Accountable Fine-Grained Blockchain Rewriting in the Permissionless Setting

Anonymous Author(s)

ABSTRACT

Blockchain rewriting with fine-grained access control allows a user to create a transaction associated with a set of attributes, while a modifier who possesses sufficient rewriting privileges from a trusted authority satisfying the attribute set can anonymously rewrite the transaction. However, it lacks accountability and is not designed for open blockchains that require no centralized trust authority. In this work, we introduce accountable fine-grained blockchain rewriting in a permissionless setting. The property of accountability allows the modifier's identity and their rewriting privileges to be held accountable for the modified transactions in case of malicious rewriting. Our contributions are three-fold. First, we present a generic framework for secure blockchain rewriting in the permissionless setting. Second, we present an instantiation of our framework and show its practicality through evaluation analysis. Last, we demonstrate that our proof-of-concept implementation can be effectively integrated into open blockchains.

KEYWORDS

Blockchain Rewriting, Accountability, Open Blockchains

ACM Reference Format:
Anonymous Author(s). 2022. Accountable Fine-Grained Blockchain Rewriting in the Permissionless Setting. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Blockchains have received tremendous attention from research communities and industries in recent years. The concept was first introduced in the context of Bitcoin [39], where all payment transactions are appended in a public ledger, and each transaction is ordered and verified by network nodes in a peer-to-peer manner. Blockchain ledgers grow by one block at a time, where the new block in the chain is decided by a consensus mechanism (e.g., Proof-of-Work in Bitcoin [28]) executed by the network nodes. Usually, blockchains deploy hash-chains as an append-only structure, where the hash of a block is linked to the next block in the chain. Each block includes a set of valid transactions which are accumulated into a single hash value using the Merkle tree [38], and each transaction contains certain content which needs to be registered in the blockchain.

Blockchain was originally designed to be immutable, such that the registered content cannot be modified once they are appended. However, blockchain rewriting is required in practice, or even legally necessary in data regulation laws such as GDPR in Europe [1]. Since a blockchain platform in the permissionless setting is open, it is possible some users append transactions into a chain containing illicit content such as sensitive information, stolen private keys, and inappropriate videos [36, 37]. The existence of illicit content in the chain poses a significant challenge to law enforcement agencies like Interpol [50].

Blockchain rewriting can be realized by replacing a standard hash function, used for generating transaction hash in the blockchain, by a trapdoor-based chameleon hash [31]. The users who are given the trapdoor, which we call modifiers, can modify a mutable transaction. In other words, the same mutable transaction can be modified by multiple modifiers with the same privilege. Nonetheless, for most real-life blockchain applications, blockchain rewriting with fine-grained access control is more desired. In fine-grained access control, each mutable transaction is associated with a set of attributes, and each modifier is associated with a policy representing their rewriting privilege. A mutable transaction can be potentially modified by multiple modifiers if their rewriting privileges satisfy the set of attributes associated with the transaction. Such fine-grained access control implies anonymity, meaning that modifiers cannot be identified/traced after rewriting mutable transactions.

Motivation. Blockchain rewriting with fine-grained access control has been studied in the permissioned setting [17, 49]; however, the proposed solution is not suitable for open blockchains in the permissionless setting for two reasons: 1) It requires a trusted authority to distribute rewriting privileges; however, such authority does not exist in the permissionless setting. 2) It lacks the accountability of identifying who are responsible for malicious modification of blockchain. For example, modifiers may add illicit or malicious content to mutable transactions, or delete legitimate or benign mutable transactions from blockchain. The main motivation of this work is to make fine-grained blockchain rewriting accountable in the permissionless setting.

In the permissionless setting, it is desired to achieve public accountability for fine-grained blockchain rewriting without relying on any trusted authority. Public accountability should enable any user in the public to identify responsible modifiers who have made malicious modifications directly to mutable transactions. In the case of indirect modifications, authorized modifiers may generate an access device like a blackbox by packaging their rewriting privileges and distribute it to other users, who use the access device in making malicious modifications to mutable transactions. In this case, public accountability should enable the pubic to identify both responsible users who make malicious modifications, and the responsible rewriting privileges included in the access device. This work aims to balance the anonymity and accountability of fine-grained blockchain rewritings in the permissionless setting. The modifiers stay anonymous when no modification occurs and hold accountable for the in/directly modified transactions.

Our Contributions. We introduce a new framework of accountable fine-grained blockchain rewriting in the permissionless setting. First, our framework relies on dynamic proactive secret sharing (DPSS) [35] to achieve strong security without relying on any trusted authority. We replace the trusted authority by a committee of multiple users for granting rewriting privileges, where each user holds a share of trust. We allow any user to join in and leave from a committee in any time epoch. We adapt the key-policy attribute-based encryption (KP-ABE) [43] to ensure fine-grained access control without a central authority. In particular, our adaption includes the following: 1) The master secret key in the framework is split into multiple key shares so that each user in a committee holds a single key share. 2) A certain number of shareholders in a committee can collaboratively recover the master secret key and distribute rewriting privileges to modifiers. 3) Any user can freely join/leave a committee, and the master secret key remains fixed across different committees. Our framework achieves strong security because its master secret key remains secure even if no more than a threshold number of shareholders are compromised in any committee.

Second, our framework achieves public accountability based on a novel combination of digital signature scheme, commitment scheme, and KP-ABE with public traceability (ABET for short). First, the digital signature helps the public to link a modified transaction to a modifier (or modifier's public key), as they sign the modified transaction using their signing keys, and the signed transaction is publicly verifiable. Second, the commitment scheme helps the public to link modifiers' public keys to responsible committees. Third, the ABET scheme helps the public to obtain a set of rewriting privileges from interacting with an access device in the case that an unauthorized user applies the access device in rewriting mutable transactions. Since there is no existing ABET to achieve this goal, we propose a new ABET scheme and apply it in open blockchains. The major contributions of this work are summarized as follows.

- *Generic Framework.* We introduce a new generic framework of accountable fine-grained blockchain rewriting, which is based on the chameleon hash function. A unique feature of this framework is that it allows the fine-grained blockchain rewriting to be performed in the permissionless setting.
- *Public Accountability.* We introduce a new notion called public accountability. The modifiers' public keys and their rewriting privileges are publicly held accountable for the modified transactions.
- *New Primitive.* We present a new ABET scheme, which is of independent interest. The proposed ABET scheme is the first KP-ABE scheme with public traceability designed for decentralized systems.
- *Integration to Open Blockchains.* The proof-of-concept implementation shows that blockchain rewriting based on our approach incurs almost no overhead to chain validation when compared to the immutable blockchain.

2 PRELIMINARY

In this section, we present the complexity assumptions and the key building blocks, which are used in our proposed generic construction and instantiation.

2.1 Complexity Assumptions

Bilinear Maps. Let (g, h) denote two group generators, which takes a security parameter λ as input and outputs a description of a group $\mathbb{G}, \mathbb{H}$. We define the output of (g, h) as $(q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \hat{e})$, where q is a prime number, $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$ are cyclic groups of order q, and $\hat{e} : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$ is an asymmetric bilinear map such that: (1) Bilinearity: $\forall g, h \in \mathbb{G}$ and $a, b \in \mathbb{Z}_q$, we have $\hat{e}(g^a, h^b) = e(g, h)^{ab}$; (2) Non-degeneracy: $\exists g \in \mathbb{G}$ such that $\hat{e}(g, h)$ has order q in $\mathbb{G}_T$.
We introduce a new assumption below, which is used to prove the semantic security of the proposed ABET scheme. The new assumption is proven secure in the generic group model [47].

Definition 2.1 (Extended q-type Assumption). Given group generators $g \in \mathbb{G}$ and $h \in \mathbb{H}$, define the following distribution:

$$\begin{aligned}
\mathsf{Adv}^{q'}_{\mathcal{A}} \quad = \quad & |\Pr[\mathsf{Adv}(1^\lambda, \mathsf{par}, D, T_0) = 1] \\
& - \Pr[\mathsf{Adv}(1^\lambda, \mathsf{par}, D, T_1) = 1]|, where \\
& \mathsf{par} = (q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \hat{e}, g, h) \leftarrow \mathsf{GroupGen}(1^\lambda) \\
& a, b, c, d \leftarrow \mathbb{Z}_q^*, s, \{z\} \leftarrow \mathbb{Z}_q; D = (g^a, h^b, g^c, \\
& g^{(ac)^2}, g^{abd}, g^{d/ab}, h^{abd}, h^{abcd}, h^{d/ab}, h^c, h^{cd/ab}, \\
& g^{z_i}, g^{acz_i}, g^{ac/z_i}, g^{a^2cz_i}, g^{b/z_i^2}, g^{b^2/z_i^2}, \forall i \in [q] \\
& g^{acz_i/z_j}, g^{bz_i/z_j^2}, g^{abcz_i/z_j}, g^{(ac)^2 z_i/z_j}, \forall i, j \in [q], \\
& i \neq j); T_0 = g^{abc}, T_1 = g^s.
\end{aligned}$$

The extended q-type (or q'-type) assumption is secure if $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ is negligible λ.

The detailed theorem and proof are shown in Appendix A.

2.2 Attribute-Based Encryption with Public Traceability

An attribute-based encryption with public traceability is shown as follows.

- $\mathsf{Setup}(1^\lambda)$: It takes a security parameter λ as input, outputs a master key pair $(\mathsf{msk}, \mathsf{mpk})$.
- $\mathsf{KeyGen}(\mathsf{msk}, \Lambda)$: It takes the master secret key msk, an access policy Λ as input, outputs a decryption key sk_{Λ_i}, which is associated with a unique index i. We assume an index space $\{1, \cdots, k\}$, where k denotes the maximal number of the index.
- $\mathsf{Enc}(\mathsf{mpk}, m, \delta, j)$: It takes the master public key mpk, a message m, a set of attributes $\delta \in \mathcal{U}$, and an index $j \in \{1, \cdots, k+1\}$ as input, outputs a ciphertext C. Note that C contains δ, not index j, and $\mathcal{U}$ is an attribute universe.
- $\mathsf{Dec}(\mathsf{mpk}, C, \mathsf{sk}_{\Lambda_i})$: It takes the master public key mpk, a ciphertext C, and the decryption key sk_{Λ_i} as input, outputs the message m if $1 = \Lambda_i(\delta) \wedge j \leq i$.
- $\mathsf{Trace}(\mathsf{mpk}, \mathcal{O}, \epsilon)$: It takes master public key mpk, a policy-specific decryption device $\mathcal{O}$, and a parameter $\epsilon > 0$ as input, outputs a set of indexes $\{1, \cdots, k\}$, where $\{1, \cdots, k\}$ denotes the index set

of the accused decryption keys, and ϵ denotes the lower-bound of O's decryption ability.

Public Traceability. Given a policy-specific decryption device that includes a set of decryption keys, the tracing algorithm, which treats the decryption device as an oracle, can identify the accused decryption keys that have been used in constructing the decryption device. The policy-specific decryption device is designed to decrypt ciphertexts with a specific access policy Λ. The decryption device is available to the public [13, 14, 32, 33], as malicious users may intentionally expose their decryption keys in the form of a decryption device on eBay for financial gain.

The ABET scheme requires an (encryption) index-hiding property to ensure the tracing algorithm works. Specifically, the encryptor must generate a ciphertext on a message associated with a set of attributes and a *hidden* index $j \in \{1, \cdots, k+1\}$. In other words, the generated ciphertext reveals no information about index j to any third parties. Later, the Trace algorithm will use $j \in \{1, \cdots, k+1\}$ in generating ciphertext for tracing. The informal description of the tracing process is shown below, while the Trace algorithm's formal definition is referred to [13, 14, 33]. In this work, we denote index-hiding as ciphertext anonymity. We denote policy-specific decryption device as access device because it accumulates various rewriting privileges for blockchain rewriting.

The tracing process can be described as follows: 1) the algorithm generates a ciphertext on a message under a set of attributes δ that satisfies Λ (i.e., $1 = \Lambda(\delta)$), and a hidden index from $\{1, \cdots, k+1\}$. 2) the algorithm sends the ciphertext to the decryption device and checks whether the decryption is successful. If decryption succeeds, the user outputs the accused index; otherwise, the user generates a new ciphertext under the same attribute set δ and a new index. 3) the algorithm continues this process until finding a set of accused indexes in $\{1, \cdots, k\}$.

2.3 Dynamic Proactive Secret Sharing

A dynamic proactive secret sharing DPSS consists of Share, Redistribute, and Open [8] protocols. It allows a dealer to share a secret s among a group of n_0 users such that the secret is secure against a *mobile* adversary, and allow any group of n_0-t users to recover the secret, where t denotes a threshold. The proactive security means that the execution of the protocol is divided into epochs, and a mobile adversary is allowed to corrupt users across all epochs, under the condition that no more than a threshold number of users are corrupted in any given epoch [41]. The Share and Open protocols can be realized via a secret sharing scheme (SSS) (e.g., Shamir's [46]). The Redistribute protocol prevents the mobile adversary from disclosing or destroying the secret and allows the set of the users and the threshold to change. Assuming that for each epoch i, no more than t users are corrupted, the following three properties hold:

- *Termination:* All honest users engaged in the protocol complete each execution of Share, Redistribute, and Open.
- *Correctness:* All honest users output a secret s' upon completing of Open, such that $s' = s$ if the dealer was honest during the execution of Share.
- *Secrecy:* If the dealer is honest, then s leaks no information to the adversary.

The definition described in [8] is for information-theoretically (or perfectly) secure protocols. We merely require DPSS scheme to be computationally secure in this work. Dynamic allows the set of users in a group (or committee) to be dynamically changed, which is useful in the permissionless blockchains. The Redistribute protocol has two processes: resharing the key shares to change the committee membership and threshold, updating the key shares across epochs to tackle mobile adversary.

- *Resharing the Key Shares [18].* We rely on a bivariate polynomial to share a secret s: $f(x,y) = \underline{s} + a_{0,1}x + a_{1,0}y + a_{1,1}xy + \cdots + a_{t_x,t_y}x^{t_x}y^{t_y}$, where t_x, t_y denote different thresholds. So there are two ways to share the secret s:
  (1) If we fix $y = 0$, then the key shares include $\{f(i_0,0), f(i_1,0), \cdots, f(i_{t_x},0)\}$;
  (2) If we fix $x = 0$, then the key shares include $\{f(0,j_0), f(0,j_1), \cdots, f(0,j_{t_y})\}$.

  We show how to transfer the ownership of the shareholders from committee A to committee B. First, we distribute key shares $\{f(i,y)\}$ to all users in committee A. Second, each user in committee A generates a set of temporary shares by running Shamir's SSS [46] on his own key share. In other words, his key share is the secret for SSS. Third, users in committee A send those temporary shares to users in committee B. Now, users in the committee B accumulate the received temporary shares and obtain another form of key shares $\{f(x,j)\}$ via interpolation of t_y temporary shares. To this end, the transfer between the two committees is successful. Note that either key shares $\{f(i,y)\}$ or $\{f(x,j)\}$ can be used to recover the secret s.
- *Updating the Key Shares [27].* Suppose that a bivariate polynomial is used to share the secret s: $f(x,y) = \underline{s} + a_{0,1}x + a_{1,0}y + a_{1,1}xy + a_{0,2}x^2 + a_{2,0}y^2 + a_{2,2}x^2y^2 + \cdots + a_{t_x,t_y}x^{t_x}y^{t_y}$. To update $f(x,y)$, we need another bivariate polynomial: $f'(x,y) = \underline{0} + a'_{0,1}x + a'_{1,0}y + a'_{1,1}xy + \cdots + a'_{t_x,t_y}x^{t_x}y^{t_y}$, which takes 0 as the secret. The reason is that the secret s in $f(x,y)$ will not be changed after updating by $f'(x,y)$. A crucial point is, we allow users in a new committee to collaboratively generate a polynomial $f'(x,y)$, thus the shareholders between old and new committees become independent. Note that t_x may not equal to t_y because the threshold between committees can be different, and we call it asymmetric bivariate polynomial.

