Trustworthy and Transparent Third Party Authority

RUNHUA XU and JAMES JOSHI, University of Pittsburgh, United States

Recent advances in *cryptographic* approaches, such as Functional Encryption, Attribute-based Encryption and their variants, have shown significant promise for enabling public clouds to provide secure computation and storage services for users' sensitive data. A crucial component of these approaches is a *third party authority (TPA)* that must be trusted to set up public parameters, provide private key service, etc. Components of deployed cryptographic mechanisms such as the *certificate authorities (CAs)*, which are the TPAs of the underlying PKI for the SSL/TLS protocol, have faced several types of attacks (e.g., *stealthy targeted* and *censorship* attacks), and certificate *mis-issuance* problems. Such practical challenges indicate that the successful deployment of newer emerging cryptographic schemes will also significantly depend on the trustworthiness of the TPAs. Furthermore, recently proposed decentralized TPA approaches that lower the threshold on the conditions required for an entity to become an authority can make the trust issue much worse. To address this issue, we propose an *authority transparency* framework to ensure the trustworthiness of TPAs of recent and emerging advanced cryptographic schemes. The framework includes a formal model and a *secure logging* based approach to implement the framework. Further, to address the issues related to privacy, we also present a *privacy-preserving authority transparency* approach. We present security analysis and performance evaluation to show that *authority transparency* achieves the security and performance goals.

CCS Concepts: • **Security and privacy → Usability in security and privacy**; **Trust frameworks**; **Domain-specific security and privacy architectures**; *Access control*; *Key management*; *Public key (asymmetric) techniques*; *Security services*.

Additional Key Words and Phrases: transparency, audit, trusted authority, access control, secure computation

ACM Reference Format:
Runhua Xu and James Joshi. 2020. Trustworthy and Transparent Third Party Authority. *ACM Trans. Internet Technol.* 1, 1 (March 2020), 23 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The trust that people place on Internet services or IoT devices has been decreasing because of the increased number of data breaches, digital certificate issues, and threats from IoT devices seen in recent times [1, 24, 28, 42, 48]. On the other hand, with the explosion of data, it is difficult for users to store, manage, and process all their data without the help of such third party services. Hence, it is critical that trust on third party services, especially those that store sensitive user data is very critical. To tackle such a crisis of trust, two alternative approaches have emerged in the literature: (i) adopting cryptographic mechanisms to protect outsourced data, and (ii) increasing trust on service providers requiring them to provide openness and accountability. Even though systems that maintain users' outsourced data are compromised, existing cryptographic approaches such as the mechanisms proposed in [4, 7, 10, 23, 27, 29, 30, 39, 45] can ensure the confidentiality of the data while supporting access control and secure computation. The provisioning of *openness* and *accountability*, also referred to as *transparency*

This research work is supported by the National Science Foundation grant DGE-1438809.
Authors' address: Runhua Xu, runhua.xu@pitt.edu; James Joshi, jjoshi@pitt.edu, School of Computing and Information, University of Pittsburgh, 135 North Bellefield Avenue, Pittsburgh, Pennsylvania, United States, 15260.

1533-5399/2020/3-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

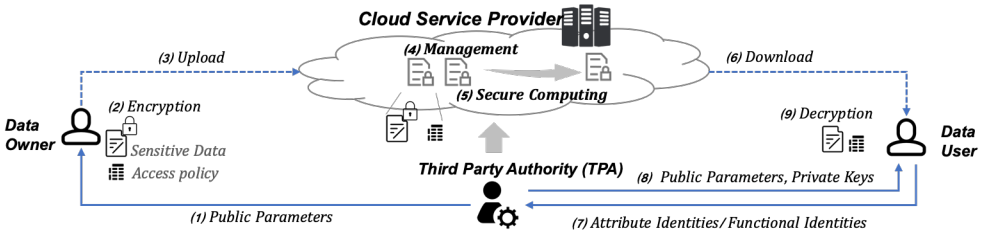
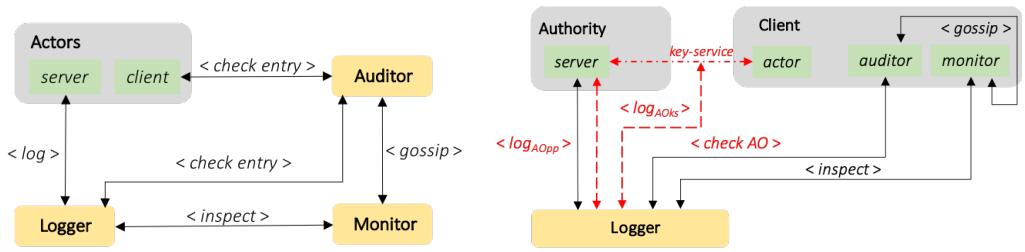


Fig. 1. An illustration of advanced crypto schemes for secure computation and access control on encrypted sensitive data outsourced to a cloud. The numbers indicate a typical sequence of operations.

in recent literature [16, 21, 22, 32, 34, 38, 40], can additionally help increase users' trust or confidence on service providers with respect to the protection of their sensitive data.

Recent advances in cryptography-based access control and secure computation approaches such as the *predicate encryption* schemes [27, 29], *functional encryption* schemes [10, 23, 45], *attribute based encryption* schemes [2, 15, 26, 37, 39], and *access control encryption* schemes [4, 30] have enabled a public cloud to provide secure computation, and secure storage and management services for users' sensitive outsourced data. For simplicity, we refer to these as ***advanced crypto schemes*** in the rest of the paper. By"***advanced***", we typically mean cryptographic schemes that cannot necessarily be built from the traditional primitives and use more modern/emerging approaches that rely on a *third party authority (TPA)* to provide a private key service. These *advanced crypto schemes* support data confidentiality with additional features such as access control, and secure computation over encrypted data. For instance, a *functional encryption* approach proposed in [10] allows computation of several functions directly over encrypted data. The *ciphertext policy attribute-based encryption* scheme proposed in [7] is ideal for supporting fine-grained access control over sensitive data outsourced in the cloud storage, while an *access control encryption* scheme proposed in [30] can restrict information flow in terms of users' *read* and *write* permissions. Fig. 1 illustrates a typical application of such *advanced crypto schemes* for secure computation and access control. A user can encrypt his sensitive data with a specified access policy over a set of attributes, and outsource it to a public cloud for secure storage, access management and secure computation. If a user needs to access such data, he can be given a private key that is generated by the associated *trusted TPA* based on his identities. If the user's attributes satisfy the access policy, he can access the data by decrypting the encrypted data using this private key.

In these advanced crypto schemes, the *trusted TPA* is a critical component that usually sets up the public parameters and generates private keys for the users. Trust on the TPA is an important assumption in these systems. However, such a trust assumption is not practical for real world deployment [35, 36]. Although such systems are not widely used over the Internet, various attacks [5, 11, 47] and mis-issuance problems [17, 31, 41] have been encountered in existing deployed TPA components such as *certificate authorities (CAs)*. The CAs are the underlying public key infrastructure for SSL/TLS protocol, and such possible attacks indicate that establishing trustworthiness of TPAs is critical for the success of any scheme that relies on a TPA. According to the survey in [31] related to the existing CA infrastructures, even though only 0.02% of total certificates violated the standards from IETF and CA/Browser Forum in 2017, the mid-sized and a long tail of small authorities made a variety of errors resulting in mis-issuance of more than 10% of their respective certificates. Such similar misbehavior or mistakes in the collaborative TPA infrastructure increases the trust concerns. Furthermore, an authority can also become untrusted when it is under *stealth targeted* and/or *censorship* attacks from insider threat agents (e.g., malicious staff managing the authority); this issue can get much worse in a collaborative TPA environment. To the best of our knowledge, solutions to ensuring trust on the TPAs within the context of advanced crypto schemes have not been investigated yet.



(a) transparency overlay illustration (b) authority transparency illustration

Fig. 2. An overview of *transparency overlay* and *authority transparency*. Note that the angle brackets denote the interaction protocols required in the transparency framework. Compared to *transparency overlay*, the crucial improvement in *authority transparency* is integrating auditor and monitor into client and introducing secure three-party log interaction to audit the key service process in advanced cryptography systems.

In this paper, we propose a notion of ***authority transparency*** to address the above mentioned ***trust issue*** in a *third party authority* component of advanced crypto schemes. The proposed notion is inspired by the recently proposed notions of *certificate transparency* [34] and *transparency overlay* [16] that focus on *certificate authorities*.

Transparency overlay is an initial formal study of the *certificate transparency* framework previously proposed in [32, 34]. As illustrated in Fig. 2, we differentiate the notion of *transparency overlay* with that of *authority transparency*. The *transparency overlay* framework can support a SSL certification system, but cannot be used in advanced crypto schemes because of the following issues: (i) unlike in conventional public key system, where the certificate authority only needs to issue a certificate that proves the *"identity-to-public-key"* binding, the TPA in an advanced crypto scheme is responsible for more complex tasks such as setting public parameters, managing fine-grained identities/attributes, generating private keys for users, etc.; (ii) the SSL certificate is the only object that is submitted by an issuer for auditing in the *certificate transparency* framework, while the audited objects in our *authority transparency* capture multiple rounds of interactions between the TPA and the client; (iii) the volume of audited objects such as attribute identity related public parameters can be much larger than the volume of certificates, thus, increasing the complexity of the auditing tasks. A separate auditor/monitor setting in a *transparency overlay* framework may not handle such a case efficiently. In the proposed work, we integrate the role of *auditor* and *monitor* into a *client*. As a result, it makes each *actor* (i.e., the service user of TPAs) to be an audit participant in the transparency operations. More detailed formalization and framework is presented in Section 3 and Section 4. We summarize our key *contributions* as follows:

(1) We present several attacks on the existing TPA infrastructures that may be possible from insider threat agents, and illustrate trust issues caused by such attacks.
(2) To address the issue of trusting a TPA, we propose a new notion of *authority transparency* and present an implementation approach, namely, *secure logging based transparency* framework. Essentially, unlike *certificate transparency*, our proposed *authority transparency* approach audits not only the objects such as certificates and public keys, but also the interactions such as those in a private key service procedure.
(3) We also present an attack to illustrate the privacy issue in the proposed authority transparency framework, and propose a privacy-preserving authority transparency model.
(4) We present an analysis of security properties and performance evaluation of our proposed authority transparency framework.

Organization. The rest of the paper is organized as follows. In Section 2, we introduce the motivation of our proposed work. We present the proposed notion of *authority transparency* with formal description and the

privacy-preserving authority transparency in Section 3. We present a secure logging based framework in Section 4, and the security analysis and performance evaluation in Section 5. We review the related work in Section 6 and present the conclusions in Section 7.

2 MOTIVATION

Emerging advanced crypto schemes support more features, such as cryptography-based access control (CBAC) and secure data processing, than conventional public key schemes. As mentioned earlier, a *TPA* is a fundamental component of most existing advanced crypto schemes and the TPA's trustworthiness is critical for its successful deployment. Essentially, the *TPA* here is responsible for setting up the public keys including common components and parameters related to users' attribute identities, and generating the private keys based on users' attribute identities. However, existing TPAs still bear several limitations that hinder wider deployment of advanced crypto schemes. Here, we summarize such limitations as follows:

i *Large and evolving sets of attributes.* A large organization typically has a huge set of attributes. This increases the complexity of attribute management, attribute authentication and attribute-related computation. Furthermore, a set of organizations served by the same TPA may constantly change as newer organizations seek services from the TPA and the existing ones leave. This makes S_U, as in Example 3.1, to constantly change, introducing additional burden to the TPA.
ii *Single point of failure.* The central authority becomes a central point of failure or the main target of attacks. A compromised or malicious TPA can continue serving the users/organizations without being detected until significant damage has been done.
iii *Trustworthiness of TPA.* The fact that an authority needs to be fully trusted by all the organizations and users is a strong assumption and is not practical in real scenarios where various threats, including from insiders, exist.

