An Accountable Decryption System Based on Privacy-Preserving Smart Contracts

Rujia Li[1,2], Qin Wang[3], Feng Liu[1], Qi Wang[1], and David Galindo[2,4]

[1] Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China.
liuf2017@mail.sustech.edu.cn, wangqi@sustech.edu.cn
[2] University of Birmingham, Edgbaston, B15 2TT, Birmingham, United Kingdom
rxl635@student.bham.ac.uk, d.galindo@cs.bham.ac.uk
[3] Swinburne University of Technology, Melbourne, VIC 3122, Australia
qinwang@swin.edu.au
[4] Fetch.AI, St John's Innovation Center, Cambridge, CB4 0WS, United Kingdom

Abstract. Accountability is a fundamental after-the-fact approach to detect and punish illegal actions during the execution of a warrant for accessing users' sensitive data. To achieve accountability in a security protocol, a trusted authority is required, denoted as *judge*, to faithfully cooperate with the rest of the entities in the system. However, malicious judges or uncooperative protocol participants may void the accountability mechanism in practice, for example by fabricating fake evidence or by refusing to provide any evidence at all. To provide remediation to these issues, in this paper we propose *Fialka*, a novel accountable decryption system based on privacy-preserving smart contracts (PPSC). The neutrality that is inherent to a secure blockchain platform is inherited by PPSC which are then used in our approach as an accountable key manager as well as a transparent judge. To the best of our knowledge, we present the first PPSC-based accountable decryption system to increase the transparency of warrant execution with formal definitions and proofs. Furthermore, we provide and evaluate a prototype implementation using the PPSC-enabled platform Oasis Devnet, which additionally demonstrates the feasibility of Fialka.

Keywords: Accountability, Privacy-Preserving Smart Contract, Blockchain

1 Introduction

Accountable cryptographic protocol is increasingly crucial in sensitive personal data protection. We focus on the following scenario. Law enforcement or intelligence agencies may demand access to personal encrypted data held by service providers, and sometimes even require access to the communication metadata that is closely related to sensitive information of individuals. In most cases, a granted warrant is needed from a legal authority. However, data owners have no way to know when and how law enforcement collects and accesses their sensitive

data. In particular, abuses of granted warrant of decryption may easily happen since the overseers cannot verify whether the practical investigation activities match the scope permitted in the document. Therefore, accountability mechanisms is a critical *after-the-fact* remediation technique to deter investigators, since it provides an instant evidence to detect malicious or deviant behaviors, which increases the transparency of warrant execution.

However, achieving accountability is tricky, and requires additional roles involved. The investigators cannot autonomously convince others of the accountability of their actions. They need to resort to one or more neutral trusted parties, usually named *judge(s)*, to audit their actions. More specifically, an accountability mechanism requires each investigator to generate evidence on their warrant execution. This evidence is then examined by the judge to detect dishonest behaviors or declare the examined participant compliant. This approach relies heavily on faithful cooperation of the judge and the investigator, as a malicious judge or dishonest investigator may undermine the accountability mechanism. If the investigator rejects to cooperate with the judge in order to provide the required evidence, or if the judge themselves examine fake evidence or apply the wrong examination procedure, outsides cannot audit investigators' decryption actions. In this paper, we generalise the above example as a standard case, in which an investigator obtains an order from a court, and his access of users' data needs to be audited by the judge. The discussed challenges lead to the following research question:

Is it possible to design an accountability mechanism guaranteeing that (1) the judge honestly checks the evidence; (2) the investigator does not refuse to provide the evidence trail of their actions?

Based on the previous discussion, the answer would intuitively be "NO". Firstly, it is difficult to guarantee that a judge will always be secure and reliable. Even if the judge claims to be neutral, she faces the threat of being attacked or provided with misleading evidence. Once the judge is compromised, the accountability mechanism fails as it cannot be applied. Undoubtedly, multiple judges may mitigate such concerns, but the judge collusion issue cannot be effortlessly overcome. Secondly, asking the investigator to neutrally create a piece of honest evidence also confronts difficulties. The isolated local execution environment makes it potentially easy and profitable for the investigator to generate fake evidence while incurring a low risk of being detected. Several proposals [3, 9] employed a certain trusted hardware to aid the evidence generation. Intuitively, physical hardware is more secure and reliable since the evidence logic and its measurement are hardcoded in non-volatile storage. However, the risk of compromised hardware still exists [17].

Blockchain-based smart contracts [26, 27] have been used in [4, 14, 23] as a building block to implement the judge. Roughly speaking, a smart contract is composed of a set of protocols to be automatically executed in a distributed network, which naturally guarantees the neutrality and behaviour of the judge thus obtained. However, the input/output data of the smart contract is transparent to the public, which limits its usage in some scenarios. For instance, private

key-dependent protocols such as decryption are executed in an isolated local environment. A transparent smart contract cannot prevent the investigator from producing fake evidence if it does not have access to the secret key material. But in the latter case, the secrecy of the private key material would be compromised. Privacy-preserving smart contracts (PPSC) [6, 13, 16, 25, 29] inherit the security, availability and neutrality benefits of smart contracts while additionally protecting the privacy of the contract data. It naturally could act as a high-level cryptographic primitive to aid in the evidence generation involving local protocol executions. For example, PPSC could be used to implement a private key manager to make decryption accountable.

In this paper, we propose *Fialka*, a novel transaction-triggering accountability framework using PPSC to make investigators accountable for executing decryption calls. Our framework prevents the decryption queries evidence from being maliciously generated (e.g. hidden) while guaranteeing the authenticity of the evidence. More precisely, *Fialka* combines PPSC with an IND-CCA secure public key encryption (PKE) scheme [15] at the protocol level to construct an accountability mechanism. PPSC cryptographically hides a secret random number used as an additional decryption key, where external investigators have to interact with PPSC for the execution of decryption warrant. The secret key will be extracted by invoking the decryption-related smart contract, which consequently generates a transaction-based evidence as an on-chain record. After that, another smart contract plays the role of the judge who transparently checks the transaction to decide whether the decryption is legal in a specific setting. The accountability is thereby achieved. Additionally, our framework inherits the benefit of high availability from the underlying blockchain protocol. This further improves reliability of the PPSC-based judge. Our contributions are summarized here:

- We propose an accountable decryption system called *Fialka* that combines the techniques of PPSC and PKE.
- We formally define our system and provide a security analysis of its accountability properties, namely *fairness* and *completeness*.
- We provide a prototype implementation based on the PPSC platform Oasis Devnet [1, 8], and evaluate its running time and gas cost.

The rest of our paper is structured as follows. Some related studies are discussed in Section 2. Definitions and building blocks are detailed in Section 3. In Section 4, we present the formal model with its property definitions. In Section 5, we provide the design of Fialka. Both the proof and security analysis are presented in Section 6. Implementations and evaluations are discussed in Section 7 and Section 8. Finally, Section 9 presents summaries and future work.

2 Related work

The smart contract-based accountability approach has been studied comprehensively recently. Xu *et al.* [28] proposed a remotely decentralized data auditing scheme for network storage service, where accountability is achieved by involving

smart contract as a third-party auditor to notarize the integrity of outsourced data. Azaria *et al.* proposed MedRec [4], in which an Ethereum [27] smart contract is used as a meta-data agent to manage the permission of data usage, making patients' choices accountable. Neisse *et al.* [23] proposed a blockchain-based framework for data accountability and provenance tracking. However, a pure smart contract does not provide a complete accountable protocol, since it cannot guarantee the authenticity of the input (i.e. the submitted evidence). In other words, even if the smart contract is neutral and trustful, a client may provide fake evidence to the smart contract without being detected.

