank in a distributed ledger. We include central bank and commercial banks under a common channel but only the central bank is allowed to publish message (transaction) into the channel. Bank receives settled transaction from central bank and stores it locally. All honest banks will be consistent in their local ledgers. All the operation of *Redis* is controlled through *Hiredis* library.

Our evaluation responds to the following questions:

a) How expensive is it to manage identity in a Merkle tree (formation, update and retrieval)? (**Experiment 1**)

b) What's the efficiency of transaction verification in AudiTEE and how does AudiTEE system scale with the number of banks? (**Experiment 2**)

c) What's the efficiency of different audit functions in online and offline mode? (**Experiment 3**)

d) What's the performance of *UTXO* tracing in online and offline mode? (**Experiment 4**)

Experiment 1 The first experiment concerns the maintenance of Merkle tree. Experiment 1 is divided into three parts: 1) Tree formation: During the construction of the tree, each user information *epk, usrSpecif, r* (assuming 128 Bytes in total) will be hashed into *info* to become a leaf and then the tree will be fully calculated and stored locally. 2) Retrieval: In this part, some members of the tree will be fetched as that in Section 2.3. *mix* refers to number of the mixing *accounts*. 3) Update: In the update test, a specific *info* will be changed, entailing certain update of the tree. The first test was run once while the latter two run 10, 000 times. We recorded the performance under different parameter setting.

According to our experiment outcome in Figure 8, the formation of the tree will grow exponentially with k in time. We stress the formation of the tree, although most costly, is a one-time process for R. The fetching with anonymity set only grows slightly when k increases. The exponentially growth of anonymity set leads to a similar growth in fetching time. The update of a leaf grows linearly when k increases. We stress that all the above processes can be separated into several servers to boost velocity.

Experiment 2 The second experiment pertains to the performance of transaction verification and the overall throughput of AudiTEE. We first measure the verification component in both online and offline mode. Only one server with single thread is interacting with one client during online test. In offline test, we put all transaction requests locally and server simply fetches them from disk. Transaction verification confirms: 1) the request is from a valid *account*; 2) the ownership of *UTXO* spent; 3) the payment is correct and policy-conformed. More details have been included in Algorithm 3 in Figure 7. All the above tests were performed on mimic request for 1000 times online and 10,000 offline. Then

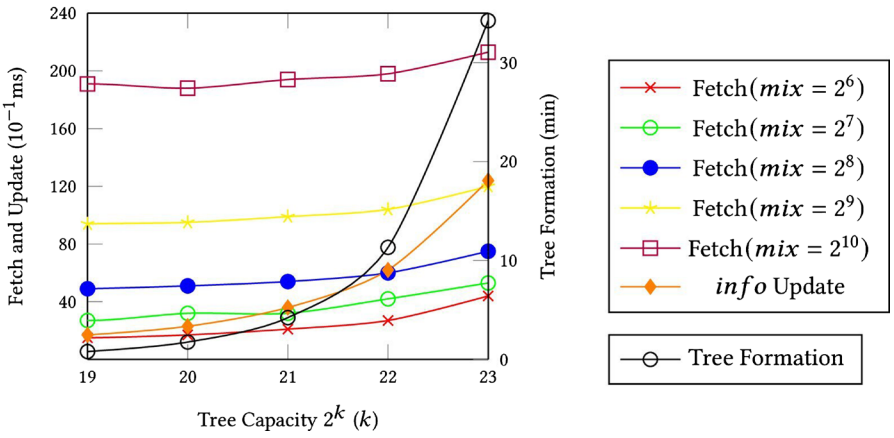

Figure 8. Merkle tree operation.

we make things more integral, with central bank, banks and clients to have a more authentic measurement on its efficiency. Each server is run with four threads. To be emphasized, each socket between client and bank will be used only once to be more realistic. We adopt the optimization in Section 3.3 and all the data sent is encrypted under AES and RSA encryption except the broadcast of half done transaction from SGX and the settled one from central bank.

Table 4 shows the desirable speed of transaction verification. The comparison of online and offline mode points out the underuse of verification function. This may be resolved by an SGX shared by several server threads. The data required by verification and the transaction are both with desirable size. In Figure 9, the throughput result itself manifests its capability of high frequency application such as trading and exchange. As service provider is potential to be equipped with high-end server machine, the performance in practice can be more powerful than we've been shown.

Experiment 3 This experiment simulates various audit functions inside SGX. The auditing outcome contains the hash of epk_A and all input c_a and the secret result under the encryption of epk_A. Without loss of generality, we omit the process of *remote attestation* and recovery of K inside SGX for their unnoticeable effect when transactions under audit are substantial. In offline mode, SGX will directly fetch desirable c_a from the local. Online mode is required when the

Table 4. Transaction verification.