2.4 Polynomial Commitments

We adapt a polynomial commitment scheme [29] to ensure it works in the asymmetric pairings.

- $\mathsf{Setup}(1^\lambda, t)$: It takes a security parameter λ and t as input, outputs a key pair $(\mathsf{msk}, \mathsf{mpk})$, where $\mathsf{msk} = \alpha$, $\mathsf{mpk} = (g, g^\alpha, \cdots, g^{\alpha^t}, h, h^\alpha, \hat{e})$.
- $\mathsf{Commit}(\mathsf{mpk}, f(x))$: It takes the public key mpk, and a polynomial $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_tx^t$ as input, outputs $C = \prod_{j=0}^{t}(g^{\alpha^j})^{a_j}$ as the commitment to $f(x)$.
- $\mathsf{CreateWitness}(\mathsf{mpk}, C, f(x))$: It takes the public key mpk, and the polynomial $f(x)$ as input, outputs a tuple $(i, f(i), w_i)$. Specifically, it computes a polynomial $\frac{f(x)-f(i)}{x-i}$ (note that the coefficients of the resulting quotient polynomial are $(\widehat{a_0}, \widehat{a_1}, \cdots, \widehat{a_t})$), and a witness $w_i = \prod_{j=0}^{t}(g^{\alpha^j})^{\widehat{a_j}}$.

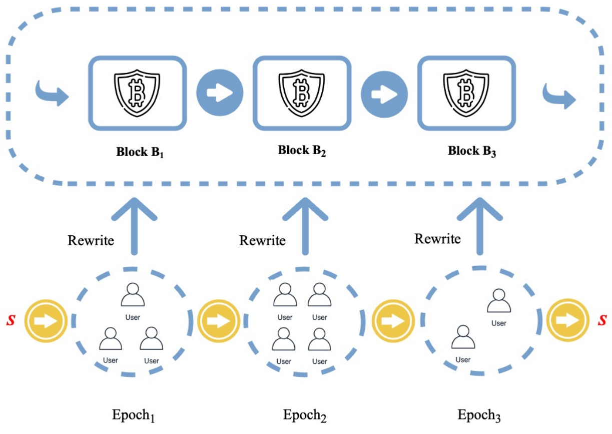

Figure 1: Blockchain rewriting with dynamic committees. Users may join in or leave from a committee, and a designated modifier in a committee can rewrite the blockchain. The secret s remains fixed across different committees.

- $\mathsf{VerifyEval}(\mathsf{mpk}, C, i, f(i), w_i)$: It takes the public key mpk, a commitment C, and the tuple $(i, f(i), w_i)$ as input, outputs 1 if $\hat{e}(C/g^{f(i)}, h) = \hat{e}(w_i, h^{\alpha}/h^{i})$.

The witness w_i proves that $f(i)$ is a correct evaluation at $i \in \mathbb{Z}_q$, without revealing the polynomial $f(x)$. If the KZG commitment scheme is used in DPSS, the committee members can be held accountable in a committee. In particular, the KZG commitment scheme is publicly verifiable if we append commitments and witnesses to the blockchain. The appended commitments and witnesses can be confirmed in the blockchain using Proof-of-Work (PoW) consensus (it is not difficult to extend this assumption to other consensus like Proof of Stake [3], Proof of Space [22]).

3 THE PROPOSED CONSTRUCTION

In this section, we present the system model for accountable fine-grained blockchain rewriting in the permissionless setting. Next, we present the definition and the generic construction, respectively.

3.1 System Model

The system model involves three types of entities: user, modifier, and miner, in which the entities can intersect, such as a user can be a modifier and/or a miner. The communication model considers both on-chain and off-chain settings. The on-chain setting is the permissionless blockchain, where *read* is public, but *write* is granted to anyone who can show PoW. The off-chain setting assumes that every user has a point-to-point (P2P) channel with every other users. One may use Tor or transaction ghosting to establish a P2P channel [35]. Such P2P channel works in a synchronous model, i.e., any message sent via this channel is received within a known bounded time-period.

The system proceeds in fixed time periods called epochs. In the first epoch, a committee election protocol (e.g., Algorand's *BA∗* protocol [26], or other methods [34, 54]) is executed, so that a set of users can agree on an initial committee with Byzantine fault tolerance (e.g., up to 1/3 malicious members). The secret s in the initial committee can be generated by an honest user (e.g., committee leader) or in a distributed fashion [25]. The secret s is shared among the committee members. Similarly, the setup of the commitment scheme can be performed by an honest user in the initial committee.

In Figure 1, a blockchain is generated by users who append their hashed contents to the blockchain. Later, modifiers with sufficient rewriting privileges are required to rewrite the hashed contents. We stress that the link of hash-chain remains intact after rewriting, and the secret remains fixed across different committees. We assume at most n users (i.e., protocol participants) exist in each epoch. We consider k dynamic committees, each of which has a varying number of committee members, and we denote $n_0 \leq n$ as a committee's size. The parameters n and k are independent. We also consider dynamic churn (i.e., join/leave) of the protocol participants. In particular, we do not assume that k committees exist in each different epoch (or we allow several committees to exist in the same epoch).

Remark. To prevent a malicious user from controlling a committee by launching Sybil attacks [20], we rely on the PoW-based identity generation mechanism [6, 34]. The mechanism allows all users to establish their identities in a committee, yet limiting the number of Sybil identities created by a malicious user. In Elastico [34], each user locally generates/establishes an identity consisting of a public key, an IP address, and a PoW solution. The user must solve a PoW puzzle which has publicly verifiable solutions to generate the final component of the identity. A PoW solution also allows other committee members to verify and accept the identity of a user. Because solving PoW requires computation, the number of identities that the malicious user can create is limited by a fraction of malicious computational power. One can refer to [34, 53, 54] for the detailed discussion on Byzantine fault resiliency.

3.2 Definition

An accountable fine-grained chameleon hash with dynamic committees consists of the following algorithms.

- $\mathsf{Setup}(1^{\lambda})$: It takes a security parameter λ as input, outputs a master key pair $(\mathsf{msk}, \mathsf{mpk})$. Note that msk is securely distributed to an initial committee C_0.
- $\mathsf{KeyGen}(\mathrm{C}_i, \Lambda)$: It takes a committee C_i, and a policy Λ as input, outputs a secret key sk_{Λ_i}. The committee index $i \in \{1, \cdots, k\}$, where k denotes the total number of committees.
- $\mathsf{Hash}(\mathsf{mpk}, m, \delta, j)$: It takes the master public key mpk, a message $m \in \mathcal{M}$, a set of attributes $\delta \in \mathcal{U}$, and an index $j \in \{1, \cdots, k+1\}$ as input, outputs a chameleon hash h, a randomness r, and a signature σ. Note that $\mathcal{M} = \{0,1\}^*$ denotes a general message space.
- $\mathsf{Verify}(\mathsf{mpk}, h, m, r, \sigma)$: It takes the master public key mpk, chameleon hash h, message m, randomness r, signature σ as input, output a bit $b \in \{0, 1\}$.
- $\mathsf{Adapt}(\mathsf{sk}_{\Lambda_i}, h, m, m', r, \sigma)$: It takes the secret key sk_{Λ_i}, chameleon hash h, messages m and m', randomness r, and signature σ as input, outputs r' and σ' if $1 = \Lambda(\delta)$ and $j \leq i$.
- $\mathsf{Judge}(\mathsf{mpk}, T, T', \mathcal{O})$: It takes the master public key mpk, two transactions (T, T'), and an access device $\mathcal{O}$ that involved users' secret keys as input, outputs a transaction-committee pair (T', C_i),

where $T' = (h, m', r', \sigma')$. It means a user with a secret key from committee C_i has modified transaction $T = (h, m, r, \sigma)$.

Correctness. The *correctness* is held if: 1) For all λ, for all $\delta \in \mathcal{U}$, all keys $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(1^\lambda)$, for all $\delta \in \Lambda$, for all $j \leq i$, for all $\mathsf{sk}_{\Lambda_i} \leftarrow \mathsf{KeyGen}(\mathsf{C}_i, \Lambda)$, for all $m \in \mathcal{M}$, for all $(h, r, \sigma) \leftarrow \mathsf{Hash}(\mathsf{mpk}, m, \delta, j)$, for all $m' \in \mathcal{M}$, for all $(r', \sigma') \leftarrow \mathsf{Adapt}(\mathsf{sk}_{\Lambda_i}, m, m', h, r, \sigma)$, we have $1 = \mathsf{Verify}(\mathsf{mpk}, h, m, r, \sigma) = \mathsf{Verify}(\mathsf{mpk}, h, m', r', \sigma')$. 2) The modified transaction is linked to a user and a committee $(T', \mathsf{C}_i) \leftarrow \mathsf{Judge}(\mathsf{mpk}, T, T', O)$.

3.3 Security Models

We consider three security guarantees, including indistinguishability, adaptive collision-resistance, and accountability.

Indistinguishability. Informally, an adversary cannot decide whether for a chameleon hash its randomness was freshly generated using Hash algorithm or was created using Adapt algorithm even if the secret key is known. We define a formal experiment between an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$ in Figure 2. The security experiment allows $\mathcal{A}$ to access a HashOrAdapt oracle, which ensures that the randomness does not reveal whether it was obtained from Hash or Adapt algorithm. The hashed messages are chosen from the same message space $\mathcal{M}$ by $\mathcal{A}$. During setup, $\mathcal{S}$ distributes msk to an initial committee. The initial committee can be updated to multiple committees while msk is fixed.

We require $1 = \Lambda(\delta)$ and $\mathsf{Verify}(\mathsf{mpk}, m', h_0, r_0, \sigma_0) = \mathsf{Verify}(\mathsf{mpk}, m, h_1, r_1, \sigma_1) = 1$. We define the advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{IND}}_{\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{IND}}_{\mathcal{A}}(1^\lambda) \rightarrow 1] - 1/2|.$$

Definition 3.1. The proposed generic framework is indistinguishable if for any probabilistic polynomial-time (PPT) $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IND}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

Adaptive Collision-Resistance. Informally, a mobile adversary can find collisions for a chameleon hash if she possesses a secret key satisfies an attribute set associated with the chameleon hash (this condition is modelled by KeyGen' oracle). We define a formal experiment in Figure 3. We allow $\mathcal{A}$ to see collisions for arbitrary access policies (i.e., KeyGen" and Adapt' oracles). We also allow $\mathcal{A}$ to corrupt a threshold number of shareholders (i.e., Corrupt oracle) in any committee.

In this experiment, $\{f(x, y)_i\}$ means key shares $\{f(x, y)\}$ at i-th committee. The key shares can be securely transferred between committee members while msk is fixed. We define the advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{ACR}}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{\mathsf{ACR}}_{\mathcal{A}}(1^\lambda) \rightarrow 1].$$

Definition 3.2. The proposed generic framework is adaptively collision-resistant if for any PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ACR}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

Accountability. Informally, an adversary cannot generate a bogus message-signature pair for a chameleon hash that points to a user and an accused committee. The adversary wins if the user has obtained a secret key from the accused committee but never modified the transactions. We define a formal experiment in Figure 4. We allow $\mathcal{A}$ to see whether a modified transaction links to a committee (i.e., Judge' oracle). Let set Q record the modified transactions and the committees produced by the Judge' oracle.

```
Experiment $\mathsf{Exp}^{\mathsf{IND}}_{\mathcal{A}}(\lambda)$
$(\mathsf{C}_i, \mathsf{mpk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $b \leftarrow \{0, 1\}$
$b^* \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt}(\mathsf{C}_i, \cdots, b)}(\mathsf{msk})$
  where $\mathsf{HashOrAdapt}(\mathsf{C}_i, \cdots, b)$ on input $m, m', \delta, \Lambda, j$ :
    $\mathsf{sk}_{\Lambda_i} \leftarrow \mathsf{KeyGen}(\mathsf{C}_i, \Lambda)$
    $(h_0, r_0, \sigma_0) \leftarrow \mathsf{Hash}(\mathsf{mpk}, m', \delta, j)$
    $(h_1, r'_1, \sigma'_1) \leftarrow \mathsf{Hash}(\mathsf{mpk}, m, \delta, j)$
    $(r_1, \sigma_1) \leftarrow \mathsf{Adapt}(\mathsf{sk}_{\Lambda_i}, m, m', h_1, r'_1, \sigma'_1)$
    return $\perp$ if $r_0 = \perp \vee r_1 = \perp$
    return $(h_b, r_b, \sigma_b)$
return 1, if $b^* = b$; else, return 0.
```

Figure 2: Indistinguishability.

```
Experiment $\mathsf{Exp}^{\mathsf{ACR}}_{\mathcal{A}}(\lambda)$
$(\mathsf{C}_i, \mathsf{mpk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $Q_1, Q_2, Q_3 \leftarrow \emptyset$
$(m^*, r^*, m^{*'}, r^{*'}, h^*, \sigma^*, \sigma^{*'}) \leftarrow \mathcal{A}^{O}(\mathsf{mpk})$
  where $O \leftarrow \{\mathsf{KeyGen}', \mathsf{KeyGen}'', \mathsf{Hash}', \mathsf{Adapt}', \mathsf{Corrupt}\}$
  and $\mathsf{KeyGen}'(\mathsf{C}_i, \cdot)$ on input $\Lambda$ :
    $\mathsf{sk}_{\Lambda_i} \leftarrow \mathsf{KeyGen}(\mathsf{C}_i, \Lambda)$
    $Q_1 \leftarrow Q_1 \cup \{\Lambda\}$
    return $\mathsf{sk}_{\Lambda_i}$
  and $\mathsf{KeyGen}''(\mathsf{C}_i, \cdot)$ on input $\Lambda$ :
    $\mathsf{sk}_{\Lambda_i} \leftarrow \mathsf{KeyGen}(\mathsf{C}_i, \Lambda)$
    $Q_2 \cup \{(i, \mathsf{sk}_{\Lambda_i})\}$
    $i \leftarrow i + 1$
  and $\mathsf{Hash}'(\mathsf{mpk}, \cdots)$ on input $m, \delta, j$ :
    $(h, r, \sigma) \leftarrow \mathsf{Hash}(\mathsf{mpk}, m, \delta, j)$
    $Q_3 \leftarrow Q_3 \cup \{(h, m, \delta)\}$
    return $(h, r, \sigma)$
  and $\mathsf{Adapt}'(\mathsf{mpk}, \cdots)$ on input $m, m', h, r, i, \sigma$ :
    return $\perp$, if $(i, \mathsf{sk}_{\Lambda_i}) \notin Q_2$ for some $\mathsf{sk}_{\Lambda_i}$
    $(r', \sigma') \leftarrow \mathsf{Adapt}(\mathsf{sk}_{\Lambda_i}, m, m', h, r, \sigma)$
    if $(h, m, \delta) \in Q_3$ for some $\delta$,
    let $Q_3 \leftarrow Q_3 \cup \{(h, m', \delta)\}$
    return $(r', \sigma')$
  and $\mathsf{Corrupt}(\mathsf{mpk}, \cdot)$ on input $\mathsf{C}_i$ :
    return $\{f(x, y)_i\}$
return 1, if
$1 = \mathsf{Verify}(\mathsf{mpk}, m^*, h^*, r^*, \sigma^*) = \mathsf{Verify}(\mathsf{mpk}, m^{*'}, h^*, r^{*'}, \sigma^{*'})$
$\wedge (h^*, \cdot, \delta) \in Q_3$, for some $\delta \wedge m^* \neq m^{*'}$
$\wedge \delta \cap Q_1 = \emptyset \wedge (h^*, m^*, \cdot) \notin Q_3$
else, return 0.
```

Figure 3: Adaptive Collision-Resistance.

```
Experiment $\mathsf{Exp}^{\mathsf{ACT}}_{\mathcal{A}}(\lambda)$
$(\mathsf{C}_i, \mathsf{mpk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $Q \leftarrow \emptyset$
$(T^*, \mathsf{C}_i) \leftarrow \mathcal{A}^{\mathsf{Judge}'(\mathsf{mpk}, \cdots)}(\mathsf{mpk})$
  where $\mathsf{Judge}'(\mathsf{mpk}, \cdots)$ on input $T, \Lambda, m'$ :
    $\mathsf{sk}_{\Lambda_i} \leftarrow \mathsf{KeyGen}(\mathsf{C}_i, \Lambda)$
    $(r', \sigma') \leftarrow \mathsf{Adapt}(\mathsf{sk}_{\Lambda_i}, T, m')$
    $(T', \mathsf{C}_i) \leftarrow \mathsf{Judge}(\mathsf{mpk}, T, T', Q)$
    let $Q \leftarrow Q \cup \{(T', \mathsf{C}_i)\}$
    return $(T', \mathsf{C}_i)$
return 1, if $(\cdot, \mathsf{C}_i) \in Q \wedge (T^*, \cdot) \notin Q$
else, return 0.
```

Figure 4: Accountability.