Recent research efforts related to TPA infrastructures have mainly focused on improving efficiency [15, 26, 37]. For instance, a group of collaborative TPAs enable or support applications where one party can share data using attribute identities from different domains and organizations. Hierarchical or decentralized TPA schemes have been proposed in the literature [15, 26] to tackle limitation (i); however, they still require a global or central authority to coordinate distributed authority servers. The dependency on the global authority, and the need to fully trust it, make it the bottleneck of the system. To address the limitation (ii), a *decentralized TPA* approach such as in [37] enables an organization or an individual to simply act as a TPA without any need of a global coordination. Thus, any entity can become a TPA that manages and authenticates the attributes to provide a key service.

Although approaches proposed in [15, 26, 37] provide a significant promise for a more practical TPA for an advanced crypto scheme, neither traditional TPA nor collaborative TPA addresses limitation (iii). Essentially, TPA approaches proposed in [15, 26] increase complexity of TPA architectures, while the approach proposed in [37] relaxes the strict condition required for becoming an authority, and it allows any party to become a TPA. Thus, these newer approaches further increase the concerns related to limitation (iii).

3 AUTHORITY TRANSPARENCY

To address the above-mentioned concerns, we propose *authority transparency* as the mitigation approaches. Here we first present related notations and terminologies. Then we propose attacks on TPAs and introduce the details of our proposed work to prevent such attacks.

3.1 Notation and Terminology

3.1.1 Generalization of Advanced Crypto Schemes. Advanced crypto schemes allow selective access and secure computing over encrypted data via a combination of key management and functional cryptographic schemes.

Phases. A typical advanced cryptography scheme can be generalized to have the following four phases:

(i) *Setup/Initialization phase.* The TPA sets up public parameters according to a universal attribute set for the specified crypto system.
(ii) *Encryption phase.* A data owner encrypts the data by using a set of public parameters or a specified access policy over the attribute identity set for access control purpose.
(iii) *Private key service phase.* The TPA generates users' private keys based on their authenticated attribute identities.
(iv) *Decryption phase.* An authorized user decrypts the ciphertext by using the private key received from the TPA based on his attribute identity as per (iii).

Roles. In any advanced crypto scheme, there are three typical entities, *data owner* ($\mathsf{C}_{\text{owner}}$), *data user* ($\mathsf{C}_{\text{user}}$), and the *TPA* ($\mathsf{T}$). $\mathsf{C}_{\text{owner}}$ employs an *Encrypt* algorithm to protect the outsourced sensitive data. C_{user} employs a *Decrypt* algorithm and the private keys obtained from the TPA to process/decrypt the sensitive data. The TPA is responsible for phases (i) and (iii) above. Here, we generalize two types of *TPAs* from the existing work, namely, *Traditional TPA* and *Collaborative TPA*, as illustrated in Example 3.1 and Example 3.2, respectively.

Example 3.1. Traditional TPA. Suppose there are three organizations, org_x, org_y and org_z that use attribute sets S_x, S_y and S_z, respectively. They employ T for their deployed advanced crypto scheme. For the initial setup, T first needs to collect the attribute identity sets from all the organizations; i.e., $S_U = S_x \lceil S_y \lceil S_z$. Then, the TPA generates the corresponding public parameters based on S_U and provides the private key service.

Example 3.2. Collaborative TPA. Suppose that organizations org_x, org_y and org_z manage attribute sets S_x, S_y and S_z, respectively. Each organization can set up its own TPA, T_x, T_y, and T_z, respectively, also called self-authority, to provide advanced cryptography services. Suppose that $\mathsf{C}_{\text{owner}}$ wants to outsource his data using a CBAC mechanism with a specified access policy; let the policy be: "*anyone who is a manager of* org_x*, or a professor of* org_y*, and is also a member of* org_z *can access the data*". Here we use $\text{att}_a \; 2 \; S_x$, $\text{att}_b \; 2 \; S_y$ and $\text{att}_c \; 2 \; S_z$ to represent the attribute identities as follows: $\text{att}_a \; \$ \; \mathsf{T}_x{:}\text{org}_x{:}\text{manager}$, $\text{att}_b \; \$ \; \mathsf{T}_y{:}\text{org}_y{:}\text{professor}$, $\text{att}_c \; \$ \; \mathsf{T}_z{:}\text{org}_z{:}\text{member}$. Thus, the above policy can be expressed as: (att_a *OR* att_b) *AND* att_c. To acquire access privilege, $\mathsf{C}_{\text{owner}}$ needs to request public parameters from T_x, T_y, and T_z, respectively. Any C_{user} who wants to access the data also needs to request for a key service from the corresponding authorities.

3.1.2 Generalization of Identity. To maintain the generality of the TPA in the advanced crypto schemes, we formalize the parameters and identities that will be used in the rest of the paper. We classify the identity information in an advanced crypto system as *unique identity* and *attribute identity*.

Definition 3.3. ***Unique Identity***. Let $id_{uni}{}^{1\ 0}$ be the unique identity function capable of generating a unique identifier of an entity e. The unique identity of e is denoted as $id_{uni}{}^{1}e^{0}$.

Definition 3.4. ***Attribute Identity***. Let $id_{attr}{}^{1\ 0}$ be the attribute identity function that generates the attribute value of a in the advanced crypto system. The attribute identity of a is denoted as $id_{attr}{}^{1}a^{0}$.

Each unique identity represents one and only one entity in the digital world. For instance, the email account could be a unique identity. In practice, $id_{uni}{}^{1\ 0}$ can be represented by a collision resistant hash function. To avoid naming conflicts, the domain information is introduced as a prefix into the attribute identity. For instance, in the attribute identity "$id_{uni}{}^{1}auth^{0}{:}id_{uni}{}^{1}org^{0}{:}a$", the prefixes $id_{uni}{}^{1}auth^{0}$ and $id_{uni}{}^{1}org^{0}$ represent the authority and the organization that the attribute a belongs to, respectively. Unlike unique identity, an attribute identity is

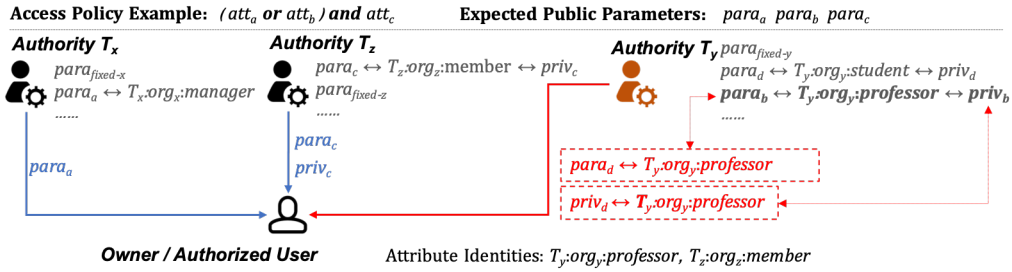


Fig. 3. An illustration of an attack on public parameter distribution and private key service.

capable of representing a group of entities in the digital world. An access policy consists of a set of attribute identities and logical conjunctions. Here, we use an example to illustrate the identities.

3.2 Attacks and Adversary Model

Here, we propose two kinds of attacks on TPAs that can be launched by an adversary such as a malicious/compromised or a misbehaving insider. Below, we use the *Collaborative TPAs* (as in Example 3.2) as the authorities.

Attack 1. ***Attack on distribution of public parameters.*** The target of such an attack could be any data owner $\mathsf{C}_{\text{owner}}$. Suppose that $\mathsf{C}_{\text{owner}}$ employs an advanced crypto scheme to outsource sensitive data with access policy: "$(att_a$ *OR* $att_b)$ *AND* att_c", as shown in Fig. 3. Then, he acquires related public parameters, e.g., $para_{\text{fixed-x}}$, $para_{\text{fixed-y}}$, $para_{\text{fixed-z}}$, $para_a$, $para_b$, $para_c$, from T_x, T_y, and T_z, respectively. At this time, a malicious insider or a compromised TPA, say T_y, can replace $para_b$ with $para_d$. The data owner thinks that he has encrypted the data using the parameter related to attribute att_b, while the data has actually been protected under attribute identity "T_y*:*org_y*: student*". As a result, any user who is a member of org_y, with extra attribute identity, namely, "T_y*:*org_y*:student*", rather than "T_y*:*org_y*:professor*" can now access the data.

Attack 2. ***Attack on private key service.*** The target of such an attack could be any user C_{user}. Suppose that C_{user} needs to access the encrypted data with access policy: "$(att_a$ *OR* $att_b)$ *AND* att_c", and he has two attribute identities "T_y*:*org_y*:professor*" and "T_z*:*org_z*:member*", as shown in Fig. 3. He sends his attribute identities to authority T_y and T_z to request the corresponding private keys, $priv_b$ and $priv_c$, and the expected private key $priv_b$ may be replaced by an invalid one, $priv_d$, or it may never be generated. This can be caused by an insider or a compromised T_y. This impacts the expected functioning of the advanced crypto system by denying a valid access or processing. Here, users may think their attribute identities have been revoked (because of unsuccessful decryption), even when they are actually valid.

Essentially, an adversary can launch Attack 1 by distributing compromised public parameters to a target user without being detected by that user. A success of Attack 2 indicates that there is a lack of a mechanism to ensure the correct functioning of the key service of a TPA. Attack 1 can be classified as *identity-to-public-key-binding stealthy targeted* attack, and Attack 2 is a *private-key-service censorship* attack. Such attacks exacerbate the issue of trusting a TPA. Hence, establishing trust through openness and accountability of a TPA is critical in ensuring users' confidence in using advanced crypto schemes. Potential solutions to address this issue, to the best of our knowledge, have not been investigated in the literature.

Adversary Model. As illustrated in Attacks 1 and 2, we consider an adversary to be any entity within a advanced crypto scheme that is *dishonest*. We assume that such a dishonest adversary may pretend to behave honestly without being detected by other participants. Adversaries may not follow the specification in the protocols, and/or attempt to conceal their activities. A *dishonest* adversary may be a *TPA*, a *user*, or a *logger*. In particular, a

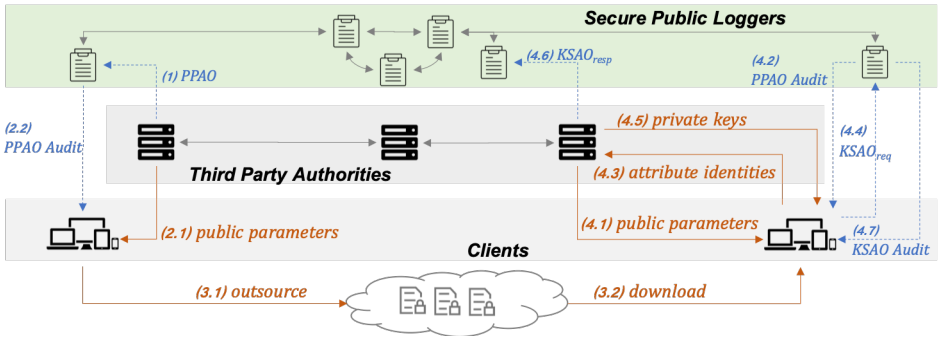


Fig. 4. An overview of secure logging based authority transparency framework. Note that the steps (i.e., 2.1, 3.1, 3.2, 4.1, 4.3, 4.5) represent a typical cryptography based access control scenario. The blue dotted lines are the integrated authority transparency features. The sequence numbers indicate the operation flow of the scenario where a data owner shares the sensitive data within the authority transparency framework.

dishonest *logger* may try to present inconsistent versions of the log to other entities; a dishonest *TPA* may attempt to forge a key service *proof of work* without actually providing a valid key service; and, a dishonest entity may try to incorrectly blame other entities for misbehavior. Note that misbehavior may be related to non-malicious misuse by normal entities or the behavior of the compromised entities controlled by an attacker.

