Policy-based Chameleon Hash for Blockchain Rewriting with Black-box Accountability

Yangguang Tian
Singapore University of Technology and Design
Singapore
yangguang_tian@stud.edu.sg

Nan Li
University of Newcastle
Australia
nan.li@newcastle.edu.au

Yingjiu Li
University of Oregon
USA
yingjiul@uoregon.edu

Pawel Szalachowski
Singapore University of Technology and Design
Singapore
pawel@sutd.edu.sg

Jianying Zhou
Singapore University of Technology and Design
Singapore
jianying_zhou@sutd.edu.sg

ABSTRACT

Policy-based chameleon hash is a useful primitive for blockchain rewriting. It allows a party to create a transaction associated with an access policy, while another party who possesses enough rewriting privileges satisfying the access policy can rewrite the transaction. However, it lacks accountability. The chameleon trapdoor holder may abuse his/her rewriting privilege and maliciously rewrite the hashed object in the transaction without being identified. In this paper, we introduce policy-based chameleon hash with black-box accountability (PCHBA). Black-box accountability allows an attribute authority to link modified transactions to responsible transaction modifiers in case of dispute, in which any public user identifies those transaction modifiers from interacting with an access device/blackbox. We first present a generic framework of PCHBA. Then, we present a practical instantiation, showing its practicality through implementation and evaluation analysis.

KEYWORDS

Chameleon Hash, Blockchain Rewriting, Black-box Accountability

ACM Reference Format:
Yangguang Tian, Nan Li, Yingjiu Li, Pawel Szalachowski, and Jianying Zhou. 2020. Policy-based Chameleon Hash for Blockchain Rewriting with Black-box Accountability. In *Annual Computer Security Applications Conference (ACSAC 2020), December 7–11, 2020, Austin, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3427228.3427247

1 INTRODUCTION

Blockchains have received tremendous attention from research communities and industries in recent years. A blockchain, as an append-only data structure combined with a consensus algorithm, was first introduced in the context of Bitcoin [38], where all payment transactions are appended to a public ledger, and each transaction is verified by network nodes in a peer-to-peer manner. A blockchain grows by one block at a time, where each new block in the chain is decided by a consensus mechanism (e.g., Proof-of-Work in Bitcoin [27]) executed by the network nodes. Blockchains can be viewed as hash-chains, where the hash of a block is linked to the next block in the chain. Each block includes a set of valid transactions that are usually accumulated into a single hash value using the Merkle tree [36], and each transaction contains an object of interest that needs to be registered in the blockchain.

A blockchain is designed to be immutable, such that registered objects cannot be modified once they appear in the blockchain. However, blockchain rewriting is necessary in practice, or even legally obliged [4]. It is possible certain users append transactions into a chain containing illicit objects such as sensitive information, stolen private keys, and inappropriate videos [34, 35]. The existence of illicit objects in the chain could pose a challenge to law enforcement agencies like Interpol [46].

Blockchain rewriting is possible if the trapdoor-based chameleon hash [29] replaces the traditional hash function used in blockchain. A user who holds a chameleon trapdoor can rewrite registered objects in the blockchain. Such trapdoor holder can modify a registered object without changing its hash output, and without breaking the link of the hash-chain. There are two types of blockchain rewriting. Type one is block-level rewriting, an entire block in the hash-chain is modified, where the chameleon hash replaces the traditional hash used in the root node of the Merkle tree for that block.

Type two is transaction-level rewriting, which means the rewriting occurs on a specific transaction inside a block. Derler et al. [23] introduced policy-based chameleon hash (PCH) for blockchain rewriting, which includes the following features. First, PCH supports transaction-level blockchain rewriting. A party uses PCH to hash an object for creating a transaction, then another party (i.e., transaction modifier) with PCH trapdoor can modify the hashed object of the transaction. Second, PCH achieves fine-grained blockchain rewriting by employing attribute-based encryption (ABE) [13]. Specifically, an access policy is embedded into a transaction and

ACSAC 2020, December 7–11, 2020, Austin, USA

ACM ISBN 978-1-4503-8858-0/20/12…$15.00
https://doi.org/10.1145/3427228.3427247

a transaction modifier possessing a PCH trapdoor can modify the transaction if her attributes corresponding the PCH trapdoor satisfy the embedded access policy.

Motivation. PCH-based blockchain rewriting yields a security concern: a transaction modifier may abuse her rewriting privilege, and maliciously rewrite the blockchain without being identified. The transaction modifier can deny her malicious rewriting on a transaction, as other transaction modifiers with different PCH trapdoors may satisfy the same access policy embedded in the transaction. This security threat exists because the cryptographic primitive ABE used in PCH has an inherent property: anonymity.