We denote $T = (h, m, r, \sigma)$ and $T' = (h, m', r', \sigma')$ as original and modified transactions with respect to chameleon hash h. We also denote the linked transaction-committee pair as (T', C_i). In this experiment, $T^* = (h^*, m^*, r^*, \sigma^*)$. We define the advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{ACT}}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{\mathsf{ACT}}_{\mathcal{A}}(1^\lambda) \rightarrow 1].$$

Definition 3.3. The proposed generic framework is accountable if for any PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ACT}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

3.4 Generic Construction

The proposed generic construction consists of the following building blocks.

- A chameleon hash scheme CH = (Setup, KeyGen, Hash, Verify, Adapt).
- An attribute-based encryption scheme with public traceability ABET = (Setup, KeyGen, Enc, Dec, Trace).
- A dynamic proactive secret sharing scheme DPSS = (Share, Redistribute, Open).
- A digital signature scheme Σ = (Setup, KeyGen, Sign, Verify).

High-level Description. We assume that every user has a key pair $(\mathsf{sk}, \mathsf{pk})$ and that users' public keys are known to all users in a committee. Each modifier possesses a set of attributes, and more than a threshold number of users in a committee can collectively grant a rewriting privilege to a modifier based on their attribute set. For example, a user with pk creates a transaction T, including a chameleon hash, a ciphertext that is labeled with a set of attributes, and a signature (i.e., signs T using his secret key sk). A modifier with pk' who is granted the rewriting privilege from a committee, can rewrite the transaction T. During rewriting, the modifier signs the modified transaction using her secret key sk'. We assume the t-out-of-n_0 DPSS scheme to be executed over off-chain P2P channels and let all k committees have the same parameters (t, n_0). The proposed construction is shown below.

- $\mathsf{Setup}(1^\lambda)$: A user takes a security parameter λ as input, outputs a public parameter $\mathsf{PP} = (\mathsf{mpk}_{\mathsf{ABET}}, \mathsf{PP}_\Sigma, \mathsf{PP}_{\mathsf{CH}})$, and a secret key $\mathsf{msk}_{\mathsf{ABET}}$, where $(\mathsf{msk}_{\mathsf{ABET}}, \mathsf{mpk}_{\mathsf{ABET}}) \leftarrow \mathsf{Setup}_{\mathsf{ABET}}(1^\lambda)$, $\mathsf{PP}_\Sigma \leftarrow \mathsf{Setup}_\Sigma(1^\lambda)$, $\mathsf{PP}_{\mathsf{CH}} \leftarrow \mathsf{Setup}_{\mathsf{CH}}(1^\lambda)$. The key shares $\{f(x,y)_0\} \leftarrow \mathsf{Share}_{\mathsf{DPSS}}(\mathsf{msk}_{\mathsf{ABET}})$ are distributed to users within committee C_0, where each user holds a key share. Besides, each user holds a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}_\Sigma(\mathsf{PP}_\Sigma)$.
- $\mathsf{KeyGen}(\mathsf{C}_i, \Lambda)$: A group of t+1 users in committee C_i take their key shares $\{f(x,y)_i\}^{t+1}$ and a policy Λ as input, output a secret key sk_{Λ_i} for a modifier, where $\mathsf{sk}_{\Lambda_i} \leftarrow \mathsf{KeyGen}_{\mathsf{ABET}}(\mathsf{msk}_{\mathsf{ABET}}, \Lambda)$, $\mathsf{msk}_{\mathsf{ABET}} \leftarrow \mathsf{Open}_{\mathsf{DPSS}}(\{f(x,y)_i\}^{t+1})$, and key shares $\{f(x,y)_i\} \leftarrow \mathsf{Redistribute}_{\mathsf{DPSS}}(\{f(x,y)_{i-1}\})$. The modifier can be either one of committee members or an external party.
- $\mathsf{Hash}(\mathsf{PP}, m, \delta, j)$: A user appends a message m, a set of attributes δ, and an index j to the blockchain, performs the following operations
  1. generate a chameleon hash $(h_{\mathsf{CH}}, \mathsf{r}) \leftarrow \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m)$, where $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \leftarrow \mathsf{KeyGen}_{\mathsf{CH}}(\mathsf{PP}_{\mathsf{CH}})$.
  2. generate a ciphertext $C \leftarrow \mathsf{Enc}_{\mathsf{ABET}}(\mathsf{mpk}_{\mathsf{ABET}}, \mathsf{sk}_{\mathsf{CH}}, \delta, j)$, where $\mathsf{sk}_{\mathsf{CH}}$ denotes the encrypted message.
  3. generate a message-signature pair (c, σ_Σ), where $\sigma_\Sigma \leftarrow \mathsf{Sign}_\Sigma(\mathsf{sk}, c)$, and message c is derived from sk and $\mathsf{sk}_{\mathsf{CH}}$.
  4. output $(h, m, \mathsf{r}, \sigma)$, where $h \leftarrow (h_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}, C)$, and $\sigma \leftarrow (c, \sigma_\Sigma)$.
- $\mathsf{Verify}(\mathsf{PP}, h, m, \mathsf{r}, \sigma)$: It outputs 1 if $1 \leftarrow \mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, h_{\mathsf{CH}}, \mathsf{r})$ and $1 \leftarrow \mathsf{Verify}_\Sigma(\mathsf{pk}, c, \sigma_\Sigma)$, and 0 otherwise.
- $\mathsf{Adapt}(\mathsf{sk}_{\Lambda_i}, h, m, m', \mathsf{r}, \sigma)$: A modifier with a secret key sk_{Λ_i} and a new message m', performs the following operations
  1. check $1 \overset{?}{=} \mathsf{Verify}(\mathsf{PP}, h, m, \mathsf{r}, \sigma)$.
  2. compute $\mathsf{sk}_{\mathsf{CH}} \leftarrow \mathsf{Dec}_{\mathsf{ABET}}(\mathsf{mpk}_{\mathsf{ABET}}, C, \mathsf{sk}_{\Lambda_i})$.
  3. compute a new randomness $\mathsf{r}' \leftarrow \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', h, \mathsf{r})$.
  4. generate a ciphertext $C' \leftarrow \mathsf{Enc}_{\mathsf{ABET}}(\mathsf{mpk}_{\mathsf{ABET}}, \mathsf{sk}_{\mathsf{CH}}, \delta, j)$.
  5. generate a message-signature pair (c', σ'_Σ), where c' is derived from sk' and $\mathsf{sk}_{\mathsf{CH}}$.
  6. output $(h, m', \mathsf{r}', \sigma')$, where $h \leftarrow (h_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}, C')$, and $\sigma' \leftarrow (c', \sigma'_\Sigma)$.
- $\mathsf{Judge}(\mathsf{PP}, T, T', O)$: It takes the public parameter PP, two transactions (T, T'), and an access device O as input, outputs a transaction-committee pair (T', C_i) if the modified transaction T' links to a committee C_i, where $T' = (h, m', \mathsf{r}', \sigma')$.

Correctness and Security. The Judge algorithm allows any public user to identify the responsible modifiers and their rewriting privileges given a modified transaction. The modified transaction can be easily linked to the modifier (or modifier's public key) because a digital signature scheme is used in the construction. Below, we explain the connection between the modified transaction and the responsible rewriting privileges (or committee indexes).

First, any public user verifies a connection between a transaction T and its modified version T'. The connection can be established, since both message-signature pair (c, σ_Σ) in T and message-signature pair (c', σ'_Σ) in T', are derived from the same chameleon trapdoor $\mathsf{sk}_{\mathsf{CH}}$. We require the underlying Σ scheme (e.g., Schnorr signature [44] and Waters signature [51]) to have homomorphic property regarding keys and signatures. We rely on this homomorphic property to find the connection between a transaction and its modified versions. The chameleon trapdoor $\mathsf{sk}_{\mathsf{CH}}$ is used in many modified versions of a mutable transaction because different modifiers may modify the same transaction. Here, we consider a single modified transaction T' for simplicity.

Second, any public user obtains a set of accused committees from interacting with an access device O, such that $\{1, \cdots, k\} \leftarrow \mathsf{Trace}_{\mathsf{ABET}}(\mathsf{mpk}_{\mathsf{ABET}}, O, \epsilon)$. Specifically, the public sends a ciphertext encrypted a message under a set of attributes (which satisfies the access policy involved in O) and a committee index from $\{1, \cdots, k+1\}$ to O. The public outputs the committee index (we call it accused committee) if decryption succeeds. The public repeats this tracing procedure until output a set of accused committees.

Third, if a user with pk' acts as a modifier in an accused committee, the public outputs (T', C_i). It means that a transaction T' is indeed modified by the user pk' whose rewriting privilege is granted from committee C_i. Because we allow the commitment scheme to be used in DPSS, the user pk' is held accountable in a committee. More specifically, user pk' joins in committee C_i by showing a commitment on her key share to other committee members, and further detail is given in the instantiation. If user pk' acts as modifiers for many accused committees, the public outputs $(T', \{\mathsf{C}_i\})$. However, if user pk' did not join in any accused committees, the public still outputs the indexes of the accused committees. This is the second case of blockchain rewriting: an unauthorized user has no granted rewriting privileges from any committee but uses an access device to rewrite mutable transactions.

To conclude, we achieve public accountability via three steps: 1) Verify a modified transaction; 2) Find an accused committee; 3) Link the modified transaction to the accused committee. We also consider that a committee may have multiple modifiers equipped with different rewriting privileges, but they should have the same

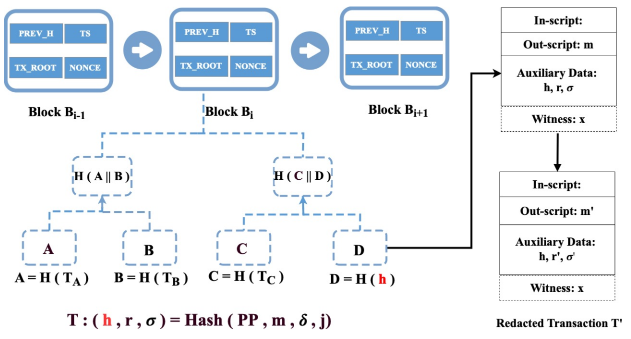

Figure 5: Application in an open blockchain.

committee index. In this case, the public still identifies the responsible modifiers in the same committee as the modifiers are required to sign the modified transactions. The security result of our proposed construction is referred to Appendix B.

3.5 Application in Open Blockchains

In this sub-section, we show how to apply our proposed solution to open blockchains, including the idea of redactable blockchain in the permissionless setting, the structure of a mutable transaction (see Figure 5), and the modification process.

- *Basic Idea.* In an open blockchain, each block stores a compact representation of a set of transactions. The root hash of a Merkle tree (i.e., *TX_ROOT*) accumulates all transactions associated with a block. If a user appends a mutable transaction T to the blockchain, the message m must be hashed via $(h, m, \mathsf{r}, \sigma) = \mathsf{Hash}(\mathsf{PP}, m, \delta, j)$, where $h = (h_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}, C)$. The ciphertext C contains an attribute set δ, which is chosen by the user who created T. The other immutable transactions are processed using collision-resistant hash function H such as SHA-256. If the transaction T needs to be modified, a modifier with a secret key sk_{Λ_i} (issued by a committee C_i) satisfying δ can replace the message-randomness pair (m, r) by (m', r'). The modifier will broadcast (m', r'), an identifier *id* and a signature σ' to the blockchain network. All system users are supposed to verify the correctness of (m', r') and σ', and update their local copy of the blockchain accordingly. Note that the identifier *id* will help system users to identify which transaction needs to be updated, the signature σ' will help the public to identify the responsible modifier.
- *Structure.* Compared to the immutable transaction, each mutable transaction should contain an auxiliary data. The auxiliary data stores (h, r, σ), and it will be updated after each modification. If a miner receives a mutable transaction, he needs to verify the transaction (i.e., the Verify algorithm) and witness (i.e., a signature computed on transaction). If they are valid, the miner will take the chameleon hash value h_{CH} as a leaf node of Merkle tree.
- *Modification Process.* We assume the message m as illicit content, and use a mutable transaction T containing m to explain the modification process. Since the message m is hashed under $\mathsf{mpk}_{\mathsf{ABET}}$, it can be modified by the key holder $\mathsf{msk}_{\mathsf{ABET}}$. But, $\mathsf{msk}_{\mathsf{ABET}}$ was distributed in a committee using the DPSS protocol and a threshold number of committee members will grant various rewriting privileges to different modifiers based on certain access policies. Thus, a modifier with sufficient rewriting privilege can redact transaction T by replacing (m, r) with (m', r'). After modification, the modifier broadcasts (m', r'), *id* and σ' in the blockchain network, all users will perform the following checks.
  – whether the message m needs to be modified as m' according to the rule, like GDPR.
  – whether the message-randomness pairs (m, r) and (m, r') are mapping to the same chameleon hash value h_{CH}.
  – whether the signature $\sigma' = (c', \sigma'_{\Sigma})$ is valid under a public key pk' and a message c'.

  If all the above requirements are met, all users will update their local copy of the blockchain by replacing (m, r) with (m', r'). Also, we consider a scenario where the modified transaction T' contains illicit message. This is because the modifier may refuse to redact/delete message m or choose a different illicit message m' to rewrite T. In this case, we let future modifiers redact/delete illicit message. One may consider a revocation mechanism to prevent the malicious modifiers from obtaining rewrite privileges in future committees. Our proposed solution only supports traceability. Any user can trace the responsible modifiers and their rewriting privileges given a modified transaction containing legal or illicit content. A traceable and revocable solution for open blockchains can be a potential future work.