3.3 Overview of Authority Transparency

To tackle the trust issue of TPA, we propose the notion of *authority transparency* and an implementation approach, i.e., secure logging based authority transparency framework, as illustrated in Fig. 4. The secure logging based *authority transparency* framework consists of the following entities:

(i) *Third Party Authority*. It refers to a TPA in advanced crypto schemes (see Section 3.1.1). However, the TPA has more obligations to fulfill in our framework including (a) submitting its unique authority identity and attribute identity bindings to the secure logging system, and (b) reporting its fulfillment of obligations in the key service process by sending a response key service snapshot for each user's request.
(ii) *Secure Public Logger*. Loggers initialize and maintain a *publicly auditable append-only ledger* that will be introduced later. Loggers are also required to respond to the audit queries by sending different cryptographic proofs such as *audit proof* and *consistency proof*.
(iii) *Owner/User*. Owners/users employ encryption and decryption algorithms.
(iv) *Client*. Users employ *client* software on their trusted devices, hence, our framework does not address the issues such as compromised clients. The *clients* act on behalf of owners/users in the framework. It also acts as the monitor and auditor, as in the *certificate transparency* framework.

Additional duties of *clients* in our framework are as follows: (i) on behalf of a user, they request public parameters or private keys based on users' attribute identities; (ii) they monitor and audit the public ledger using *gossiping*, *audit proof* and *consistency proof* protocols that have been discussed in [16, 21, 34]; (iii) They participate in the key service audit obligation process.

3.4 Model of Authority Transparency

The proposed *authority transparency* framework increases the transparency of a TPA infrastructure in advanced crypto systems. In particular, it supports monitoring and auditing of the public credentials generated and the

private key service provided by a TPA. Here, we first define two notations, namely, *public parameter audit obligations* and *key-service audit obligations*.

Definition 3.5. ***Identity Binding***. An identity binding is a key-value pair, namely, $\mathsf{B} = {}^{1}k_{\text{id}}, v_{\text{para}}{}^{o}$, where k_{id} represents the identity defined in Definition 3.3 and Definition 3.4, and v_{para} denotes the corresponding public parameter.

Definition 3.6. ***Public Parameter Audit Obligation (PPAO)***. A PPAO element O_{pp} is a two-tuple ${}^{1}\mathsf{B}_{type}, \text{Sig}_{e}{}^{o}$ where B is an identity binding that needs to be audited and $type$ 2 f$id_{uni}{}^{1}{}^{o}, id_{attr}{}^{1}{}^{o}$g. Sig_{e} indicates the signature of entity e.

Note that the identity binding has been used in *certificate transparency* [34] and its variant [38]. Here, we extend the scope of *identity* to refer to not only a unique fact related to who or what an entity is, but also to a more general concept of a group of entities. For instance, "social security number" or "email account" that represents a unique user could be an identity; similarly, "faculty member of university x" that represents a group of professors is also an identity. An attribute can also be an identity, referred to as *attribute identity*. In addition, the TPA has the obligation to send the PPAO element to the logger for auditing. O_{pp} consists of identity bindings related to a specific TPA. Here, we use an example to illustrate these.

Example 3.7. Suppose that an authority T_x provides services for a university org_a and a medical center org_b. The possible general identity binding and attribute identity binding are as follows:

$$\mathsf{B}_{\text{unique}} = {}^{1}id_{uni}{}^{1}\mathsf{T}_x{}^{o}, \text{pk}_{\mathsf{T}_x,\text{RSA}}{}^{o} \qquad \mathsf{B}_{\text{attribute}} = {}^{1}id_{attr}{}^{1}\text{student}^{o}, \text{pk}_{\mathsf{T}_x,\Pi}\text{»para}_1\text{¼}^{o}$$

Thus, possible public parameter audit obligations O_{pp} are as follows: ${}^{1}\mathsf{B}_{\text{unique}}, \text{Sig}_{\mathsf{T}_x}{}^{o}$ and ${}^{1}\mathsf{B}_{\text{attribute}}, \text{Sig}_{\mathsf{T}_x}{}^{o}$, where $\text{pk}_{\mathsf{T}_x,\text{RSA}}$ is a RSA public key of TPA. T_x and $\text{pk}_{\mathsf{T}_x,\Pi}$»para$_1$¼ indicate the public key component corresponding to attribute identity $id_{attr}{}^{1}student^{o}$ using advanced crypto scheme Π.

Here, we define *key service audit obligations* as below.

Definition 3.8. ***Key Service Snapshot***. A key service snapshot is a 7-tuple as follows:

$$\mathsf{S} = {}^{1}v, id_{uni}{}^{1}e_{\text{source}}{}^{o}, id_{uni}{}^{1}e_{\text{target}}{}^{o}, t, r, \sigma, \text{Sig}_{e_{\text{source}}}{}^{o},$$

where $v = h_{CRHF}{}^{1}S_{\text{attr}}, r^{o}$ is a unique value derived from an attribute set S_{attr} that is used in the key request service interaction and r is the protocol execution indicator indicating a specific round of the key service. $h_{CRHF}{}^{1}{}^{o}$ takes attribute set S_{attr} as the input and generates a unique value. $id_{uni}{}^{1}e_{\text{source}}{}^{o}$ and $id_{uni}{}^{1}e_{\text{target}}{}^{o}$ are the unique identity components of the service source and target entity, respectively. t is the timestamp component of the service. σ indicates the *proof of work*. $\text{Sig}_{e_{\text{source}}}$ is the signature of the snapshot signed by the source entity.

Definition 3.9. ***Key Service Audit Obligation (KSAO)***. A *KSAO* element $\mathsf{O}_{\text{ks}} = {}^{1}\mathsf{S}_{\text{req}}, \mathsf{S}_{\text{resp}}{}^{o}$, consists of a pair of *key service* snapshots, namely, a *request key service* snapshot, S_{req}, and a *response key service* snapshot, S_{resp}, such that, (1) $\mathsf{S}_{\text{req}}.v = \mathsf{S}_{\text{resp}}.v$, (2) $\mathsf{S}_{\text{req}}.id_{uni}{}^{1}e_{\text{source}}{}^{o} = \mathsf{S}_{\text{resp}}.id_{uni}{}^{1}e_{\text{target}}{}^{o}$, (3) $\mathsf{S}_{\text{req}}.id_{uni}{}^{1}e_{\text{target}}{}^{o} = \mathsf{S}_{\text{resp}}.id_{uni}{}^{1}e_{\text{source}}{}^{o},$, (4) $\mathsf{S}_{\text{req}}.t < \mathsf{S}_{\text{resp}}.t$, (5) $\mathsf{S}_{\text{resp}}.t \quad \mathsf{S}_{\text{req}}.t < \Delta_t$, where Δ_t is the threshold of timestamp difference indicating the expected time of processing of the key service request by the TPA.

Note that the *key service snapshot* is a record that can be used to represent *request* and *response* snapshots of key service generated by users and authorities, respectively. One possible way to implement the snapshot function $h_{CRHF}{}^{1}{}^{o}$ is to use a collision-resistant hash function (CRHF), e.g., SHA-2 or SHA-3. S_{req} represents the key service snapshot of the key service requester, i.e., clients, and S_{resp} denotes the key service snapshot of the key service responder, the TPA. The TPA and client both have the obligation to send their parts of the KSAO elements to the logger for auditing. Constraint (1) ensures matching a user's request with the corresponding

Authority Transparency $AT_{O}^{T,L,C}$

1 : $(S_{O_{pp}}, S_{O_{ks}}) \leftarrow \mathsf{Run}(1^{\lambda}, \mathsf{Gen}_{O}, \{T, C.actor\})$

2 : $(b_T, \varepsilon) \leftarrow \mathsf{Run}(1^{\lambda}, \mathsf{Log}_{O_{pp}}, \{T, L\}, (S_{O_{pp}}, \varepsilon))$

3 : $(b_T, b_C, \varepsilon) \leftarrow \mathsf{Run}(1^{\lambda}, \mathsf{Log}_{O_{ks}}, \{T, C.actor, L\}, (\varepsilon, O_{ks}[S_C], O_{ks}[S_T]))$

4 : $(\varepsilon, b_{C.actor}, b_{C.auditor}) \leftarrow \mathsf{Run}(1^{\lambda}, \mathsf{Check}_{O}, \{L, C.actor, C.auditor\}, (\varepsilon, O, \varepsilon))$

5 : $(b_L, \varepsilon) \leftarrow \mathsf{Run}(1^{\lambda}, \mathsf{Inspect}, \{L, C.monitor\}, (\varepsilon, \varepsilon))$

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

Fig. 5. The interactive protocols for authority transparency. Note that we adopt the notation from [6]. The order of parameters in the input tuple and the order of elements in the output are consistent with participant entities. Taking second interactive protocol as an example, participant entity T has the input $S_{\mathsf{O}_{pp}}$ and outputs b_{T}, while entity L has no input and output denoted as ε, where $S_{\mathsf{O}_{pp}}$ and $S_{\mathsf{O}_{ks}}$ are the set of O_{pp} and O_{ks} elements, respectively.

TPA response on the same attribute set. Constraints (2) and (3) check to ensure that the obligation is a pair of request and response. Time related constraints (i.e., (4) and (5)) ensure that the S_{resp} is a fresh response to the corresponding S_{req}.

In our model, we separate the *PPAO* into two parts, namely, the *unique PPAO* of the TPA, $\mathsf{O}_{\text{pp,authority}}$, and the *attribute PPAO* of the TPA, $\mathsf{O}_{\text{pp,attribute}}$, because the set of authority identity bindings is relatively more stable than the set of attribute identity bindings in O_{pp}. In particular, (i) once the identity of an authority has been set up and broadcast, it does not need to be updated; (ii) on the other hand, the attribute identities managed by the authority may occasionally vary as users' privileges can be revoked - hence, we divide O_{pp} into two separate streams; (iii) the key service snapshot is taken in each round of key request/response; hence, the key service audit obligation varies in each interval. Thus, three types of audit obligations need to be recorded separately. Here, we compare them with existing approaches:

(i) $\mathsf{O}_{\text{pp,authority}}$ includes the authority's unique identity bindings, which is similar to digital certificates in *certificate transparency* in [34];
(ii) $\mathsf{O}_{\text{pp,attribute}}$ includes all the attribute identity bindings, which is similar to public keys of end users in *key transparency* in [38], but it is more complex than users' public keys since the attribute identity related parameters are the basic elements of existing advanced crypto schemes to provide access control and secure computing features;
(iii) *KSAO* is another concept we have proposed; it has not been studied in the existing literature.