Several solutions have been proposed to ensure the authenticity of the submitted evidence. Among them, equipping entities with secure hardware devices [3, 24, 17] is an attractive approach. Alder *et al.* [3] employed Intel SGX [10] to produce a verifiable measurement of the resource usage in each function invocation. Luo *et al.* [21] applied the Intel SGX with blockchain to a data sharing scheme, where the decryption process also relied on the confidentiality of secure hardware devices. The hardware-based approach is intuitively reliable and robust, since trusted hardware devices cannot change the evidence generation rules once loaded. However, the security cannot be guaranteed when adversaries successfully attack the hardware. The approach using multiple hardware may mitigate such security concerns to a certain extent. Unfortunately, the efficiency issue and incentives issue cannot be easily overcome. Another promising approach is directly employing the protocol execution result as the evidence, such as using the ciphertext and the private key as evidence. A typical example is accountable identity-based encryption [11, 12, 19], where a judge can decide whether a PKG is malicious by showing cryptographic proofs that contain the decryption key. However, such an approach lacks practicality.

Privacy-preserving smart contract (PPSC) is a special contract that aims to make the contract state private. The techniques on PPSC have been studied extensively in the recent years. Enigma [29] provided a decentralized confidential computation platform by employing multi-party computation. Hawk [16], Zether [6] and Zkay [25] realized privacy-preserving smart contract by heavily relying on zero-knowledge proofs. Ekiden [8] and Microsoft Coco framework [22] employed Intel SGX to achieve confidential smart contracts. Essentially, PPSC is a decentralized confidential computing technology, which inherits the benefit of transparent execution from a smart contract while additionally protecting the privacy of contract data. Our accountable system leverages the main benefits of PPSC. The transparent execution of smart contracts ensures the judge honestly checks the evidence, and the trigger mechanism of contract execution enforces investigators to invoke PPSC through transactions, which ensures investigators neutrally provide the evidence.

3 Preliminaries

Let λ be the security parameter, and $negl(\lambda)$ be a negligible function. The challenger and adversary are represented as $\mathcal{C}$ and $\mathcal{A}$, respectively. We use the nota-

tion $\eth$ to denote the game in security deduction, *adv* to represent the advantage that the adversary holds, and the notation "$\approx$" to show these two games are computationally indistinguishable. The message space is denoted as $\mathcal{M}$.

3.1 Privacy-Preserving Smart Contract

Smart contract was first proposed by Nick Szabo [26] and further developed by Ethereum [27] in the blockchain system. A blockchain-based smart contract consists of two mutually interacting components: contract state and operational code [27]. The contract state covers the input and output of the operational code, while operational code specifies operations/commands to store or transfer the contract state. The intuitive target for a privacy-preserving smart contract is to make the contract state private. However, purely protecting the privacy of the state against the public is not sufficient, especially when multiple entities are involved in one contract to finish complex cryptographic tasks. The state in a contract is required to reach a new consensus view after the execution of operational codes, in which this rule is followed by PPSC projects such as Zether [6], Ekiden [8] and *Oasis Devnet* [1]. Thus, we capture two main PPSC principles: **P.1** the contract state should be protected against the public; **P.2** the authorized entities should see the same private data view.

PPSC-based Accountability. The initial contract state and the operational code will reach consensus after the successful deployment. After that, two approaches can trigger the execution of the operational code: the internal schedule code and the external message call. The first approach allows the operational code to execute periodically. However, it cannot complete a complex task due to massive gas consumption [27]. Thus, to trigger the execution, an external message call with sufficient gas is crucial. PPSC inherits the state triggering mechanism from smart contracts, namely, the state-changing is based on external message call. For example, *Oasis Devnet* [1] requires an external caller to firstly build a secure channel with the TEE-protected smart contract, and then the transition of the private state is accomplished through this channel when a transaction call is provided. *Origo Network* [2] reveals the private input to an off-line executor and then allows the executor to provide a ZKP-proof transaction for online state transferring. Zether [6] funds the Zether tokens (ZTH) by sending some Ethereum [27] tokens (ETH) and converts ZTH back to ETH by sending a ZK-proof transaction. In summary, a transaction is required to trigger the execution and obtain the state from PPSC. Therefore, by tracing the sender who sends the transaction, the auditor implicates the wrongdoing of the contract caller. Based on the above analysis, we give a formal definition of PPSC.

Definition 1 ($\widehat{\boldsymbol{PPSC}}$) *A Privacy-Preserving Smart Contract (PPSC) is a private state machine built on top of a blockchain system and can be modeled by* 5*-tuple* $(\mathcal{S}, \mathcal{S}', \mathcal{T}, \mathsf{s}, \mathbb{B})$ *and a transition function* $f : \mathcal{S} \otimes \mathcal{T} \xrightarrow{\mathbb{B}} \mathcal{S}'$*, where* $\mathcal{S}$ *represents a set of private state with the initial state* s*,* $\mathcal{S}'$ *is the new state set after the specified operations,* $\mathcal{T}$ *means the publicly visible transactions that can trigger the*

execution of a contract, and $\mathbb{B}$ *represents the blockchain oracle which provides the execution environment.*

- **Deploy** $< \mathsf{bytecode} > \otimes \mathsf{Tx} \rightarrow (< \mathsf{opcode} >, < \mathsf{reqcode} >, \mathsf{s})$: The deployment is triggered by a transaction Tx, where $\mathsf{Tx} \in \mathcal{T}$. It takes the binary code $< \mathsf{bytecode} >$ as input, and outputs the private state s. The contract is compiled into $< \mathsf{opcode} >$ and $< \mathsf{reqcode} >$, where $< \mathsf{opcode} >$ specifies the operation set to be executed and $< \mathsf{reqcode} >$ defines the conditions depending on which the operation of $< \mathsf{opcode} >$ can be conducted.
- **Transfer** $< \mathsf{input} > \otimes \mathcal{S} \otimes \mathsf{Tx} \xrightarrow{\mathbb{B}} \mathcal{S}'$: By sending a transaction Tx with the input $< \mathsf{input} >$, the current private state $\mathcal{S}$ is transited to the new private state $\mathcal{S}'$ under the blockchain oracle $\mathbb{B}$. The new state $\mathcal{S}'$ returns only when Tx satisfies the condition defined in $< \mathsf{reqcode} >$, *i.e.*, $\mathsf{Tx} \in < \mathsf{reqcode} >$.
- **Access** $\mathcal{S} \otimes \mathsf{Tx} \xrightarrow{\mathbb{B}} \mathcal{S}$: By sending a query transaction Tx through the blockchain oracle $\mathbb{B}$, the private state returns only when Tx satisfies the condition predefined in $< \mathsf{reqcode} >$, *i.e.*, $\mathsf{Tx} \in < \mathsf{reqcode} >$.

Based on the above syntax, we provide four PPSC security properties: *state-privacy*, *state-consistency*, *transaction-transparency* and *transaction-unforgeability*. The state-privacy guarantees that the state is protected against the public (Principle **P.1**). Only the caller who satisfies the predefined conditions can learn the state. Meanwhile, the state-consistency ensures that a smart contract shares the same data view after operational code is executed (Principle **P.2**). The transaction-transparency ensures that transactions triggering the execution of PPSC can be freely queried, while the transaction-unforgeability guarantees the transactions (as evidence) are reliable and authentic without being forged or cheated. PPSC is secure when these four security properties are all satisfied.