Accountability is essential to blockchain rewriting. It is desirable that a transaction modifier A who has maliciously modified a transaction, is unable to accuse another transaction modifier B whose PCH trapdoor satisfies the same policy of modifying the transaction. Accountability in blockchain rewriting should enable a third party to resolve any dispute over modified transactions.

This Work. We introduce Policy-based Chameleon Hash with Black-box Accountability PCHBA, which is used to secure blockchain rewriting. PCHBA achieves anonymity and accountability together, such that a transaction modifier remains anonymous if there is no dispute over the modified transactions. In particular, PCHBA considers black-box accountability for blockchain rewriting in a permissioned setting (e.g., in Ripple [7] and Hyperledger [5]). The black-box accountability has two levels of meanings: 1) any public user can identify a set of accused transaction modifiers whose PCH trapdoors have been used in generating an *access device/blackbox*. The access blackbox includes various rewriting privileges from different transaction modifiers, which helps those accused transaction modifiers to get better financial gain in the market or have fewer chance of getting caught. 2) An attribute authority (AA) can link a modified transaction to a responsible transaction modifier (i.e., one-to-one mapping) in case of dispute over this modified transaction. Since different modifiers may modify the same transaction in the PCH-based blockchain rewriting, we allow AA to link each modified version of the transaction to a responsible transaction modifier in case of dispute over many modified versions of the same transaction.

Our Contributions. The major contributions of this work are summarized as follows.

- *Generic Framework.* We introduce the *first* generic framework of policy-based chameleon hash with black-box accountability PCHBA for blockchain rewriting. A unique feature of this framework is that it disallows a transaction modifier to deny her malicious rewriting on any transaction modified by the modifier. This property of accountability of our proposed framework is essential to blockchain rewriting because it helps thwart malicious rewriting by any party who has the rewriting privilege.
- *Practical Instantiation.* We present a *practical* instantiation of PCHBA, and validate its practicality with implementation and evaluation. The evaluation shows that our instantiation incurs a moderate performance overhead as compared to PCH in [23]. Besides, we show that PCHBA can be effectively integrated into the mutable blockchains.

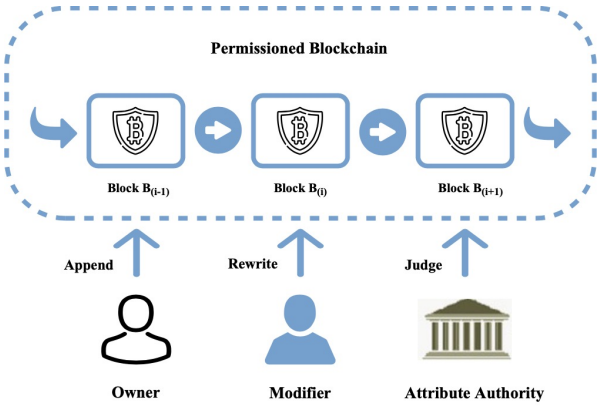

Figure 1: Accountable blockchain rewriting.

2 OVERVIEW

In this section, we provide an overview of PCHBA and the intuition of behind it. The PCHBA is constructed based on attribute-based encryption with black-box traceability (ABET), chameleon hash with ephemeral trapdoor (CHET), and digital signature. To show the intuition of constructing PCHBA, we consider a simple system that includes a transaction owner, a transaction modifier, and an attribute authority, as shown in Figure 1. When an owner appends a mutable transaction to the blockchain, he outputs a policy-based chameleon hash that additionally includes a ciphertext and a signature. In this process, the owner encrypts an ephemeral trapdoor, and signs on a randomized version of the ephemeral trapdoor. The fine-grained blockchain rewriting can be achieved by restricting who can decrypt ephemeral trapdoor. To generate a signature in the policy-based chameleon hash, the owner randomizes the ephemeral trapdoor using his signing key and signs on it. Later, a modifier is allowed to rewrite the mutable transaction if the modifier possesses both the chameleon secret key given by the attribute authority and the ephemeral trapdoor by decrypting the ciphertext in the policy-based chameleon hash. To complete the process of rewriting, the modifier also randomizes the same ephemeral trapdoor using her signing key and signs on it. The intention is to use the same ephemeral trapdoor to link message-signature pairs so as to achieve accountability.

