Blockchain-based Transparency Framework for Privacy Preserving Third-party Services

Runhua Xu, *Member, IEEE,* Chao Li, *Member, IEEE,* and James Joshi, *Senior Member, IEEE*

Abstract—Increasingly, information systems rely on computational, storage, and network resources deployed in third-party facilities such as cloud centers and edge nodes. Such an approach further exacerbates cybersecurity concerns constantly raised by numerous incidents of security and privacy attacks resulting in data leakage and identity theft, among others. These have, in turn, forced the creation of stricter security and privacy-related regulations and have eroded the trust in cyberspace. In particular, security-related services and infrastructures, such as Certificate Authorities (CAs) that provide digital certificate services and Third-Party Authorities (TPAs) that provide cryptographic key services, are critical components for establishing trust in crypto-based privacy-preserving applications and services. To address such trust issues, various *transparency* frameworks and approaches have been recently proposed in the literature. This paper proposes `TAB` framework that provides transparency and trustworthiness of third-party authority and third-party facilities using blockchain techniques for emerging crypto-based privacy-preserving applications. `TAB` employs the Ethereum blockchain as the underlying public ledger and also includes a novel smart contract to automate accountability with an incentive mechanism that motivates users to participate in auditing, and punishes unintentional or malicious behaviors. We implement `TAB` and show through experimental evaluation in the Ethereum official test network, Rinkeby, that the framework is efficient. We also formally show the security guarantee provided by `TAB`, and analyze the privacy guarantee and trustworthiness it provides.

Index Terms—Transparency, Trustworthiness, Third-party Authority, Blockchain, Ethereum, Smart Contract, Functional Encryption

✦

1 INTRODUCTION

INCREASINGLY, information systems are being built on the third-party facilities or use external services. This is beneficial to many enterprises as it lowers costs and allows them to keep their focus on business missions. On the other hand, increasing cybersecurity incidents such as cybersecurity attacks including those leading to data leakage and identity theft are amplifying users' concerns with regards to their sensitive personal data that is collected, stored, and processed on the third-party facilities. Furthermore, regulations such as General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA) introduce stricter compliance requirements for enterprise information systems. Fig. 1 illustrates the architecture of a typical privacy-preserving third-party service enabled system, as illustrated in a variety of existing work [1], [2], [3], [4], where the personal data is protected by a cryptosystem, and the encrypted data is collected and processed by a third-party IaaS, while the public key and private key services are provided by the third-party authority (TPA). Usually, the third-party entities are assumed to be *honest-but-curious*, and a TPA is typically *fully trusted*.

To address the trust and compliance issues on these service providers, especially, the security-related service providers such as certificate authorities (CAs), key directories (KDs) and TPAs that provide services of certification,
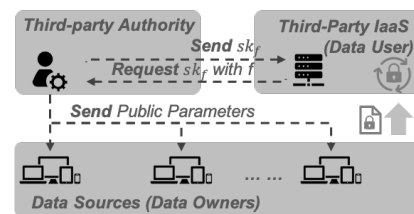

Fig. 1. Illustration of privacy-preserving third-party service enabled system.

public key lookup and private key generation in existing cryptographic key infrastructures, various *transparency* approaches that provision *openness* and *accountability* have been recently proposed [5], [6], [7], [8], [9], [10] to increase users' trust or confidence in such cryptographic key infrastructures. For instance, the CAs, as the underlying public key infrastructure for SSL/TLS protocol, are responsible for issuing digital certificates that certify the ownership of a public key by the named principal of the certificate and allows others to rely upon signatures made by the private key corresponding to the certified public key.

Recent research have demonstrated that a variety of attacks [11], [12] and mis-issuance problems [13], [14] might cause complications during certificate issuance procedures. For instance, Kumar et al. [13] analyze CAs' certificate mis-issuance incidents using a certificate linter named ZLint and Scheitle et al. [14] focus on issues with certification authority authorization (CAA) DNS records. To further mitigate the threat of attack and mis-issuance, notions of *certificate transparency* [5], [6], *CertChain* [15], *verifiable key directory* [7], [9], [10], *transparency overlay* [8] and *authority transparency* [16] have been proposed. To be more precise,

• *Runhua Xu is with the IBM Research, San Jose, CA, United States, 95120. Chao Li is with Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing, China, 100044. Runhua Xu and Chao Li are corresponding authors. James Joshi is with School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, United States, 15260.*
E-mail:runhua@ibm.com, li.chao@bjtu.edu.cn, jjoshi@pitt.edu

certificate transparency frameworks such as those presented in [5], [6] are intended to increase the transparency of users' certificates, whereas the CONIKS [7], [9] and SEEMless [10] are intended to be used with general key directories in end-to-end encryption systems. Unlike the above-mentioned conventional transparency frameworks, which focus on the static binding of a public key and an identity, emerging authority transparency [16] focuses on the dynamic key generation interactions in TPA that is requisite by modern cryptosystems such as attribute-based encryption (ABE) [17], [18], functional encryption (FE) [19], [20], and multi-key homomorphic encryption (HE) [21], [22].

However, the initial and formal design of authority transparency has considerable limitations that hinder its deployment and application in several areas, such as emerging third-party service enabled privacy-preserving applications [1], [2], [3], [4], [23]. Specifically, these limitations include: (i) the definitions and protocols designed in authority transparency model only work on, and relies on, the ABE cryptosystems; and (ii) the implementation of authority transparency framework is based on a secure logging system. In short, existing authority transparency proposal does not directly support other emerging cryptosystems such as the FE and multi-key HE families that have been used to build secure computation protocols [2], [3], [4], [24].

Besides the *identity-to-public-key-binding stealthy targeted attack* and the *private-key-service censorship attack* as illustrated in [16], FE or multi-key HE based applications have additional privacy threats; for instance, there is a potential inference attack by manipulating a malicious functionality-related vector, as illustrated in [2], [3]. Furthermore, the deployment of secure logging system based authority transparency solution pose a challenge with regards to being broadly accepted by the Internet community because: (i) it requires several commercial companies or non-profit organizations that have the computation and storage capabilities to deploy a publicly auditable secure logging system such as that used in the certificate transparency community (e.g., secure logging systems deployed by Google and Mozilla); (ii) there is also a lack of a concrete mechanism for the entities to participate in a transparency framework and monitor and audit *unintentional* or *malicious* behaviors.

To address the aforementioned limitations, in this paper, we propose an approach that provides transparency and trustworthiness of third-party authority and IaaS using blockchain techniques - in short, `TAB` - for emerging third-party service-enabled crypto-based privacy-preserving applications. For simplicity, we use FE-based privacy-preserving systems proposed in [2], [3], [4] as the underlying application to illustrate the `TAB` approach. In particular, to achieve the transparency and trustworthiness goal, `TAB` employs the Ethereum blockchain as the underlying public ledger infrastructure, and also includes a novel and well designed Ethereum smart contract to support automatic accountability with an additional incentive mechanism to motivate participants to participate in the auditing process and punish unintentional misbehaviors or malicious behaviors. We summarize our key **contributions** as follows:

We first revisit the notion of *authority transparency* model and propose our formal `TAB` model with new definitions

and protocols to address the entity trust issues considering the scenarios of generic crypto-based privacy-preserving applications where the cryptographic infrastructure TPA or centralized key server is commonly assumed to be *fully trusted* and the application entities (e.g., third-party IaaS and data sources as illustrated in Fig. 1) is usually assumed to be *honest-but-curious*.

Next, we design a novel smart contract to achieve automatic accountability based on our design of the `TAB` model and employ the Ethereum blockchain as the underlying public ledger infrastructure.

We also design an incentive mechanism in the smart contract to (i) reward a TPA if it fulfills its obligation; (ii) punish any entity that violates its responsibility, and (iii) encourage other entities to help audit and inspect the potential malicious behaviors caused by the *assumed fully trusted* TPA and *assumed honest* participants.

We finally analyze the security guarantee of `TAB` and present the experimental evaluation on the smart contract implemented in the Ethereum official test network - Rinkeby. The evaluation result shows that `TAB` is efficient and provides security and privacy guarantees.

2 BACKGROUND AND PRELIMINARIES

Here, we briefly present preliminaries and background of related concepts such as functional encryption and its related applications, authority transparency, blockchain, Ethereum, and smart contract.

2.1 TPA-based Cryptosystems and Applications

Emerging modern cryptographic schemes, especially those that rely on a *third-party authority (TPA)* to provide key services, are being adopted in privacy-preserving applications, where data is encrypted and the data management operations such as querying, accessing control, and computation are over the encrypted data. A TPA is a critical component in these cryptosystems, and it is generally assumed to be fully trusted. Such an assumption is very common in cryptography research community. However, deploying such a trusted TPA component in a real scenario is still a challenge because there is a lack of (i) incentive mechanisms to encourage a participant (i.e., a third-party entity) to play the role of the *authority* and (ii) a transparent mechanism to ensure that such a TPA works as expected when considering the attacks such as *identity-to-public-key-binding stealthy targeted* attack and *private-key-service censorship attack* as illustrated in [16].

Beyond *authority transparency* that addresses the trust issues in a TPA caused by the aforementioned attacks, we address incentive issues related to participants' engagement in a transparency framework via Ethereum blockchain techniques, and tackle additional privacy leakage issue caused by assuming that participants are *hones-but-curious* in crypto-based privacy-preserving applications [2], [3]. Our proposed blockchain-based `TAB` framework can support various TPA-based cryptosystems such as ABE-enabled applications as illustrated in [16] and the emerging FE-enabled privacy-preserving applications. Here, we briefly introduce the FE cryptosystem and FE-based applications; we also differentiate between the ABE and FE structures/components.