Tree Capacity	Data for Verification (Plaintext/Ciphertext)	Online	Offline
2^{23}	1.87 kB/2 kB	206 ms/tx	0.3 ms/tx
Halfdone Transaction Size		Settled Size	
0.39 kB		0.55 kB	

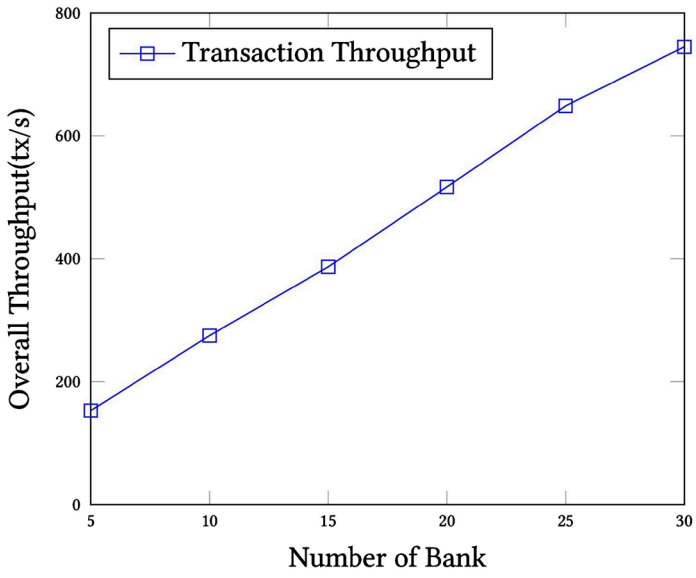

Figure 9. AudiTEE throughput.

auditor asks for extra confidentiality. c_a will be sent under a symmetric key shared between bank's SGX and the auditor. As a result, bank can have no idea which transaction is currently being audited. We performed three types of audit: 1) summing (calculate balance of a user); 2) matching (how many transaction happened between two specified users); 3) exception detection (how many transaction of a user is beyond some value). Each audit type was performed online and offline for 1,000,000 times.

Table 5 shows a satisfactory speed for audit. Realistically, an audit only concerns relatively small part of c_a on the ledger rather than all. What's more, several audit functions can actually be carried out in one audit and one audit can also be separately performed in several SGXs. Thus, there is still a considerable room for optimization.

Experiment 4 The final experiment is on the tracing of *UTXO*. In our implementation, *UTXO* is a hash commitment of epk_U, bank identifier $bank_U$, value *v*, *ktag* and a secret randomness *s*. $Bank_U$, *v*, *s* are encrypted under the receiver's private key in c_p. Thus, to identify the ownership of a *UTXO*, the user first tries to decrypt c_p to get $bank_U$, *v*, *s*. Then it hashes, together with its epk_U and *ktag* to see if $UTXO = H(epk_U, bank_U, v, ktag, s)$. Once again, the tracing may be carried out locally (offline mode), or can be passed from a bank to save disk space (online mode). Notice that this process is performed outside the SGX. Each mode was performed 1,000,000 times.

The outcome is shown in Table 6. A user only has to trace those transactions with its chosen *ktag*. A *ktag* tells the possible recipients of a transaction. As *ktag* is self-defined as part of a user's *account,* $(epk_U, bank_U, ktag)$, to receive payment, when *ktag* indicates a larger subtree, thus with a higher anonymity, it's more likely that a transaction with user-chosen *ktag* may not actually belong to *U*, but other users under the same subtree. This means a lot of redundancy, or extra effort will take place when the possible transactions are to be transferred from a full node to and verified by user to confirm their real ownership. Thus, a user can properly choose a desirable *ktag* based on the cost of network communication and

Table 5. Transaction audit.

Data Size: 68 Bytes/tx (offline)/80 Bytes/tx (online)			
Audit Type	**Summing**	**Matching**	**Exception Detection**
Online	0.017 ms/tx	0.016 ms/tx	0.017 ms/tx
Offline	0.002 ms/tx	0.002 ms/tx	0.002 ms/tx

Table 6. *UTXO* tracing.

Data Size: 160 Bytes (offline)/180 Bytes (online)	
***UTXO* Tracing**	**Time Cost**
Online	0.095 ms/tx
Offline	0.088 ms/tx

verification during the tracing of *UTXO*.

5. Conclusion and Future Work

AudiTEE proposes a practical framework for speedy, general-purpose and confidential audit on privacy-preserving distributed ledger. It can be implemented in various applications to enable user privacy with effective regulation. Powered by KAMT protocol, AudiTEE tackles several concerns for TEE in realising efficient *account* management in large scale. Meanwhile, the user-defined anonymous authentication for a user only relies on the verification of Merkle path, which is much faster than signature-based method. Specifically, a special *ktag* defined in KAMT protocol helps to realize fast audit on user while still preserving *unlinkability* and completeness. AudiTEE also allows various policies on user from regulator. Our evaluation shows that AudiTEE has satisfactory performance in all its core functionalities.