We achieve black-box accountability via the following steps. The first step is to find the linked transactions. Given a set of transactions, any public user can link a transaction to its modified version if it exists in the set via the corresponding message-signature pairs. The public can link these message-signature pairs because they are associated with the same ephemeral trapdoor. The second step is to find the accused modifiers. Given an access blackbox, any public user can obtain a set of accused modifiers from interacting with the access blackbox due to ABET's black-box traceability. The third step is to link the modified transaction to a responsible modifier. Since the anonymity of PCHBA ensures that a mutable transaction leaks no information about the transaction modifier's identity to the public, only the attribute authority (AA) can link the modified transaction to a responsible transaction modifier using its master secret key.

The ABET scheme is a critical component in constructing PCHBA. We show a generic approach to construct the ABET scheme from a ciphertext-policy attribute-based encryption (CP-ABE) scheme and an anonymous hierarchy identity-based encryption (HIBE) scheme. The proposed ABET scheme works as follows: 1) the master key pair combines the key pairs from CP-ABE and HIBE; 2) the decryption key is associated with a set of attributes and identity at depth i in the hierarchy of identities in HIBE, and the ciphertext is associated with an access policy and another identity at depth j in the hierarchy; 3) the decryption works if the attribute set satisfies the access policy and if the depth $j \leq i$ holds. Note that depth i is relatively close to the root node compared to depth j, and depth j is relatively close to the leaf node. The intention is to let the decryption key at a higher-level decrypt the ciphertext at a lower-level. Such condition $j \leq i$ is also used to ensure the black-box traceability so that any public user can obtain a set of accused identities from interacting with an access blackbox.

For constructing a practical ABET, we rely on the most efficient CP-ABE scheme [9] and a HIBE scheme [14]. We extend the intertwined ABET to an anonymized version using asymmetric pairings, i.e., $\hat{\mathsf{e}} : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$. The basic idea is, the identity-based elements in a modifier's decryption key belong to group $\mathbb{G}$. The identity-based elements in a ciphertext belong to group $\mathbb{H}$ so that the ciphertext can conceal the modifier's identity if the master secret key is unknown. We prove the semantic security of the proposed ABET scheme under a new composite assumption: decisional linear exponent (DLE), which is based on standard decisional linear (DLIN) [9] and bilinear Diffie-hellman exponent (BDHE) [14]. We also prove the ABET scheme's ciphertext anonymity under another new assumption: extended decisional Diffie-hellman (eDDH), which extends the standard DDH in the asymmetric setting. The proposed ABET scheme and the new assumptions can be regarded as the main technical contribution of this work.

3 PRELIMINARIES

In this section, we present the complexity assumptions and the building blocks, which are used in our proposed PCHBA protocol.

3.1 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 $(q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \hat{\mathsf{e}})$, where q is a prime number, $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$ are cyclic groups of order q, and $\hat{\mathsf{e}} : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$ is a bilinear map such that: (1) Bilinearity: $\forall g, h \in \mathbb{G}$ and $a, b \in \mathbb{Z}_q$, we have $\hat{\mathsf{e}}(g^a, h^b) = e(g, h)^{ab}$; (2) Non-degeneracy: $\exists g \in \mathbb{G}$ such that $\hat{\mathsf{e}}(g, h)$ has order q in $\mathbb{G}_T$. We assume that group operations in $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$ and bilinear map $\hat{\mathsf{e}}$ are computable in polynomial time with respect to λ. We refer to $\mathbb{G}$ and $\mathbb{H}$ as the source groups and $\mathbb{G}_T$ as the target group.

We introduce a new composite assumption, which is based on decisional linear assumption [9] and bilinear Diffie-hellman exponent assumption [14]. We use it to prove the semantic security of the proposed attribute-based encryption with black-box traceability (ABET) scheme.

Definition 3.1 (Decisional Linear Exponent (DLE)). Given group generators $g \in \mathbb{G}$ and $h \in \mathbb{H}$, define the following distribution:

$$
\begin{aligned}
\mathsf{Adv}^{DLE}_{\mathcal{A}} \quad = \quad & |\Pr[\mathsf{Adv}(1^\lambda, \mathsf{par}, D, T_0) = 1] \\
& - \Pr[\mathsf{Adv}(1^\lambda, \mathsf{par}, D, T_1) = 1]|, \textit{where} \\
& \mathsf{par} = (q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \hat{\mathsf{e}}, g, h) \leftarrow \mathsf{GroupGen}(1^\lambda) \\
& a_1, a_2, s_1, s_2 \leftarrow \mathbb{Z}_q^*, s \leftarrow \mathbb{Z}_q; \\
& D = (g^{a_1}, g^{a_2}, h^{a_1}, h^{a_2}, g^{a_1 \cdot s_1^k}, g^{a_2 \cdot s_2^k}, h^{a_1 \cdot s_1^k}, h^{a_2 \cdot s_2^k}, \\
& \underline{\{g^{s_1}, \cdots, g^{s_1^{k-1}}, g^{s_1^{k+1}}, \cdots, g^{s_1^{2k}}\}}, \\
& \underline{\{g^{s_2}, \cdots, g^{s_2^{k-1}}, g^{s_2^{k+1}}, \cdots, g^{s_2^{2k}}\}}); \\
& T_0 = (g^{s_1^k + s_2^k}, h^{s_1^k + s_2^k}), T_1 = (g^s, h^s).
\end{aligned}
$$