2.1.1 Functional Encryption (FE)

FE is a generalization of public-key encryption in which any party with an issued functional secret key allows us to compute a function of what a ciphertext is encrypting. A FE scheme for functionality $\mathcal{F}$ is a tuple $\mathcal{E}_{FE}$ = *(Setup, KeyDerive, Encrypt, Decrypt)* of four algorithms [19], [20], where the *Setup* and *KeyDerive* algorithms are run by a TPA that is assumed to be *fully trusted*. A data owner can adopt the *Encrypt* algorithm to protect its data, while a data user with the functional decryption key issued by its TPA can compute the function over the ciphertext to acquire the function result without learning the original data via *Decrypt* algorithm.

2.1.2 FE-based Application and Potential Privacy Leakage

The feature of computing over encrypted data makes functional encryption a promising approach for employing secure multi-party protocols for privacy-preserving machine learning (PPML) [2], [3]. While employing FE, a PPML also inherits the assumption of a *trusted* TPA. Besides, PPML techniques typically assume that the aggregator or coordinator (that is, the decryption party when PPML uses a FE scheme) is *honest-but-curious*.

Security guarantee provided by a FE scheme can ensure that the encrypted data cannot be compromised by an adversary [20]. However, there is still potential privacy leakage in PPML approaches that use FE schemes, as demonstrated in [2], [3]; here, an authorized *honest-but-curious* decryption party may exploit a manipulated vector to request a functional decryption key to repeatedly execute the decryption algorithm over the encrypted data and store the intermediate data to infer partial information in the encrypted data. For the specific inference attack, we refer the readers to [2], [3] for more details.

2.1.3 Comparing ABE and FE

ABE is also a type of public-key encryption in which ciphertexts are dependent upon access policy over a set of attribute credentials (e.g., age, affiliation, etc.) and any party with proper attribute credentials can be issued a secret key to access the encrypted data. A *(ciphertext-policy) attribute-based encryption (ABE)* scheme for access policy $\mathcal{A}$ is a tuple $\mathcal{E}_{ABE}$ = *(Setup, KeyGeneration, Encrypt, Decrypt)* of four algorithms [25], [26]. As in ABE, the *Setup* and *KeyGeneration* algorithms are run by a TPA that is assumed to be *fully trusted*. A data owner uses the *Encrypt* algorithm with a specified access policy to protect her data, while the data user with proper attribute credentials that satisfy the access policy can access (Decrypt) the encrypted data.

The main difference between ABE and FE is the credentials that are used to generate or derive the private key. In ABE, the private key is generated based on a set of attributes of a data user, while the functional decryption key is derived from a function-related vector in FE for the functionality of inner-product scheme. Besides, the adoption of FE may introduce potential inference threats as illustrated in [3].

Remark. Unlike *authority transparency* [16] that builds on the ABE scheme, for simplicity, in this paper, we use the recently proposed FE-based applications [2], [3], [4] as underlying examples to illustrate the key features of TAB. Specifically, TAB focuses on providing transparency in cases related to above-discussed assumptions, namely, a **trusted** TPA and a **honest-but-curious** participants, to increase users' trust in a system. In Section 3.4, we analyze the applicability of TAB in other TPA-based cryptosystems.

2.2 Authority Transparency

Authority transparency is defined as a publicly auditable set of a TPA's activities. The goal is to ensure that a TPA fulfills its auditing obligations ($\mathcal{O}$) related to public parameter distribution ($\mathcal{O}_{pp}$) and trustworthy key service ($\mathcal{O}_{ks}$), continuously and transparently. We formally define authority transparency as below; here, we adopt the notation from [27].

Definition 2.1 (Authority Transparency [16]). Let $\mathcal{T}, \mathcal{L}$ and $\mathcal{C}$ denote a third-party authority, a log server, and a client, respectively, that use a set of interactive protocols. Let $\mathcal{C}.actor, \mathcal{C}.auditor$ and $\mathcal{C}.monitor$ represent the roles of the actor, auditor, and monitor that execute the application, auditing, and monitoring modules, respectively. We define *authority transparency*, $\mathcal{AT}_{\mathcal{O}}^{\mathcal{T},\mathcal{L},\mathcal{C}}$, as a set of six *interactive* protocols:

$$\mathcal{AT}_{\mathcal{O}}^{\mathcal{T},\mathcal{L},\mathcal{C}} = (\text{Gen}_{\mathcal{O}}, \text{Log}_{\mathcal{O}_{pp}}, \text{Log}_{\mathcal{O}_{ks}}, \text{Check}_{\mathcal{O}}, \text{Inspect}, \text{Gossip}),$$

and each protocol is defined as follows:

$$(S_{\mathcal{O}_{pp}}, S_{\mathcal{O}_{ks}}) \leftarrow \text{Run}(1^\lambda, \text{Gen}_{\mathcal{O}}, \{\mathcal{T}, \mathcal{C}.actor\}, (\varepsilon, \varepsilon)) \tag{1}$$

$$(b_{\mathcal{T}}, \varepsilon) \leftarrow \text{Run}(1^\lambda, \text{Log}_{\mathcal{O}_{pp}}, \{\mathcal{T}, \mathcal{L}\}, (S_{\mathcal{O}_{pp}}, \varepsilon)) \tag{2}$$

$$(b_{\mathcal{T}}, b_{\mathcal{C}}, \varepsilon) \leftarrow \text{Run}(1^\lambda, \text{Log}_{\mathcal{O}_{ks}}, \{\mathcal{T}, \mathcal{C}.actor, \mathcal{L}\}, (\varepsilon, \mathcal{O}_{ks}.S_{\mathcal{C}}, \mathcal{O}_{ks}.S_{\mathcal{T}})) \tag{3}$$

$$(\varepsilon, b_{\mathcal{C}.auditor}) \leftarrow \text{Run}(1^\lambda, \text{Check}_{\mathcal{O}}, \{\mathcal{L}, \mathcal{C}.auditor\}, (\varepsilon, \varepsilon)) \tag{4}$$

$$(b_{\mathcal{L}}, \varepsilon) \leftarrow \text{Run}(1^\lambda, \text{Inspect}, \{\mathcal{L}, \mathcal{C}.monitor\}, (\varepsilon, \varepsilon)) \tag{5}$$

$$(\text{evidence}) \leftarrow \text{Run}(1^\lambda, \text{Gossip}, \{\mathcal{C}.auditor, \mathcal{C}.monitor\}, (\varepsilon, \varepsilon)) \tag{6}$$

The order of parameters in the input tuple and the order of elements in the output are consistent with participating entities. For instance, in protocol $(b_{\mathcal{T}}, \varepsilon) \leftarrow \text{Run}(1^\lambda, \text{Log}_{\mathcal{O}_{pp}}, \{\mathcal{T}, \mathcal{L}\}, (S_{\mathcal{O}_{pp}}, \varepsilon))$, there exists two participants: $\mathcal{T}$ has the input $\mathcal{O}_{pp}$ while $\mathcal{L}$ has no input, as denoted by ε.

We briefly introduce each interactive protocol as follows:

(1) *Gen$_{\mathcal{O}}$* is a protocol between $\mathcal{T}$ and $\mathcal{C}.actor$ that generates the audit obligations to be logged;

(2) *Log$_{\mathcal{O}_{pp}}$* is a protocol between $\mathcal{T}$ and $\mathcal{L}$ that is used to record $\mathcal{O}_{pp}$ in the public log;

(3) *Log$_{\mathcal{O}_{ks}}$* is a protocol involving $\mathcal{T}$, $\mathcal{L}$ and $\mathcal{C}.actor$ that is used to record $\mathcal{O}_{ks}$ in the public log;

(4) *Check$_{\mathcal{O}}$* is a protocol involving $\mathcal{L}$, $\mathcal{C}.actor$ and $\mathcal{C}.auditor$ that is used to check whether or not an audit obligation $\mathcal{O}_{pp}$ or $\mathcal{O}_{ks}$ is in the log;

(5) *Inspect* is a protocol between $\mathcal{L}$ and $\mathcal{C}.monitor$ that is used to allow the monitor to inspect the contents of the log and find suspicious audit obligations $\{\mathcal{O}_i\}$;

(6) *Gossip* is a protocol between $\mathcal{C}.auditor$ and $\mathcal{C}.monitor$ that is used to compare different versions of a log and detect any inconsistencies caused by misbehavior of a participant or on behalf of the log server.

Unlike the authority transparency approach proposed in [16] that is built on the secure logging system, our

proposed TAB relies on the Ethereum blockchain. Thus, the above-mentioned protocols are not directly applicable in our blockchain-based TAB framework. We will present our relevant definitions in Section 3.3.

2.3 Blockchain, Ethereum and Smart Contract

A blockchain is a growing list of records (a.k.a, blocks) that are linked via cryptographic techniques, where each block contains a cryptographic hash of the previous block, a timestamp, and the transaction data. In particular, the blockchain is a public distributed database of records, transactions, or digital events that have been executed and shared among various participants. In our proposed work we, employ a blockchain as the underlying public ledger infrastructure instead of the secure logging system adopted in [16].