In the future, the current work can be extended in several aspects. Firstly, AudiTEE will be implemented in more applications for testing and improving. Secondly, extra attention should be focused on possible side-channel attacks on TEE and corresponding solutions. Finally, further research will be required to appease the need for a central trusted party, namely, regulator in AudiTEE system model to make the framework even more robust.

Acknowledgements

This work was in part supported by National Natural Science Foundation of China (Grant Nos. 61932011), Guangdong Basic and Applied Basic Research Foundation (Grant No. 2019B1515120010), Guangdong Key R&D Plan2020 (No. 2020B0101090002), Key-Area Research and Development Program of Guangdong Province (No. 2020B0101090004) and Special Funds for the Cultivation of Guangdong College Students' Scientific and Technological Innovation ("Climbing Program" Special Funds.) (No. PDJH2021A0050).

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Maull, R., Godsiff, P., Mulligan, C., Brown, A. and Kewell, B. (2017) Distributed Ledger Technology: Applications and Implications. *Strategic Change*, **26**, 481-489. https://doi.org/10.1002/jsc.2148

[2] Rauchs, M., Glidden, A., Gordon, B., Pieters, G.C., Recanatini, M., Rostand, F., Vagneur, K. and Zhang, B.Z. (2018) Distributed Ledger Technology Systems: A Conceptual Framework. https://doi.org/10.2139/ssrn.3230013

[3] Wikipedia Audits. https://en.wikipedia.org/wiki/Audit

[4] American Society for Quality. What Is Auditing? https://asq.org/quality-resources/auditing

[5] Investor.gov. The Laws That Govern the Securities Industry. https://www.investor.gov/introduction-investing/investing-basics/role-sec/laws-govern-securities-industry

[6] REUTERS (2020) China Fines Luckin Coffee and Linked Firms a Total of $9 Million. https://www.reuters.com/article/us-luckin-coffee-investigation-fine-idUSKCN26D07G

[7] IRS Charity and Nonprofit Audits. https://www.irs.gov/charities-non-profits/exempt-organizations-audit-process

[8] AICPA What Is a Governmental Audit? https://www.aicpa.org/interestareas/governmentalauditquality/information-on-governmental-audits.html

[9] Schmitz, J. and Leoni, G. (2019) Accounting and Auditing at the Time of Blockchain Technology: A Research Agenda. *Australian Accounting Review*, **29**, 331-342. https://doi.org/10.1111/auar.12286

[10] Mitani, T. and Otsuka, A. (2020) Confidential and Auditable Payments. *International Conference on Financial Cryptography and Data Security*, Kota Kinabalu, 10-14 February 2020, 466-480. https://doi.org/10.1007/978-3-030-54455-3_33

[11] Androulaki, E., Camenisch, J., Caro, A.D., Dubovitskaya, M., Elkhiyaoui, K. and Tackmann, B. (2020) Privacy-Preserving Auditable Token Payments in a Permissioned Blockchain System. *Proceedings of the* 2*nd ACM Conference on Advances in Financial Technologies*, New York, 21-23 October 2020, 255-267. https://doi.org/10.1145/3419614.3423259

[12] Cecchetti, E., Zhang, F., Ji, Y., Kosba, A., Juels, A. and Shi, E. (2017) Solidus: Confidential Distributed Ledger Transactions via PVORM. *Proceedings of the* 2017 *ACM SIGSAC Conference on Computer and Communications Security*, Dallas, 30 October-3 November 2017, 701-717. https://doi.org/10.1145/3133956.3134010

[13] Chen, Y., Ma, X.C., Tang, C. and Au, M.H. (2020) PGC: Decentralized Confidential Payment System with Auditability. *European Symposium on Research in Computer Security*, Guildford, 14-18 September 2020, 591-610. https://doi.org/10.1007/978-3-030-58951-6_29

[14] Naganuma, K., Yoshino, M., Sato, H. and Suzuki, T. (2017) Auditable Zerocoin. 2017 *IEEE European Symposium on Security and Privacy Workshops*, Paris, 26-28 April 2017, 59-63. https://doi.org/10.1109/EuroSPW.2017.51

[15] Garman, C., Green, M. and Miers, I. (2016) Accountable Privacy for Decentralized Anonymous Payments. *International Conference on Financial Cryptography and Data Security*, Christ Church, 22-26 February 2016, 81-98. https://doi.org/10.1007/978-3-662-54970-4_5

[16] Narula, N., Vasquez, W. and Virza, M. (2018) zkledger: Privacy-Preserving Auditing for Distributed Ledgers. 15*th USENIX Symposium on Networked Systems Design and Implementation*, Renton, 9-11 April 2018, 65-80.