The DLE assumption is secure if $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ is negligible in λ.

Since $(g^{s_1^k}, g^{s_2^k})$ or $(h^{s_1^k}, h^{s_2^k})$, are missing from D, the additional terms (underline part) are no help in computing $g^{s_1^k + s_2^k}$ or $h^{s_1^k + s_2^k}$. Now, we introduce an extended version of the decisional Diffie-hellman assumption, which is used to prove the ciphertext anonymity of the proposed ABET scheme.

Definition 3.2 (Extended Decisional Diffie-Hellman (eDDH)). Given group generators $g \in \mathbb{G}$ and $h \in \mathbb{H}$, define the following distribution:

$$
\begin{aligned}
\mathsf{Adv}^{eDDH}_{\mathcal{A}} \quad = \quad & |\Pr[\mathsf{Adv}(1^\lambda, \mathsf{par}, D, T_0) = 1] \\
& - \Pr[\mathsf{Adv}(1^\lambda, \mathsf{par}, D, T_1) = 1]|, \textit{where} \\
& \mathsf{par} = (q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \hat{\mathsf{e}}, g, h) \leftarrow \mathsf{GroupGen}(1^\lambda) \\
& a, b, c \leftarrow \mathbb{Z}_q^*, s \leftarrow \mathbb{Z}_q; D = (g^a, g^b, \underline{g^{ab}, h^c, h^{ab}}, \\
& \underline{h^{1/ab}, h^{abc}}); T_0 = h^{c/ab}, T_1 = h^s.
\end{aligned}
$$

The eDDH assumption is secure if $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ is negligible in λ.

We prove the proposed DLE and eDDH assumptions in the generic group model [44]. The detailed theorems and proofs are referred to Appendix A.

3.2 Ciphertext-policy Attribute-based Encryption with Black-box Traceability

Access Structure. Let attribute universe be $\mathcal{U}$. An access structure Λ is a collection of non-empty subsets of $\mathcal{U}$ (i.e., $\Lambda \subseteq 2^{\mathcal{U}} \backslash \{\phi\}$). It is called monotone if $\forall B, C$: if $B \in \Lambda$ and $B \subseteq C$ then $C \in \Lambda$.

Monotone Span Program (MSP). A secret-sharing scheme $\prod$ with domain of secrets realizing access structure Λ is called linear over $\mathbb{Z}_q$ if: (1) The shares of a secret $s \in \mathbb{Z}_q$ for each attribute form a vector over $\mathbb{Z}_q$; (2) For each access structure Λ, there exist a matrix $\mathbf{M}$ with n_1 rows and n_2 columns called the share-generating matrix for $\prod$. For $u = 1, ..., n_1$, we define a function π labels row u of $\mathbf{M}$ with attribute $\pi(u)$ from the attribute universe $\mathcal{U}$. We consider the column vector $\vec{v} = (s, r_2, ..., r_{n_2})^\top$, where $s \in \mathbb{Z}_q$ is the secret to be shared and $r_2, ..., r_{n_2} \in \mathbb{Z}_q$ are chosen at random. Then $\mathbf{M}\vec{v} \in \mathbb{Z}_q^{n_1 \times 1}$ is the vector of n_1 shares of the secret s according to $\prod$. The share $(\mathbf{M}\vec{v})_u$ belongs to attribute $\pi(u)$, where $u \in [n_1]$.

According to [12], every linear secret-sharing scheme has the linear reconstruction property, which is defined as follows: we

assume that $\prod$ is a MSP for the access structure Λ, $1 = \Lambda(\delta)$ is an authorized set and let $I \subset \{1, 2, ..., n_1\}$ be defined as $I = \{u \in [n_1] \wedge \pi(u) \in \delta\}$. There exist the constants $\{\gamma_u \in \mathbb{Z}_q\}_{u \in I}$ such that for any valid share $\{\lambda_u = (\mathbf{M}\vec{v})_u\}_{u \in I}$ of a secret s according to $\prod$, $\sum_{u \in I} \gamma_u \lambda_u = s$. Meanwhile, these constants $\{\gamma_u\}_{u \in I}$ can be found in time polynomial in the size of the share-generating matrix $\mathbf{M}$. For any unauthorized set δ', no such $\{\gamma_u\}$ exist.