Ethereum is an open-source and public blockchain-based distributed computing platform supporting smart contracts [28]. Usually, there are two types of accounts in Ethereum, namely External Owned Accounts (EOAs) controlled by private keys associated with users and Contract Accounts assigned to smart contracts. A smart contract in Ethereum refers to a piece of code, for instance, a Solidity[1] program code that usually consists of multiple functions, few parameters and perhaps some modifiers. To deploy a smart contract, an ordinary user can compile the contract to generate the corresponding bytecodes and application binary interface (ABI), and then send a contract creation transaction to the Ethereum network with the bytecodes and ABI. Upon receiving a transaction, the miners of the Ethereum network will include the bytecodes into the newly coming block being added. Each successfully deployed contract account can be viewed as a small decentralized computation and storage unit that can execute specific functions defined in the contract and also store data allowed by the contract. As a result, the transactions, messages, as well as the inputs of the functions are all recorded by the Ethereum blockchain, and, hence, the outputs of the functions are deterministic because the distributed miners can ensure that. Note that it is not free to either deploy a smart contract or to call a function of existing smart contracts in Ethereum. A user needs to pay Gas[2] that can be exchanged with Ether, the cryptocurrency used in Ethereum.

3 TAB FRAMEWORK

3.1 Overview of TAB

3.1.1 Entities in TAB

Fig. 2 illustrates the architecture of TAB framework. Note that the dashed lines represent the procedures of crypto-based privacy-preserving applications, while the solid lines denote the procedures of the TAB framework. TAB consists of the following entities:

TPA. The TPA is the same role as in the ordinary FE cryptosystem, but in TAB it has additional responsibilities to fulfill, including: (a) submitting the public parameters obligations (in particular, identity-to-public-key bindings), (b) reporting its fulfillment of obligations in the key service

1. https://github.com/ethereum/solidity
2. https://github.com/ethereum/wiki/wiki/Whisper

process, and (c) verifying that the submitted/reported obligations are permanently recorded in the blockchain.

Actors. Actors include all users of an crypto-based privacy-preserving application, namely, the entities (e.g., *data owner*) that employ the encryption algorithm and the entities (e.g., *data user*) that perform secure computation or access control via the decryption algorithm. Besides, the actors may also need to fulfill the obligations of key service because they are involved in interaction with other actors and/or the TPA.

Monitors. Monitors are responsible for inspecting the contents of the recorded auditing obligations to find suspicious obligations. In TAB, the encryption entities or the additional independent entities play the role of the monitors.

Administrator. An administrator is responsible for the deployment, maintenance, and administration of the smart contract. The smart contract mainly includes three modules: (a) the obligation record module that provides various interaction functions for the entities to carry out recording, auditing and inspection requirements related to the obligations, (b) the incentive mechanism that provides the payment and reward functions to the participants, and (c) the *inference prevention module (IPM)*, previously deployed in a TPA as illustrated in [3]. Note that the Ethereum blockchain can ensure the trustworthiness of smart contracts; it can also ensure that the recorded obligations are distributed, open, and tamper-proof. Note that once the smart contract deployed it does not need a centralized administration.

3.1.2 Notations and Use Scenarios

To elaborate our TAB, we first present the notations, entities, and scenarios of applying our TAB framework in a crypto-based privacy-preserving environment. Here we use FE as the underlying crypto scheme to present TAB framework. In Section 3.4, we analyze the applicability of TAB in other TPA-based cryptosystems. Suppose that we have a group of data owners $\{\mathcal{C}_i^{owner}\}_{i \in [n]}$ that will share their private data $\boldsymbol{x} = \{x_i\}_{i \in [n]}$ encrypted by an FE scheme where for simplicity we assume that $\mathcal{C}_i^{owner}$ owns data x_i , a group of data users $\{\mathcal{C}_j^{user}\}_{j \in [m]}$, where each data user has a vector $\boldsymbol{y}_j$ and needs to acquire the inner-product $\langle \boldsymbol{x}, \boldsymbol{y}_j \rangle$ over the ciphertext of $\boldsymbol{x}$, and a TPA $\mathcal{A}$ that provides public and private key services for these data owners and users. Furthermore, let $\{\mathcal{C}_k^{owner}\}_{k \in [l]}$ be the monitors. We use $\mathcal{B}$ to represent the Ethereum blockchain, and let $\mathcal{B}_{\text{SC}}^{\text{TAB}}$ denotes our proposed smart contract deployed in the blockchain.

3.2 Threat Model

Threat Model of Privacy-Enhanced Applications: Existing crypto-based privacy-preserving applications are usually based on some common assumptions: (i) a centralized TPA or key server is assumed to be *fully trusted* and (ii) both decryption and encryption entities are assumed to be *honest-but-curious*. Hence, the threat models in such cases typically focus on an adversary who attempts to compromise the encrypted data and a *curious* entity that launches potential privacy attacks (e.g., infer the private information), while *honestly* following the protocols/algorithms.

Threat Model of TAB: As illustrated by the *identity-to-public-key-binding stealthy targeted attack* and the *private-key-service censorship attack* in [16], a TPA or key server may not
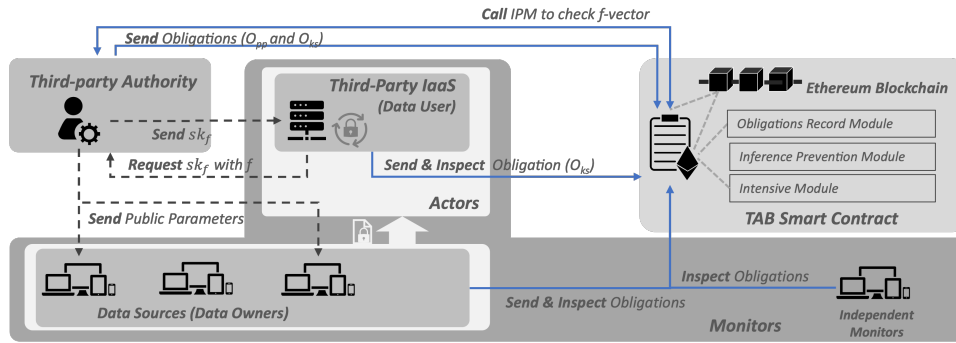
Fig. 2. Overview of the TAB framework. Note that the dashed lines represent the procedures of the crypto-based privacy-preserving applications, while the solid lines denote the procedures of TAB framework.

be trusted because of its *unintentional misbehaviors* and/or *malicious behaviors*. Similarly, the *honest-but-curious* entities may also behave dishonestly. In addition to addressing above threats, TAB focuses on increasing entities' trust on the TPAs and other *honest-but-curious* entities through transparency approach. In particular, TAB mitigates the dependence of crypto-based privacy-preserving applications on the assumptions of a *trusted* TPA or key server and an *honest* entities.

To be more precise, we assume that such a dishonest adversary may pretend to behave honestly without being detected by other entities. Adversaries may not follow the specifications in the protocols, and/or attempt to conceal their activities. In general, the *dishonest* adversary includes the TPA and actors, where a dishonest TPA may attempt to forge a key service *proof-of-work* without actually providing a valid key service; and, a dishonest actor may try to incorrectly blame other entities for misbehavior. Note that misbehavior may be related to non-malicious misuse by normal actors or the behavior of compromised actors controlled by an attacker.

We note that the case of potential collusion between a *dishonest* TPA and *honest-but-curious* actors is not fully considered in this paper. Rather than forbidding or preventing collusion through technical means, such collusion between two stakeholders (i.e., TPA and actors) can be solved by resorting to game theory and incentive mechanism designed in the smart contracts [29]. TAB also involves the incentive mechanism from the aspects of each entity, and hence it can prevent such collusion in a game theory manner. We will not discuss that in the reset of the paper, and readers can refer to [29] for more details.

Furthermore, unlike the secure logging system based *authority transparency* framework in [16], where the *logger* is treated as a potential dishonest adversary, in TAB, the Ethereum smart contract is adopted as the public ledger infrastructure that has been proved to be a trusted computation platform.

3.3 Proposed *TAB* Framework

3.3.1 TAB Model

Unlike the authority transparency approach in [16] that builds on the secure logging system for ABE cryptosystem, TAB uses the Ethereum blockchain, and to keep consistency,

we adopt the similar concepts/notions of the authority transparency but it considers generic crypto-based privacy-preserving scenarios including emerging FE-based applications and a blockchain-based public ledger infrastructure.

Suppose that each entity e in TAB is issued or self-generates an identity-based public and private key pair $\langle \mathrm{pk}_e, \mathrm{sk}_e \rangle$. Note that the key service interaction occurs between entity $\mathcal{C}^{\mathrm{actor}}$ and authority $\mathcal{A}$, where each entity has already received its public and private key pair. For instance, let $\langle \mathrm{pk}_{\mathrm{actor}}, \mathrm{sk}_{\mathrm{actor}} \rangle$ and $\langle \mathrm{pk}_{\mathrm{TPA}}, \mathrm{sk}_{\mathrm{TPA}} \rangle$ represent the public/private key pairs of the actor and the TPA, respectively. Here, we first present the notion of *public parameter audit obligation* and *key service audit obligation*, and then present the formal definition of TAB.

Definition 3.1 (Public Parameter Audit Obligation (PPAO)). A PPAO $\mathcal{O}_{pp}$ of e is a map structure as follows:

$$\mathcal{O}_{pp}^e := \mathrm{H}(e_{id}) : \langle e_{id}, \mathrm{pk}_e, \mathrm{Sig}_{\mathrm{sk}_e}(e_{id}, \mathrm{pk}_e) \rangle,$$

where e_{id} represents the descriptive identifier of e, $\mathrm{H}(\cdot)$ is a hash function, pk_e denotes the public key binding of entity e, and $\mathrm{Sig}_{\mathrm{sk}_e}$ is the signature using sk_e.