  Now, we present our conclusions. First, the proposed solution incurs no overhead to chain validation. This is because, rewrite the message in T has no effect on the consensus mechanism, as the chameleon hash output h is used for computing the transaction hash for Merkle tree leaves. Second, the proposed solution is compatible with existing blockchain systems. The only change which CH-based blockchain rewriting requires is to replace the standard the hash function H by CH for generating a chameleon hash value h before validating the transactions in each block. Besides, DPSS is designed for open blockchains, and decentralized systems [35]. Third, we only allow blockchain rewriting that does not affect a transaction's past and future events. If a modifier removes a transaction entirely or changes spendable data of a transaction, it may lead to serious transaction inconsistencies in the chain [19]. If any committee member rewrites blockchain maliciously, their public keys are held accountable to the modified transactions due to the signature scheme is used in the construction. Last, our proof-of-concept implementation [2] indicates that the proposed solution can act as an additional layer on top of any open blockchains to perform accountable and fine-grained rewritings. Specifically, we append mutable transactions using the proposed solution to the blockchain, and we allow dynamic committees to grant rewriting privileges for rewriting those mutable transactions.

4 INSTANTIATION AND IMPLEMENTATION

In this section, we explain the proposed ABET scheme first. Then, we show the proposed instantiation, give the implementation and evaluation analysis.

4.1 The Proposed ABET Scheme

For constructing a practical ABET, we require that the underlying KP-ABE scheme should have a minimal number of elements in master secret key, while the size of the ciphertext is constant (i.e., independent of the number of committees). Therefore, we rely on a KP-ABE scheme [43] and a hierarchy identity-based encryption (HIBE) scheme [12] to construct our ABET scheme, respectively. First, the KP-ABE [43] can be viewed as the stepping stone to construct ABET. Its master secret key has a single element, which requires a single execution of the DPSS. Its security is based on q-type assumption in the standard model, and it works in prime-order group. One may use more efficient ABE schemes such as [4]. However, their master secret key includes several elements, which requires several executions of the DPSS. Second, the HIBE scheme [12] has constant-size ciphertext. Specifically, the ciphertext has just three group elements, and the decryption requires only two pairing operations. In particular, the HIBE's master secret key has one element, which can be shared with KP-ABE. We note that several ABET schemes have been proposed in [32, 33, 40, 49], but they are cipher-policy ABE based. They cannot be applied to open blockchains with decentralized access control, such that a committee of multiple users share a common secret and manage access control. Our proposed ABET scheme makes it possible based on the KP-ABE scheme.

The ABET scheme described above lacks anonymity because its ciphertext reveals committee's index information to the public. As a result, the index-hiding required in the ABET scheme cannot be held (see Section 2.2). We realize the ABET scheme with ciphertext anonymity using asymmetric pairings, i.e., $\hat{e} : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$ (which is used in [21]). The basic idea is, the index-based elements in a modifier's decryption key belong to group $\mathbb{G}$. The index-based elements in a ciphertext belong to group $\mathbb{H}$ so that the ciphertext can hide the committee's index if the master secret key is unknown. The proposed ABET scheme can be found in the instantiation. The security analysis is referred to Appendix C.

4.2 Instantiation

First, we modify the discrete logarithm (DL)-based chameleon hash [31] and apply it to our instantiation. Specifically, we modify the hash process as $h = g^r \cdot \mathsf{pk}^m$, and the adapt process becomes $r' = r + (m - m') \cdot \mathsf{sk}$, where $\mathsf{pk} = g^{\mathsf{sk}}$. The adapt process will have a better performance compared to [31] because the cost of multiplication is lower than the division. Second, we use the proposed ABET scheme to construct our instantiation. Specifically, the Setup and KeyGen of ABET are directly used in the instantiation. The Enc, Dec and Trace of ABET are presented in the step (2) of Hash, Adapt and Judge, respectively. Third, we rely on a recent work [35] to initiate the DPSS scheme. We particularly show an instantiation of DPSS with a KZG commitment scheme [29], which allows users to be held accountable in a committee. Last, we use Schnorr signature [44] to instantiate digital signature scheme due to its inherent homomorphic properties.

We denote an index space as $\{I_1, \cdots, I_k\} \in (\mathbb{Z}_q)^k$, which is associated with k committees. We define a hierarchy as follows: index i is close to the root node k, and index j is close to the leaf node. We assume each committee has n_0 users, and the threshold is t, where $t < n_0/2$ according to [35]. Let $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$ be a hash function, and the size of hash output H is l. The concrete instantiation is shown below.

- $\mathsf{Setup}(1^\lambda)$: It takes a security parameter λ as input, outputs a master public key $\mathsf{mpk} = (g, u, v, w, h, \hat{e}(g,h)^\alpha, \{g_1^\alpha, \cdots g_k^\alpha\}, \{h_1^\alpha, \cdots h_k^\alpha\}, g^\beta, h^{1/\alpha}, h^{\beta/\alpha}, \hat{e}(g,h)^{\theta/\alpha})$, and a master secret key $\mathsf{msk} = (\alpha, \beta, \theta)$, where $(\alpha, \beta, \theta) \in \mathbb{Z}_q^* \, \{z_1, \cdots, z_k\} \in \mathbb{Z}_q, (u, v, w) \in \mathbb{G}, \{g_1, \cdots, g_k\} = \{g^{z_1}, \cdots, g^{z_k}\}, \{h_1, \cdots, h_k\} = \{h^{z_1}, \cdots, h^{z_k}\}$. The master key generation is essentially the same as Setup of ABET.
  Regarding DPSS, the key shares of α and θ are distributed to users in committee C_0 by running the Share protocol of the DPSS scheme.
- $\mathsf{KeyGen}(\mathsf{C}_i, (\mathbf{M}, \pi))$: It inputs a committee C_i with index $(I_1, \cdots, I_i)$, and an access policy $(\mathbf{M}, \pi)$ ($\mathbf{M}$ has n_1 rows and n_2 columns, $\pi : \{1, \cdots, n_1\} \to \mathcal{U}$ denotes a mapping function), outputs a secret key sk_{Λ_i} for a modifier. Specifically, a group of t+1 members in committee C_i first recover secrets α and θ. Then, they pick $\{t_1, \cdots, t_{n_1}\} \in \mathbb{Z}_q$, for all $\tau \in [n_1]$, compute $\mathsf{sk}_{(\tau,1)} = g^{s_\tau} w^{t_\tau}, \mathsf{sk}_{(\tau,2)} = (u^{\pi(\tau)} v)^{-t_\tau}, \mathsf{sk}_{(\tau,3)} = h^{t_\tau}$, where s_τ is a key share from α. Eventually, they pick $\{r_1, \cdots, r_{n_1}\} \in \mathbb{Z}_q$, compute $\mathsf{sk}_0 = (g^{t^*/\alpha}, g^{r^*}), \mathsf{sk}_1 = g^\theta \cdot \widehat{i}^{t^*} \cdot g^{\beta \cdot r^*}, \mathsf{sk}_2 = \{g_{i-1}^{\alpha \cdot t^*}, \cdots g_1^{\alpha \cdot t^*}, g^{1/\alpha}\}$ (note that $g^{1/\alpha}$ is used only for delegation[1]), where $t^* = \sum_{\tau \in |n_1|}(t_\tau)$, $r^* = \sum_{\tau \in |n_1|}(r_\tau)$, and $\widehat{i} = g_k^{\alpha I_1} \cdots g_i^{\alpha I_i} \cdot g \in \mathbb{G}$ is associated with the committee C_i indexed $(I_1, \cdots, I_i)$. The secret key is $\mathsf{sk}_{\Lambda_i} = (\{\mathsf{sk}_\tau\}_{\tau \in [n_1]}, \mathsf{sk}_0, \mathsf{sk}_1, \mathsf{sk}_2)$. The secret key generation is essentially the same as KeyGen of ABET.
  Regarding DPSS, a group of t+1 members should recover the secrets α and θ by running the Open protocol of the DPSS scheme, prior to issuing a secret key to a modifier. Besides, the key shares of α and θ can be redistributed between committees by running the Redistribute protocol of the DPSS scheme (see the correctness of Redistribute protocol below).
- $\mathsf{Hash}(\mathsf{mpk}, m, \delta, j)$: To hash a message $m \in \mathbb{Z}_q$ under a set of attributes δ, and an index $(I_1, \cdots I_j)$, a user performs the following operations
  1. choose a randomness $\mathsf{r} \in \mathbb{Z}_q^*$, and a trapdoor R, compute a chameleon hash $b = g^{\mathsf{r}} \cdot p'^m$ where $p' = g^e$, $e = \mathsf{H}(\mathsf{R})$. Note that R denotes a short bit-string.
  2. generate a ciphertext on message $M = \mathsf{R}$ under a set of attributes $\delta = \{A_1, \cdots, A_{|\delta|}\}$ and index $(I_1, \cdots I_j)$. It first picks $s, r_1, r_2, \cdots, r_{|\delta|} \in \mathbb{Z}_q$, for $\tau \in |\delta|$ computes $ct_{(\tau,1)} = h^{r_\tau}$ and $ct_{(\tau,2)} = (u^{A_\tau} v)^{r_\tau} w^{-s}$. Then, it computes $ct = (\mathsf{R}||0^{l-|\mathsf{R}|}) \oplus \mathsf{H}(\hat{e}(g,h)^{\alpha s}) \oplus \mathsf{H}(\hat{e}(g,h)^{\theta s/\alpha})$, $ct_0 = (h^s, h^{s/\alpha}, h^{\beta \cdot s/\alpha})$, and $ct_1 = \widehat{j}^s$, where $\widehat{j} = h_k^{\alpha I_1} \cdots h_j^{\alpha I_j} \cdot h \in \mathbb{H}$. Eventually, it sets $C = (ct, \{ct_{(\tau,1)}, ct_{(\tau,2)}\}_{\tau \in [\delta]}, ct_0, ct_1)$.
  3. generate a signature $epk = g^{esk}, \sigma = esk + \mathsf{sk} \cdot \mathsf{H}(epk||c)$, where (esk, epk) denotes an ephemeral key pair, and $c = g^{\mathsf{sk} + (\mathsf{R}||0^{l-|\mathsf{R}|})}$ denotes a signed message.
  4. output $(m, p', b, \mathsf{r}, C, c, epk, \sigma)$.

[1]We discover that the underlying ABE scheme [43] cannot support key delegation. We add $g^{1/\alpha}$ to users' decryption keys, so the key holders can privately delegate their decryption keys (similar to the technique used in [11]).

- $\mathsf{Verify}(\mathsf{mpk}, m, p', b, \mathsf{r}, c, epk, \sigma)$: Anyone can verify whether a given hash (b, p') is valid, it outputs 1 if $b = g^{\mathsf{r}} \cdot p'^{m}$, and $g^{\sigma} = epk \cdot \mathsf{pk}^{\mathsf{H}(epk||c)}$.
- $\mathsf{Adapt}(\mathsf{sk}_{\Lambda_i}, m, m', p', b, \mathsf{r}, C, c, epk, \sigma)$: A modifier with a secret key sk_{Λ_i}, and a new message $m' \in \mathbb{Z}_q$, performs the following operations
  (1) check $1 \overset{?}{=} \mathsf{Verify}(\mathsf{mpk}, m, p', b, \mathsf{r}, c, epk, \sigma)$.
  (2) run the following steps to obtain trapdoor R:
    (a) generate a delegated key w.r.t an index $(I_1, \cdots I_{i+1})$. It picks $t' \in \mathbb{Z}_q$, computes $\mathsf{sk}_0 = (g^{(t^*+t')/\alpha}, g^{r^*}), \mathsf{sk}_1 = g^{\theta} \cdot \widehat{i}^{t^*} \cdot g^{\beta \cdot r^*} \cdot (g_{i-1}^{\alpha \cdot t^*})^{I_{i+1}} \cdot (g_k^{\alpha \cdot I_1} \cdots g_{i-1}^{\alpha \cdot I_{i+1}} \cdot g)^{t'}, \mathsf{sk}_2 = \{g_{i-2}^{\alpha \cdot t^*} \cdot g_{i-2}^{\alpha \cdot t'}, \cdots g_1^{\alpha \cdot t^*} \cdot g_1^{\alpha \cdot t'}, g^{1/\alpha}\}$. The delegated secret key is $\mathsf{sk}_{\Lambda_{i+1}} = (\{\mathsf{sk}_\tau\}_{\tau \in [n_1]}, \mathsf{sk}_0, \mathsf{sk}_1, \mathsf{sk}_2)$.
    (b) if the attribute set δ involved in the ciphertext satisfies the policy $(\mathbf{M}, \pi)$, then there exists constants $\{\gamma_\mu\}_{\mu \in I}$ according to [9]. It computes B as follows.

$$\begin{aligned} B &= \prod_{\mu \in I} (\hat{\mathsf{e}}(\mathsf{sk}_{(\mu,1)}, ct_{(0,1)}) \hat{\mathsf{e}}(\mathsf{sk}_{(\mu,2)}, ct_{(\mu,1)}) \\ &\quad \hat{\mathsf{e}}(ct_{(\mu,2)}, \mathsf{sk}_{(\mu,3)}))^{\gamma_\mu} \\ &= \hat{\mathsf{e}}(g,h)^{s \sum_{\mu \in I} \gamma_\mu s_\mu} = \hat{\mathsf{e}}(g,h)^{\alpha s}, where \sum_{\mu \in I} \gamma_\mu s_\mu = \alpha. \end{aligned}$$

Since the key delegation process will not affect the attribute-based components $\{\mathsf{sk}_\tau\}_{\tau \in [n_1]}$, B is computed once. Note that $ct_{(0,1)}, ct_{(0,2)}$ denote the first and second element of ct_0, and the same rule applies to sk_0.
    (c) check $(\mathsf{R}||0^{l-|\mathsf{R}|}) \overset{?}{=} ct \oplus \mathsf{H}(B) \oplus \mathsf{H}(A)$, where $A = \frac{\hat{\mathsf{e}}(\mathsf{sk}_1, ct_{(0,2)})}{\hat{\mathsf{e}}(\mathsf{sk}_{(0,1)}, ct_1)\hat{\mathsf{e}}(\mathsf{sk}_{(0,2)}, ct_{(0,3)})}$. The format "$||0^{l-|\mathsf{R}|}$" is used to ensure that the encrypted value R can be decrypted successfully with certainty $1 - 2^{l-|\mathsf{R}|}$. Since this technique can decide when the delegation process terminates, the hidden index j of C is known to the modifier.
  (3) compute a new randomness $\mathsf{r}' = \mathsf{r} + (m - m') \cdot e$, where $e = \mathsf{H}(\mathsf{R})$.
  (4) generate a new ciphertext C' on the same message $M = \mathsf{R}$ using the attribute set δ and index $(I_1, \cdots, I_j)$.
  (5) generate a signature $epk' = g^{esk'}, \sigma' = esk' + \mathsf{sk}' \cdot \mathsf{H}(epk'||c')$, where $c' = g^{\mathsf{sk}' + (\mathsf{R}||0^{l-|\mathsf{R}|})}$.
  (6) output $(m', p', b, \mathsf{r}', C', c', epk', \sigma')$.
- $\mathsf{Judge}(\mathsf{mpk}, T, T', \mathcal{O})$: Given two transactions (T, T') and an access device $\mathcal{O}$, where $T = (m, b, p', C, c, epk, \sigma,), T' = (m', b, p', C', c', epk', \sigma')$, any public user performs the following operations.
  (1) verify the connection between T' and T' as follows
    – verify chameleon hash $b = g^{\mathsf{r}} \cdot p'^{m} = g^{\mathsf{r}'} \cdot p'^{m'}$.
    – verify message-signature pair (c, σ) under (epk, pk), and message-signature pair (c', σ') under (epk', pk').
    – verify $\mathsf{pk}' = \mathsf{pk} \cdot \Delta(\mathsf{sk})$, where $\Delta(\mathsf{sk}) = c'/c = g^{\mathsf{sk}' - \mathsf{sk}}$. Note that $\Delta(\mathsf{sk})$ means the difference or shift between two keys, and (c, c') are derived from the same chameleon trapdoor R.
  (2) obtain a set of accused committees from interacting with an access device $\mathcal{O}$. This interaction process is referred to ABET's public traceability at Section 2.2.
  (3) output a transaction-committee pair (T', C_i). We rely on the KZG commitment and PoW consensus to hold a modifier pk' accountable in an accused committee C_i (see the correctness of KZG commitment below).