Definition 2 *PPSC achieves state-privacy, if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $negl(\lambda)$ *such that* $adv^{\mathfrak{I}_{privacy}}_{\mathcal{A},ppsc}(\lambda) < negl(\lambda)$*, where* $adv^{\mathfrak{I}_{privacy}}_{\mathcal{A},ppsc}(\lambda)$ *is the advantage that* $\mathcal{A}$ *successfully obtains the private state without satisfying the condition predefined in* $< \mathsf{reqcode} >$*.*

Definition 3 *PPSC achieves state-consistency, if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $negl(\lambda)$ *such that* $adv^{\mathfrak{I}_{cons}}_{\mathcal{A},ppsc}(\lambda) < negl(\lambda)$*, where* $adv^{\mathfrak{I}_{cons}}_{\mathcal{A},ppsc}(\lambda)$ *is the advantage that* $\mathcal{A}$ *obtains a valid state* $\mathcal{S}^{\star}$ *through the algorithm* ***Transfer*** *in which* $s^{\star}$ *does not belong to the state set* $\mathcal{S}$*, i.e.,* $s^{\star} \notin \mathcal{S}$*.*

Definition 4 *PPSC achieves transaction-transparency, if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $negl(\lambda)$ *such that* $adv^{\mathfrak{I}_{tran}}_{\mathcal{A},ppsc}(\lambda) < negl(\lambda)$*, where* $adv^{\mathfrak{I}_{tran}}_{\mathcal{A},ppsc}(\lambda)$ *is the advantage that* $\mathcal{A}$ *successfully obtains a transaction* $\mathsf{Tx}^{\star}$ *through calling the algorithm* ***Transfer*** *with the condition* $\mathsf{Tx}^{\star} \notin \mathcal{T}$*.*

Definition 5 *PPSC achieves transaction-unforgeability, if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $negl(\lambda)$ *such that* $adv^{\mathfrak{I}_{unforg}}_{\mathcal{A},ppsc}(\lambda) < negl(\lambda)$*, where* $adv^{\mathfrak{I}_{unforg}}_{\mathcal{A},ppsc}(\lambda)$ *is the advantage that* $\mathcal{A}$ *successfully forges a transaction* $\mathsf{Tx}^{\star}$ *and obtains the state through* $\mathsf{Tx}^{\star}$*.*

3.2 Decision Linear Assumption

Decision Linear Assumption [5, 15] is based on the Linear Problem. Due to limitation of space, we skip a full definition and refer to Appendix A for details.

4 General Construction

4.1 System Overview

Our system consists of four entities (see Fig 1.a): common users (sender/receiver), investigator, key management smart contract (PPSC-KM), and auditor smart contract (PPSC-AD). PPSC-KM is used to manage investigators' decryption keys. PPSC-AD is employed as a "judge" to decide whether the event of the investigator's decryption is conducted under the court-issued order. A detailed workflow is shown as follows. The sender encrypts messages with a random number, which is hidden in PPSC-KM, and then it sends the encrypted message to the receiver. The receiver decrypts the ciphertext as normal. Meanwhile, the investigator who obtained a court-issued order decrypts the ciphertext by fetching the random number from PPSC-KM. When a query is sent to PPSC-KM, the actions will be recorded through a transaction as the evidence. Next, PPSC-AD will check the evidence to report malicious decryption. In our protocol, PPSC-KM and PPSC-AD are, respectively, abbreviated as $\widehat{\mathsf{c}}_{\mathsf{km}}$ and $\widehat{\mathsf{c}}_{\mathsf{ad}}$ for simplicity. Formally, we provide the general construction as follows.

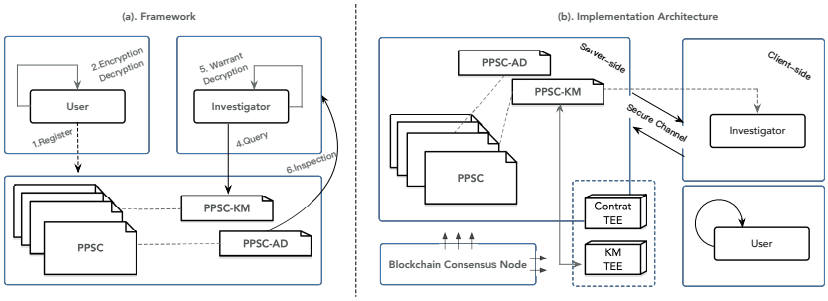

Fig. 1. System Framework & Architecture

Setup $(pms, \widehat{\mathsf{c}}_{\mathsf{km}}, \widehat{\mathsf{c}}_{\mathsf{ad}}) \leftarrow \mathsf{Setup}(1^\lambda, codes)$. The algorithm takes as input a security parameter λ and binary codes $codes$, and returns public parameters pms and two contracts $\widehat{\mathsf{c}}_{\mathsf{km}}$, $\widehat{\mathsf{c}}_{\mathsf{ad}}$.

Key Generation $(pk, sk, tk) \leftarrow \mathsf{KeyGen}(pms)$. The algorithm takes as input pms, and returns receiver's key pair (pk, sk), and a secret tag key tk.

Registration $\mathsf{s} \xleftarrow{\mathbb{B}} \mathsf{Register}(tk, \breve{P})$. The algorithm takes as input a master key tk and accountability policies $\breve{P}$, and returns an initial state $\mathsf{s} \in \mathcal{S}$. The tk and policies $\breve{P}$ are added to $\widehat{\mathsf{c}}_{\mathsf{km}}$ and $\widehat{\mathsf{c}}_{\mathsf{ad}}$, respectively.

Encryption $ct \leftarrow \mathsf{Encrypt}(tk, pk, m)$. This algorithm takes as input tk, pk, and a message m, and then returns a ciphertext ct.

Decryption $m \leftarrow \mathsf{Decrypt}(sk, ct)$. The algorithm takes as input sk, ct, and returns $m \in \mathcal{M}$.

Warrant Decryption $(m, \mathsf{Tx}) \xleftarrow{\mathbb{B}} \mathsf{WDecrypt}(r, r_1, \mathsf{s}, ct)$. This algorithm takes as input a random number r, r_1, and s (including tk) calls the algorithm **Transfer** described in Section 3.1, and returns $m \in \mathcal{M}$.

Inspection $\mathsf{true/false} \xleftarrow{\mathbb{B}} \mathsf{Inspect}(\mathsf{Tx}, \breve{P})$. This algorithm takes as input $\breve{P}$ and Tx, and returns the inspection result. The result true indicates that the authorized decryption is legitimately executed under the warrant, and vice versa.

The procedure of **Decryption** represents normal decryption run by offline users, whereas **Warrant Decryption** is run by the investigators who are forced to leave evidence each time of decryption. Meanwhile, the access control conditions in $\widehat{\mathsf{c}}_{\mathsf{km}}$ and the accountability policies in $\widehat{\mathsf{c}}_{\mathsf{ad}}$ are set as the same. We notice that the logic of **Warrant Decryption** might be confusing: the $\widehat{\mathsf{c}}_{\mathsf{km}}$ has defined the access control conditions for investigators. Is the accountability necessary for investigators' decryption? We clarify that access control and accountability in our system play different roles. The access control condition in $\widehat{\mathsf{c}}_{\mathsf{km}}$ is similar to an order issued by the court, which describes the actions that an investigator should do but not yet, whereas the accountability policies in $\widehat{\mathsf{c}}_{\mathsf{ad}}$ are responsible for checking the actions an investigator has done (e.g., whether an investigator has executed the decryption under a warrant). We define *malicious decryption* as: the investigator's decryption does not match the actions permitted in the issued orders.