Definition 3.2 (Key Service Audit Obligation (KSAO)). A KSAO $\mathcal{O}_{ks}^{\mathcal{C}^{\mathrm{actor}}, \mathcal{A}}$ is a map structure consisting of a pair of key service snapshots

$$\mathcal{O}_{ks}^{\mathcal{C}^{\mathrm{actor}}, \mathcal{A}} := \mathrm{H}(\mathcal{C}_{id}^{\mathrm{actor}}, \mathcal{A}_{id}, r) : \langle \mathcal{S}_{\mathrm{req}}, \mathcal{S}_{\mathrm{resp}} \rangle,$$

where each snapshot is a 4-tuple as follows:

$$\mathcal{S}_{\mathrm{req}} := \mathrm{H}(\mathcal{C}_{id}^{\mathrm{actor}}, \mathcal{A}_{id}, r) : \langle r, f, t_{\mathcal{C}^{\mathrm{actor}}}, \mathrm{Sig}_{\mathrm{sk}_{\mathrm{actor}}}(r, f, t_{\mathcal{C}^{\mathrm{actor}}}) \rangle,$$
$$\mathcal{S}_{\mathrm{resp}} := \mathrm{H}(\mathcal{C}_{id}^{\mathrm{actor}}, \mathcal{A}_{id}, r) : \langle r, \sigma, t_{\mathcal{A}}, \mathrm{Sig}_{\mathrm{sk}_{\mathrm{TPA}}}(r, \sigma, t_{\mathcal{A}}) \rangle,$$

such that

$$\mathcal{S}_{\mathrm{req}}.\mathrm{H}(\mathcal{C}_{id}^{\mathrm{actor}}, \mathcal{A}_{id}, r) = \mathcal{S}_{\mathrm{resp}}.\mathrm{H}(\mathcal{C}_{id}^{\mathrm{actor}}, \mathcal{A}_{id}, r)$$
$$\mathcal{S}_{\mathrm{resp}}.t_{\mathcal{A}} - \mathcal{S}_{\mathrm{req}}.t_{\mathcal{C}^{\mathrm{actor}}} > 0$$
$$\mathcal{S}_{\mathrm{req}}.t_{\mathcal{A}} - \mathcal{S}_{\mathrm{resp}}.t_{\mathcal{C}^{\mathrm{actor}}} < \delta_t$$

where r is a nonce selected by the key service requester, t is the timestamp of key service processed by each entity, f denotes the request content such as function related vector, σ represents the proof-of-work that TPA has issued the key, δ_t is the threshold of timestamp difference indicating the expected time of processing of the key service request by the TPA.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3179698

6

Remark. In particular, $\mathcal{O}_{pp}$ is an *identity-to-public-key* binding with the issuer's signature, while $\mathcal{O}_{ks}^{\mathcal{C}^{\mathrm{actor}},\mathcal{A}}$ is the *proof-of-key-service*. In the $\mathcal{O}_{ks}^{\mathcal{C}^{\mathrm{actor}},\mathcal{A}}$, for simplicity, to provide the *proof-of-work* of issuing the functional decryption key sk_f for the function related materials f, let $\mathrm{Sig}ma$ be $\mathrm{H}(\mathrm{sk}_f)$.

Based on the notion of *public parameter audit obligation* and *key service audit obligation*, we present the formal TAB model as follows:

Definition 3.3 (TAB Model). Let $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$ denote a third-party <u>A</u>uthority, a <u>B</u>lockchain, and an <u>A</u>ctor, respectively, which are parties involved in the interactive protocols. Let $\mathcal{C}.actor$ and $\mathcal{C}.monitor$ represent the roles of the actor and monitor that execute the functional and monitoring modules, respectively. We define TAB model, $\mathcal{M}$, as a set of five interactive protocols:

$$\mathcal{M}_{\mathcal{O}}^{A,B,C} = (\mathrm{Gen}_{\mathcal{O}}, \mathrm{Log}_{\mathcal{O}_{pp}}, \mathrm{Log}_{\mathcal{O}_{ks}}, \mathrm{Inspect}),$$

and each protocol is defined as follows:

$$(S_{\mathcal{O}_{pp}}, S_{\mathcal{O}_{ks}}) \leftarrow \mathrm{Run}(1^\lambda, \mathrm{Gen}_{\mathcal{O}}, \{\mathcal{A}, \mathcal{C}.actor\})$$
$$(b_{\mathcal{A}}, \varepsilon) \leftarrow \mathrm{Run}(1^\lambda, \mathrm{Log}_{\mathcal{O}_{pp}}, \{\mathcal{A}, \mathcal{B}\}, (S_{\mathcal{O}_{pp}}, \varepsilon))$$
$$(b_{\mathcal{A}}, b_{\mathcal{C}}, \varepsilon) \leftarrow \mathrm{Run}(1^\lambda, \mathrm{Log}_{\mathcal{O}_{ks}}, \{\mathcal{A}, \mathcal{C}.actor, \mathcal{B}\}, (\mathcal{O}_{ks}.S_{\mathcal{A}}, \mathcal{O}_{ks}.S_{\mathcal{C}}, \varepsilon))$$
$$(b_{\mathcal{B}}, \varepsilon) \leftarrow \mathrm{Run}(1^\lambda, \mathrm{Inspect}, \{\mathcal{B}, \mathcal{C}.monitor\}, (\varepsilon, \varepsilon))$$

Theorem 3.1 presents similar security guarantee as used in [16]. We present the details in Section 4.1.

Theorem 3.1. If the hash function is collision-resistant and the signature scheme is unforgeable, then TAB model comprises a secure transparency framework.

Remark. Note that the formal definition of our TAB model is inherited from the *authority transparency* model [16] with needed changes considering the underlying Ethereum blockchain infrastructure. Specifically, in the authority transparency model, the *gossip* protocol essentially ensures the consistency of distributed logs without being tampered by an adversary, while the *check* protocol guarantees that the submitted obligations are recorded by the logging system. As TAB adopts the Ethereum blockchain as the underlying public ledger infrastructure, there is no need to run the *gossip* and *check* protocols because these logging-related functions are the features provided by the Ethereum smart contract.

3.3.2 Design of $\mathcal{B}_{\mathrm{SC}}^{\mathrm{TAB}}$

The TAB smart contract is a critical component in our framework. To support the goal of TAB framework, $\mathcal{B}_{\mathrm{SC}}^{\mathrm{TAB}}$ includes various types of modules: *administrative module, access control module, obligation module, inspection module*, and *incentive module*. As illustrated in Fig. 3, the access control module verifies a user's role within an application and determines whether or not to execute corresponding modules or functions. The administrative module keeps track of the TAB smart contract's status and registration procedures. The obligation module is concerned with the collection of various obligation records and the prevention of inference in privacy-preserving applications. The inspection module enables authorized users to examine misbehavior or malicious activity. The incentive module coordinates the above modules in order to reward or punish users using the
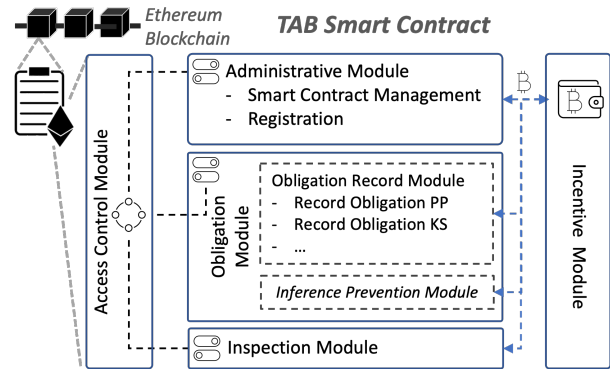

Fig. 3. Overview of the TAB Smart Contract Interface.

Ethereum network's payment functionalities. We discuss each module in details next.

Administrative module. This module allows the administrator role to deploy the smart contract into the Ethereum network. The module also includes functions such as opening and locking the enrollment, and allowing the participants to drop out.

Access control module. This module supports a basic role based access control (RBAC) mechanism that allows the account (a.k.a, the participating entities) have role-related permissions to call various functions. In $\mathcal{B}_{\mathrm{SC}}^{\mathrm{TAB}}$, we define four types of roles: the *TPA*, the *actors of data owner*, the *actors of data user*, the *monitors* and the *administrator* (i.e., the smart contract owner). The administrative entity that deploys the smart contract becomes the *smart contract owner*. The ownership can be transferred to a new account if necessary. Besides, it is also possible to relinquish this administrative privilege, which is a common pattern after an initial stage when there is a need for a decentralized administration. After the deployment, each entity needs to register to its role by calling the corresponding function before it can use the ordinary features of the smart contract.

Obligation module. This module assists in recording the *audit obligation* into the public ledger. It also publishes its *identity-to-public-key* binding to the Ethereum blockchain, as illustrated above. Note that the identity of the entity is the unique public address (i.e., 42 hex string characters without case-sensitivity) of the blockchain account, which is derived from the entity's private key. With regards to the key service audit obligation procedure, the key service requestor (i.e., data owner) can call the corresponding function (that includes role verification) with a randomly generated request identifier, the key-related request parameters, and the corresponding signature. The function then automatically analyzes the request parameters via the *inference prevention module (IPM)*. Note that the IPM, previously deployed inside the centralized trusted TPA in vanilla FE-based applications, is now deployed in the smart contracts in a decentralized trust setting in the TAB. Upon receiving the key service request with the request identifier, the TPA first checks the verification result of IPM. If the request passes the verification, the TPA will issue the functional decryption key and then publish a response snapshot to fulfill the key service obligation.

Inspection module. This module mainly inspects the com-

pleteness of a pair of the key service snapshots to check whether the TPA has fulfilled its key service obligation or not. Besides, it also allows checking for the published *identity-to-public-key* binding. Besides the inspection module that can prevent potential misbehaviors, we have introduced the RBAC mechanism to prevent partial misbehaviors and malicious behaviors as each entity only will be allowed to call corresponding functions with limited privilege.