Correctness of the Redistribute protocol. Two secrets need to be distributed: (α, θ). We assume the readers are familiar with Shamir's SSS; thus, we omit the correctness of Share and Open protocols in our instantiation. Now, we show users in committee C_{i-1} securely *handoff* their key shares of secret α to users in committee C_i. According to the DPSS scheme in [35], an asymmetric bivariate polynomial is used: $f(x,y) = \underline{\alpha} + a_{0,1}x + a_{1,0}y + a_{1,1}xy + a_{1,2}xy^2 + \cdots + a_{t,2t}x^t y^{2t}$. Each user in committee C_{i-1} holds a *full* key share after running Share protocol. For example, a user with pk holds a key share $f(i,y)$, which is a polynomial with dimension t. Overall, the handoff (i.e., Redistribute protocol) includes three phases: share reduction, proactivization, and full-share distribution.

- *Share Reduction.* It requires each user in committee C_{i-1} reshares its full key share. For example, user pk derives a set of *reduced* shares $\{f(i,j)\}_{j \in [1,n_0]}$ from its key share $f(i,y)$ using SSS. Then, each user distributes the reduced shares to users in committee C_i, which includes a user with pk'. As a result, each user in C_i obtains a reduced share $f(x,j)$ by interpolating the received shares $\{f(i,j)\}_{i \in [1,t]}$. Note that the dimension of $f(x,j)$ is $2t$, and $2t$+1 of these reduced key shares $\{f(x,j)\}_{j \in [1,2t+1]}$ can recover α (see Section 2.3). The goal of this dimension-switching (from t to $2t$) is to achieve optimal communication overhead, such that only $2t$+1 users in committee C_i are required to update $f(x,j)$.
- *Proactivization.* It requires $F(x,j) = f(x,j) + f'(x,j)$, where $f'(x,y)$ is a new asymmetric bivariate polynomial with dimension $(t, 2t)$ and $f'(0,0) = 0$. We provide more details of $f'(x,y)$ later.
- *Full-share Distribution.* It requires each user in committee C_i to recover its full key share with dimension t. For example, a full key share $F(i,y)$ is recovered by interpolating the reduced shares $\{F(i,j)\}_{j \in [1,2t+1]}$ in committee C_i. This full key share $F(i,y)$ belongs to user pk', and t+1 of these full key shares can recover α.

Now we show the generation of an asymmetric bivariate polynomial $f'(x,y)$ with dimension $(t, 2t)$ such that $f'(0,0) = 0$, which is used to update the reduced key shares $f(x,j)$ during proactivization. We denote a subset of C_i as $\mathcal{U}'$, which includes $2t$+1 users. The generation of $f'(x,y)$ requires two steps: univariate zero share, and bivariate zero share.

- *Univariate Zero Share.* It requires each user in $\mathcal{U}'$ to generate a key share $f_j'(y)$ from a common univariate polynomial with dimension $2t$. First, each user i generates a univariate polynomial $f_i'(y) = 0 + a_1'y + a_2'y^2 + \cdots + a_{2t}'y^{2t}$, and broadcasts it to all users in $\mathcal{U}'$. Second, each user in $\mathcal{U}'$ generates a common univariate polynomial $f'(y) = \sum_{i \in [1,2t+1]} f_i'(y)$ by combining all received polynomials, and obtains a key share $f_j'(y)$ from $f'(y)$.
- *Bivariate Zero Share.* It requires each user in committee C_i to generate a key share $f'(x,j)$ from a common bivariate polynomial with dimension $(t, 2t)$. First, each user in $\mathcal{U}'$ generates a set of reduced shares $\{f'(i,y)\}_{i \in [1,n_0]}$ with dimension t from its key share $f_j'(y)$ (i.e., resharing process), where $f'(i,y) =$

$0 + a'_{1,0}y + a'_{2,0}y^2 + \cdots + a'_{2t,0}y^{2t}$. Since the reduced shares are distributed to all users in committee C_i, a common bivariate polynomial with dimension $(t, 2t)$ is established: $f'(x,y) = 0 + a'_{0,1}x + a'_{1,0}y + a'_{1,1}xy + a'_{1,2}xy^2 + \cdots + a'_{t,2t}x^t y^{2t}$. Second, each user in committee C_i obtains a reduced key share $f'(x,j)$ by interpolating the received shares $\{f'(i,j)\}_{j\in[1,2t+1]}$. The key share $f'(x,j) = 0 + a'_{0,1}x + a'_{0,2}x^2 + \cdots + a'_{0,t}x^t$ is used to update $f(x,j)$ in the proactivization.

The asymmetric bivariate polynomial $f'(x,y)$ can be reused in another proactivization when sharing secret θ. In other words, multiple handoff protocols with respect to different secrets can be updated using the same bivariate polynomial, with the condition that these handoff protocols are executed within the same committee.

Correctness of KZG commitment. We show that the committee members can be held accountable in a committee.

- *Share Reduction.* We require user pk' in committee C_i to generate a commitment $C_{f(x,j)}$, which is a KZG commitment to the reduced key shares $\{f(i,j)\}_{j\in[1,2t+1]}$, and a set of witnesses $\{w_{f(i,j)}\}_{j\in[1,2t+1]}$. A witness $w_{f(i,j)}$ means the witness to evaluation of $f(x,j)$ at i. Note that $i \in [1, 2t+1]$ indicates the order of user pk''s public key in committee C_i (we order nodes lexicographically by users' public keys and choose the first $2t+1$).
- *Full-share Distribution.* We require user pk' in committee C_i to generate a commitment $C_{F(x,j)}$, which is a KZG commitment to the reduced key shares $\{F(i,j)\}_{j\in[1,2t+1]}$, and a set of witnesses $w_{F(i,j)}$. A witness $w_{F(i,j)}$ means the witness to evaluation of $F(x,j)$ at i.
- *PoW Consensus.* We require user pk' to hash the KZG commitment and the set of witnesses, store them to an immutable transaction, and put them on-chain for PoW consensus.

The commitment and witness can also ensure the correctness of handoff described in the DPSS scheme above. Specifically, new committee members can verify the correctness of reduced shares from old committee members, thus the correctness of dimension-switching. The proof of correctness is publicly verifiable, such that any public user can verify that $f(i,j)$ (or $F(i,j)$) is the correct evaluation at i (i.e., user pk') of the polynomial committed by $C_{f(x,j)}$ (or $C_{F(x,j)}$) in committee C_i.

4.3 Implementation and Evaluation

We evaluate the performance of the proposed solution based on a proof-of-concept implementation in Python and Flask framework. We create a mutable open blockchain system with basic functionalities and a PoW consensus mechanism. The simulated open blockchain system is "healthy", satisfying the properties of persistence and liveness [24]. The system is specifically designed to include ten blocks, each block consists of 100 transactions. Please note, that our implementation can easily extend it to real-world applications such as a block containing 3500 transactions. We simulate ten nodes in a peer-to-peer network, each of them is implemented as a lightweight blockchain node. They can also be regarded as the users in a committee. A chain of blocks is established with PoW mechanism by consolidating transactions broadcast by the ten nodes. Our implementation code is available on GitHub [2].

First, if users append mutable transactions to blockchain, they use the proposed solution to hash the registered message m. Later, a miner uses the conventional hash function SHA-256 H to hash the chameleon hash output h and validates $\mathsf{H}(h)$ using a Merkle tree. Note that the non-hashed components such as randomness r, are parts of a mutable transaction $T = (\mathsf{pk}_{\mathsf{CH}}, m, h, \mathsf{r}, C, c, \sigma)$. As a consequence, a modifier can replace T by $T' = (\mathsf{pk}_{\mathsf{CH}}, m', h, \mathsf{r}', C', c', \sigma')$ without changing the hash output $\mathsf{H}(h)$.

Second, we mimic a dynamic committee that includes five users, we split the master secret key into five key shares so that each user in a committee holds a key share. We simulate the basic functionality of DPSS, including resharing and updating key shares. Any user can join in or leave from a committee by transmitting those key shares between committee members. In particular, we simulate three users in a committee can collaboratively recover the master secret key and grant rewriting privileges to the modifiers.

We implement our proposed solution using the Charm framework [5] and evaluate its performance on a PC with Intel Core i5 (2.7GHz×2) and 8GB RAM. We use Multiple Precision Arithmetic Library, Pairing-Based Cryptography (PBC) Library, and we choose MNT224 curve for pairing, which is the best Type-III paring in PBC. We instantiate the hash function and the pseudo-random generator with the corresponding standard interfaces provided by the Charm framework.

First, we made an overhead comparison against [17, 49], showing that our storage cost is slightly higher than [17] because our scheme includes digital signature and ABET, and it is lower than [49] if a mutable transaction involves less than 100 attributes. We also implemented the scheme in [49], showing that the execution times of our keygen, hash, and adapt algorithms are slightly higher than [49]. The execution time of KeyGen, Hash, and Adapt algorithms are measured and shown in Figure 6 (a-c).

Second, we evaluate the execution time of a t-out-of-n_0 DPSS protocol, where n_0 indicates the number of users in a committee and t is the threshold. Let $t < n_0/2$ be a safe threshold. The overhead includes the distribution cost between committee members, and the polynomial calculation cost. The evaluation in [35] provides storage overhead only, but we provide the execution time of DPSS in Figure 6 (d). Our evaluation shows that the maximal threshold is t=13 if we run SSS on a standard computer. If t=14, a secret share such as $f(23) = secret + (23)^1 + \cdots + (23)^{14}$ will overflow.

To conclude, the implementation performs the resharing twice and updating once regarding two shared secrets. Besides, the number of updating process is constant in a committee, independent of the number of shared secrets used in ABET. Since only two secrets are needed to be shared and recovered, we argue that the proposed ABET scheme is the most practical one if ABET includes DPSS. On the security-front, because every committee has at most $\frac{n_0}{3}$ malicious members [34] and $\frac{n_0}{2}$+1 committee members recover the shared secrets [35], the malicious committee members cannot dictate the committee and control the rewriting privileges. For the storage cost, each mutable transaction needs to store $T = (\mathsf{pk}_{\mathsf{CH}}, m, h, \mathsf{r}, C, c, \sigma)$. So, the storage cost of a mutable transaction includes: 1) $2\mathcal{L}_{\mathbb{Z}_q} + 3\mathcal{L}_{\mathbb{G}}$ regarding DL-based chameleon hash; 2) $\mathcal{L}_{\mathbb{Z}_q} + |\delta| \times \mathcal{L}_{\mathbb{G}} + (|\delta| + 4) \times \mathcal{L}_{\mathbb{H}}$ regarding ABET; 3) $\mathcal{L}_{\mathbb{Z}_q} + 2\mathcal{L}_{\mathbb{G}}$

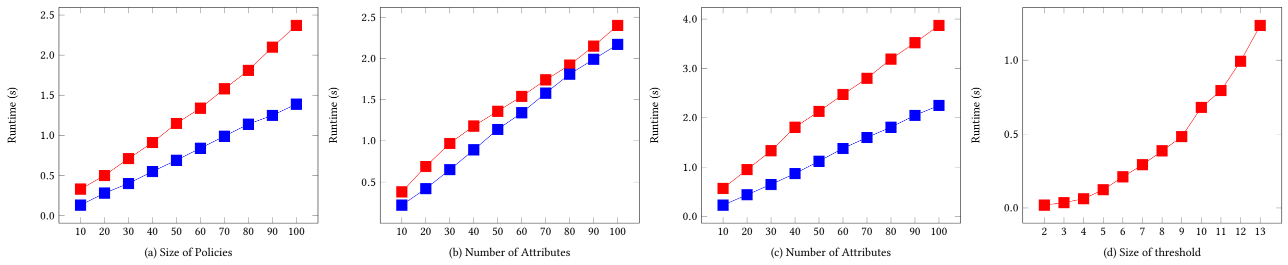

Figure 6: Execution time of KeyGen, Hash, Adapt **algorithms, and** DPSS **scheme [35]. Red line (with solid square) is for this work, while blue line (with solid dot) is for [49].**

regarding digital signature. The committee's on-chain storage cost regarding DPSS [35] is $2(t+1) \times [\mathcal{L}_{\mathbb{G}} + (2t+1)(\mathcal{L}_{\mathbb{Z}_q} + \mathcal{L}_{\mathbb{H}})]$.

5 RELATED WORK

Blockchain Rewriting. Ateniese et al. [7] introduced the notion of blockchain rewriting. Their proposal is to replace the regular SHA256 hash function by a chameleon hash (CH) in blockchain generation [31]. The hashing of CH is parametrized by a public key pk, and CH behaves like a collision-resistant hash function if the chameleon secret key sk (or trapdoor) is unknown. A trapdoor holder (or modifier) can find collisions and output a new message-randomness pair without changing the hash value.

Camenisch et al. [15] introduced a new cryptographic primitive: chameleon hash with ephemeral trapdoor (CHET). CHET states that a modifier should have two trapdoors to find collisions: one trapdoor sk is associated with the public key pk; the other one is an ephemeral trapdoor *etd* chosen by the party who initially computed the hash value. CHET provides more control in rewriting in the sense that the party, who computed the hash value, can decide whether the holder of sk shall be able to rewrite the hash by providing or withholding the ephemeral trapdoor *etd*.

Derler et al. [17] proposed policy-based chameleon hash (PCH) to achieve fine-grained blockchain rewriting. The proposed PCH replaces the public key encryption scheme in CHET by a ciphertext-policy ABE scheme, such that a modifier must satisfy a policy to find collisions given a hash value. Later, Tian et al. proposed an accountable PCH (PCHBA) for for blockchain rewritings [49]. The proposed PCHBA enables the modifiers of transactions to be held accountable for the modified transactions. In particular, PCHBA allows a third party (e.g., key generation center) to resolve any dispute over modified transactions.

In another work, Puddu et al. [42] proposed μchain: a mutable blockchain. A transaction owner introduces a set of transactions, including an active transaction and multiple inactive transactions, where the inactive transactions are possible versions of the transaction data (namely, mutations) encrypted by the transaction owner, and the decryption keys are distributed among miners using Shamir's SSS [46]. The transaction owner enforces access control policies to define who is allowed to trigger mutations in which context. Upon receiving a mutation-trigger request, a set of miners runs a Multi Party Computation (MPC) protocol to recover the decryption key, decrypt the appropriate version of the transaction and publish it as an active transaction. μchain incurs considerable overhead due to the use of MPC protocols across multiple miners. It works at both permissioned and permissionless blockchains.