4.2 Security Definitions

Our *Fialka* system is denoted by Π, and above algorithms are abbreviated as: Set, Gen, Reg, Enc, Dec, WDec, and Insp, respectively. We assume an investigator has already obtained a warrant from a court, and his access to users' plaintext needs to be audited by the judge. Inspired by [18], the investigator should obtain fair treatment, neither being framed for the legitimate investigation nor being escaped from the punishment for wrongdoings. We captures two properties *w.r.t* accountability: *fairness* and *completeness*.

Fairness. This property prevents the judge from framing honest investigators. An honest investigator should follow the pre-defined policies and return true. We consider the adversary $\mathcal{A}$ who imitates an honest investigator, and then maliciously executes the warrant/order attempting to frame him.

Definition 6 (Fairness) *Fialka satisfies fairness, if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that $adv^{\Im_{fair}}_{\mathcal{A},\Pi}(\lambda) < negl(\lambda)$ where $adv^{\Im_{fair}}_{\mathcal{A},\Pi}(\lambda)$ is the advantage of $\mathcal{A}$ wins the game $\Im_{fair}$ defined as,*

- **Initialization***. The system configures the parameters $pms = \perp$, and creates $\widehat{\mathsf{c}}_{\mathsf{km}}$ and $\widehat{\mathsf{c}}_{\mathsf{ad}}$ by running the algorithm Set. Then, $\mathcal{C}$ generates the secret key tk by running the algorithm Gen. Next, $\mathcal{C}$ registers the tk and decryption policies $\breve{P}$ to the $\widehat{\mathsf{c}}_{\mathsf{km}}$ and $\widehat{\mathsf{c}}_{\mathsf{ad}}$, respectively.
- **Actions***. At each round, the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ execute the following algorithms. (1) $\mathcal{A}$ generates the key pair (sk_a, pk_a) by running the algorithm Gen. (2) $\mathcal{C}$ inputs the public key pk_a, message m, a random numbers r and a secret key tk, and then obtains the ciphertext ct by running the algorithm Enc. (3) $\mathcal{C}$ runs the algorithm **Transfer**, and then returns r_2 and Tx to the $\mathcal{A}$. (4) $\mathcal{A}$ inputs r_2, the ciphertext ct, and outputs the message m by running the algorithm WDec. (5) $\widehat{\mathsf{c}}_{\mathsf{ad}}$ executes the algorithm Insp with the input Tx, and return the inspection result.
- **Challenge**. Assume that $\mathcal{A}$ executes above actions at most for l times, and obtains a set $\mathcal{T} = \{\mathsf{Tx}_0, \mathsf{Tx}_1, ..., \mathsf{Tx}_l\}$. $\mathcal{A}$ wins if $\mathcal{A}$ generates a transaction $\mathsf{Tx}^\star$ satisfying the conditions: $\mathsf{false} \leftarrow \mathsf{Insp}(\mathsf{Tx}^\star, \breve{P}) \wedge \mathsf{Tx}^\star \notin \mathcal{T}$.

Completeness. This property guarantees that the judge always punishes the users who misbehave. To define *completeness*, we consider an adversary $\mathcal{A}$ aims to evade the responsibility of illegally executing the authorized decryption.

Definition 7 (Completeness) *Fialka satisfies completeness, if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that $adv^{\ni_{comp}}_{\mathcal{A},\Pi}(\lambda) < negl(\lambda)$, where $adv^{\ni_{comp}}_{\mathcal{A},\Pi}(\lambda)$ is the advantage of $\mathcal{A}$ wins $\ni_{comp}$ defined as,*

- **Initialization** and **Actions**. The steps are same with that in fairness game labeled with $(\star)$.
- **Challenge**. Assume that $\mathcal{A}$ executes the above action at most for l times, and then obtains a set of ciphertext-transaction tuple $\{\mathcal{C}, \mathcal{T}\} = \{(ct_0, \mathsf{Tx}_0), (ct_1, \mathsf{Tx}_1), ..., (ct_l, \mathsf{Tx}_l)\}$. $\mathcal{A}$ wins if $\mathcal{A}$ successfully generates a new tuple $(ct^\star, \mathsf{Tx}^\star)$ that satisfying the conditions: $\mathsf{true} \leftarrow \mathsf{Insp}(\mathsf{Tx}^\star, \breve{P}) \wedge \mathsf{WDec}(r, \mathsf{s}, ct^\star) = m^\star \wedge (ct^\star, \mathsf{Tx}^\star) \notin \{\mathcal{C}, \mathcal{T}\}$.

5 Concrete Instantiation

In this section, we present an instantiation of *Fialka* based on Kiltz's PKE protocol [15] and the Oasis Devnet [1, 8]. Kiltz's PKE is an efficient and IND-CCA secure scheme with a tight security reduction, while Oasis Devnet is an SGX-backed PPSC platform with a rigorous security proof under the Universal Composability (UC) framework [7]. In this instance, PPSC-KM manages a secrete random number as the investigator's decryption key and its access permission through the SGX enclave, and the PPSC-AD audits the transactions, and then reports the investigator's malicious decryption. Specifically, a decryption key using for investigation is loaded in PPSC-KM and hidden in an enclave, which forces the outside investigator to fetch it, and further leaves the transaction-based evidence that will be audited by PPSC-AD. Note that SGX-based PPSC

is an example by hiding the secret key inside the hardware, and other approaches can also achieve the same goal, such as cryptographically hiding secret key by ZKP. Our framework is compatible with various aforementioned PPSC technologies [6, 13, 16, 25, 29]. Importantly, our construction can easily be extended to other accountable PKE protocols without significant modifications.

Setup $(pms, \widehat{\mathsf{c}}_{\mathsf{km}}, \widehat{\mathsf{c}}_{\mathsf{ad}}) \leftarrow \mathsf{Setup}(1^{\lambda}, codes)$. The algorithm takes as input a security parameter λ, and returns public parameters including the multiplicative cyclic group $\mathbb{G}$ with prime order p. Then, it chooses two collision resistant hash functions $\mathcal{H}_1 : \{0,1\}^{\star} \rightarrow \mathbb{Z}_p$ and $\mathcal{H}_2 : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{Z}_p$. Next, it takes as input contract binary codes, and calls the algorithm **Deploy** (defined in Section 3.1), and finally returns two contracts $\widehat{\mathsf{c}}_{\mathsf{km}}$ and $\widehat{\mathsf{c}}_{\mathsf{ad}}$.

Key Generation $(pk, sk, tk) \leftarrow \mathsf{KeyGen}(pms)$. The algorithm is run by the sender and receiver. The receiver runs the algorithm to generate her key pair (pk, sk), and the sender runs the algorithm to obtain a secret tag key tk.

$$\begin{aligned} tk, x_1, x_2, y_1, y_2 &\leftarrow \mathbb{Z}_p^*; \\ \text{Choose } (g_1, g_2, z) \in \mathbb{G}, &\text{ satisfying } g_1^{x_1} = g_2^{x_2} = z; \\ u_1 \leftarrow g_1^{y_1}; u_2 \leftarrow g_2^{y_2};\ pk \leftarrow (\mathbb{G}, p, g_1, g_2, z, u_1, u_2);\ &sk \leftarrow (x_1, x_2, y_1, y_2). \end{aligned}$$

Registration $\mathsf{s} \overset{\mathbb{B}}{\leftarrow} \mathsf{Register}(tk, \breve{P})$: The algorithm is run by the sender. It takes as input tk and policies $\breve{P}$, and outputs contract initial state s. In particular, the tag key tk is registered into $\widehat{\mathsf{c}}_{\mathsf{km}}$. The policies $\breve{P}$ are added to $\widehat{\mathsf{c}}_{\mathsf{ad}}$ by the means of external message calls (see Section 3.1). The privacy of tk and s are protected by the SGX enclave. More details can be found in our implementation.