Incentive module. This module, as part of $\mathcal{B}_{SC}^{TAB}$, includes several functions to enforce the incentive mechanism, as depicted in Fig. 3. The incentive mechanism is based on payment features of the Ethereum network, where the token can be exchanged for real currency. As illustrated in Fig. 3, we design several functions as "publicly payable", which indicates that the smart contract is able to receive the transaction value (e.g., the Ether) when the function is successfully called and executed.

In general, m data users need to pay equally for the cost of calling the registration function for themselves as well as for n data owners and the TPA. Each data user also needs to pay for the cost of calling the *request obligation record* function and that of calling the *response obligation record* function by the TPA. Additionally, there exists a mechanism to punish the misbehaviors and malicious activities by a *dishonest* TPA and data users. To achieve that, the data owners and the TPA first need to register and pay the cost by themselves. The data users make a deposit equally for all the entities' registration cost after the enrollment phase. Then, the data owners and the TPA can call the *disposable* reward function to withdraw the registration cost. Besides, we make the TPA and data users make a *guaranteed deposit* after the registration phase. The monitors can register and pay the cost by themselves, and then calls the inspection function to check the suspicious behaviors. If monitors find the malicious behaviors, they will acquire the reward from a fine to the corresponding entity (i.e., the guaranteed deposit of the entity). Without the guaranteed deposit, the corresponding entity is not allowed to operate in/join the system. We discuss the quantitative analysis of the cost of each entity in $\mathcal{B}_{SC}^{TAB}$ in Section 4.2.

3.3.3 TAB Procedures

As depicted in Fig. 4, we illustrate the four phases of the TAB framework with specific procedures in a typical FE-based privacy-preserving application scenario. Note that the dashed arrows represent the functional procedures of a typical FE-based application, while the solid arrows denote procedures specific to TAB. In our design, each entity in the FE-based application can also play the role of the auditor and monitor, and we also allow additional monitors to help inspect the misbehaviors and malicious behaviors. Below, we present the specific procedures of each phase in TAB.

Phase I: entity initialization: For each entity e with role e_{role} and identifier e_{id} in the framework, it generates a public and private key pair $\langle pk_e, sk_e \rangle$. Then, entity e registers its role e_{role} to $\mathcal{B}_{SC}^{TAB}$, and publishes its *id-to-public-key binding* $\langle e_{id}, pk_e \rangle$ with its signature $Sig_{sk_e}(e_{id}, pk_e)$ to $\mathcal{B}_{SC}^{TAB}$.

Phase II: FE (crypto) initialization. The TPA $\mathcal{A}$ sets up the FE cryptosystem with the master public key and master private key pair $\langle mpk^{FE}, msk^{FE} \rangle$. Using the master key, the TPA generates and sends the common public key pk_{com}^{FE} for all entities (i.e., data owners and data users) in the FE-based application. Then, the TPA publishes the binding $\langle \mathcal{A}_{id}, pk_{com}^{FE} \rangle$ with its signature $Sig_{sk_{\mathcal{A}}}(\mathcal{A}_{id}, pk_{com}^{FE})$ to $\mathcal{B}_{SC}^{TAB}$.

Phase III: secure data publishing. For each data owner $\mathcal{C}_i^{owner}$, it first selects a nonce r as the key service identifier. Then $\mathcal{C}_i^{owner}$ requests the entity-specific public key $pk_{\mathcal{C}_i^{owner}}^{FE}$ from the TPA with r. Meanwhile, $\mathcal{C}_i^{owner}$ also sends a request key service snapshot $\mathcal{S}_{req}^{\mathcal{C}_i^{owner}}$ to $\mathcal{B}_{SC}^{TAB}$ as follows:

$$\mathcal{S}_{req}^{\mathcal{C}_i^{owner}} = \langle r, 0, t_{\mathcal{C}_i^{owner}}, Sig_{sk_{\mathcal{C}_i^{owner}}}(r, 0, t_{\mathcal{C}_i^{owner}}) \rangle.$$

Then, the TPA generates $pk_{\mathcal{C}_i^{owner}}^{FE}$ for $\mathcal{C}_i^{owner}$ using its master keys, and also publishes a corresponding response key service snapshot $\mathcal{S}_{resp}^{\mathcal{A}}$ to $\mathcal{B}_{SC}^{TAB}$ to fulfill its key service audit obligation $\mathcal{O}_{ks}^{\mathcal{C}_i^{owner}, \mathcal{A}}$ with mapping key $H(\mathcal{C}_{i,id}^{owner}, \mathcal{A}_{id}, r)$ as follows:

$$\mathcal{S}_{resp}^{\mathcal{A}} = \langle r, H(pk_{\mathcal{C}_i^{owner}}^{FE}), t_{\mathcal{A}}, Sig_{sk_{\mathcal{A}}}(r, H(pk_{\mathcal{C}_i^{owner}}^{FE}), t_{\mathcal{A}}) \rangle.$$

Each data owner then uses $pk_{e_i^{owner}}^{FE}$ to encrypt its data as follows: $\{Enc_{pk_{e_i^{owner}}^{FE}}(x_i)\}_{i \in [n]}$. Finally, the data owner publishes a receipt for the received $pk_{e_i^{owner}}^{FE}$.

Phase IV: secure data computation. Suppose that a data user $\mathcal{C}_j^{user}$ who has a vector $\boldsymbol{y}_j = (y_1, ..., y_n)_j$ would apply inner-product functionality over the encrypted data $\{Enc(x_1), ..., Enc(x_n)\}$. $\mathcal{C}_j^{user}$ also selects a key service identifier r' first, and then requests the functional decryption key $sk_{\boldsymbol{y}_j}^{FE}$ to the TPA with the vector $\boldsymbol{y}_j$ and r'. At the same time, $\mathcal{C}_j^{user}$ also sends the request key service snapshot $\mathcal{S}_{req}^{\mathcal{C}_j^{user}}$ to $\mathcal{B}_{SC}^{TAB}$ as follows:

$$\mathcal{S}_{req}^{\mathcal{C}_j^{user}} = \langle r', \boldsymbol{y}_j, t_{\mathcal{C}_j^{user}}, Sig_{sk_{\mathcal{C}_j^{user}}}(r', \boldsymbol{y}_j, t_{\mathcal{C}_j^{user}}) \rangle.$$

Unlike the approaches proposed in [2], [3] that deploy the *inference prevention module (IPM)* within a TPA, we propose to deploy IPM in a smart contract as the TPA is not fully trusted in TAB. Thus, the TPA needs to query $\mathcal{B}_{SC}^{TAB}$ to check the validity of $\boldsymbol{y}_i$. If $\boldsymbol{y}_i$ is valid, the TPA generates $sk_{\boldsymbol{y}_j}^{FE}$ for $\mathcal{C}_j^{user}$ using its master keys, and then publishes a corresponding response key service snapshot $\mathcal{S}_{resp}^{\mathcal{A}}$ to $\mathcal{B}_{SC}^{TAB}$ to fulfill its key service audit obligation $\mathcal{O}_{ks}^{\mathcal{C}_j^{user}, \mathcal{A}}$ with mapping key $H(\mathcal{C}_j^{user}, \mathcal{A}, r')$ as follows:

$$\mathcal{S}_{resp}^{\mathcal{A}} = \langle r', H(sk_{\boldsymbol{y}_j}^{FE}), t_{\mathcal{A}}, Sig_{sk_{\mathcal{A}}}(r', H(sk_{\boldsymbol{y}_j}^{FE}), t_{\mathcal{A}}) \rangle.$$

Otherwise, the TPA refuses the key service and also publishes key service snapshot indicating that it has refused the key service, $\mathcal{S}_{resp}^{\mathcal{A}, refuse}$, with refusing symbol $\perp$ to $\mathcal{B}_{SC}^{TAB}$ to fulfill its key service audit obligation as follows:

$$\mathcal{S}_{resp}^{\mathcal{A}, refuse} = \langle r', H(\perp, \boldsymbol{y}_j), t_{\mathcal{A}}, Sig_{sk_{\mathcal{A}}}(r', H(sk_{\boldsymbol{y}_j}^{FE}), t_{\mathcal{A}}) \rangle.$$

With the received $sk_{\boldsymbol{y}}^{FE}$, a data user can compute the inner-product of $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ by decrypting as follows:

$$\langle \boldsymbol{x}, \boldsymbol{y}_j \rangle = Dec_{sk_{\boldsymbol{y}_j}^{FE}}(\{Enc_{pk_{\mathcal{C}_i^{owner}}^{FE}}(x_i)\}_{i \in [n]}).$$