3.4.1 Authority Transparency. We model *authority transparency* as a publicly auditable set of TPA's activities. In particular, the goal is to ensure that the TPA fulfills its auditing obligations related to public parameter distribution and trustworthy key service, continuously and transparently. Towards this, we specify the interactive protocols as below.

Definition 3.10. ***Authority Transparency.*** Let T, L and C denote the third party authority, logger server, and client, respectively, which are parties involved in the interactive protocols. Let $\mathsf{C}.actor$, $\mathsf{C}.auditor$ and $\mathsf{C}.monitor$ represent the roles of the actor, auditor and monitor that execute the functional, auditing and monitoring modules, respectively. We define *authority transparency* as a set of six interactive protocols:

$$\mathsf{AT} = {}^{1}\mathrm{Gen}_{\mathsf{O}}, \mathrm{Log}_{\mathsf{O}_{pp}}, \mathrm{Log}_{\mathsf{O}_{ks}}, \mathrm{Check}_{\mathsf{O}}, \mathrm{Inspect}, \mathrm{Gossip}^{\mathsf{o}},$$

where each protocol is as specified in Fig. 5.

As depicted in Fig. 5, the protocols are as follows: (1) Gen_{O} is an interactive protocol between T and $\mathsf{C}.actor$ that generates the audit obligations to be logged; (2) $Log_{\mathsf{O}_{pp}}$ is an interactive protocol between T and L that is used to record O_{pp} in the public log; (3) $Log_{\mathsf{O}_{ks}}$ is an interactive protocol involving T, L and $\mathsf{C}.actor$ that is used to record O_{ks} in the public log; (4) $Check_{\mathsf{O}}$ is an interactive protocol involving L, $\mathsf{C}.actor$ and $\mathsf{C}.auditor$ that is used to check whether or not an audit obligation O_{pp} or $\mathsf{O}_{ks}$is in the log; (5) *Inspect* is an interaction between L and $\mathsf{C}.monitor$ that is used to allow the monitor to inspect the contents of the log and find suspicious audit obligations $\{\mathsf{O}_i\}$; (6) *Gossip* is an interaction between $\mathsf{C}.auditor$ and $\mathsf{C}.monitor$ that is used to compare different versions of the log and detect any inconsistencies caused by misbehavior on behalf of the log server.

Note that we adopt the notation of interactive protocol presented in [6]. In general, the behavior of a stateful participant p with input m during the i-th round of the j-th execution of a protocol P can be defined as $(\text{state}_p, \text{out}) \leftarrow p_{rcv}(\cdot), p_{rcv}, \text{out}) \leftarrow p(\cdot)$ $\mathsf{P}(p, i, j, 1^\lambda, \text{state}_p, \text{in}_p)$, where $\text{out} \leftarrow p_{rcv}(\cdot)$ denotes the message sent to receiver p_{rcv} in the i-th round of the j-th execution. Thus, the execution of the entire interactive protocol can be defined by $outputs \quad Run(1^\lambda, \mathsf{P}, \{p_1, p_2, ..., p_n\}, inputs)$, where 1^λ is the secure parameter indicating the number of bits for secure related parameters, and $\{p_1, p_2, ..., p_n\}$ indicates the participant entities.

We also adopt the notation of a primitive called *dynamic list commitment (DLC)* that abstracts the secure log infrastructure that has been formalized in *transparency overlay* in [16]. The specific DLC algorithms are introduced in the Appendix. In practice, the DLC can be viewed as a generalization of a rolling hash chain or hash tree.

The proposed *authority transparency* components have overlaps with *transparency overlay* components such as *Gossip* and *Inspect* protocols. Furthermore, the protocols $Log_{\mathsf{O}_{pp}}$ and $Check_{\mathsf{O}}$ are similar to the protocols *Log* and *CheckEntry* presented in [16], respectively. Hence, we present these modified protocols in the appendix rather than here.

We elaborate on the tripartite protocol $\text{Log}_{\mathsf{O}_{ks}}^{\mathsf{T},\mathsf{L},\mathsf{C}}$ depicted in Fig. 6, that tackles the issue of auditing the key service procedure. A *request key service snapshot* S_{C} is generated by the client and sent to the logger that first stores the S_{C} temporarily. Then the logger provides a receipt and sends it back to the client for checking (lines 1-5). Simultaneously, the client initiates the key service request to the TPA. The TPA first handles the key request and generates the key according to the attribute identities it receives. It forms the *response key service snapshot* S_{T} as defined in Definition 3.8 (line 7). Then the TPA sends S_{T} and the combination of S_{T} and sk_{attr} back to the logger and the client, respectively (lines 8 and 12). The logger generates a receipt and sends back to the TPA for checking (lines 9-11), while the client verifies the S_{T} by using its received private key sk_{attr} and sends the newly formed key service snapshot $\mathsf{S}_{\mathsf{C}}^{'}$ to the logger (lines 13-14). Finally, the logger verifies and forms a valid O_{ks} according to received $\mathsf{S}_{\mathsf{C}}^{'}$, S_{C}, and S_{T}, and then commits to the log system by adopting the *dynamic list commitments* primitive with two receipts for the client and the TPA, respectively (lines 15-17). The client and TPA finalize *key service audit obligation* by combining the results from checking the receipts they received to confirm whether or not they believe the logger behaved honestly (line 18). We generalize $\{id_{attr}, a_i\}$ and sk_{attr} to represent the messages of a key service procedure (lines 6 and 12 in blue color). The operations of each participant presented in the protocol $\text{Log}_{\mathsf{O}_{ks}}^{\mathsf{T},\mathsf{L},\mathsf{C}}$ are defined in Fig. 6.

$GenKSS_{\mathsf{C}}$ takes the set of attribute identities as input. It first generates a random nonce r as the unique identifier for the interaction. Then it returns the audit obligation as defined in Definition 3.8. Note that the client is not the provider of the service, hence, we keep the component σ as null.

GenStageRcpt takes the *key service snapshot* S_{T} *or* S_{C} *as the input.* It first adopts the *stage* algorithm to store S_{T}. If the snapshot is stored successfully, the logger will generate a receipt that includes the public key, timestamp and a signature. Note that the key service snapshot is the component to form the element of KSAO, hence, the *stage* algorithm aims at storing the key service snapshot temporarily rather than permanently. The function *CheckRcpt* takes the receipt and the snapshot as input to verify the logger's signature.

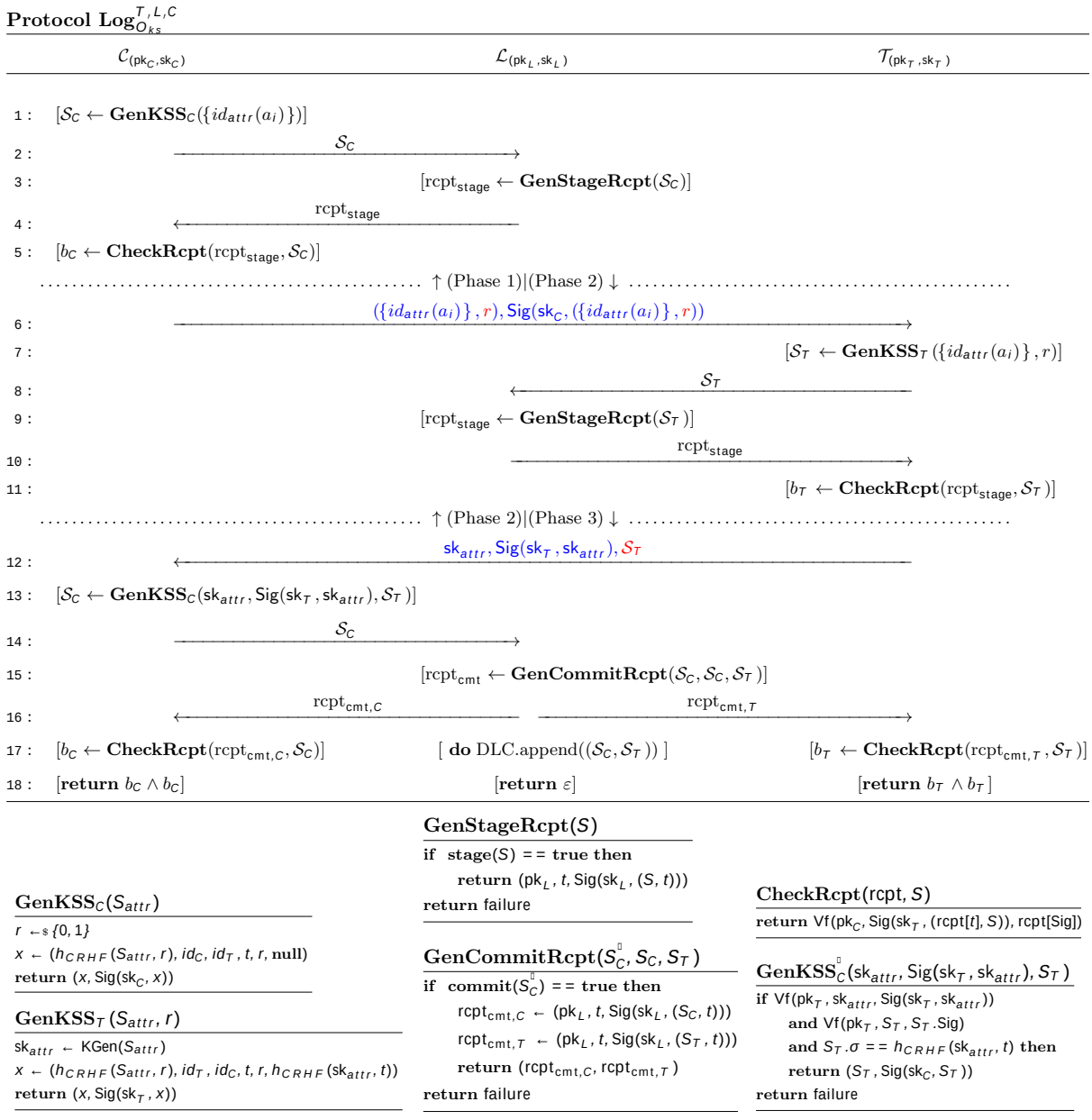


Fig. 6. The $\mathrm{Log}^{\mathsf{T},\mathsf{L},\mathsf{C}}_{\mathsf{O}_{ks}}$ protocol and related operations for authority transparency. Note that the text on the top of the arrow indicates the message flow between entities and the square brackets denote the subsequent operations by the participant entity. The dotted line indicates the division of the three-phase commitment. Sig and Vf represent a pair of signature and verify functions, respectively. KGen denotes the key generation algorithm adopted in the CBAC system.

GenKSS$_{\mathsf{T}}$ takes the set of attribute identity and the random nonce r as input. It first employs the key generation algorithm *KGen* of an advanced crypto scheme to generate the key sk_{attr}. Then it forms the *response key service snapshot* as per Definition 3.8.

GenKSS$^{\emptyset}_{\mathsf{C}}$ takes key sk_{attr} received from T, the signature of a pair of private key and corresponding attribute identity set, and the response S_{T} as input. It first validates the key and the response S_{T}. Then it verifies the key-service proof of work component σ (see Definition 3.8), i.e., compares the CRHF results of received private keys and request timestamp. If these validations are passed, it returns the verified response S_{T} with the client's signature.

GenCommitRcpt takes the three key service snapshots as input. It first adopts *commit* algorithm to submit the snapshot, and then generates the receipts for the client and the authority to confirm the success of enrolling O_{ks} in the log.