Encryption $ct \leftarrow \mathsf{Encrypt}(tk, pk, m)$. This algorithm is run by the sender. It takes as input tk, pk, and a message m, returns a ciphertext ct.

$$\begin{aligned} pk = (\mathbb{G}, p, g_1, g_2, z, u_1, u_2);\ r_1, r &\leftarrow \mathbb{Z}_p; \\ r_2 \leftarrow \mathsf{H}_1(tk|r); C_1 \leftarrow g_1^{r1}; C_2 \leftarrow g_2^{r2};\ \tau \leftarrow \mathsf{H}_2(C_1, C_2);\ V &\leftarrow r_1; \\ D_1 \leftarrow z^{\tau r_1} u_1^{r_1}; D_2 \leftarrow z^{\tau r_2} u_2^{r_2}; K \leftarrow z^{r_1 + r_2};\ E &\leftarrow mK; \\ ct &\leftarrow (C_1, C_2, D_1, D_2, E, V). \end{aligned}$$

Decryption $m \leftarrow \mathsf{Decrypt}(sk, ct)$. This algorithm is run by the receiver. It takes as input the receiver's secret key sk, the ciphertext ct, and returns $m \in \mathcal{M}$.

$$\begin{aligned} Parse\ ct\ as (C_1, C_2, D_1, D_2, E, V); \\ s_1, s_2 \leftarrow \mathbb{Z}_p;\ \tau \leftarrow \mathsf{H}_2(C_1, C_2); \\ K' \leftarrow \frac{C_1^{x_1 + s_1(\tau x_1 + y_1)} C_2^{x_2 + s_2(\tau x_2 + y_2)}}{D_1^{s_1} D_2^{s_2}};\ m \leftarrow E(K')^{-1}. \end{aligned}$$

Warrant Decryption $(m, \mathsf{Tx}) \overset{\mathbb{B}}{\leftarrow} \mathsf{WDecrypt}(r, r_1, \mathsf{s}, ct)$. This algorithm is run by the investigator. It takes as input r, r_1 and the private state s (including tk), and then calls the **Transfer** algorithm to execute the function $r_2 \leftarrow \mathsf{H}_1(tk|r)$

in an isolated environment provided by the SGX. This calling progress is represented in the form of a transaction Tx.

$$\begin{aligned} Parse\ ct\ as(C_1, C_2, D_1, D_2, E, V); \\ r_2, \mathsf{Tx} \leftarrow \mathbf{Transfer}(\mathsf{s}, r); \\ K'' = z^{r_1+r_2};\ m \leftarrow E(K'')^{-1}. \end{aligned}$$

Inspection $\mathsf{true/false} \xleftarrow{\mathbb{B}} \mathsf{Inspect}(\mathsf{Tx}, \breve{P})$. This algorithm is run by $\widehat{\mathsf{c}}_{\mathsf{ad}}$. It takes as input $\breve{P}$ and Tx, and returns inspection result. The true indicates the warrant decryption satisfying the policies, and vice versa.

Here, the **correctness** of our construction is easy to check as we have

$$\begin{aligned} K' =& \frac{C_1^{x_1+s_1(\tau x_1+y_1)} C_2^{x_2+s_2(\tau x_2+y_2)}}{D_1^{s_1} D_2^{s_2}} = C_1^{x_1} C_2^{x_2} \left(\frac{C_1^{\tau x_1+y_1}}{z^{tr_1} u_1^{r_1}}\right)^{s_1} \left(\frac{C_2^{\tau x_2+y_2}}{z^{\tau r_2} u_2^{r_2}}\right)^{s_2} \\ =& C_1^{x_1} C_2^{x_2} \left(\frac{g_1^{r_1(\tau x_1+y_1)}}{g_1^{r_1(x_1\tau+y_1)}}\right)^{s_1} \left(\frac{g_2^{r_2(\tau x_2+y_2)}}{g_2^{r_2(x_2\tau+y_2)}}\right)^{s_2} = g_1^{r_1 x_1} g_2^{r_2 x_2}. \end{aligned}$$

Note that the random numbers s_1 and s_2 are used for implicitly testing if the ciphertext is consistent with tag τ [15]. We see that $K = z^{r_1+r_2} = g_1^{x_1 r_1 + x_1 r_2} = g_1^{x_1 r_1} g_2^{x_2 r_2}$. Then, we observe that $K = K' = K''$. Thus, both the receiver and investigator can obtain the message m by

$$\mathsf{Dec}(sk, ct) = E(K)^{-1} = mK(K)^{-1} = m.$$

6 Security Proof

Theorem 1 (Fairness) *Assume that the SGX-based PPSC is secure, our construction Fialka satisfies the property of fairness.*

Proof Suppose that there exists an adversary $\mathcal{A}$ who wins the fairness game $\eth_{\text{fair}}$ with a non-negligible advantage. Then, we transform an adversary $\mathcal{A}$ against *Fairness* into adversaries against PPSC security. Next, we describe a sequence of games to finish the proof.

Lemma 1 (SGX-based PPSC [8, 20]) *Our SGX-based platform is a secure instantiation of PPSC whose protocols match the ideal functionality in the UC framework. More details can be found in [8].*

Game $\eth_0$. This is an unmodified game. Trivially, the winning probability of this game equals the advantage of $\mathcal{A}$ against fairness game, namely, $adv_{\mathcal{A},\Pi}^{\eth_{\text{fair}}}(\lambda)$.

Game $\eth_1$. In this game, when $\mathcal{A}$ calls $\mathcal{C}$, we disallow $\mathcal{C}$ to call contract $\widehat{\mathsf{c}}_{\mathsf{km}}$.

Game $\eth_2$. In this game, when $\mathcal{A}$ calls $\mathcal{C}$, the transaction-based evidence is not allowed to be given to the $\widehat{\mathsf{c}}_{\mathsf{ad}}$. Instead, the evidence is randomly selected for auditing.

Obviously, the winning probability of the game $\eth_2$, denoted as $adv_{\mathcal{A},\Pi}^{\eth_2}(\lambda)$, is negligible, since the transaction-based evidence is randomly selected. Next, to find out the differences between these games, we define the following events.