Finally, the data owner publishes a receipt for the received $sk_{\boldsymbol{y}}^{FE}$
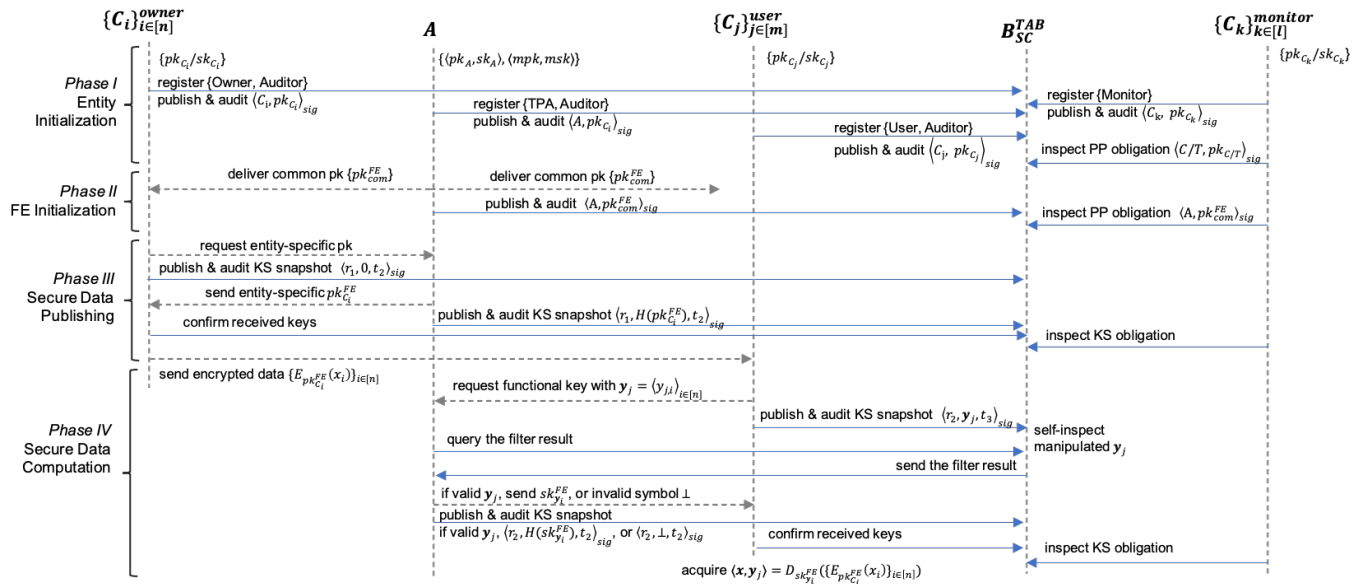
Fig. 4. Illustration of the four phases with specific procedures in TAB in a FE-based privacy-preserving application scenario.

Remark. To avoid redundant description, we do not present the roles of *auditor* and *monitor* in the above-mentioned procedures. In particular, as illustrated in Fig. 4, the *data users*, *data owners* and the *TPA* also play the role of auditor that checks whether the audit obligations are recorded into the blochchain permanently. In our design, the *data owners* also play the role of a monitor to check the suspicious obligations caused by misbehaviors and malicious behaviors from the TPA and adversarial *data users*. For instance, as illustrated in [2], [3], an adversarial data user may infer the private vector $\boldsymbol{x}$ by manipulating a vector to request the functional decryption key. The monitor can inspect $\mathcal{O}_{ks}^{e^{\text{user}}, e^{\text{TPA}}}$ to find the adversary's suspicious behaviors. Furthermore, our design can also address the case of intentionally issuing an incorrect functional decryption key. For example, suppose the key request material (i.e., $\boldsymbol{x}$) from the authorized user is correct after IPM checking, while the key is incorrect. In TAB, the data user is also the monitor/auditor and can file a claim by manually calling the inspect function based on existing logged key service materials, where the inspection is based on the rule defined in the formal model presented above.

3.4 Applicability of *TAB*

TAB is applicable to other popular TPA-based cryptosystems as well. Specifically, we analyze the applicability of TAB in the attribute-based encryption (ABE) scheme that is the focus of *authority transparency* proposed in [16].

Differences in key service audit obligations in FE and ABE schemes are as shown in TABLE 1. The main difference in the key service is the credential type, namely, the function-related vector and the attributes that are usually represented in a character string. These credentials are used to generate or derive the private key in the *Setup* and *KeyGeneration* phases. As presented in Section 3.3, TAB is a general framework and is not restricted to the type of audit obligation that builds on different key service credentials. Specifically, the request content f and response proof-of-

TABLE 1
Different key service audit obligations in FE and ABE

Types	obligations in FE	obligations in ABE
Setup-Output	private key	public parameter
KeyGen-Input	function-related vector	attribute set
KeyGen-Output	functional decryption key	attribute private key

work σ in Definition 3.2 are not limited to a function-related vector and corresponding hashed generated key as illustrated in Section 3.3.3. TAB is applicable to ABE-based applications by replacing the following audit obligations:

$$\mathcal{S}_{\text{req}}^{\mathcal{C}_j^{\text{user}}} = \langle r', \boldsymbol{y}_j, t_{\mathcal{C}_j^{\text{user}}}, \text{Sig}_{\text{sk}_{\mathcal{C}_j^{\text{user}}}}(r', \boldsymbol{y}_j, t_{\mathcal{C}_j^{\text{user}}}) \rangle,$$

$$\mathcal{S}_{\text{resp}}^{\mathcal{A}} = \langle r', \text{H}(\text{sk}_{\boldsymbol{y}_j}^{\text{FE}}), t_{\mathcal{A}}, \text{Sig}_{\text{sk}_{\mathcal{A}}}(r', \text{H}(\text{sk}_{\boldsymbol{y}_j}^{\text{FE}}), t_{\mathcal{A}}) \rangle,$$

by the corresponding audit obligations:

$$\mathcal{S}_{\text{req}}^{\mathcal{C}_j^{\text{user}}} = \langle r', \boldsymbol{S}_j, t_{\mathcal{C}_j^{\text{user}}}, \text{Sig}_{\text{sk}_{\mathcal{C}_j^{\text{user}}}}(r', \boldsymbol{S}_j, t_{\mathcal{C}_j^{\text{user}}}) \rangle,$$

$$\mathcal{S}_{\text{resp}}^{\mathcal{A}} = \langle r', \text{H}(\text{sk}_{\boldsymbol{S}_j}^{\text{ABE}}), t_{\mathcal{A}}, \text{Sig}_{\text{sk}_{\mathcal{A}}}(r', \text{H}(\text{sk}_{\boldsymbol{S}_j}^{\text{FE}}), t_{\mathcal{A}}) \rangle,$$

where $\boldsymbol{S}_j$ is the attribute set and $\text{H}(\text{sk}_{\boldsymbol{S}_j}^{\text{ABE}})$ is the corresponding access control private key generated by the TPA using the attribute set $\boldsymbol{S}_j$ in the hash format.

4 EVALUATION

4.1 Security, Privacy and Trustworthiness

4.1.1 Security Guarantee

The security for the transparency framework is defined in terms of three properties [8], [16]: (i) *log-consistency* - a dishonest public ledger cannot remain undetected if it tries to present inconsistent versions of the recorded obligations; (ii) *unforgeable-service* - a dishonest TPA cannot forge a key service by sending valid key service snapshots, but not provide the key service to the actors; (iii) *non-fabrication* - a dishonest TPA or actors cannot blame the public ledger for

misbehavior if it has behaved honestly, and dishonest actors cannot prove the TPA for misbehavior if it has behaved honestly.

We note that *log-consistency* relies on the security properties of the Ethereum blockchain. The *unforgeable-service* and *non-fabrication* properties depend on the designed smart contract functions and the adopted signature scheme. Here, we use the game simulation-based reduction methodology to prove Theorem 3.1.

Proof. TAB is built on three fundamental security components: the Ethereum blockchain as the public ledger infrastructure, the Secure Hash Algorithm 3 (SHA3) as the collision-resistance hash function, the Elliptic Curve Digital Signature Algorithm (ECDSA) to sign and validate the origin and integrity of messages. The security of three components has been proved in corresponding related work [28], [30], [31]. We only prove the above-mentioned three security properties.

Log-consistency. Unlike the existing transparency framework, [8], [16], that relies on the customized public ledger, TAB uses the public blockchain that has already been proved to provide secure consistency feature [28], and hence we do not present it here to avoid redundancy.

Unforgeable-service. In TAB, there are two possible issues related to forgeable-service:

- a dishonest TPA may publish $\mathcal{S}_{\text{resp}}^{\mathcal{A}}$ to the blockchain, but does not send the key sk_f to the actors;
- the dishonest TPA may send an invalid key sk_f' to the actors, but publishes correct $\mathcal{S}_{\text{resp}}^{\mathcal{A}}$ generated from the valid key sk_f.

For the first issue, the confirmation phase of key service audit obligation cannot be accomplished in our designed smart contract, and then such adversarial behavior is easily detected by the monitors. For the second issue, suppose that the dishonest TPA has the non-negligible advantage ϵ to break the *unforgeable-service* security guarantee, and hence it can forge the hashed key component $\text{H}^{\text{SHA3}}(\text{sk}_f')$ for sk_f with advantage $\mathbf{Adv}_{\text{H}^{\text{SHA3}}(\text{sk}_f') \rightarrow \text{sk}_f}^{\mathcal{A}} \geq \epsilon$.

To achieve that, the dishonest TPA hence needs the ability to find potential collision $\text{H}^{\text{SHA3}}(\text{sk}_f) = \text{H}^{\text{SHA3}}(\text{sk}_f')$. According to the security promise of SHA3, it is impossible to find that collision with non-negligible advantage [30]. Thus, dishonest TPA does not have a non-negligible advantage to provide an unforgeable key service without being detected.

Non-fabrication. In TAB, a possible fabrication case is that dishonest actors may attempt to blame the TPA by publishing $\mathcal{S}_{req}^{\mathcal{C}^{\text{actor}}}$ to the blockchain but does not actually send the key request to the TPA. Suppose that a dishonest actor has the non-negligible advantage ϵ to break the *non-fabrication* security promise. To launch the fabrication case, the dishonest actor needs to forge a fake $\mathcal{S}_{\text{resp}}^{\mathcal{A}}$ so that it can accomplish the confirmation phase. Thus, the dishonest actor is able to forge a fake signature of the TPA with advantage $\mathbf{Adv}_{\text{sk}_{\mathcal{A}}}^{\mathcal{C}^{\text{actor}}} \geq \epsilon$. However, it is impossible to break the ECDSA [31], as has been proved, namely, the unforgeability of the signature scheme. Thus, the actors do not have a non-negligible advantage to frame up the TPA. □

4.1.2 Privacy Guarantee

Typically, authorized users request public and private keys from a TPA using (attribute) identities in privacy-preserving applications built on cryptosystems like as ABE, FE, and multi-key HE. The major objective of *authority transparency* [16] and TAB is to audit the interactive key service discussed previously without invading the original key service. As a result, privacy assurance of TAB focuses on preventing data leaking from audit materials.