3.5 Privacy-Preserving Authority Transparency

Our proposed authority transparency has security guarantee under the presented adversary model. To tackle the potential privacy disclosure issue under the adversary model, we present a privacy-preserving authority transparency considering the cases of applying our proposed work in some privacy-sensitive scenarios.

Privacy Challenge. The PPAO and KSAO elements include components derived from the attribute identities that potentially disclose users' privacy. For instance, the attribute identity binding "$auth_a{:}org_x{:}salary > 10k \ \$ \ para_i$", may disclose users' income. Even though the bindings do not link to any particular user, it is still possible to establish the link and leak the user's privacy. Here, we present Attack 3 to illustrate the privacy issue.

Attack 3. The purpose of this attack is to link the attribute identity to a particular user. Suppose that there is a set of O_{pp} elements used for audit; hence, the set of attribute identity bindings $\mathsf{O}_{pp}.\mathsf{B}_{attr}$ is also public for all clients. There are two possible ways to establish the link: *(i)* A curious adversary can remember a user's audit query for PPAO elements to find what attribute identity bindings a user wants to audit. Thus, an adversary knows with a higher probability that these attribute identities may belong to the user. *(ii)* A curious adversary can monitor a user's audit query for the KSAO elements. As presented in Definition 3.8, v is the hash of a set of attribute identities and the protocol execution indicator (rounds); thus, there is no leakage of attribute privacy. However, the hash function $h^{1\,0}$ and the PPAOs are public and the space of hash input, namely, the size of all attribute identities, is limited. The adversary can use the attribute identity set that is gathered in PPAOs as the input space to reverse v by brute-force.

The public ledger that is maintained by *loggers* only provide audit service for clients; hence, it is still possible to launch the privacy attack.

Privacy-Preserving Authority Transparency. The privacy-preserving authority transparency model adopts an augmented key service snapshot as follows:

$$\mathsf{S} = \mathsf{h}v_{h_k{}^1S_{\text{attr}},r^0}, id_{uni}{}^1e_{\text{source}}{}^0, id_{uni}{}^1e_{\text{target}}{}^0, t, r, \sigma, \text{Sig}_{e_{\text{source}}}\mathsf{i}$$

Unlike the snapshot depicted in Definition 3.8, the augmented model uses a keyed cryptographic hash function $h_k{}^1S_{\text{attr}}{}^0$.

To deal with privacy issue in (i) of Attack 3, our augmented framework increases the scope of audit PPAO by each client. Suppose that a user has attribute identity set $S_{\text{attr}_{\text{user}}}$. In non-privacy-preserving design, his client only audits the part of PPAO that is related to $S_{\text{attr}_{\text{user}}}$, which makes the approach (i) possible. To avoid such privacy leakage, the client can audit PPAO related to $S_{\text{attr}_{\text{user}}} \ [\ S_{\text{attr}_{\text{other}}}$. Here, $S_{\text{attr}_{\text{other}}}$ is the additional attribute identity set to perturb the users' attribute set to avoid the inference of privacy sensitive information.

To address the privacy issue in (ii) of Attack 3, our augmented model uses a cryptographic hash function $h_k{}^{1\,0}$ to replace collision-resistant hash function $h^{1\,0}$. For instance, a hash-based message authentication code function,

i.e., $HMAC_k$[1] [0], which can generate the hash value of specified attribute set with a key. Here the shared key k is established between the user and the authority, which can be constructed using existing key agreement/exchange protocols. As the shared key k is secret for adversaries, they cannot recover the attribute identity set by a brute force attack.

4 SECURE LOGGING BASED FRAMEWORK

The successful deployment of certificate transparency framework proposed in [32, 34] indicates the feasibility of adopting a publicly auditable secure logging system. Adopting that approach, we propose our authority transparency framework built on a secure logging framework. Unlike in *certificate transparency* where the framework only needs to process relatively a smaller number of issued certificates and their revocations, our authority transparency framework faces new challenges because of the volume and variety of requirements related to managing *public parameters* and *key service*, and the associated audit obligations.

Note that compared to the volume of digital certificates managed in approaches proposed in [34, 38], our framework needs to manage a much higher volume of identity bindings. Here, we integrate the auditor, monitor, and gossiper roles together in our framework. All the clients can work together to verify the consistency of the public append-only ledger and gossip with each other to check the misbehavior of loggers; thus all the clients participant in the transparency procedure. To reduce the auditing burden, each client is only responsible for auditing the unique/attribute identity bindings and KSAO elements that are related to its associated users. From the perspective of a particular identity binding, the audit processes from clients on that identity binding are overlapping. As a result, the frequency of auditing for a particular identity binding is based on the number of users who have that identity. For instance, suppose that there are 1000 faculty members who need authority service from au_x in a university org_x. The attribute identity binding, namely, "au_x:org_x:*faculty* \$ $para_i$", will be audited 1000 times at each epoch.

4.1 Secure Public Ledger

The secure public ledger techniques such as hash chain based and Merkle tree based approaches have been studied in [34] and detailed research on formalization and security proof related to them have been provided in [16]. However, it is not easy to adopt them in our proposed authority transparency framework. In our framework, three types of audit obligations need to be recorded separately (see Section 3.4).

The public ledger in our secure logging based framework inherits the data structure proposed in CONIKS [38]. Actually, the data structure combines the hash chain and the Merkle tree. The Merkle tree is used to store all the identity bindings, while the hash chain is used to manage the roots of the tree that represent the history of identity bindings. In our proposed approach, we replace the Merkle prefix tree by *Merkle Patricia trie (MPT)* in the public ledger to accommodate the structural characteristics of the attribute information.

The MPT, adopted in Ethereum community [46], provides a cryptographically authenticated data structure that can be used to store all key-value bindings with $O^1log^1n^{00}$ efficiency for inserts and look-ups. Generally speaking, MPT integrates the characteristics of Radix tree and Merkle tree. Merkle trees provide an efficient means to prove the inclusion and absence of specific bindings, while the Radix trees support efficiently inserting and locating specified bindings in the scenario where keys have a similar prefix. Hence, it helps in adaptively managing the attribute identity bindings. For instance, the binding "au_a:org_x:*faculty* \$ $para_i$" and "au_a:org_x:*student* \$ $para_{i,1}$" have the same prefix on the key, i.e., attribute prefix "au_a:org_x:". The values $para_i$ and $para_{i,1}$ will be organized as neighbors in the MPT with similar paths.

Algorithm 1: pseudocode of stage and commit function

```
1  initialize global staging hash maps M_stag,C := ; and M_stag,T := ;;
2  function stage(S)
3      if type of S is C then return result of putting (S.v, S) into M_stag,C;
4      if type of S is T then return result of putting (S.v, S) into M_stag,T;
5  function commit(S_C^0)
6      if S_C^0.v in M_stag,C and S_C^0.v in M_stag,T then
7          S_C := M_stag,C »S_C^0.v½; S_T := M_stag,T »S_C^0.v½;
8          if S_C^0 == S_T and pair-validation(S_C, S_T) then
9              DLC.appendLog(S_C, S_T); remove S_C from M_stag,C; remove S_T from M_stag,T; return true;
10     return false;
11 function pair-validation(S_req, S_resp)
12     if S_req.id_source != S_resp.id_target or S_req.id_target != S_resp.id_source then return false;
13     if S_req.t >= S_resp.t or S_resp.t   S_req.t >= Δ_t then return false;
14     return true
```

4.2 KSAO Commitment

Note that a key service audit obligation consists of pairs of request and response key service snapshots that are generated by a *client* and the *TPA*, respectively. The *logger* cannot record a KSAO element to the public ledger in one-round of interaction; hence, we design a three-phase commit mechanism to record a complete KSAO element, as depicted in Fig. 6. Generally speaking, we design three statuses, namely: *initial*, *staging*, *commit*, for each key service snapshot[1]. The status of newly created key service snapshot is *initial* status. The status of a snapshot that has been sent to the *logger* from *clients/TPAs* will change to *staging* status. The *logger* maintains a temporary storage and manage the staging snapshots, and confirms a complete KSAO element to record into the public ledger.

Here, we present the algorithms of **stage** and **commit** as shown in Algorithm 1 that is formalized in protocol $\text{Log}^{\mathsf{T},\mathsf{L},\mathsf{C}}_{\mathsf{O}_{ks}}$ (see Section 3.4.1). It is run by the *logger* to show how to form the key service audit obligations and commit them to the public ledger. The algorithms maintain two global hash maps (namely, the staging area), take the key service snapshots they receive as input and output a result indicating whether or not the operation has succeeded. Specifically, the algorithm first initializes two empty global hash maps that are used to store the staging key service snapshots received from the client and the authority (line 1). In our design, the request and response key service snapshots have the same "key" in the key-value based staging area, where the "key" is derived from the set of attribute identities and the protocol execution indicator. The *stage* function identifies the type of key service snapshot and stores it to the corresponding staging area temporarily (lines 2-4). The *commit* function first needs to confirm that both the request and response key service snapshots are in the staging area (lines 5-7). Then it checks whether or not the received S^0_C is the same as S_T; this is to verify that the authority does provide the key service rather than just generate the key service snapshot. Finally, the *commit* function does the validation for a pair of request and response key service snapshots and adds a KSAO to the log (lines 8-9). The validation function is presented in lines 11-14 according to constraints mentioned in Definition 3.9.

Hierarchical Storage. Compared to PPAO elements, the storage for KSAO elements grows rapidly; hence, one Merkle-Patricia trie is not adequate. To store the KSAO elements, we adopt a hierarchical Merkle tree to manage

[1]The notion of *staging* comes from Git system [14]. The Git system uses staging area to store information about what will go into your next commit.

data, namely, a parent tree with several trees as its children. Specifically, a child tree only stores one user's related KSAO elements, while the leaf node of the parent tree stores the root of a child tree.

5 DISCUSSION AND EVALUATION

5.1 Security Proof of Authority Transparency

Security Guarantee. We define security for the authority transparency in terms of three properties: (i) *log-consistency*, which denotes that a dishonest log server L cannot remain undetected if it tries to present inconsistent versions of the log to the client, namely, auditor and monitor components; (ii) *unforgeable-service*, which says that a dishonest authority T cannot forge a key service by sending valid key service snapshots, but not really provide the key service to C; (iii) *non-fabrication*, which ensures that a dishonest authority T or a client C cannot blame L for misbehavior if it has behaved honestly, and a dishonest client C cannot reprove T for misbehavior if it has behaved honestly.

In general, the *log-consistency* property relies on the security properties of the primitive, namely, the *dynamic list commitment*. The *Unforgeable-service* property follows from the three-phase commits with components that include proof of key service work. The *Non-fabrication* depends on the unforgeability of the signature scheme and the security property of DLC such as *proof of non-inclusion*. Theorem 5.1 generalizes the security guarantee as follows:

Theorem 5.1. *If the primitive DLC is secure, the hash function is collision-resistant and the signature scheme is unforgeable, then the protocols in Definition 3.10 comprise a secure authority transparency.*

Analysis Methodology. Our security proof is a game simulation based reduction proof. Suppose that the adversary has non-negligible advantage ϵ to break the protocol. Then we reduce the game simulator to break the security assumption by the adversary's ability with advantage ϵ, which leads to a contradiction. As a result, the adversary does not have such a non-negligible advantage.

Proof. As mentioned in Theorem 5.1, we have the following three assumptions: (i) the primitive DLC is secure; (ii) the hash function is collision-resistant; (iii) the signature scheme is unforgeable.