[17] Dinh, T.T.A., Saxena, P., Chang, E.-C., Ooi, B.C. and Zhang, C.W. (2015) M2r: Enabling Stronger Privacy in Mapreduce Computation. 24*th USENIX Security Symposium* (*USENIX Security* 15), Washington DC, 12-14 August 2015, 447-462.

[18] Zheng, W.T., Dave, A., Beekman, J.G., Popa, R.A., Gonzalez, J.E. and Stoica, I. (2017) Opaque: An Oblivious and Encrypted Distributed Analytics Platform. 14*th USENIX Symposium on Networked Systems Design and Implementation*, Boston, 27-29 March 2017, 283-298.

[19] Arasu, A., Blanas, S., Eguro, K., Kaushik, R., Kossmann, D., Ramamurthy, R. and Venkatesan, R. (2013) Orthogonal Security with Cipherbase. CIDR.

[20] Baumann, A., Peinado, M. and Hunt, G. (2015) Shielding Applications from an Untrusted Cloud with Haven. *ACM Transactions on Computer Systems* (*TOCS*), **33**, 1-26. https://doi.org/10.1145/2799647

[21] Bajaj, S. and Sion, R. (2013) TrustedDB: A Trusted Hardware-Based Database with Privacy and Data Confidentiality. *IEEE Transactions on Knowledge and Data Engineering*, **26**, 752-765. https://doi.org/10.1109/TKDE.2013.38

[22] Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G. and Russinovich, M. (2015) VC3: Trustworthy Data Analytics in the Cloud Using SGX. 2015 *IEEE Symposium on Security and Privacy*, San Jose, 21-22 May 2015, 38-54. https://doi.org/10.1109/SP.2015.10

[23] Milutinovic, M., He, W., Wu, H. and Kanwal, M. (2016) Proof of Luck: An Efficient Blockchain Consensus Protocol. *Proceedings of the* 1*st Workshop on System Software for Trusted Execution*, Trento, 12-16 December 2016, 1-6. https://doi.org/10.1145/3007788.3007790

[24] Lind, J., Eyal, I., Kelbert, F., Naor, O., Pietzuch, P. and Sirer, E. (2017) Teechain: Scalable Blockchain Payments Using Trusted Execution Environments.

[25] Nakamoto, S. (2019) Bitcoin: A Peer-to-Peer Electronic Cash System. Manubot.

[26] Fedorov, S., Li, W.T. and Karame, G. (2019) Efficient Validation of Transaction Policy Compliance in a Distributed Ledger System. Google Patents, US Patent App. 15/916,293.

[27] Zhao, C.C., Saifuding, D., Tian, H.L., Zhang, Y. and Xing, C.X. (2016) On the Performance of Intel Sgx. 2016 13*th Web Information Systems and Applications Conference* (*WISA*), Wuhan, 23-25 September 2016, 184-187. https://doi.org/10.1109/WISA.2016.45

[28] Göttel, C., Pires, R., Rocha, I., Vaucher, S., Felber, P., Pasin, M. and Schiavoni, V. (2018) Security, Performance and Energy Trade-Offs of Hardware-Assisted Memory Protection Mechanisms. 2018 *IEEE* 37*th Symposium on Reliable Distributed Systems* (*SRDS*), Salvador, 2-5 October 2018, 113-142. https://doi.org/10.1109/SRDS.2018.00024

[29] Danezis, G. and Meiklejohn, S. (2015) Centrally Banked Cryptocurrencies. https://doi.org/10.14722/ndss.2016.23187

[30] Walker, P. and Venables, P.J. (2017) Cryptographic Currency for Securities Settlement. Google Patents, US Patent 9,704,143.

[31] Beimel, A. (2011) Secret-Sharing Schemes: A Survey. *International Conference on Coding and Cryptology*, Qingdao, 30 May-3 June 2011, 11-46. https://doi.org/10.1007/978-3-642-20901-7_2

[32] Shamir, A. (1979) How to Share a Secret. *Communications of the ACM*, **22**, 612-613. https://doi.org/10.1145/359168.359176

[33] Blakley, G.R. (1979) Safeguarding Cryptographic Keys. *International Workshop on Managing Requirements Knowledge* (*MARK*), New York, 4-7 June 1979, 313. https://doi.org/10.1109/MARK.1979.8817296

[34] Shen, Y.R., Tian, H.L., Chen, Y., Chen, K., Wang, R.J., Xu, Y., Xia, Y.B. and Yan, S.M. (2020) Occlum: Secure and Efficient Multitasking inside a Single Enclave of Intel Sgx. *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, Lausanne, 16-20 March 2020, 955-970. https://doi.org/10.1145/3373376.3378469