◇ $\mathbb{E}[a1]$: forging an evidence. The event $\mathbb{E}[a1]$ implies that the adversary $\mathcal{B}_1$ forges a valid transaction Tx^* without update $\widehat{\mathsf{c}}_{\mathsf{km}}$, denoted as $\neg$**Transfer**.

$$\left.\begin{array}{r} r_2, \mathsf{Tx}^* \xleftarrow{\mathbb{B}} \neg\mathbf{Transfer}(\mathsf{s}, r) \wedge \\ \mathsf{WDec}\,(r, r_1, \mathsf{s}, \mathsf{Enc}(tk, pk, m)) = m \wedge \\ \mathsf{false} \xleftarrow{\mathbb{B}} \mathsf{Insp}(\mathsf{Tx}^*, \breve{P}) \end{array}\right] \Rightarrow \mathbb{E}[a1]\,.$$

◇ $\mathbb{E}[a2]$: forging an inspection result. The event $\mathbb{E}[a2]$ implies that the adversary $\mathcal{B}_2$ forges an inspection result, where the originally "true" in the algorithm $\mathsf{Inspect}$ is modified to be "false".

$$\left.\begin{array}{r} r_2, \mathsf{Tx} \xleftarrow{\mathbb{B}} \mathbf{Transfer}(\mathsf{s}, r) \wedge \\ \mathsf{WDec}\,(r, r_1, \mathsf{s}, \mathsf{Enc}(tk, pk, m)) = m \wedge \\ \mathsf{false} \xleftarrow{\mathbb{B}} \mathsf{Insp}(\mathsf{Tx}, \breve{P}) \end{array}\right] \Rightarrow \mathbb{E}[a2]\,.$$

Game $\eth_0 \approx$ **Game** $\eth_1$. The winning condition for $\eth_0$ is equal to the winning condition for $\eth_1$ if and only if the event $\mathbb{E}[a1]$ does not happen. The probability of $\mathbb{E}[a1]$ happening is identical to the advantage of breaking the promise of transaction-unforgeability. Thus, we have

$$|\Pr[\eth_0] - \Pr[\eth_1]| = \Pr[\mathbb{E}[a1]] = adv^{\eth_{\mathrm{unforg}}}_{\mathcal{B}_1,\Pi}(\lambda).$$

Game $\eth_1 \approx$ **Game** $\eth_2$. The winning condition for $\eth_1$ is equal to the winning condition for $\eth_2$ if and only if the event $\mathbb{E}[a2]$ does not happen. We consider the possibility of $\mathbb{E}[a2]$, and it is identical to the advantage of breaking the promise of state-consistency. Thus, we obtain

$$|\Pr[\eth_1] - \Pr[\eth_2]| = \Pr[\mathbb{E}[a2]] = adv^{\eth_{\mathrm{cons}}}_{\mathcal{B}_2,\Pi}(\lambda).$$

Putting everything together, we conclude that

$$\begin{aligned} adv^{\eth_{\mathrm{fair}}}_{\mathcal{A},\Pi}(\lambda) \leq & \Pr[\mathbb{E}[a1]] + \Pr[\mathbb{E}[a2]] + adv^{\eth_2}_{\mathcal{B},\Pi}(\lambda) \\ \leq & adv^{\eth_{\mathrm{unforg}}}_{\mathcal{B}_1,\Pi}(\lambda) + adv^{\eth_{\mathrm{cons}}}_{\mathcal{B}_2,\Pi}(\lambda) + adv^{\eth_2}_{\mathcal{A},\Pi}(\lambda) \leq negl(\lambda). \end{aligned}$$

Theorem 2 (Completeness) *Assume that SGX-based PPSC is secure and Kiltz's full PKE scheme [15] is secure against chosen-ciphertext attacks, Fialka satisfies completeness.*

Proof The concrete proof can be found at Appendix B.

7 Implementation

In this section, we discuss the implementation[1] of our instantiation based on the SGX-based PPSC platform Oasis Devnet [1, 8] (version 2.0). Our implementation

[1] A demo site and reference source code are accessible at http://www.fialka.top.

(see Figure 1.b) has two components: the *client-side* and the *server-side*. The client-side is run by the sender, receiver and investigator, while the server-side is run by the PPSC platform. The client-side covers four algorithms: Set, Gen, Enc, Dec. They are implemented by 1000+ lines of Javascript codes in total, containing the packages of `client` and `client-connector`. The `client` implements basic operations executed by end-users at local, while `client-connector` builds a bridge between the client-side and the server-side. The server-side consists of two pieces of privacy-preserving smart contracts: PPSC-KM and PPSC-AD. PPSC-KM covers the algorithms CGen, Reg and Trans[2], while PPSC-AD includes the algorithm Insp. Both of them are implemented in Rust. PPSC-KM protects private decryption keys by using the enclave technology from Intel SGX [10], while PPSC-AD determines whether the decryption is legal or not by checking the security policies.

To be specific, after a successful deployment of the contract PPSC-KM and PPSC-AD, the evidence inspection algorithm Insp and the investigator's key generation algorithm (by revoking Trans) as well as their access conditions, will be compiled as the binary codes and replicated to the enclaves [10] in SGX-powered blockchain nodes. Then, an encrypted contract state containing the investigator's key $\mathsf{H}_1(tk|r)$ reaches an agreement across distributed blockchain nodes. After that, to obtain the key from PPSC-KM, two requirements must be fulfilled: (1) a transaction with the input satisfying access conditions should be provided; (2) An encrypted and authenticated channel connected to enclaves should be established (after a successful attestation [1, 8, 10]). Then, an invoked progress will be executed in the form of a transaction, and remains visible and immutable which could be publicly accessed. Each entity is able to see/witness the progress of obtaining the investigator's key, but no entity except the contract caller, knows the exact output (key) of the smart contract. Subsequently, PPSC-AD audits the transactions through an internal query to detect suspicious activities. Essentially, the privileges of the Trans algorithm are protected and managed in a CPU-level by Intel SGX. Only designated investigators should be allowed to access this secret key. We also notice that our implementation only provides one-off auditing, since it can only trace the records when the first time an investigator extracts the secret key. Our implementation provides a prototype to demonstrate the feasibility.

8 Evaluation

We first provide the performance evaluation on average CPU-time, representing the consumed time since the operation starts. The evaluation contains all the algorithms, and the testing environment is set as follows. The client-side runs on a Dell precision 3630 Tower with 16GB of RAM and one 3.7GHz six-core i7-8700K processors running Ubuntu 18.04. The server-side runs on a blockchain node, which is provided by Oasis SDK [1, 8].

[2] Trans (**Transfer** algorithm) calculates the investigator's key and it belongs to WDec.

Table 1. The average CPU-time, Gas cost and Latency of Operations.

Operations	**CPU-time**/ms	**Cost**/gas	**Latency**/ms
Set	1.16	-	-
Gen	50.04	-	-
CGen†	0.0880	5129943	5683
Reg	0.0104	494553	3960
Enc	102.35	-	-
Dec	64.86	-	-
Trans	0.0325	342514	3643
Insp	0.0027	251971	2450

†: CGen means contract generation

CPU-time. The evaluation results illustrate some critical points. The offline operation Enc is the most time-consuming operation since the encryption covers the seven exponentiations. The offline operation Dec takes approximately half the time of that in encryption because it processes four exponentiations. On the contrary, blockchain-related operations CGen, Reg, Trans and Insp take much less than offline operations, since they do not have group mathematics computation. In particular, the operation Insp is the fastest operation, which indicates the efficiency of our accountability protocol. However, CPU-time is close to testing environment, inefficient to convince that our framework is practical. Therefore, we provide further evaluations on *gas cost* and *latency* for real-world scenarios.