The security proof of DLC primitive was presented in [16]. The collision-resistance of hash function and unforgeability of signature scheme depend on the respective security proofs of the particular schemes that are chosen in the deployment phase of the authority transparency framework. We now focus on the three security properties mentioned earlier, namely, *log-consistency*, *unforgeable-service* and *non-fabrication*. We do not analyze all protocols as shown in Fig. 5 here, as some of protocols are similar or straight-forward modifications of those presented in [16, 21]. We refer the authors to those early work for the proof sketch. Here, we only focus on the security proof of the newly proposed three-party log protocol, namely, $\mathrm{Log}_{\mathsf{O}_{ks}}^{\mathsf{C},\mathsf{L},\mathsf{T}}$.

Log-consistency. Our *authority transparency* adopts the DLC primitive from *transparency overlay*[16], and we do not change the *Inspect* and *Gossip* protocols compared to those used in the *transparency overlay* scheme. Thus, the security proof of *log-consistency* property is also similar to that provide in [16], hence, we do not present it here.

Non-fabrication. In the protocol $\mathrm{Log}_{\mathsf{O}_{ks}}^{\mathsf{C},\mathsf{L},\mathsf{T}}$, there are two possible fabrication cases:

(i) The adversary A^{C} tries to blame T by sending S_{C} to L but does not actually send the key request to T.
(ii) The adversary $\mathsf{A}^{\mathsf{f}\mathsf{C},\mathsf{T}\mathsf{g}}$ tries to blame L for misbehavior even when it has behaved honestly.

Suppose that the adversary A has the non-negligible advantage ϵ to break the *non-fabrication* security promise. To achieve the fabrication case (i), the adversary A^{C} needs to forge a fake S_{T} that includes a signature signed by the private key of T. Thus, A^{C} has the ability to forge a fake signature with advantage $\mathbf{Adv}_{\mathsf{A}^{\mathsf{C}}}^{\text{fabrication}} \quad \epsilon$. For

fabrication case (ii), as depicted in the protocol, the receipt is formalized as $^1\mathrm{pk}_{\mathsf{L}}, t, \mathrm{Sig}^1\mathrm{sk}_{\mathsf{L}}, {}^1\mathsf{S}, t^{000}$. To forge a fake receipt, $\mathsf{A}^{\{\mathsf{C},\mathsf{T}\}}$ should also have a non-negligible advantage $\mathbf{Adv}^{\mathrm{Sig}}_{\mathsf{A}^{\{\mathsf{C},\mathsf{T}\}}}$ to forge the signature.

However, it is impossible to break the security assumption (iii), namely, the unforgeability of signature scheme, according to the security assumption. Thus, $\mathsf{A}^{\{\mathsf{C},\mathsf{T}\}}$ and A^{C} do not have non-negligible advantage ϵ to frame up L and T in the protocol, respectively.

Unforgeable-service. In protocol $\mathrm{Log}^{\mathsf{C},\mathsf{L},\mathsf{T}}_{\mathsf{O}_{ks}}$, there are two possible forgeable-service challenges:

(i) Adversary A^{T} sends S_{T} to L but does not send the key sk_{attr} to C;
(ii) Adversary A^{T} sends C an invalid key sk^{0}_{attr}, but the "correct" S_{T} derived from valid is actually key sk_{attr};

Suppose that adversary A^{T} has the non-negligible advantage ϵ to break the *unforgeable-service* security promise. For challenge (i), the third phase commitment (see Fig. 6) cannot be accomplished because of a failed verification in the function $GenKSS^{0}_{\mathsf{C}}$. For challenge (ii), A^{T} has the ability to forge the proof of work component $\sigma = h_{CRHF}{}^{1\,0}$ with advantage $\mathbf{Adv}^{\text{forgeable-service},{}^1ii^0}_{\mathsf{A}^{\mathsf{T}}}$ ϵ. To achieve that, A^{T} hence needs the ability to find potential collision $\sigma = h_{CRHF}{}^{1\,0}$. According to security assumption (ii), it is impossible to break $\sigma = h_{CRHF}{}^{1\,0}$. As a result, A^{T} does not have non-negligible advantage $\mathbf{Adv}^{\text{forgeable-service},{}^1ii^0}_{\mathsf{A}^{\mathsf{T}}}$ ϵ to provide an unforgeable key service without being detected.

□

5.2 Trustworthiness and Privacy Goals

5.2.1 Trustworthiness Goals. As we mentioned before, the purpose of *authority transparency* is dealing with the trust issue in the TPA infrastructure in case of malicious insiders. Here, we present our analysis of how our proposed *authority transparency* can prevent such attacks and hence establish trust in the TPA infrastructure.

Defense against Attack 1. The attack on public parameter distribution is actually a *stealthy* targeted attack [13]. Specifically, the malicious insiders distribute parameters such as tampered attribute identity key bindings to a targeted entity without being detected, even though the targeted entity has a valid attribute identity.

In the proposed *authority transparency* approach, a TPA is required to publish its public parameters to the public ledger through a secure logging system. The public ledger supports public audit; hence, each client can request an audit proof to verify whether a specific content, e.g., attribute identity parameters that are related to its user, is in the ledger or not. In the proposed framework, each client is also designed to monitor the consistency of the ledger using a consistency proof. As the public ledger is append-only and cannot be tampered with, the client can find misbehavior or malicious activities in public parameter distribution phase by comparing received public parameters to the newest public ledger periodically. As a result, the TPA cannot distribute different parameters related to the same attribute identity to data owner and data user without being detected. So the misbehavior or malicious activities can be detected easily.

Defense against Attack 2. The attack on private key service is essentially a variant of DoS attack, called a *censorship* attack [19]. To be specific, a malicious insider in the TPA infrastructure tries to treat a part of users differently, e.g., refusing to provide key services, without being notified/detected by those users or all the users.

Note that clients and TPAs are required to send the request and response key service snapshots, respectively. The proposed mechanism ensures that a pair of key service snapshots is recorded in the public ledger. If a TPA does not fulfill its obligation of responding to key services by sending $\mathsf{S}_{\mathrm{resp}}$, there will be no corresponding response record in the public ledger for $\mathsf{S}_{\mathrm{req}}$ that is sent by the client, and hence such incomplete O_{ks} (i.e., the snapshot pairing) will be detected by audit entities. As a result, the TPA's censorship attack will be detected by audit entities in the next epoch of public ledger.

5.2.2 Privacy Goals. Defense against Attacks 3. Even though the proposed *authority transparency* approach tackles the trust issues related to the stealthy targeted attack (Attack 1) and the censorship attack (Attack 2), it

Table 1. Simulated settings

settings	symbols	data
#storage slots	m	2^{32}
#changes	n	2^{25}
#attr-identity	x	10
#collab-TPA	y	3
#epoch per day	c	24

Table 2. Client bandwidth cost based on settings in Table 1

	#hashes	transmission
audit(epoch)	$^1x \cdot y^0 log_2{}^1 m^0 \cdot 1$	1668 B
audit(day)	$c^{11}x \cdot y^0 log_2{}^1 m^0 \cdot 1^0$	39.09 KB
monitor(epoch)	$log_2{}^1 n^0 \cdot 1$	104 B
monitor(day)	$c^1 log_2{}^1 n^0 \cdot 1^0$	2.44 KB

also has privacy issue as we described in Attack 3. In the proposed *privacy-preserving authority transparency*, the cryptographic hash function ensures that the component v cannot be reversed by brute force attack because of the hash key, even though the hash input set, namely, the set of attribute identities, is available to the adversary. Thus, the key service snapshots will not disclose any privacy-sensitive information of the users.

5.3 Performance Analysis

Here, we present theoretical analysis on performance of our secure logging based framework with simulated settings.

The underlying storage structure, i.e., Merkle Prefix Tree (MPT), in our implemented secure public ledger for storing PPAO/KSAO elements is the same structure adopted in CONIKS [38], and hence we use the similar evaluation parameters as shown in Table 1, i.e., parameters m, n, c. Specifically, we suppose that the logger might support m storage slots, namely, the number of leaf nodes in the MPT, to store elements of PPAO/KSAO. Besides, we assume that there will be n changes on bindings in each epoch, and there will be c epochs for each day. Unlike the settings in the CONIKS framework, the audit objects in our proposed authority transparency framework include not only the public-key bindings as supported in CONIKS framework but also the private-key service process. Such audit objects are related to the number of attribute identities and TPAs in our experimental crypto scheme instance. Thus, we suppose that a user has x attribute identities that come from y TPAs, which decides the number of elements of PPAO/KSAO.

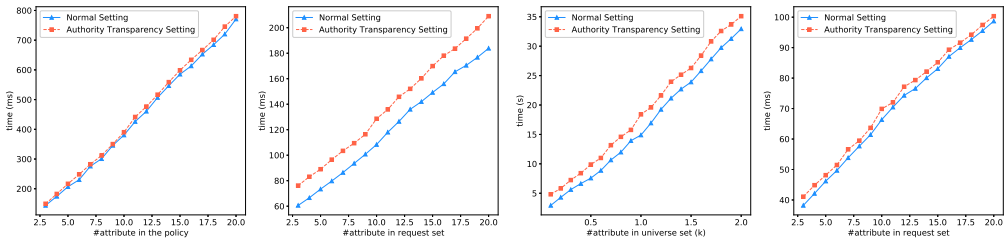
5.3.1 Auditing and monitoring bandwidth cost. We analyze the theoretical bandwidth cost for each client based on simulated settings described in Table 1. The results are presented in Table 2.

Auditing Cost. Suppose that a client needs to audit the attribute identity bindings related to its user. It might download the current root of the tree, and a proof of inclusion for the authority/attribute identities. For each identity binding, the proof of inclusion needs about $log_2{}^1 m^0$ hashes. Thus, the estimated bandwidth size for a client will be $^1x \cdot y^0 log_2{}^1 m^0 \cdot 1$ hashes for each epoch. Suppose that the length of the hash value is 32 bit, the total simulated transmission size is about 1668 bytes, as shown in Table 2.

We assume the attributes do not have the same prefix; this indicates that the leaf nodes that store identity bindings do not have the same parent or ancestor nodes in the authentication paths in the Merkle tree. However, the attribute identities of a particular user usually have the same prefix in the real scenarios, as shown in Example 3.2. Hence, the estimated bandwidth size presented in Table 1 is the upper bound of the interval.

Monitoring Cost. Suppose that a client needs to monitor the consistency of the bindings for each epoch, it might download the root of the tree and the changes in the tree for each epoch. Suppose that there are n changes in the Merkle tree. For a specified authentication path in the tree, there will be $log_2{}^1 n^0 \cdot 1$ hashes in the changed set of nodes. Thus, the estimated bandwidth size under the simulated setting will be 104 bytes in case of 32 bit hash length.

Storage Size of KSAO. Compared to the public parameters distribution operation, the private key service request is a higher frequency operation. Thus, the storage size of KSAO will grow rapidly. As we mentioned in Section



(a) encryption phase (b) decryption phase (c) initialization phase (d) key service phase

Fig. 7. The performance impacts of authority transparency setting on cryptography based access control scheme.

4, the storage used for KSAOs is a hierarchical Merkle tree, namely, each leaf node stores the root of another Merkle tree. For a specific record in KSAO, the auditing cost and monitoring cost will be doubled compared to the cost presented in Table 1.

5.3.2 Influence of Authority Transparency Setting. We implement a prototype of secure logging based framework to evaluate the influence of introducing authority transparency into cryptography based access control schemes. In particular, we compare the time cost with and without authority transparency setting for each of the phases of a specific advanced cryptosystem instance.