Deuber et al. [19] introduced an efficient redactable blockchain in the permissionless setting. The proposed protocol relies on a consensus-based e-voting system [30], such that the modification is executed in the chain if a modification request from any public user gathers enough votes from miners (we call it V-CH for convenience). In a follow-up work, Thyagarajan et al. [48] introduced a protocol called Reparo to repair blockchains, which acts as a publicly verifiable layer on top of any permissionless blockchain. The unique feature of Reparo is that it is immediately integrable into open blockchains in a backward compatible fashion (i.e., any existing blockchains already containing illicit contents can be redacted).

There are mainly two types of blockchain rewritings in the literature: CH-based [7, 15–17, 49], and non CH-based [19, 42, 48]. CH-based blockchain rewritings allow one or more trusted modifiers to rewrite blockchain. The non-CH-based solution requires a threshold number of parties (or miners) to rewrite the blockchain. We stress that both aim to secure blockchain rewritings and one can apply both of them to redactable blockchains.

Table 1 shows a comparison between blockchain rewriting related solutions. In this work, we use chameleon hash cryptographic primitive to secure the blockchain rewriting. Our proposed solution supports fine-grained and accountable rewriting for open blockchains in the permissionless setting. It holds both the modifiers' public keys and their rewriting privileges accountable for the modified transactions. We stress that it is necessary to hold rewriting privileges accountable to the modified transactions; for example, a modifier may obtain various rewriting privileges from different committees. Overall, this work takes a significant step forward by allowing fine-grained blockchain rewriting to be performed in the permissionless setting compared to [17, 49].

Dynamic Proactive Secret Sharing. Proactive security was first introduced by Ostrovsky and Yung [41], which is refreshing secrets to withstand compromise. Later, Herzberg et al. [27] introduced proactive secret sharing (PSS). The PSS allows the distributed key shares in a SSS to be updated periodically, so that the secret remains secure even if an attacker compromises a threshold number of shareholders in each epoch. However, it did not support dynamic committees because users may join in or leave from a committee dynamically. Desmedt and Jajodia [18] introduced a scheme that

Table 1: The comparison between various blockchain rewriting solutions. CH-based means blockchain rewriting is realized via the chameleon hash function. Fine-grained (access control) means that a transaction is associated with an attribute set (or an access policy), and the transaction can be modified by anyone whose rewriting privilege satisfies the attribute set (or the access policy).

	CH [7]	μchain [42]	PCH [17]	V-CH [19]	PCHBA [49]	Ours
CH-based	✓	×	✓	×	✓	✓
Permissionless	✓	✓	×	✓	×	✓
Fine-grained	×	×	✓	×	✓	✓
Accountability	✓	✓	×	✓	✓	✓

redistributes secret shares to new access structure (or new committee). Specifically, a resharing technique is used to change the committee and threshold in PSS. However, the scheme is not verifiable, which disallows PSS to identify the faulty (or malicious) users. The property of verifiability is essential to PSS (i.e., verifiable secret sharing such as Feldman [23]), which holds malicious users accountable. So, the dynamic proactive secret sharing (DPSS) we considered in this work includes verifiability.

There exist several DPSS schemes in the literature. Wong et al. [52] introduced a verifiable secret redistribution protocol that supports dynamic committee. The proposed protocol allows new shareholders to verify the validity of their shares after redistribution between different committees. Zhou et al. [55] introduced an APSS, a PSS protocol for asynchronous systems that tolerate denial-of-service attacks. Schultz et al. [45] introduced a resharing protocol called MPSS. The MPSS supports mobility, which means the group of shareholders can change during resharing. Baron et al. [8] introduced a DPSS protocol that achieves a constant amortized communication overhead per secret share. In CCS'19, Maram et al. [35] presented a practical DPSS: CHURP. CHURP is designed for open blockchains, and it has very low communication overhead per epoch compared to the existing schemes [8, 45, 52, 55]. Specifically, the total number of bits transmitted between all committee members in an epoch is substantially lower than in existing schemes. Recently, Benhamouda et al. [10] introduced anonymous secret redistribution. The benefit is to ensure sharing and resharing of secrets among small dynamic committees.

DPSS can be used to secure blockchain rewriting, such as μchain [42]. μchain relies on encryption with secret sharing (ESS) to hide illegal content, as certain use-cases aim to prevent distribution of illegal content (e.g., child pornography) via the blockchain. ESS allows all the mutable transactions containing illegal content to be encrypted using transaction-specific keys. The transaction-specific keys are split into shares using DPSS [8], and these resulting shares are distributed to a number of miners, which then reshare the keys among all online miners dynamically. In this work, we use KP-ABE with DPSS to ensure blockchain rewiring with fine-grained access control. The master secret key in KP-ABE is split into key shares, and these key shares are distributed to all users in a committee. The key shares can be securely redistributed across dynamic committees. To the best of our knowledge, ours is the first attempt to distribute the master secret key in KP-ABE for decentralized systems.

6 CONCLUSION

In this paper, we proposed a new framework of accountable fine-grained blockchain rewriting. The proposed framework is designed for open blockchains that require no trust assumptions. Besides, the proposed framework achieves public accountability, meaning that the modifiers' public keys and their rewriting privileges can be publicly held accountable for the modified transactions. We presented a practical instantiation, and showed that the proposed solution is suitable for open blockchain applications. In particular, the proof-of-concept implementation demonstrated that our proposed solution can be easily integrated into the existing open blockchains.

REFERENCES

[1] [n.d.]. General Data Protection Regulation. https://eugdpr.org.

[2] [n.d.]. Our Source Code. https://github.com/lbwtorino/Fine-Grained-Blockchain-Rewriting-in-Permissionless-Setting.

[3] [n.d.]. Proof of Stake. https://en.wikipedia.org/wiki/Proof_of_stake.

[4] Shashank Agrawal and Melissa Chase. 2017. FAME: fast attribute-based message encryption. In *CCS*. 665–682.

[5] 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.

[6] Marcin Andrychowicz and Stefan Dziembowski. 2015. Pow-based distributed cryptography with no trusted setup. In *CRYPTO*. 379–399.

[7] Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton Andrade. 2017. Redactable blockchain–or–rewriting history in bitcoin and friends. In *EuroS&P*. 111–126.

[8] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. 2015. Communication-optimal proactive secret sharing for dynamic groups. In *ACNS*. 23–41.

[9] Amos Beimel. 1996. *Secure schemes for secret sharing and key distribution.* Ph.D. Dissertation. PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel.

[10] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. 2020. Can a Public Blockchain Keep a Secret?. In *TCC*. 260–290.

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

[12] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. 2005. Hierarchical identity based encryption with constant size ciphertext. In *CRYPTO*. 440–456.

[13] Dan Boneh, Amit Sahai, and Brent Waters. 2006. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*. 573–592.

[14] Dan Boneh and Brent Waters. 2006. A fully collusion resistant broadcast, trace, and revoke system. In *CCS*. 211–220.

[15] Jan Camenisch, David Derler, Stephan Krenn, Henrich C Pöhls, Kai Samelin, and Daniel Slamanig. 2017. Chameleon-hashes with ephemeral trapdoors. In *PKC*. 152–182.

[16] David Derler, Kai Samelin, and Daniel Slamanig. 2020. Bringing Order to Chaos: The Case of Collision-Resistant Chameleon-Hashes. In *PKC*. 462–492.

[17] David Derler, Kai Samelin, Daniel Slamanig, and Christoph Striecks. 2019. Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based. In *NDSS*.

[18] Yvo Desmedt and Sushil Jajodia. 1997. *Redistributing secret shares to new access structures and its applications.* Technical Report.

[19] Dominic Deuber, Bernardo Magri, and Sri Aravinda Krishnan Thyagarajan. 2019. Redactable blockchain in the permissionless setting. In *IEEE S&P*. 124–138.

[20] John R Douceur. 2002. The sybil attack. In *International workshop on peer-to-peer systems.* 251–260.

[21] Léo Ducas. 2010. Anonymity from asymmetry: New constructions for anonymous HIBE. In *CT-RSA*. 148–164.

[22] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. 2015. Proofs of space. In *CRYPTO*. 585–605.

[23] Paul Feldman. 1987. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*. 427–438.

[24] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*. 281–310.

[25] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT*. 295–310.

[26] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles.* 51–68.

[27] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. 1995. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO*. 339–352.

[28] Markus Jakobsson and Ari Juels. 1999. Proofs of work and bread pudding protocols. In *Secure information networks*. 258–272.

[29] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*. 177–194.

[30] Tadayoshi Kohno, Adam Stubblefield, Aviel D Rubin, and Dan S Wallach. 2004. Analysis of an electronic voting system. In *IEEE S&P*. 27–40.

[31] Hugo Krawczyk and Tal Rabin. 2000. Chameleon Signatures. In *NDSS*.

[32] Junzuo Lai and Qiang Tang. 2018. Making any attribute-based encryption accountable, efficiently. In *ESORICS*. 527–547.

[33] Zhen Liu, Zhenfu Cao, and Duncan S Wong. 2013. Blackbox traceable CP-ABE: how to catch people leaking their keys by selling decryption devices on ebay. In *CCS*. 475–486.

[34] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. In *CCS*. 17–30.

[35] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. 2019. Churp: Dynamic-committee proactive secret sharing. In *CCS*. 2369–2386.

[36] Roman Matzutt, Jens Hiller, Martin Henze, Jan Henrik Ziegeldorf, Dirk Müllmann, Oliver Hohlfeld, and Klaus Wehrle. 2018. A quantitative analysis of the impact of arbitrary blockchain content on bitcoin. In *FC*. 420–438.

[37] Roman Matzutt, Oliver Hohlfeld, Martin Henze, Robin Rawiel, Jan Henrik Ziegeldorf, and Klaus Wehrle. 2016. Poster: I don't want that content! on the risks of exploiting bitcoin's blockchain as a content store. In *CCS*. 1769–1771.

[38] Ralph C Merkle. 1989. A certified digital signature. In *CRYPTO*. 218–238.

[39] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[40] Jianting Ning, Zhenfu Cao, Xiaolei Dong, Junqing Gong, and Jie Chen. 2016. Traceable CP-ABE with short ciphertexts: How to catch people selling decryption devices on ebay efficiently. In *ESORICS*. 551–569.

[41] Rafail Ostrovsky and Moti Yung. 1991. How to withstand mobile virus attacks. In *ACM PODC*. 51–59.

[42] Ivan Puddu, Alexandra Dmitrienko, and Srdjan Capkun. 2017. μchain: How to Forget without Hard Forks. *IACR Cryptology ePrint Archive* 2017 (2017), 106.

[43] Yannis Rouselakis and Brent Waters. 2013. Practical constructions and new proof methods for large universe attribute-based encryption. In *CCS*. 463–474.

[44] Claus-Peter Schnorr. 1991. Efficient signature generation by smart cards. *Journal of cryptology* 4, 3 (1991), 161–174.

[45] David A Schultz, Barbara Liskov, and Moses Liskov. 2008. Mobile proactive secret sharing. In *ACM PODC*. 458–458.

[46] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[47] Victor Shoup. 1997. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*. 256–266.

[48] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Bernardo Magri, Daniel Tschudi, and Aniket Kate. 2020. Reparo: Publicly Verifiable Layer to Repair Blockchains. *arXiv preprint arXiv:2001.00486* (2020).

[49] Yangguang Tian, Nan Li, Yingjiu Li, Pawel Szalachowski, and Jianying Zhou. 2020. Policy-based Chameleon Hash for Blockchain Rewriting with Black-box Accountability. In *ACSAC*. 813–828.

[50] Giannis Tziakouris. 2018. CryptocurrenciesâĂŤa forensic challenge or opportunity for law enforcement? an interpol perspective. *IEEE S&P* 16, 4 (2018), 92–94.

[51] Brent Waters. 2005. Efficient identity-based encryption without random oracles. In *EUROCRYPT*. 114–127.

[52] Theodore M Wong, Chenxi Wang, and Jeannette M Wing. 2002. Verifiable secret redistribution for archive systems. In *First International IEEE Security in Storage Workshop, 2002. Proceedings*. 94–105.

[53] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. 2020. Ohie: Blockchain scaling made simple. In *IEEE (S&P)*. 90–105.

[54] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. Rapidchain: Scaling blockchain via full sharding. In *CCS*. 931–948.

[55] Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. 2005. APSS: Proactive secret sharing in asynchronous systems. *ACM (TISSEC)* 8, 3 (2005), 259–286.

A SECURITY ANALYSIS OF NEW ASSUMPTION

Theorem A.1. *Let $(\epsilon_1, \epsilon_2, \epsilon_T) : \mathbb{Z}_q \to \{0,1\}^*$ be three random encodings (injective functions) where $\mathbb{Z}_q$ is a prime field. ϵ_1 maps all $a \in \mathbb{Z}_q$ to the string representation $\epsilon_1(g^a)$ of $g^a \in \mathbb{G}$. Similarly, ϵ_2 for $\mathbb{H}$ and ϵ_T for $\mathbb{G_T}$. If $(a,b,c,d,\{z_i\}_{i\in[1,q']}) \xleftarrow{R} \mathbb{Z}_q$ and encodings $\epsilon_1, \epsilon_2, \epsilon_T$ are randomly chosen, then we define the advantage of the adversary in solving the q'-type with at most Q queries to the group operation oracles O_1, O_2, O_T and the bilinear pairing $\hat{e}$ as*

$$
\begin{aligned}
\mathsf{Adv}^{q'\text{-}type}_{\mathcal{A}}(\lambda) \quad = \quad & |\Pr[\mathcal{A}(q, \epsilon_1(1), \epsilon_1(a), \epsilon_1(c), \epsilon_1((ac)^2),\\
& \epsilon_1(abd), \epsilon_1(d/ab), \epsilon_1(z_i), \epsilon_1(acz_i),\\
& \epsilon_1(ac/z_i), \epsilon_1(a^2cz_i), \epsilon_1(b/z_i^2),\\
& \epsilon_1(b^2/z_i^2), \epsilon_1(acz_i/z_j), \epsilon_1(bz_i/z_j^2),\\
& \epsilon_1(abcz_i/z_j), \epsilon_1((ac)^2z_i/z_j),\\
& \epsilon_2(1), \epsilon_2(b), \epsilon_2(abd), \epsilon_2(abcd),\\
& \epsilon_2(d/ab), \epsilon_2(c), \epsilon_2(cd/ab),\\
= \quad & b : (a,b,c,d,\{z_i\}_{i\in[1,q']}, s \xleftarrow{R} \mathbb{Z}_q, b \in (0,1),\\
& t_b = abc, t_{1-b} = s)]\\
& -1/2| \le \frac{16(Q + q' + 22)^2}{q}
\end{aligned}
$$

Proof. Let $\mathcal{S}$ play the following game for $\mathcal{A}$. $\mathcal{S}$ maintains three polynomial sized dynamic lists: $L_1 = \{(p_i, \epsilon_{1,i})\}, L_2 = \{(q_i, \epsilon_{2,i})\}, L_T = \{(t_i, \epsilon_{T,i})\}$, the $p_i \in \mathbb{Z}_q[A,B,C,D,Z_i,Z_j,T_0,T_1]$ are 8-variate polynomials over $\mathbb{Z}_q$ (note that $i \neq j$), such that $p_0 = 1, p_1 = A, p_2 = C, p_3 = (AC)^2, p_4 = ABD, p_5 = D/AB, p_6 = Z_i, p_7 = ACZ_i, p_8 = AC/Z_i, p_9 = A^2CZ_i, p_{10} = B/Z_i^2, p_{11} = B^2/Z_i^2, p_{12} = ACZ_i/Z_j, p_{13} = BZ_i/Z_j^2, p_{14} = ABCZ_i/Z_j, p_{15} = (AC)^2Z_i/Z_j, q_0 = 1, q_1 = B, q_2 = ABD, q_3 = ABCD, q_4 = D/AB, q_5 = C, q_6 = CD/AB, p_{16} = T_0, p_{17} = T_1, t_0 = 1$, and $(\{\epsilon_{1,i}\}_{i=0}^{16} \in \{0,1\}^*, \{\epsilon_{2,i}\}_{i=0}^{5} \in \{0,1\}^*, \{\epsilon_{T,0}\} \in \{0,1\}^*)$ are arbitrary distinct strings. Therefore, the three lists are initialized as $L_1 = \{(p_i, \epsilon_{1,i})\}_{i=0}^{17}, L_2 = \{(q_i, \epsilon_{2,i})\}_{i=0}^{6}, L_T = (t_0, \epsilon_{T,0})$.