Gas cost. The gas cost measures the amount of computational effort that a blockchain takes to execute an algorithm. The gas cost evaluation includes the operations of CGen, Reg, Trans and Insp. The operation CGen costs the most gas among all since the initial configuration of a smart contract has to be loaded. Fortunately, this bottleneck can be ignored, because each contract is created only once and can be reused multiple times. The cost of Reg is relatively high, since the public parameters are needed to store on smart contracts. The cost of Trans and Insp are relatively low due to simple online calculations, which indicates that our accountability protocol is financially feasible[3]. In a real-world setting, different investigators may call the functions in a same PPSC for decryption and auditing simultaneously. To demonstrate the practicability of our system, we simulate a distributed environment by increasing the number of invocations from different investigators. Specifically, we test the gas cost of Trans and of Insp along with a maximum 1000 invocations simultaneously. As shown in Fig.2, the outputs remains relatively stable under the variations, and the average cost of Trans reaches approximately 340k while Insp is about 250k. It matches our intuitive expectation, since the gas cost is theoretically independent with the

[3] Estimates on real value of gas cost are omitted, since the Oasis token has not been officially released at the time of writing.

number of investigators. Based on such results, our accountability framework is practically affordable which could be widely adopted.

Latency. Our latency test covers all the blockchain-related operations including CGen, Reg, Trans and Insp. Among them, CGen is the most time consuming, as the contract codes need to be compiled into the blockchain. The operation Reg also takes a long time because all parameters have to be configured into contracts. In contrast, the operations Trans and Insp are in low latency, due to the fact that they do not have sophisticated on-chain computations. We also provide a simulation by increasing the invocations in a distributed environment. Our simulation includes the most two frequently used functions in PPSC, namely Trans and Insp. As shown in Fig.2, the results turn out that the latency stably increases along with the growing number of invocations. Theoretically, numerous invocations will impose a heavy burden onto the distributed network, which may even cause the network failure or transaction stuck. We set an upper bound of invoking transactions with 1000 users at the peak. The testing results confirm our expectations.

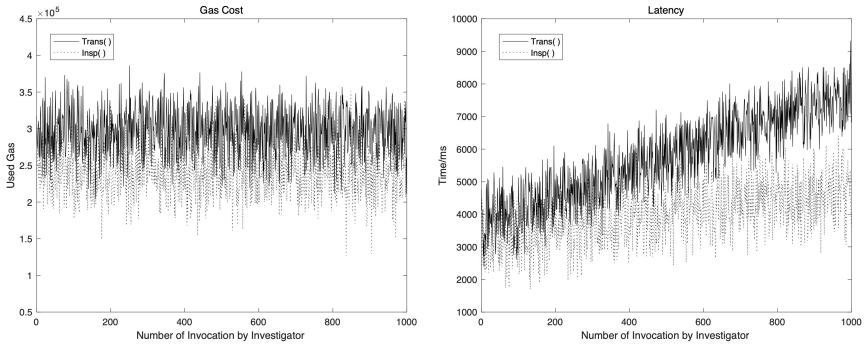

Fig. 2. Gas and latency evaluation

Weakness. The average latency of the operation Trans reaches approximately five seconds, which is the primary drawback of our implementation. Frankly speaking, our system, at least built on the current version of Oasis Devnet (version 2.0), cannot compatibly support the applications that require fast decryption, due to the latency constraints. However, it is worth noting that the execution of warrant focuses on finding criminal evidence, which is latency insensitive.

9 Conclusion

In this paper, we propose *Fialka*, a novel transaction-triggering accountable decryption system based on the privacy-preserving smart contracts. Our system

utilizes PPSC to trace and detect the decryption evidence, which makes warrant execution accountable. To the best of our knowledge, we present the first PPSC-based accountability mechanism with formal definitions and proofs. The security analysis shows that our system holds the accountability properties of fairness and completeness. The implementation based on Oasis Devnet with the detailed evaluation indicates that our system is feasible and applicable.

Future Work. Fialka is a composite framework containing PKE and PPSC. The definition and proof of our instantiation are complete and sound. However, SGX-based PPSC and its underlying blockchain are inherently hybrid systems with a sophisticated mechanism, making our assumption inevitably strong. The possibility to weaken the assumption will be further explored.

Acknowledgments. R. Li, F. Liu and Q. Wang were supported by the National Science Foundation of China under Grant No. 61672015 and Guangdong Provincial Key Laboratory (Grant No. 2020B121201001). D. Galindo was partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 779391 (FutureTPM).

References

1. Oasis labs: A safer way to use data (2020), https://www.oasislabs.com/
2. Origo: the privacy preserving platform for decentralized applications (2020), https://origo.network/
3. Alder, F., Asokan, N., et al.: S-faas: Trustworthy and accountable function-as-a-service using intel sgx. In: CCSW'19. pp. 185–199 (2019)
4. Azaria, A., Ekblaw, A., Vieira, T.: Medrec: Using blockchain for medical data access and permission management. In: OBD'16. pp. 25–30. IEEE (2016)
5. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: CRYPTO'04. pp. 41–55. Springer (2004)
6. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: FC'20 (2020)
7. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS'01. pp. 136–145. IEEE (2001)
8. Cheng, R., et al.: Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: EuroSP'19. pp. 185–200. IEEE (2019)
9. Contractor, D., Patel, D.R.: Accountability in cloud computing by means of chain of trust. IJ Network Security **19**(2), 251–259 (2017)
10. Costan, V., Devadas, S.: Intel sgx explained. IACR Cryptology ePrint Archive **2016**(086), 1–118 (2016)
11. Goyal, V., Lu, S., Sahai, A., Waters, B.: Black-box accountable authority identity-based encryption. In: ACM CCS'08. pp. 427–436. ACM (2008)
12. Guo, H., Zhang, Z., et al.: Generic traceable proxy re-encryption and accountable extension in consensus network. In: ESORICS'19. pp. 234–256. Springer (2019)
13. Juels, A., Kosba, A., Shi, E.: The ring of gyges: Investigating the future of criminal smart contracts. In: ACM CCS'16. pp. 283–295. ACM (2016)
14. Kaaniche, N., Laurent, M.: A blockchain-based data usage auditing architecture with enhanced privacy and availability. In: NCA' 17. pp. 1–5. IEEE (2017)
15. Kiltz, E.: Chosen-ciphertext security from tag-based encryption. In: TCC'06. pp. 581–600. Springer (2006)

16. Kosba, A., Miller, A., et al.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: IEEE S&P'16. pp. 839–858. IEEE (2016)
17. Kroll, J.A., Zimmerman, J., Wu, D.J., Nikolaenko, V., Felten, E.W.: Accountable cryptographic access control. In: Workshop, CRYPTO'18. vol. 2018 (2018)
18. Küsters, R., Truderung, T., Vogt, A.: Accountability: definition and relationship to verifiability. In: ACM CCS'10. pp. 526–535. ACM (2010)
19. Lai, J., Tang, Q.: Making any attribute-based encryption accountable, efficiently. In: ESORICS'18. pp. 527–547. Springer (2018)
20. Li, R., Galindo, D., Wang, Q.: Auditable credential anonymity revocation based on privacy-preserving smart contracts. In: ESORICS'19 CBT, pp. 355–371. Springer (2019)
21. Luo, Y., Fan, J., Deng, C., Li, Y., Zheng, Y., Ding, J.: Accountable data sharing scheme based on blockchain and sgx. In: CyberC' 19. pp. 9–16. IEEE (2019)
22. Microsoft: The coco framework: Technical overview (May 2019), https://github.com/Azure/coco-framework/
23. Neisse, R., Steri, G., Nai-Fovino, I.: A blockchain-based approach for data accountability and provenance tracking. In: ARES'17. p. 14. ACM (2017)
24. Ryan, M.: Making decryption accountable. In: SPW. pp. 93–98. Springer (2017)
25. Steffen, S., et al.: zkay: Specifying and enforcing data privacy in smart contracts. In: ACM CCS'19. pp. 1759–1776. ACM (2019)
26. Szabo, N.: Smart contracts: building blocks for digital markets. EXTROPY: The Journal of Transhumanist Thought,(16) **18**, 2 (1996)
27. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**(2014), 1–32 (2014)
28. Xu, Y., et al.: Blockchain empowered arbitrable data auditing scheme for network storage as a service. IEEE TSC **13**(2), 289–300 (2019)
29. Zyskind, G., Nathan, O., Pentland, A.: Enigma: Decentralized computation platform with guaranteed privacy. arXiv preprint arXiv:1506.03471 (2015)