Unlike the initial *authority transparency* that focuses on ABE-based applications where partial attribute identities are privacy-sensitive, TAB also supports the privacy-enhanced computing applications that built on FE or multi-key HE schemes. There is no privacy concern regarding the identity in an FE or multi-key HE because those identities could be any unique characters without any privacy-sensitive information. For example, the identity of each entity in TAB could be the Ethereum network's public account address, which is a random 64-character hex string produced from the entity's private key. Additionally, the TAB framework only receives the hash of attribute identifiers in ABE-based applications, not potentially sensitive attribute information. As a result, such account identifiers or hash of auditing materials do not reveal any private identifiable information.

4.1.3 Trustworthiness Goal

The purpose of the TAB approach is to deal with the trust issues raised by potential *dishonest* entities by providing transparency. TAB is able to prevent the attacks such as stealthy targeted attack and censorship attack as illustrated in [16]. Specifically, each dishonest entity needs to publish the key service snapshot to prove that it has fulfilled its obligation of public parameter distribution and private key service. The designed smart contract can ensure that each entity's submitted audit obligations can be automatically cross-validated based on our designed protocols before being honestly and permanently recorded into the blockchain. Our security analysis has shown that the misbehaviors or malicious behaviors of a TPA and the actors are easily detected. Furthermore, the IPM is a critical component in FE-based applications [2], [3] that helps to mitigate the inference threats. In TAB, the IPM, which was deployed in a TPA in the scheme proposed in [3], is moved to the smart contract, is automatically executed in a publicly auditable environment, and hence increase the transparency and trustworthiness of IPM.

4.2 Experimental Evaluation

4.2.1 Implementation and Setup

The TAB model does not rely on the specific privacy-enhanced applications that are built on FE or multi-key HE cryptosystems, and hence for generality, we only present the evaluation on a pure TAB model with the simulated audit obligations where the key-related components are generated by the FE-based application in an off-line manner.

Implemented Smart Contract: We implement the smart contracts in *Solidity* programming language using a *Truffle* development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM). Fig. 3 illustrates the core modules and

interfaces of `TAB` smart contract, while the whole implementation is publicly available on Github [3]. We direct the reader to the Github repository for implementation specifics and discuss the fundamental implementation considerations below. `TAB` mainly includes four types of functions as follows:

Access Control Modifiers. The *modifier* can be used to change the behavior of functions in a declarative way. In our implementation, we use the modifier to automatically check the privilege of each account that is defined in RBAC module prior to executing the function. We employ *Ownable* and *AccessControl* smart contracts from *OpenZeppelin*[4] as the basis for our access control mechanism. To be specific, we define various access control modifiers in which the basic RBAC functions are integrated to satisfy our access control requirement. Except for the registration related functions, other functions are restricted by these modifiers.

Administrative and Incentive Functions. We define several administrative functions such as *'enrollLock()'*, *'enrollOpen()'*, *'dropout()'* that allow the administrator to control the enrollment status. In `TAB`, each entity can register if and only if the enrollment is set as open by the administrator. After the enrollment is locked, the deposit operations are opened to the related entities. Besides, `TAB` also inherits the administrative functions such as *'transferOwnership(newOwner)'*, *'renounceOwnership()'*. These two functions allow transferring the ownership of the contract and leave the contract without owner, respectively. Furthermore, we also define several withdraw and deposit functions that help to establish a basis for the incentive and penalty mechanisms.

Registration Functions. The registration functions mainly focus on the initialization phases of `TAB` (i.e., Phases I and II, as illustrated in Section 3.3.3), where each entity is allowed to register a role, and publish its *identity-to-public-key* binding in the blockchain.

Obligation Functions. The obligation functions address the core features of the `TAB` model. As illustrated in Section 3.3.3 Phases III and IV, we use a three-phase commitment approach to achieve the obligation features. To be specific, *'recordKSPKReq'*, and *'recordKSSKReq'* allow the actors to publish the key service request snapshots, while *'recordKSPKResp'*, and *'recordKSSKResp'* allow the TPA to record corresponding key service response snapshots. Then, *'recordKSPKResp'* function allows us to confirm the receipt of the key service.

Inspection Functions. The inspection functions address the monitoring task for the recorded audit obligation as discussed in Section 3.3.3. To be specific, *'inspectObligationKS'* allows to automatically inspect the completeness of the key service obligations, while *'inspectObligationPP'* permits the monitor to verify the published *identity-to-public-key* binding. Regarding the incentive design, if a dishonest behavior is detected, the corresponding entity will be fined a fixed number of *ethers* that will be provided as an incentive reward to the monitor.

Experimental Setup: Our experiments have been performed on a Macbook Pro platform with 2.3GHz 8-Core Intel Core i9 processors and 32GB DDR4 memory. Besides, we use the Ethereum official test network, Rinkeby, as the experimental
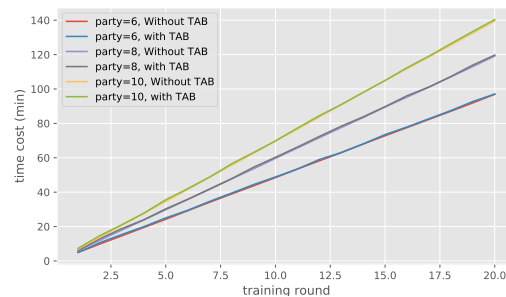

Fig. 5. The time cost of `TAB`-enhanced privacy-preserving FL

environment to deploy our smart contract. Furthermore, we write several *JavaScript* test-cases using the automated testing framework of *Truffle* that is built on Mocha[5] and provides a cleanroom environment.

Specifically, for demonstration, we use five Ethereum accounts to simulate various entities in `TAB`, namely, the role of the *administrator*, the *TPA*, the *data owner*, the *data user* and the *monitor*. With regards to various scenarios, we write corresponding test-cases to evaluate the performance (i.e., the gas cost and the time cost for scenarios such as administrative, registration, obligation, etc).

4.2.2 Experimental Results

We report the performance of `TAB` for selected functions for various test scenarios in TABLE 2. In particular, the performance includes two aspects: the gas cost and the test time. Gas is spent in Ethereum for deploying smart contracts or calling functions. As reported in TABLE 2, most functions cost very little. Specifically, except for the smart contract deployment, the cost of each function is at the level of 10^5 gas in general. Regarding the most called functions for obligation and inspection, to record an audit obligation for one key service, the functions related to three-phase commitment (i.e., *recordKSSKReq*, *recordKSSKResp*, *recordKSConfirm*) cost 3.7×10^5 gas, 8.4×10^5 gas, 4.3×10^5 gas, respectively. Besides, the cost of inspection for key service and public parameter audit obligations is 2.4×10^5 gas and 3.7×10^5 gas, respectively.

Furthermore, we also measure the time it takes to test the selected functions. Except for the administrative functions, the calling time of rest of the functions is less than $100ms$. Note that the time to test each function is measured in the Ethereum test network. The testing time is related to execution time instead of time taken to confirm the transaction . Thus, the deployment time of the smart contract is only $183ms$ rather than the general time taken to confirm a transaction, namely, about 6 minutes.

To further assess the `TAB` framework's scalability, we integrate it into a specific privacy-preserving federated learning application built on the FE cryptosystem with a TPA for key service, where the number of enrolled data owners (i.e., called party in the FL system) increased from 6 to 10 and the task of FL is to train a CNN model over MNIST dataset. Due to the fact that TAB focuses exclusively on the key interactions between the user and the TPA, and

---

3. https://github.com/iRxyzzz/tab
4. https://openzeppelin.com/contracts/

5. https://mochajs.org/

TABLE 2
The gas cost and test time of selected functions in various test case scenarios in the `TAB`.

Test Cases	Functions	Gas Cost	Time	Description
Administrative	deployment	4125603	183ms	deploy the smart contract
	enrollOpen	44126	42ms	open the enrollment
	enrollLock	14531	46ms	lock the enrollment
	dropout	28293	178ms	allow to drop out and withdraw the balance
Incentive	depositGuarantee	28083	48ms	deposit the guarantee
	rewardRegisterCost	52949	43ms	reward registration cost for non-payable entity
	rewardDeploymentCost	51584	41ms	reward deployment for the administrator
Registeration	registerAuthority	38276	80ms	register the role of third-party authority
	registerActorDataOwner	38335	71ms	register the role of data owner
	registerActorDataUser	36555	70ms	register the role of data user
	registerMonitor	36521	72ms	register the role of monitor
Obligation	recordKSSKReq	43173	96ms	publish the key service request snapshot
	recordKSSKResp	84211	55ms	publish the key service response snapshot
	recordKSConfirm	43402	49ms	confirm receipt of the key service obligation
Inspection	inspectObligationKS	24511	41ms	inspect the key service audit obligation
	inspectObligationPP	37482	46ms	check the correct of the public parameter

the time cost is negligible in comparison to the time cost of FL training, the introduction of the `TAB` framework has slight effect on the time performance of the original privacy-preserving FL training, as shown in Fig. 5. Additionally, an increase in the number of participants has an effect on the amount of time spent on the supported privacy-preserving application (i.e., FL training), but not obvious on the amount of time spent on `TAB`.