At the beginning of the game, $\mathcal{S}$ sends the encoding strings $(\{\epsilon_{1,i}\}_{i=0,\cdots,17}, \{\epsilon_{2,i}\}_{i=0,\cdots,6}, \epsilon_{T,0})$ to $\mathcal{A}$, which includes q'+26 strings. Note that the number of encoding string $\epsilon_{1,i}$ is linear to the parameter q'. After this, $\mathcal{S}$ simulates the group operation oracles O_1, O_2, O_T and the bilinear pairing $\hat{e}$. We assume that all requested operands are obtained from $\mathcal{S}$.

- O_1: The group operation involves two operands $\epsilon_{1,i}, \epsilon_{1,j}$. Based on these operands, $\mathcal{S}$ searches the list L_1 for the corresponding polynomials p_i and p_j. Then $\mathcal{S}$ performs the polynomial addition or subtraction $p_l = p_i \pm p_j$ depending on whether multiplication or division is requested. If p_l is in the list L_1, then $\mathcal{S}$ returns the corresponding ϵ_l to $\mathcal{A}$. Otherwise, $\mathcal{S}$ uniformly chooses $\epsilon_{1,l} \in \{0,1\}^*$, where $\epsilon_{1,l}$ is unique in the encoding string L_1, and appends the pair $(p_l, \epsilon_{1,l})$ into the list L_1. Finally, $\mathcal{S}$ returns $\epsilon_{1,l}$ to $\mathcal{A}$ as the answer. Group operation queries in O_2, O_T are treated similarly.
- $\hat{e}$: The group operation involves two operands $\epsilon_{T,i}, \epsilon_{T,j}$. Based on these operands, $\mathcal{S}$ searches the list L_T for the corresponding polynomials t_i and t_j. Then $\mathcal{S}$ performs the polynomial multiplication $t_l = t_i \cdot t_j$. If t_l is in the list L_T, then $\mathcal{S}$ returns the corresponding $\epsilon_{T,l}$ to $\mathcal{A}$. Otherwise, $\mathcal{S}$ uniformly chooses $\epsilon_{T,l} \in \{0,1\}^*$, where $\epsilon_{T,l}$ is unique in the encoding string L_T, and appends the pair $(t_l, \epsilon_{T,l})$ into the list L_T. Finally, $\mathcal{S}$ returns $\epsilon_{T,l}$ to $\mathcal{A}$ as the answer.

After querying at most Q times of corresponding oracles, $\mathcal{A}$ terminates and outputs a guess $b' = \{0,1\}$. At this point, $\mathcal{S}$ chooses random $a,b,c,d,z_i,z_j,s \in \mathbb{Z}_q$ and $t_b = abc$ and $t_{1-b} = s$. $\mathcal{S}$ sets $A = a, B = b, C = c, D = d, Z_i = z_i, Z_j = z_j, T_0 = t_b, T_1 = t_{1-b}$. The simulation by $\mathcal{S}$ is perfect (and reveal nothing to $\mathcal{A}$ about b) unless the abort event happens. Thus, we bound the probability of event abort by analyzing the following cases:

(1) $p_i(a,b,c,d,z_i,z_j,t_0,t_1) = p_j(a,b,c,d,z_i,z_j,t_0,t_1)$: The polynomial $p_i \neq p_j$ due to the construction method of L_1, and $(p_i - p_j)(a,b,c,d,z_i,z_j,t_0,t_1)$ is a non-zero polynomial of degree $[0,6]$, or q-2 (q-2 is produced by Z_j^{q-2}). Since $Z_j \cdot Z_j^{q-2} = Z_j^{q-1} \equiv 1(\mod q)$, we have $(AC)^2 Z_i Z_j \cdot Z_j^{q-2} \equiv (AC)^2 Z_i Z_j(\mod q)$. By using Lemma 1 in [47], we have $\Pr[(p_i - p_j)(a,b,c,d,z_i,z_j,t_0,t_1) = 0] \leq \frac{6}{q}$ because the maximum degree of $(AC)^2 Z_i / Z_j (p_i - p_j)(a,b,c,d,z_i,z_j,t_0,t_1)$ is 6. So, we have $\Pr[p_i(a,b,c,d,z_i,z_j,t_0,t_1) = p_j(a,b,c,d,z_i,z_j,t_0,t_1)] \leq \frac{6}{q}$, and the abort probability is $\Pr[\text{abort}_1] \leq \frac{6}{q}$.

(2) $q_i(a,b,c,d,z_i,z_j,t_0,t_1) = q_j(a,b,c,d,z_i,z_j,t_0,t_1)$: The polynomial $q_i \neq q_j$ due to the construction method of L_2, and $(q_i - q_j)(a,b,c,d,z_i,z_j,t_0,t_1)$ is a non-zero polynomial of degree $[0,4]$, or q-2 (q-2 is produced by $(AB)^{q-2}$). Since $AB \cdot (AB)^{q-2} = (AB)^{q-1} \equiv 1(\mod q)$, we have $CDAB \cdot (AB)^{q-2} \equiv CDAB(\mod q)$. The maximum degree of $CD/AB(q_i - q_j)(a,b,c,d,z_i,z_j,t_0,t_1)$ is 4, so the abort probability is $\Pr[\text{abort}_2] \leq \frac{4}{q}$.

(3) $t_i(a,b,c,d,z_i,z_j,t_0,t_1) = t_j(a,b,c,d,z_i,z_j,t_0,t_1)$: The polynomial $p_i \neq p_j$ due to the construction method of L_1, and $(p_i - p_j)(a,b,c,d,z_i,z_j,t_0,t_1)$ is a non-zero polynomial of degree $[0,6]$, or q-2. Since $(AC)^2 Z_i \cdot Z_j^{q-2}(t_i - t_j)(a,b,c,d,z_i,z_j,t_0,t_1)$ has degree 6, we have $\Pr[(p_i - p_j)(a,b,c,d,z_i,z_j,t_0,t_1) = 0] \leq \frac{6}{q}$. The abort probability is $\Pr[\text{abort}_3] \leq \frac{6}{q}$.

By summing over all valid pairs (i,j) in each case (i.e., at most $\binom{Q_{\epsilon_1}+18}{2} + \binom{Q_{\epsilon_2}+7}{2} + \binom{Q_{\epsilon_T}+1}{2}$ pairs), and $Q_{\epsilon_1} + Q_{\epsilon_2} + Q_{\epsilon_T} = Q + q' + 26$, we have the abort probability is

$$\begin{aligned}\Pr[\text{abort}] &= \Pr[\text{abort}_1] + \Pr[\text{abort}_2] + \Pr[\text{abort}_3] \\ &\leq [\binom{Q_{\epsilon_1}+18}{2} + \binom{Q_{\epsilon_2}+7}{2} + \binom{Q_{\epsilon_T}+1}{2}] \\ &\cdot (\frac{4}{q} + 2\frac{6}{q}) \leq \frac{16(Q + q' + 26)^2}{q}.\end{aligned}$$

□

B SECURITY ANALYSIS OF GENERIC FRAMEWORK

B.1 Proof of Theorem B.1

Theorem B.1. *The proposed generic framework is indistinguishable if the* CH *scheme is indistinguishable.*

Proof. The reduction is executed between an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$. Assume that $\mathcal{A}$ activates at most $n(\lambda)$ chameleon hashes. Let $\mathcal{S}$ denote a distinguisher against CH, who is given a chameleon public key pk^* and a HashOrAdapt oracle, aims to break the indistinguishability of CH. In particular, $\mathcal{S}$ is allowed to access the chameleon trapdoor $Dlog(\mathsf{pk}^*)$ [17]. $\mathcal{S}$ randomly chooses $g \in [1, n(\lambda)]$ as a guess for the index of the HashOrAdapt query. In the g-th query, $\mathcal{S}$'s challenger directly hashes a message $(h, \mathsf{r}) \leftarrow \mathsf{Hash}(\mathsf{pk}^*, m)$, instead of calculating the chameleon hash and randomness (h, r) using Adapt algorithm.

$\mathcal{S}$ sets up the game for $\mathcal{A}$ by distributing a master secret key to a group of users in a committee. $\mathcal{S}$ can honestly generate secret keys for any modifier associated with an access policy Λ. If $\mathcal{A}$ submits a tuple (m_0, m_1, δ) in the g-th query, then $\mathcal{S}$ first obtains a chameleon hash (h_b, r_b) from his HashOrAdapt oracle. Then, $\mathcal{S}$ simulates a message-signature pair (c, σ), and a ciphertext C on message $Dlog(\mathsf{pk}^*)$ according to the protocol specification. Eventually, $\mathcal{S}$ returns $(h_b, \mathsf{r}_b, C, c, \sigma)$ to $\mathcal{A}$. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. If $\mathcal{A}$ guesses the random bit correctly, then $\mathcal{S}$ can break the indistinguishability of CH.

□

B.2 Proof of Theorem B.2

Theorem B.2. *The proposed generic framework is adaptively collision-resistant if the* ABET *scheme is semantically secure, the* CH *scheme is collision-resistant, and the* DPSS *scheme has secrecy.*

Proof. We define a sequence of games $\mathbb{G}_i$, $i = 0, \cdots, 4$ and let Adv_i^{GF} denote the advantage of the adversary in game $\mathbb{G}_i$. Assume that $\mathcal{A}$ issues at most $n(\lambda)$ queries to Hash oracle.

- $\mathbb{G}_0$: This is the original game for adaptive collision-resistance.
- $\mathbb{G}_1$: This game is identical to game $\mathbb{G}_1$ except that $\mathcal{S}$ will output a random bit if $\mathcal{A}$ outputs a correct master secret key when no more than t users in a committee are corrupted, and the setup is honest (or the dealer is honest). The difference between $\mathbb{G}_0$ and $\mathbb{G}_1$ is negligible if DPSS scheme has secrecy.
$$\left|\mathsf{Adv}_0^{GF} - \mathsf{Adv}_1^{GF}\right| \leq \mathsf{Adv}_{\mathcal{S}}^{\mathsf{DPSS}}(\lambda). \quad (1)$$
- $\mathbb{G}_2$: This game is identical to game $\mathbb{G}_1$ except the following difference: $\mathcal{S}$ randomly chooses $g \in [1, n(\lambda)]$ as a guess for the index of the Hash$'$ oracle which returns the chameleon hash $(h^*, m^*, \mathsf{r}^*, C^*, c^*, \sigma^*)$. $\mathcal{S}$ will output a random bit if $\mathcal{A}$'s attacking query does not occur in the g-th query. Therefore, we have
$$\mathsf{Adv}_1^{GF} = n(\lambda) \cdot \mathsf{Adv}_2^{GF} \quad (2)$$
- $\mathbb{G}_3$: This game is identical to game $\mathbb{G}_2$ except that in the g-th query, the encrypted message $\mathsf{sk}_{\mathsf{CH}}$ in C^* is replaced by "$\perp$" (i.e., an empty value). Below we show that the difference between $\mathbb{G}_2$ and $\mathbb{G}_3$ is negligible if ABET scheme is semantically secure.
Let $\mathcal{S}$ denote an attacker against ABET with semantic security, who is given a public key pk^* and a key generation oracle, aims to distinguish between encryptions of M_0 and M_1 associated with a challenge index j^* and a challenge set of attributes δ^*, which are predetermined at the beginning of the game for semantic security. $\mathcal{S}$ simulates the game for $\mathcal{A}$ as follows.
  - $\mathcal{S}$ sets up $\mathsf{mpk}_{\mathsf{ABET}} = \mathsf{pk}^*$ and completes the remainder of Setup honestly, which includes user's key pairs and chameleon key pairs for hashing in CH. $\mathcal{S}$ returns all public information to $\mathcal{A}$.
  - $\mathcal{S}$ can honestly answer the queries made by $\mathcal{A}$ regarding decryption keys using his given oracle, such that $\Lambda_i(\delta^*) \neq 1, i \neq j^*$. In the g-th query, upon receiving a hash query w.r.t., a hashed message m^* from $\mathcal{A}$. $\mathcal{S}$ first submits two messages (i.e., $[M_0 = \mathsf{sk}_{\mathsf{CH}}, M_1 = \perp]$) to his challenger, and obtains a challenge ciphertext C^* under index j^* and δ^*. Then, $\mathcal{S}$ returns the tuple $(h^*, m^*, \mathsf{r}^*, C^*, c^*, \sigma^*)$ to $\mathcal{A}$. Note that $\mathcal{S}$ can simulate the message-signature pair (c^*, σ^*) honestly using user's key

pairs. Besides, $\mathcal{S}$ can simulate the adapt query successfully using $\mathsf{sk_{CH}}$.

If the encrypted message in C^* is $\mathsf{sk_{CH}}$, then the simulation is consistent with $\mathbb{G}_2$; Otherwise, the simulation is consistent with $\mathbb{G}_3$. Therefore, if the advantage of $\mathcal{A}$ is significantly different in $\mathbb{G}_2$ and $\mathbb{G}_3$, $\mathcal{S}$ can break the semantic security of the ABET. Hence, we have

$$\left|\mathsf{Adv}_2^{GF} - \mathsf{Adv}_3^{GF}\right| \leq \mathsf{Adv}_{\mathcal{S}}^{\mathsf{ABET}}(\lambda). \tag{3}$$

- $\mathbb{G}_4$: This game is identical to game $\mathbb{G}_3$ except that in the g-th query, $\mathcal{S}$ outputs a random bit if $\mathcal{A}$ outputs a valid collision $(h^*, m^{*'}, \mathsf{r}^{*'}, C^{*'}, c^{*'}, \sigma^{*'})$, and it was not previously returned by the Adapt' oracle. Below we show that the difference between $\mathbb{G}_3$ and $\mathbb{G}_4$ is negligible if CH is collision-resistant.