The prototype of our proposed secure logging based authority transparency framework is implemented using Python. The data interchange format between each entity is designed using *protocol buffers* that is Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data. This framework can be broadly deployed in heterogeneous environments as it is independent of specific crypto schemes Our framework has no dependency on any specific advanced crypto schemes. The scheme is built on the Charm framework [3] using wrappers to satisfy the interface in our authority transparency framework.

Here, we have used a typical cryptography-based access control scheme (i.e., CP-ABE [44]) as an instance for our evaluation purpose, but we note that the results are generalizable to general crypto-based access control scheme, including both CP-ABE schemes and KP-ABE schemes, where the TPA play the same role, namely, setting up the public keys and providing private key service according to the provided identity components. In secure computation schemes such as functional encryption (FE) cryptosystem, the TPA is engagged in the similar process flow and interactions. The only minor difference there is the identity components such as the attribute identity in the ABE and vector identity in the FE. Even though the method of generating public/private keys of TPA in those cryptosystems is different, the process of authority transparency audit is similar. Thus, the experimental evaluation here could be representative in those TPA-related advanced crypto schemes.

The experimental results are presented in Fig. 7. From the perspective of the client (on behalf of users), the authority transparency setting does not add significant additional time needed in the encryption phase, as shown in Fig. 7a, while it adds additional costs - on average 20 milliseconds - regardless of the number of attribute identities, as shown in Fig. 7b. From the perspective of the authority, the authority transparency setting does not increase additional time needed in the key service phase (Fig. 7d). In the system initialization phase, the authority transparency setting adds extra average cost of 2 seconds, and in the worst case, it is less than 10 seconds, as depicted in Fig. 7c. However, the initialization phase is only needed when new attribute identity sets are added, hence, it will have no significant negative influence on user experience.

6 RELATED WORK

Advanced cryptography. The emerging *advanced modern cryptography* scheme refers to one that supports both confidentiality and additional features such as access control [2, 15, 26, 37, 39] and secure computation [8–10, 12, 20, 23, 25, 45]. For instance, the variant of ABE schemes [2, 7, 39] and ACE schemes [4, 30] have enabled a public cloud to provide storage and management services for users' sensitive data. Moreover, homomorphic encryption family [8, 9, 12, 20, 25] allows one to compute arbitrary operations over outsourced encrypted data without the decryption key, while in functional encryption family [10, 23, 45], the generated private key allows one to learn a function result over a ciphertext without leaking the corresponding plaintext. Note that homomorphic encryption based secure computation approach does not need a TPA, while functional encryption based secure computation method relies on a TPA to help generate a functional private key for each function.

A TPA, as a critical component in advanced cryptography schemes, is responsible for setting up public parameters and generating private keys. The recent research on TPA infrastructure only focuses on improving efficiency by proposing multi-authority [15, 26] where several authorities are organized by hierarchical or radial architecture, and decentralized-authority [37] where any party can become a TPA and become a part of a collaborative group of TPAs. The collaborative TPAs enable or support applications where one party can share data using attribute identities from different domains and organizations. However, the trust issues caused by malicious insiders in the TPA infrastructure has not been investigated adequately.

Transparency. The concept of transparency in the digital world is used to avoid malicious activities or misbehavior in the critical infrastructures. Certificate transparency proposed in [32, 34] aims to mitigate the certificate based threats caused by fake or forged SSL certificates that are mistakenly or maliciously issued by insiders. Certificate transparency model creates an open framework for monitoring the TLS/SSL certificate system and auditing specific TLS/SSL certificates. Then, several works related to certificate transparency were proposed in [18, 21, 22, 33, 43] to deal with revocation, security and privacy issues.

Most recent and related work is *CONIKS* and *transparency overlay*. Melara et al. propose CONIKS in [38], which deals with key transparency in end-to-end encrypted communications systems where the public keys of end users are a general version of the digital certificate. Chase and Meiklejohn propose a formal framework called transparency overlay with a specific security proof in [16]. However, such transparency model and framework cannot deal with the trust and other issues we have addressed in this paper, e.g., how to ensure authorities' fulfillment of obligations in key services phases.

7 CONCLUSION

Recently proposed advanced crypto schemes show huge promise for providing user-centric, secure data management and processing in distributed environments such as clouds; however, the TPA infrastructure becomes an obstacle for their wide spread and successful deployment because of possible trust issues caused by attacks such as stealthy targeted and censorship attacks. In this paper, we have proposed a new concept of *authority transparency* as a means to address these trust issues related to the TPA component. We have proposed an *authority transparency* framework, including a formal model and an implementation approach, namely, a *secure logging based* framework. Moreover, we have analyzed the framework to show that our proposed work achieves the security goals. We also present theoretical performance evaluation and experimental results. As future direction, we plan to address practical deployment, blockchain based implementation, efficiency of each collaborative TPA, and develop ways to measure trustworthiness of the TPAs associated with various advanced crypto schemes.

REFERENCES

[1] Mohamed Hossam Afifi, Liang Zhou, Shantanu Chakrabartty, and Jian Ren. 2018. Dynamic authentication protocol using self-powered timers for passive Internet of Things. *IEEE Internet of Things Journal* 5, 4 (2018), 2927–2935.

[2] Shashank Agrawal and Melissa Chase. 2017. FAME: Fast Attribute-based Message Encryption. In *Proceedings of the ACM SIGSAC CCS*. ACM, 665–682.

[3] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. 2013. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering* 3, 2 (2013), 111–128.

[4] Christian Badertscher, Christian Matt, and Ueli Maurer. 2017. Strengthening access control encryption. In *ASIACRYPT*. Springer, 502–532.

[5] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. 2018. Design, analysis, and implementation of ARPKI: an attack-resilient public-key infrastructure. *IEEE TDSC* 15, 3 (2018), 393–408.

[6] Mihir Bellare and Sriram Keelveedhi. 2015. Interactive message-locked encryption and secure deduplication. In *PKC*. Springer, 516–538.

[7] John Bethencourt, Amit Sahai, and Brent Waters. 2007. Ciphertext-policy attribute-based encryption. In *IEEE Symposium S&P*. IEEE, 321–334.

[8] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. 2018. Threshold cryptosystems from threshold fully homomorphic encryption. In *Annual International Cryptology Conference*. Springer, 565–596.

[9] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J Wu. 2013. Private database queries using somewhat homomorphic encryption. In *International Conference on Applied Cryptography and Network Security*. Springer, 102–118.

[10] Dan Boneh, Amit Sahai, and Brent Waters. 2011. Functional encryption: Definitions and challenges. In *TCC*. Springer, 253–273.

[11] Kevin Borgolte, Tobias Fiebig, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. 2018. Cloud strife: mitigating the security risks of domain-validated certificates. In *NDSS*. Internet Society.

[12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)* 6, 3 (2014), 13.

[13] Aldo Cassola, William K Robertson, Engin Kirda, and Guevara Noubir. 2013. A Practical, Targeted, and Stealthy Attack Against WPA Enterprise Authentication.. In *NDSS*. Internet Society.

[14] Scott Chacon and Ben Straub. 2014. *Pro Git*. Apress.

[15] Melissa Chase. 2007. Multi-authority attribute based encryption. In *TCC*. Springer, 515–534.

[16] Melissa Chase and Sarah Meiklejohn. 2016. Transparency Overlays and Applications. In *Proceedings of the ACM SIGSAC CCS*. ACM, 168–179.

[17] Jing Chen, Shixiong Yao, Quan Yuan, Kun He, Shouling Ji, and Ruiying Du. 2018. CertChain: Public and Efficient Certificate Audit Based on Blockchain for TLS Connections. In *IEEE INFOCOM*. IEEE, 2060–2068.

[18] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. 2015. Efficient gossip protocols for verifying the consistency of certificate logs. In *CNS*. IEEE, 415–423.

[19] Alberto Dainotti, Claudio Squarcella, Emile Aben, Kimberly C Claffy, Marco Chiesa, Michele Russo, and Antonio Pescapé. 2011. Analysis of country-wide internet outages caused by censorship. In *Proceedings of the ACM SIGCOMM IMC*. ACM, 1–18.

[20] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*. Springer, 643–662.

[21] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. 2016. Secure logging schemes and Certificate Transparency. In *European Symposium on Research in Computer Security*. Springer, 140–158.

[22] Saba Eskandarian, Eran Messeri, Joe Bonneau, and Dan Boneh. 2017. Certificate Transparency with Privacy. *arXiv preprint arXiv:1703.02209* (2017).

[23] Ben Fisch, Dhinakaran Vinayagamurthy, Dan Boneh, and Sergey Gorbunov. 2017. Iron: functional encryption using Intel SGX. In *Proceedings of the ACM SIGSAC CCS*. ACM, 765–782.

[24] Oliver Gasser, Benjamin Hof, Max Helm, Maciej Korczynski, Ralph Holz, and Georg Carle. 2018. In Log We Trust: Revealing Poor Security Practices with Certificate Transparency Logs and Internet Measurements. In *PAM*. Springer, 173–185.

[25] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Annual Cryptology Conference*. Springer, 75–92.

[26] Nikita Gorasia, RR Srikanth, Nishant Doshi, and Jay Rupareliya. 2016. Improving Security in Multi Authority Attribute Based Encryption with Fast Decryption. *Procedia Computer Science* 79 (2016), 632–639.

[27] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. 2015. Predicate encryption for circuits from LWE. In *CRYPTO*. Springer, 503–523.

[28] The Wall Street Journal. 2017. Yahoo Triples Estimate of Breached Accounts to 3 Billion. https://www.wsj.com/articles/yahoo-triples-estimate-of-breached-accounts-to-3-billion-1507062804 Online; accessed 2018-1-19.

[29] Jonathan Katz, Amit Sahai, and Brent Waters. 2008. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*. Springer, 146–162.

[30] Sam Kim and David J Wu. 2017. Access control encryption for general policies from standard assumptions. In *ASIACRYPT*. Springer, 471–501.