A Appendix: Linear Problem

Definition 8 (Linear Problem [5, 15]) *Let $\mathbb{G}$ be a cyclic multiplicative group with prime order p, and g_1, g_2, g_3 be generators of $\mathbb{G}$. Given $g_1, g_2, g_3, g_1^a, g_2^b, g_3^c \in \mathbb{G}$, decide whether $a+b$ equals to c. If $a+b=c$, outputs true, or false otherwise. The advantage of an algorithm $\mathcal{A}$ in deciding the linear problem in $\mathbb{G}$ is*

$$adv_{\mathcal{A}}^{LP} = \left| \begin{array}{l} Pr[\mathcal{A}(g_1, g_2, g_3, g_1^a, g_2^b, g_3^{a+b}) = true: \\ \qquad\qquad g_1, g_2, g_3 \leftarrow \mathbb{G}, a, b \leftarrow \mathbb{Z}_p] \\ -Pr[\mathcal{A}(g_1, g_2, g_3, g_1^a, g_2^b, \eta) = true: \\ \qquad\qquad g_1, g_2, g_3, \eta \leftarrow \mathbb{G}, a, b \leftarrow \mathbb{Z}_p] \end{array} \right|,$$

with the probability taken over the uniform random choice of the parameters to $\mathcal{A}$ and over the coin tosses of $\mathcal{A}$.

Assumption 1 (Decision Linear Assumption) *No adversary $\mathcal{A}$ succeeds in deciding the Linear Problem in $\mathbb{G}$ with a non-negligible advantage.*

Lemma 2 *Assume H_2 is a target collision-resistant hash function, under the Decision Linear Problem, Kiltz's full PKE scheme [15] is secure against chosen-ciphertext attacks.*

B Appendix: Completeness

Proof (Theorem 2: Completeness) Suppose that there exists an adversary $\mathcal{A}$ who wins the completeness game $\eth_{\text{comp}}$ with non-negligible probability. Then, we transform an adversary $\mathcal{A}$ against *Completeness* into adversaries against PPSC security and IND-CCA security of Kiltz's PKE scheme. We describe a sequence of games to conduct the proof.

Game $\eth_0$. This is the unmodified completeness game. The winning probability equals the advantage of $\mathcal{A}$ against *Completeness* game, namely, $adv^{\eth_{\text{comp}}}_{\mathcal{A},\Pi}(\lambda)$.

Game $\eth_1$. In this game, when the adversary calls the $\mathcal{C}$, we disallow contract $\widehat{\mathsf{c}}_{\mathsf{ad}}$ to execute the algorithm Insp, and then $\widehat{\mathsf{c}}_{\mathsf{ad}}$ outputs true to the adversary.

Game $\eth_2$. In this game, we disallow $\mathcal{A}$ calls $\mathcal{C}$, and thus **Transfer** in $\widehat{\mathsf{c}}_{\mathsf{km}}$ cannot be executed, indicating $\mathcal{A}$ cannot obtain secret key from blockchain.

Clearly, without querying smart contract, the adversary's advantage of winning $\eth_2$ equals the advantage of breaking the CCA security of PKE. The adversary against security of Kiltz's PKE scheme $adv^{\eth_{\text{CCA}}}_{\mathcal{B},\Pi}(\lambda)$ is negligible, and the proof is given in Lemma 2. To find out the difference between these games, we define the events: (1) $\mathbb{E}[b1]$: blocking the transaction-based evidence. The adversary $\mathcal{B}_1$ fetches the key from the blockchain, and successfully hides the transaction $\mathsf{Tx}^{\star}$ that used for validation in the algorithm Insp. (2) $\mathbb{E}[b2]$: forging an inspection result. The adversary $\mathcal{B}_2$ forges an inspection result by executing $\neg\mathsf{Insp}$, where $\neg\mathsf{Insp}$ means the malicious behaviors of inspection and it modifies the false result as true. (3)$\mathbb{E}[b3]$: breaking the security of PPSC. The adversary $\mathcal{B}_3$ obtains a valid private key without invoking the blockchain.

Game $\eth_0 \approx$ **Game** $\eth_1$. The winning conditions for $\eth_0$ equals the winning conditions for $\eth_1$ if neither event $\mathbb{E}[b1]$ nor event $\mathbb{E}[b2]$ happen. Thus, we have $|\Pr[\eth_0] - \Pr[\eth_1]| = \Pr[\mathbb{E}[b1]] + \Pr[\mathbb{E}[b2]]$. We then consider the happening probabilities of the $\mathbb{E}[b1]$ and $\mathbb{E}[b2]$. The happening of $\mathbb{E}[b1]$ implies that the adversary $\mathcal{B}_1$ hides the transaction evidence, which contradicts the assumption of the transparency properties. Thus, the wining advantages of $\mathbb{E}[b1]$ is identical to breaking the promise of transaction-transparency. If the event $\mathbb{E}[b2]$ happens, indicating that the adversary $\mathcal{B}_2$ breaks the state-consistency of PPSC, the possibility is identical to the advantage of breaking the promise of state-consistency. Thus, we have $\Pr[\mathbb{E}[b1]] = adv^{\eth_{\text{tran}}}_{\mathcal{B}_1,\Pi}(\lambda)$ and $\Pr[\mathbb{E}[b2]] = adv^{\eth_{\text{cons}}}_{\mathcal{B}_2,\Pi}(\lambda)$.

Game $\eth_1 \approx$ **Game** $\eth_2$. The winning condition for $\eth_1$ is equal to the winning condition for $\eth_2$ if and only if event $\mathbb{E}[b3]$ does not happen. The possibility of $\mathbb{E}[b3]$ is identical to the advantages of breaking the promise of state-privacy. Thus, $|\Pr[\eth_1] - \Pr[\eth_2]| = \Pr[\mathbb{E}[b3]] = adv^{\eth_{\text{privacy}}}_{\mathcal{B}_3,\Pi}(\lambda)$.

Combining everything together, we obtain that

$$\begin{aligned} adv^{\eth_{\text{comp}}}_{\mathcal{A},\Pi}(\lambda) \leq & \Pr[\mathbb{E}[b1]] + \Pr[\mathbb{E}[b2]] + \Pr[\mathbb{E}[b3]] + adv^{\eth_{\text{no-query}}}_{\mathcal{B},\Pi}(\lambda) \\ \leq & adv^{\eth_{\text{tran}}}_{\mathcal{B}_1,\Pi}(\lambda) + adv^{\eth_{\text{cons}}}_{\mathcal{B}_2,\Pi}(\lambda) + adv^{\eth_{\text{privacy}}}_{\mathcal{B}_3,\Pi}(\lambda) + adv^{\eth_{\text{CCA}}}_{\mathcal{B},\Pi}(\lambda) \leq negl(\lambda). \end{aligned}$$