5 RELATED WORK

Privacy Enhanced Applications. Emerging FE schemes [19], [20] have been shown to be a promising candidate for secure computation in privacy-preserving application scenarios where data is encrypted and outsourced, and the computation is carried out over the encrypted data. Especially, recently proposed functional encryption for the functionality of computing the inner-product such as in [20], [32] raises the possibility of applying functional encryption in complex applications such as the federated learning and deep neural networks, as demonstrated in [2], [3], [23], [24], [33], [34].

Both HE and FE schemes are required to allow privacy-enhanced computing in crypto-based federated learning applications. While HE makes use of a centralized crypto dealer to synchronize key pairs, FE makes use of a third-party authority (TPA) to assist in the generation of public parameters and the provision of functional decryption keys for each function. The crypto dealer or TPA is a vital component of those applications and is typically regarded to be *fully trustworthy*. Furthermore, the entities such as a coordinator in federated learning are also assumed to be *honest-but-curious*. However, the trust issues caused by malicious insiders in the TPA infrastructure [8], [16] and the privacy inference issues caused by curious entities [2], [3] have not been investigated adequately.

Transparency and Blockchain. The concept of transparency issues have received more and more attention due to malicious activities or misbehavior in various secure computing infrastructures and components. For instance, the *certificate transparency* proposed in [5], [6] aims to mitigate the certificate-based threats caused by fake or forged SSL certificates that are mistakenly or maliciously issued by

insiders. Most recent and related work such as *CONIKS* [7] and its following up work *EthIKS* [9], *SEEMless* [10], and *transparency overlay* [8] target the key transparency in end-to-end encrypted communications systems and provide a formal transparency model. The work closest to this proposed work is the *authority transparency* framework proposed in [16] that addresses the issues related to potentially dishonest TPAs in ABE-based applications using a secure logging based approach. Further, blockchain based techniques have also been introduced to help increase the transparency of existing certificate transparency framework, such as in [15], [35]. However, there is still a lack of a mechanism to address the *authority transparency* issues for emerging crypto-based privacy-preserving applications without relying on the complex secure logging systems. Such a transparency approach is important to ensure the trustworthy deployment of generic crypto-enabled systems.
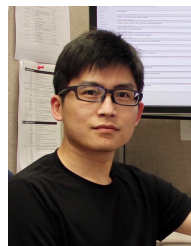
6 CONCLUSION

This paper proposed the `TAB` framework to address transparency and trustworthiness of *third-party authorities (TPAs)* and *honest-but-curious* entities for generic modern crypto enabled privacy-preserving applications, as well as other schemes that have components similar TPAs and many entities that interact with them. `TAB` employs the Ethereum blockchain as the underlying public ledger infrastructure and also incorporates a novel smart contract to support accountability with an additional incentive mechanism that motivates participants to engage in auditing and punish misbehaviors or malicious behaviors in the environment. Our evaluation shows that `TAB` is efficient in the simulated Ethereum, and achieves the security, privacy and trustworthiness goals.
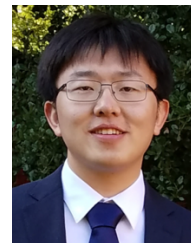
ACKNOWLEDGMENT

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3179698

12

REFERENCES

[1] E. D. Sans, R. Gay, and D. Pointcheval, "Reading in the dark: Classifying encrypted digits with functional encryption." *IACR Cryptol. ePrint Arch.*, p. 206, 2018.

[2] R. Xu, J. B. Joshi, and C. Li, "Cryptonn: Training neural networks over encrypted data," in *ICDCS*. IEEE, 2019, pp. 1199–1209.

[3] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, "Hybridalpha: An efficient approach for privacy-preserving federated learning," in *ACM AISec*, 2019, pp. 13–23.

[4] T. Ryffel, D. Pointcheval, F. Bach, E. Dufour-Sans, and R. Gay, "Partially encrypted deep learning using functional encryption," in *NeurIPS*, 2019, pp. 4517–4528.

[5] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," IETF, Tech. Rep., 2013.

[6] B. Laurie, "Certificate transparency," *Queue*, vol. 12, no. 8, p. 10, 2014.

[7] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "Coniks: Bringing key transparency to end users." in *USENIX Security*, 2015, pp. 383–398.

[8] M. Chase and S. Meiklejohn, "Transparency overlays and applications," in *ACM CCS*, 2016, pp. 168–179.

[9] J. Bonneau, "Ethiks: Using ethereum to audit a coniks key transparency log," in *IACR FC*. Springer, 2016, pp. 95–105.

[10] M. Chase, A. Deshpande, E. Ghosh, and H. Malvai, "Seemless: Secure end-to-end encrypted messaging with less trust," in *ACM CCS*, 2019, pp. 1639–1656.

[11] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski, "Design, analysis, and implementation of arpki: an attack-resilient public-key infrastructure," *TDSC*, vol. 15, no. 3, pp. 393–408, 2016.

[12] J. Yu, M. Ryan, and C. Cremers, "Decim: Detecting endpoint compromise in messaging," *TIFS*, vol. 13, no. 1, pp. 106–118, 2017.

[13] D. Kumar, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey, "Tracking certificate misissuance in the wild," in *S&P*. IEEE, 2018, pp. 785–798.

[14] Q. Scheitle, T. Chung, J. Hiller, O. Gasser, J. Naab, R. van Rijswijk-Deij, O. Hohlfeld, R. Holz, D. Choffnes, A. Mislove *et al.*, "A first look at certification authority authorization (caa)," *SIGCOMM CCR*, vol. 48, no. 2, pp. 10–23, 2018.

[15] J. Chen, S. Yao, Q. Yuan, K. He, S. Ji, and R. Du, "Certchain: Public and efficient certificate audit based on blockchain for tls connections," in *INFOCOM*. IEEE, 2018, pp. 2060–2068.

[16] R. Xu and J. Joshi, "Trustworthy and transparent third party authority," *TOIT*, 2020.

[17] N. Gorasia, R. Srikanth, N. Doshi, and J. Rupareliya, "Improving security in multi authority attribute based encryption with fast decryption," *Procedia Computer Science*, vol. 79, pp. 632–639, 2016.

[18] S. Agrawal and M. Chase, "Fame: fast attribute-based message encryption," in *ACM CCS*, 2017, pp. 665–682.

[19] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *TCC*. Springer, 2011, pp. 253–273.

[20] M. Abdalla, D. Catalano, D. Fiore, R. Gay, and B. Ursu, "Multi-input functional encryption for inner products: function-hiding realizations and constructions without pairings," in *CRYPTO*. Springer, 2018, pp. 597–627.

[21] P. Ananth, A. Jain, Z. Jin, and G. Malavolta, "Multi-key fully-homomorphic encryption in the plain model," in *IACR TCC*. Springer, 2020, pp. 28–57.

[22] H. Chen, I. Chillotti, and Y. Song, "Multi-key homomorphic encryption from tfhe," in *Asiacrypt*. Springer, 2019, pp. 446–472.

[23] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, J. Joshi, and H. Ludwig, "Fedv: Privacy-preserving federated learning over vertically partitioned data," in *ACM AISec*, 2021, pp. 181–192.

[24] R. Xu and J. Joshi, "Revisiting secure computation using functional encryption: Opportunities and research directions," in *IEEE TPS-ISA*. IEEE, 2020, pp. 226–235.

[25] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *ACM CCS*, 2006, pp. 89–98.

[26] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *S&P*. IEEE, 2007, pp. 321–334.

[27] M. Bellare and S. Keelveedhi, "Interactive message-locked encryption and secure deduplication," in *PKC*. Springer, 2015, pp. 516–538.

[28] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[29] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, and A. van Moorsel, "Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing," in *ACM CCS*, 2017, pp. 211–227.

[30] M. J. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," Tech. Rep., 2015.

[31] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[32] M. Abdalla, F. Bourse, A. De Caro, and D. Pointcheval, "Simple functional encryption schemes for inner products," in *PKC*. Springer, 2015, pp. 733–751.

[33] R. Xu, J. Joshi, and C. Li, "Nn-emd: Efficiently training neural networks using encrypted multi-sourced datasets," *IEEE TDSC*, 2021.

[34] S. R. Pokhrel and J. Choi, "Federated learning with blockchain for autonomous vehicles: Analysis and design challenges," *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 4734–4746, 2020.

[35] Z. Wang, J. Lin, Q. Cai, Q. Wang, D. Zha, and J. Jing, "Blockchain-based certificate transparency and revocation transparency," *TDSC*, 2020.

Runhua Xu is a research staff member at IBM Research. His research interests cover a range of areas: privacy-preserving machine learning, secure computing, access control, applied cryptography, blockchain, security and privacy issues in cloud/edge computing. He holds a Ph.D. degree in Information Security from School of Computing and Information, University of Pittsburgh in 2020. Prior to that, Runhua received his M.S. degree in Computer Science from Beihang University, China in 2014, and his B.E. in Software Engineering from Northwestern Polytechnical University, China in 2011.

Chao Li is an Assistant Professor in the School of Computing and Information Technology at Beijing Jiaotong University. His current research interests are focused on Blockchain and Data Privacy. He received my MSc degree from Imperial College London and my Ph.D. degree in Information Science from University of Pittsburgh.

James B.D. Joshi received the MS degree in computer science and the Ph.D. degree in computer engineering from Purdue University. He is a professor of School of Computing and Information (SCI), at the University of Pittsburgh. His research interests include Access Control Models, Security and Privacy of Distributed Systems, Trust Management and Information Survivability. He is the director of LERSAIS at the University of Pittsburgh. He is an elected fellow of the Society of Information Reuse and Integration (SIRI) and is a senior member of the IEEE and the ACM. He currently serves as a Program Director of the Secure and Trustworthy Cyberspace program at the National Science Foundation.