[31] Deepak Kumar, Zhengping Wang, Matthew Hyder, Joseph Dickinson, Gabrielle Beck, David Adrian, Joshua Mason, Zakir Durumeric, J Alex Halderman, and Michael Bailey. 2018. Tracking certificate misissuance in the wild. In *IEEE Symposium S&P*. IEEE, 785–798.
[32] Ben Laurie. 2014. Certificate transparency. *Queue* 12, 8 (2014), 10.
[33] Ben Laurie and Emilia Kasper. 2012. Revocation transparency. *Google Research* (2012).
[34] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. *Certificate transparency*. Technical Report. IETF.
[35] Neal Leavitt. 2011. Internet security under attack: The undermining of digital certificates. *Computer* 44, 12 (2011), 17–20.
[36] Brian Neil Levine, Clay Shields, and N Boris Margolin. 2006. A survey of solutions to the sybil attack. *University of Massachusetts Amherst, Amherst, MA* 7 (2006), 224.
[37] Allison Lewko and Brent Waters. 2011. Decentralizing attribute-based encryption. In *EUROCRYPT*. Springer, 568–588.
[38] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. 2015. CONIKS: Bringing Key Transparency to End Users.. In *USENIX Security*. 383–398.
[39] Yannis Rouselakis and Brent Waters. 2013. Practical constructions and new proof methods for large universe attribute-based encryption. In *Proceedings of the ACM SIGSAC CCS*. ACM, 463–474.
[40] Mark Dermot Ryan. 2014. Enhanced Certificate Transparency and End-to-End Encrypted Mail.. In *NDSS*. Internet Society.
[41] Quirin Scheitle, Taejoong Chung, Jens Hiller, Oliver Gasser, Johannes Naab, Roland van Rijswijk-Deij, Oliver Hohlfeld, Ralph Holz, Dave Choffnes, Alan Mislove, et al. 2018. A First Look at Certification Authority Authorization (CAA). *ACM SIGCOMM Computer Communication Review* 48, 2 (2018), 10–23.
[42] Quirin Scheitle, Oliver Gasser, Theodor Nolte, Johanna Amann, Lexi Brent, Georg Carle, Ralph Holz, Thomas C Schmidt, and Matthias Wählisch. 2018. The Rise of Certificate Transparency and Its Implications on the Internet Ecosystem. In *Proceedings of the ACM SIGCOMM IMC*. ACM, 343–349.
[43] Linus Sjöström and Carl Nykvist. 2017. How Certificate Transparency Impact the Performance.
[44] Brent Waters. 2011. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *PKC*. Springer, 53–70.
[45] Brent Waters. 2012. Functional encryption for regular languages. In *CRYPTO*. Springer, 218–235.
[46] Ethereum Wiki. 2018. Merkle Patricia Trie Specification. https://github.com/ethereum/wiki/wiki/Patricia-Tree Online; accessed 2018-1-31.
[47] Jiangshan Yu, Mark Ryan, and Cas Cremers. 2018. Decim: Detecting endpoint compromise in messaging. *IEEE TIFS* 13, 1 (2018), 106–118.
[48] Liang Zhou, Sri Harsha Kondapalli, Kenji Aono, and Shantanu Chakrabartty. 2019. Desynchronization of Self-powered FN Tunneling Timers for Trust Verification of IoT Supply-chain. *IEEE Internet of Things Journal* (2019).

A DYNAMIC LIST COMMITMENT

The primitive, dynamic list commitment (DLC), adopted in our paper is proposed in [16]. Here, we present brief introduction. A DLC is a collection of the following algorithms:

c $\textsc{Commit}^1 list^0$ creates the commitment c and $0 \bullet 1$ $\textsc{CheckCommit}^1 c, list^0$ checks that c is a commitment to $list$;
c_{new} $\textsc{Append}^1 list_\Delta, c_{old}{}^0$ updates the commitment to take into account the new elements in $list_\Delta$;
π $\textsc{ProveAppend}^1 c_{old}, c_{new}, list^0$ proves that c_{new} was obtained from c_{old} by appending elements $list$ and $0 \bullet 1$ $\textsc{CheckAppend}^1 c_{old}, c_{new}, pi^0$ check this proof;
π $\textsc{ProveInList}^1 c, elmt, list^0$ proves that $elmt$ is in $list$ as represented in c and $0 \bullet 1$ $\textsc{CheckInList}^1 c, elmt, \pi^0$ checks this proof.

Note that the DLC allows someone to commit a list of elements with the following two promise: (i) the list represented as the commitment can be updated only by having new elements appended to the end of the list, and (ii) given the commitment, one can efficiently prove that the list is append-only and whether or not a given element is in the list.

B PROTOCOL OF *LOG* AND *CHECK*

The detail of protocols $\text{Log}^{\mathsf{T},\mathsf{L}}_{\mathsf{O}_{pp}}$ and $\text{Check}^{\mathsf{C}.auditor,\mathsf{L}}_{\mathsf{O}}$ are presented in Fig. 8. Note that the two protocols are derived from the the *Log* and *CheckEntry* protocol presented in *transparency overlay*. To adapt to our proposed *authority transparency* model, we modified them to satisfy the requirements of *authority transparency*.

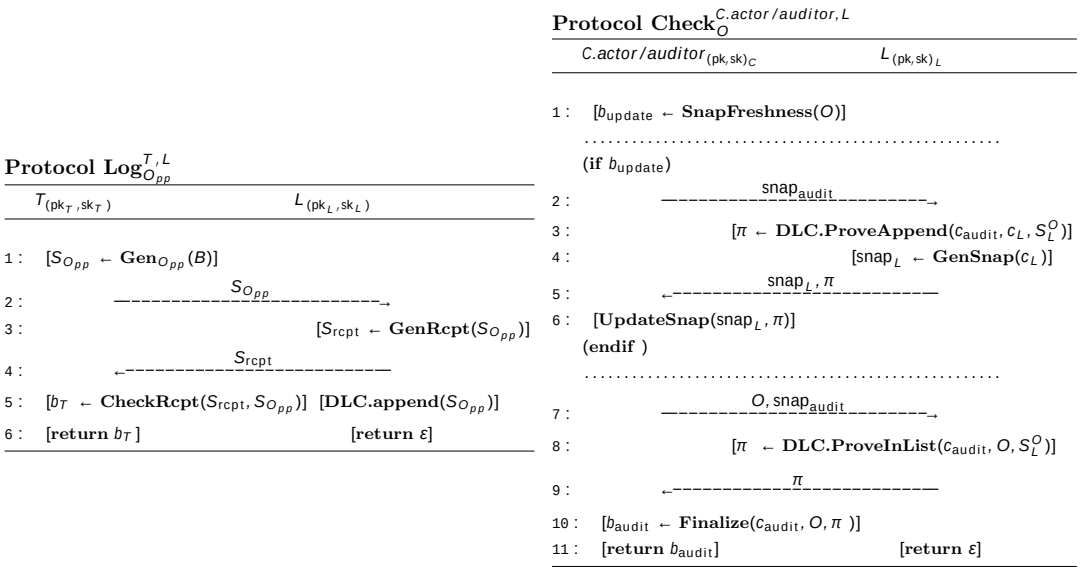


Fig. 8. The $\mathsf{Log}^{\mathsf{T},\mathsf{L}}_{\mathsf{O}_{pp}}$ protocol and the $\mathsf{Check}^{\mathsf{C}.actor\bullet auditor,\mathsf{L}}_{\mathsf{O}}$ protocol in the authority transparency.

B.1 The protocol of *Log*

$\mathsf{Log}^{\mathsf{T},\mathsf{L}}_{\mathsf{O}_{pp}}$ is an interactive protocol between T and L that is used to record *public parameter audit obligation* O_{pp} into the public log represented as a commitments DLC, as depicted in Fig. 8 left column.

The protocol of $\mathsf{Log}^{\mathsf{T},\mathsf{L}}_{\mathsf{O}_{pp}}$ is simpler than protocol $\mathsf{Log}^{\mathsf{T},\mathsf{L},\mathsf{C}}_{\mathsf{O}_{ks}}$. The authority first prepares a set of *public parameter audit obligations* and sends them to the log server (lines 1-2). Then, the log server generates the receipts for each audit obligation and sends back the receipts (lines 3-4). Finally, the authority verifies the receipts, while the log server store the audit obligations by the *append* function of the DLC primitive. The operations for each entity in the protocol $\mathsf{Log}^{\mathsf{T},\mathsf{L}}_{\mathsf{O}_{pp}}$ are presented in Fig. 9 left column.

B.2 The protocol of *Check*

$\mathsf{Check}^{\mathsf{C}.actor\bullet auditor,\mathsf{L}}_{\mathsf{O}}$ is an interactive protocol between L, $\mathsf{C}.actor$ and $\mathsf{C}.auditor$ that is used to check whether or not an audit obligation O is in the log represented as a commitment DLC. As we integrate the role of *actor* and *auditor* into one role *client*, we have two entity in the protocol, as depicted in Fig. 8 right column.

The protocol first checks the freshness of the local snapshot of the DLC commitment by comparing the timestamp between the snapshot and the receipt of audit obligation that needs to be checked (line 1). If the snapshot is out of date, the protocol updates local snapshot $\mathsf{snap}_{\mathsf{audit}}$ to a new snapshot $\mathsf{snap}_{\mathsf{L}}$ from the log server with append-only proof π (lines 2-5). Then, the client sends the audit obligation O with its snapshot $\mathsf{snap}_{\mathsf{audit}}$ to acquire the proof π^0 that proved O is in the commitment (lines 8-9). Finally, the client verify the proof π^0 to check whether or not it is a valid proof. Otherwise, the client will record O into a suspicious set that is used for gossiping with other clients. The operations for each entity in the protocol $\mathsf{Check}^{\mathsf{C}.actor\bullet auditor,\mathsf{L}}_{\mathsf{O}}$ are presented in Fig. 9 right column.

$\mathbf{Gen}_{O_{pp}}(S_B)$

$S_{O_{pp}} \leftarrow \{\ \}$
foreach B **in** S_B **do**
  $O_{pp} \leftarrow (B, \mathsf{Sig}(\mathsf{sk}_T, B))$
  $S_{O_{pp}} \leftarrow S_{O_{pp}} || O_{pp}$
endfor
return O_{pp}

GenRcpt($S_{O_{pp}}$)

$S_{\mathsf{rcpt}} \leftarrow \{\ \}$
foreach O_{pp} **in** $S_{O_{pp}}$ **do**
  $S_{\mathsf{rcpt}} \leftarrow S_{\mathsf{rcpt}} || (\mathsf{pk}_L, t, \mathsf{Sig}(\mathsf{sk}_L, (t, O_{pp})))$
endfor
return S_{rcpt}

CheckRcpt($S_{\mathsf{rcpt}}, S_{O_{pp}}$)

$b \leftarrow$ **false**
for $i = 1 \ldots$**length**(S_{rcpt}) **do**
  $\mathsf{rcpt} \leftarrow S_{\mathsf{rcpt}}[i]$
  $O_{pp} \leftarrow S_{O_{pp}}[i]$
  $b[i] \leftarrow \mathsf{Vf}(\mathsf{pk}_L, \mathsf{Sig}(\mathsf{sk}_T, (\mathsf{rcpt}[t], O_{pp})), \mathsf{rcpt}[\mathsf{Sig}])$
endfor
return b

SnapFreshness(O)

$\mathsf{rcpt} \leftarrow S_{\mathsf{rcpt}}[O]$
if $\mathsf{rcpt}[t] > t_{\mathsf{auditor}}$ **then return true**
else return false

GenSnap(c)

$t \leftarrow \mathsf{time}_{\mathsf{now}}$
return $(c, t, \mathsf{Sig}(\mathsf{sk}_L, (c, t)))$

CheckSnap(snap)

return $\mathsf{Vf}(\mathsf{pk}_L, (\mathsf{snap}[c], \mathsf{snap}[t]), \mathsf{snap}[\mathsf{Sig}])$

UpdateSnap(snap_L, π)

$b \leftarrow$ **CheckSnap**(snap_L)
$b' \leftarrow$ **DLC.CheckAppend**$(c_{\mathsf{audit}}, c_L, \pi)$
if b b' **then** $\mathsf{snap}_{\mathsf{audit}} \leftarrow \mathsf{snap}_L$
else return false

Finalize($c_{\mathsf{audit}}, O, \pi'$)

$b \leftarrow$ **DLC.CheckInList**$(c_{\mathsf{audit}}, O, \pi')$
if $b = 0$ **then**
  $S_{O,suspicious} \leftarrow S_{O,suspicious} || (O, \mathsf{rcpt})$
  return false
else
  return true

Fig. 9. The operations for each entity in the $\mathsf{Log}^{\mathsf{T},\mathsf{L}}_{\mathsf{O}_{pp}}$ protocol and $\mathsf{Check}^{\mathsf{C}.actor \bullet auditor,\mathsf{L}}_{\mathsf{O}}$ protocol.