  Let $\mathcal{S}$ denote an attacker against CH with collision-resistant, who is given a chameleon public key pk^* and an Adapt' oracle, aims to find a collision which was not simulated by the Adapt' oracle. $\mathcal{S}$ simulates the game for $\mathcal{A}$ as follows.
  - $\mathcal{S}$ sets up $\mathsf{pk_{CH}} = \mathsf{pk}^*$ for the g-th hash query, and completes the remainder of Setup honestly, which includes user's key pairs and master key pair in ABET. $\mathcal{S}$ returns all public information to $\mathcal{A}$.
  - $\mathcal{S}$ can simulate all queries made by $\mathcal{A}$ except adapt queries. If $\mathcal{A}$ submits an adapt query in the form of $(h, m, \mathsf{r}, C, c, \sigma, m')$, then $\mathcal{S}$ obtains a randomness r' from his Adapt' oracle, and returns $(h, m', \mathsf{r}', C', c', \sigma')$ to $\mathcal{A}$. In particular, $\mathcal{S}$ simulates the g-th hash query as $(h^*, m^*, \mathsf{r}^*, C^*, c^*, \sigma^*)$ w.r.t. a hashed message m^*, where $C^* \leftarrow \mathsf{Enc_{ABET}}(\mathsf{mpk}^*, \perp, \delta^*, j^*)$, c^* is derived from $\perp$ because $Dlog(\mathsf{pk}^*)$ is unknown.
  - If $\mathcal{A}$ outputs a collision $(h^*, m^*, \mathsf{r}^*, C^*, c^*, \sigma^*, m^{*'}, \mathsf{r}^{*'}, C^{*'}, c^{*'}, \sigma^{*'})$ with respect to the g-th query, and all relevant checks are succeed, then $\mathcal{S}$ output $(h^*, m^{*'}, \mathsf{r}^{*'}, C^{*'}, c^{*'}, \sigma^{*'})$ as a valid collision to CH; Otherwise, $\mathcal{S}$ aborts the game. Therefore, we have

$$\left|\mathsf{Adv}_3^{GF} - \mathsf{Adv}_4^{GF}\right| \leq \mathsf{Adv}_{\mathcal{S}}^{\mathsf{CH}}(\lambda). \tag{4}$$

Combining the above results together, we have

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{GF}(\lambda) \quad &\leq \quad n(\lambda) \cdot (\mathsf{Adv}_{\mathcal{S}}^{\mathsf{DPSS}}(\lambda) + \mathsf{Adv}_{\mathcal{S}}^{\mathsf{ABET}}(\lambda) \\ &\qquad + \mathsf{Adv}_{\mathcal{S}}^{\mathsf{CH}}(\lambda)).\end{aligned}$$

□

B.3 Proof of Theorem B.3

Theorem B.3. *The proposed generic framework is accountable if the* Σ *scheme is EUF-CMA secure, and the* DPSS *scheme has correctness.*

Proof. We define a sequence of games $\mathbb{G}_i$, $i = 0, \cdots, 2$ and let Adv_i^{GF} denote the advantage of the adversary in game $\mathbb{G}_i$.

- $\mathbb{G}_0$: This is the original game for accountability.
- $\mathbb{G}_1$: This game is identical to game $\mathbb{G}_1$ except that $\mathcal{S}$ will output a random bit if all honest users in a committee outputs a master secret key $\mathtt{msk}'$ such that $\mathtt{msk}' \neq \mathtt{msk}$, and the setup is honest. The difference between $\mathbb{G}_0$ and $\mathbb{G}_1$ is negligible if DPSS scheme has correctness.

$$\left|\mathsf{Adv}_0^{GF} - \mathsf{Adv}_1^{GF}\right| \leq \mathsf{Adv}_{\mathcal{S}}^{\mathsf{DPSS}}(\lambda). \tag{5}$$

- $\mathbb{G}_2$: This game is identical to game $\mathbb{G}_1$ except that $\mathcal{S}$ will output a random bit if $\mathcal{A}$ outputs a valid forgery σ^*, where σ^* was not previously simulated by $\mathcal{S}$ and the user is honest. The difference between $\mathbb{G}_1$ and $\mathbb{G}_2$ is negligible if Σ is EUF-CMA secure.

  Let $\mathcal{F}$ denote a forger against Σ, who is given a public key pk^* and a signing oracle O^{Sign}, aims to break the EUF-CMA security of Σ. Assume that $\mathcal{A}$ activates at most n users in the system.
  - $\mathcal{F}$ randomly chooses a user in the system and sets up its public key as pk^*. $\mathcal{F}$ completes the remainder of Setup honestly. Below we mainly focus on user pk^* only.
  - To simulate a chameleon hash for message m, $\mathcal{F}$ first obtains a signature σ from his signing oracle O^{Sign}. Then, $\mathcal{F}$ generates chameleon hash and ciphertext honestly because $\mathcal{F}$ chooses the chameleon secret key $\mathsf{sk_{CH}}$, and returns $(m, h, \mathsf{r}, C, \sigma)$ to $\mathcal{A}$. Besides, the message-signature pairs and collisions can be perfectly simulated by $\mathcal{F}$ for any adapt query. $\mathcal{F}$ records all the simulated message-signature pairs by including them to a set Q.
  - When forging attack occurs, i.e., $\mathcal{A}$ outputs $(m^*, h^*, \mathsf{r}^*, C^*, c^*, \sigma^*)$, $\mathcal{F}$ checks whether:
    * the forging attack happens to user pk^*;
    * the ciphertext C^* encrypts the chameleon trapdoor $\mathsf{sk_{CH}}$;
    * the message-signature pair (c^*, σ^*) is derived from the chameleon trapdoor $\mathsf{sk_{CH}}$;
    * the message-signature pair $(c^*, \sigma^*) \notin Q$;
    * $1 \leftarrow \Sigma.\mathsf{Verify}(\mathsf{pk}^*, c^*, \sigma^*)$ and $1 \leftarrow \mathsf{Verify}(\mathsf{pk_{CH}}, m^*, h^*, \mathsf{r}^*)$.

  If all the above conditions hold, $\mathcal{F}$ confirms that it as a successful forgery from $\mathcal{A}$, then $\mathcal{F}$ extracts the forgery via $\sigma \leftarrow M_\Sigma(\mathsf{PP}, \mathsf{pk}^*, \sigma^*, \Delta(\mathsf{sk}))$ due to the homomorphic property of Σ (regarding keys and signatures [49]), where $\Delta(\mathsf{sk})$ is derived from (c, c^*). To this end, $\mathcal{F}$ outputs σ as its own forgery; Otherwise, $\mathcal{F}$ aborts the game.

$$\left|\mathsf{Adv}_1^{GF} - \mathsf{Adv}_2^{GF}\right| \leq n \cdot \mathsf{Adv}_{\mathcal{F}}^{\Sigma}(\lambda). \tag{6}$$

Combining the above results together, we have

$$\mathsf{Adv}_{\mathcal{A}}^{GF}(\lambda) \quad \leq \quad \mathsf{Adv}_{\mathcal{S}}^{\mathsf{DPSS}}(\lambda) + n \cdot \mathsf{Adv}_{\mathcal{F}}^{\Sigma}(\lambda).$$

□

C SECURITY ANALYSIS OF ABET

C.1 Semantic Security

Informally, an ABE scheme is secure against chosen plaintext attacks if no group of colluding users can distinguish between encryption of M_0 and M_1 under an index and a set of attributes δ^* of an attacker's choice as long as no member of the group is authorized to decrypt on her own. The selective security is defined as an index, as well as a set of attributes δ^*, are chosen by attackers at the beginning security experiment. The semantic security model here is based on the selective security model defined in [43].

Theorem C.1. *The proposed ABET scheme is semantically secure in the standard model if the* q'*-type assumption is held in the asymmetric pairing groups.*

Proof. Let $\mathcal{S}$ denote a q'-type problem attacker, who is given the terms from the assumption, aims to distinguish g^{abc} and g^s. The reduction is performed as follows.

- $\mathcal{S}$ simulates master public key $\mathsf{mpk} = (g, u, v, w, h, \hat{e}(g,h)^{\alpha}, \{g_1^{\alpha}, \cdots g_k^{\alpha}\}, \{h_1^{\alpha}, \cdots h_k^{\alpha}\}, g^{\beta}, h^{1/\alpha}, h^{\beta/\alpha}, \hat{e}(g,h)^{\theta/\alpha})$ as follows: $\hat{e}(g,h)^{\alpha} = \hat{e}(g^a, h^b), \{g_1^{\alpha}, \cdots g_k^{\alpha}\} = \{g^{abdz_1}, \cdots, g^{abdz_k}\}, \{h_1^{\alpha}, \cdots h_k^{\alpha}\} = \{h^{abdz_1}, \cdots, h^{abdz_k}\}, h^{1/\alpha} = h^{d/ab}, h^{\beta/\alpha} = h^{\beta d/ab}, \hat{e}(g,h)^{\theta/\alpha} = \hat{e}(g^{\theta}, h^{d/ab})$, and (u, v, w) are simulated using the same method described in [43]. Note that $(\beta, \theta, \{z_1, \cdots, z_k\})$ are randomly chosen by $\mathcal{S}$, and α (or $1/\alpha$) is implicitly assigned as ab (or d/ab) from the q'-type assumption. $\mathcal{A}$ submits a challenge index $j^* = \{I_1, \cdots, I_j\}$ and a set of attributes δ^* to $\mathcal{S}$.
- $\mathcal{S}$ simulates decryption keys $\mathsf{sk}_{\Lambda_i} = (\{\mathsf{sk}_{\tau}\}_{\tau \in [n_1]}, \mathsf{sk}_0, \mathsf{sk}_1, \mathsf{sk}_2)$ (note that $\Lambda_i(\delta^*) \neq 1$) as follows: $\mathsf{sk}_0 = (g^{dt/ab}, g^r)$, $\mathsf{sk}_1 = g^{\theta} \cdot \widehat{i}^t \cdot g^{\beta \cdot r}$, $\mathsf{sk}_2 = \{g_{i-1}^{abdt}, \cdots, g_1^{abdt}\}$, where $\widehat{i} = g_k^{abdI_1} \cdots g_i^{abdI_i} \cdot g$. Note that $\{\mathsf{sk}_{\tau}\}_{\tau \in [n_1]}$ is simulated using the same method described in [43], and (t, r) are randomly chosen by $\mathcal{S}$.
- $\mathcal{S}$ simulates challenge ciphertext $C^* = (ct, \{ct_{\tau,1}, ct_{\tau,2}\}_{\tau \in [\delta]}, ct_0, ct_1)$ as follows: $ct = M_b \oplus \mathsf{H}(T || \hat{e}(g,h)^{\theta cd/ab})$, $ct_0 = (h^c, h^{cd/ab}, h^{\beta cd/ab})$, and $ct_1 = \widehat{j}^c = h_k^{abcdI_1} \cdots h_j^{abcdI_j} \cdot h$. Note that $\{ct_{\tau,1}, ct_{\tau,2}\}_{\tau \in [\delta]}$ are simulated using the same method described in [43], s is implicitly assigned as c from the q'-type assumption, and T can be either $\hat{e}(g,h)^{abc}$ or $\hat{e}(g,h)^s$.

Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. If $\mathcal{A}$ guesses the random bit correctly, then $\mathcal{S}$ can break the q'-type problem. □

C.2 Ciphertext Anonymity

Informally, ciphertext anonymity requires that any third party cannot distinguish the encryption of a chosen message for a first chosen index from the encryption of the same message for a second chosen index. In other words, the attacker cannot decide whether a ciphertext was encrypted for a chosen index or a random index. We prove the ABET scheme has selective ciphertext anonymity (i.e., the index is chosen prior to the security experiment). The ciphertext anonymity (or index hiding) model is referred to [13, 14, 33].

Theorem C.2. *The proposed ABET scheme is anonymous if the Extended Decisional Diffie-Hellman Assumption (eDDH) assumption is held in the asymmetric pairing groups.*

Proof. Let $\mathcal{S}$ denote an eDDH problem distinguisher, who is given $(g, h, g^a, g^b, g^{ab}, h^c, h^{ab}, h^{1/ab}, h^{abc})$, aims to distinguish $h^{c/ab}$ and h^s [49]. The reduction is performed as follows.

- $\mathcal{S}$ simulates master public key $\mathsf{mpk} = (g, u, v, w, h, \hat{e}(g,h)^{\alpha}, \{g_1^{\alpha}, \cdots g_k^{\alpha}\}, \{h_1^{\alpha}, \cdots h_k^{\alpha}\}, g^{\beta}, h^{1/\alpha}, h^{\beta/\alpha}, \hat{e}(g,h)^{\theta/\alpha})$ as follows: $\hat{e}(g,h)^{\alpha} = \hat{e}(g,h)^{ab}, \{g_1^{\alpha}, \cdots, g_k^{\alpha}\} = \{g_1^{ab}, \cdots, g_k^{ab}\}, \{h_1^{\alpha}, \cdots, h_k^{\alpha}\} = \{h_1^{ab}, \cdots, h_k^{ab}\}$, $h^{1/\alpha} = h^{1/ab}$, $h^{\beta/\alpha} = h^{\beta/ab}$, $\hat{e}(g,h)^{\theta/\alpha} = \hat{e}(g,h)^{\theta/ab}$. Note that $\mathcal{S}$ randomly chooses (u, v, w) and $(\beta, \theta, \{z_i\})$, and implicitly sets $\alpha = ab$. $\mathcal{A}$ submits a challenge index $j^* = \{I_1, \cdots, I_j\}$ to $\mathcal{S}$.
- $\mathcal{S}$ simulates a decryption key $\mathsf{sk}_{\Lambda_i} = (\{\mathsf{sk}_{\tau}\}_{\tau \in [n_1]}, \mathsf{sk}_0, \mathsf{sk}_1, \mathsf{sk}_2)$ with respect to index $i = \{I_1, \cdots, I_i\}$ as follows: $\mathsf{sk}_0 = (g^t, g^{\beta \cdot r} \cdot g^{-\Sigma(z_i I_i) t \cdot b})$, $\mathsf{sk}_1 = g^{\theta} \cdot g^{a \cdot r}$, where $(t, r) \in \mathbb{Z}_q$ are randomly chosen by $\mathcal{S}$. The components $(\{\mathsf{sk}_{\tau}\}_{\tau \in [n_1]}, \mathsf{sk}_2)$ are honestly simulated by $\mathcal{S}$. The simulated components $(g^t, g^{\beta \cdot r} \cdot g^{-\Sigma(z_i I_i) t \cdot b}, g^{\theta} \cdot g^{a \cdot r})$ are correctly distributed, because $g^{\theta} \cdot g^{a \cdot r} = g^{\theta} \cdot \widehat{i}^t \cdot g^{a[\beta \cdot r - \Sigma(z_i I_i) t \cdot b]} = g^{\theta} \cdot \widehat{i}^t \cdot g^{a \cdot \overline{r}}$, where $\overline{r} = \beta \cdot r - \Sigma(z_i I_i) t \cdot b$, and $\widehat{i} = g_k^{abI_1} \cdots g_i^{abI_i} \cdot g$. So, the simulated components $(g^t, g^{\overline{r}}, g^{\theta} \cdot \widehat{i}^t \cdot g^{a \cdot \overline{r}})$ match the real distribution.
- $\mathcal{S}$ simulates the challenge ciphertext $C^* = (ct, \{ct_{\tau,1}, ct_{\tau,2}\}_{\tau \in [\delta]}, ct_0, ct_1)$ with respect to index j^* as follows: $ct = M_b \oplus \mathsf{H}(\hat{e}(g,h)^{abc} || \hat{e}(g,T)^{\beta})$, $ct_0 = (h^c, T, T^{\beta})$, and $ct_1 = h_k^{abcI_1} \cdots h_j^{abcI_j} \cdot h^c$. Note that $\mathcal{S}$ simulates $\{ct_{\tau,1}, ct_{\tau,2}\}_{\tau \in [\delta]}$ honestly, and T can be either $h^{c/ab}$ or h^s.

Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. If $\mathcal{A}$ guesses the random bit correctly, then $\mathcal{S}$ can break the eDDH problem. □