Ciphertext-policy Attribute-based Encryption with Black-box Traceability. It consists of the following algorithms [32]: ABET = (Setup, KeyGen, Enc, Dec, Trace). It assumes an index (or identity) space: $\{1, \cdots, k\}$, where k denotes the total number of users in the system.

- Setup(1^λ): It takes a security parameter λ as input, outputs a master key pair (msk, mpk).
- KeyGen(msk, δ): It takes the master secret key msk, a user's attribute set $\delta \in \mathcal{U}$ as input, outputs a decryption key $ssk_{i,\delta}$ (or ssk_i), which is assigned by a unique index i.
- Enc(mpk, m, Λ, j): It takes the master public key mpk, a message m, an access structure Λ, and an index $j \in \{1, k+1\}$ as input, outputs a ciphertext C. Note that C contains Λ, not index j.
- Dec(mpk, C, ssk_i): It takes the master public key mpk, a ciphertext C, and the decryption key ssk_i as input, outputs the message m if $(1 = \Lambda(\delta)) \wedge (j \leq i)$.
- Trace(mpk, $\mathcal{D}$, ϵ): It takes the master public key pair mpk, a key-like decryption device $\mathcal{D}$ associated with an attribute set $\delta_{\mathcal{D}}$, and a parameter $\epsilon > 0$ as input, outputs a set of indexes $\mathbb{K}_T \in \{1, \cdots, k\}$, where ϵ is polynomially related to λ, and $\mathbb{K}_T$ denotes the index set of the decryption keys of the accused users.

Black-box Traceability. Key-like decryption device means that certain users generate a decryption device using their decryption keys, and distribute or sell it to the public. The device works as a decryption black-box, and the decryption works if a ciphertext access policy can be satisfied by the attribute set $\delta_{\mathcal{D}}$ (or decrypt using any of the decryption keys associated with $\delta_{\mathcal{D}}$ in the black-box). Given a key-like decryption device that includes a set of decryption keys, the tracing algorithm, which treats the decryption device as an oracle, can identify the accused users whose decryption keys have been used in constructing the decryption device.

Specifically, any public user generates a ciphertext on a message under a policy satisfying $\delta_{\mathcal{D}}$, and an index $j \in \{1, \cdots, k+1\}$. Then, the public sends the ciphertext to the decryption black-box and checks whether the decryption is successful. If succeeds, the public outputs the index; otherwise, it generates a new ciphertext under another policy and index. The public repeats this process until finding a set of indexes $\mathbb{K}_T \in \{1, \cdots, k\}$. The tracing algorithm is related to the traitor tracing algorithm used in the broadcast encryption; the detailed description of the tracing algorithm is referred to [16, 18].

The ABET scheme requires that the encryptor generates a ciphertext on message m, which is associated with a policy and a hidden index j. The decryptor decrypts message m if the policy is satisfied by her attribute set, and $j \leq i$. We stress that the hidden index (or index-hiding) is critical to ABET. On the one hand, the index-hiding ensures that a ciphertext created using index j reveals no information about j. On the other hand, we use index $j \in \{1, \cdots, k+1\}$ in generating ciphertext for tracing. In this work, we denote index-hiding as ciphertext anonymity, we denote key-like decryption device as access device/blackbox because it accumulates various rewriting privileges to perform blockchain rewriting.

3.3 Digital Signature

A digital signature scheme Σ = (Setup, KeyGen, Sign, Verify) is homomorphic, if the following conditions are held.

- *Simple Key Generation.* It means $(sk, vk) \leftarrow$ KeyGen(pp) and $pp \leftarrow$ Setup(1^λ), where vk is derived from sk via a deterministic algorithm $vk \leftarrow$ KeyGen$'(pp, sk)$.
- *Linearity of Keys.* It requires KeyGen$'(pp, sk + \Delta(sk)) = M_{vk}(pp,$ KeyGen$'(pp, sk), \Delta(sk))$, where M_{vk} denotes a deterministic algorithm which takes pp, a verification key vk and a "shifted" value $\Delta(sk)$ as input, outputs a "shifted" verification key vk'. Δ denotes the difference or shift between two keys.
- *Linearity of Signatures.* Two distributions are identical: $\{\sigma' \leftarrow$ Sign$(pp, sk + \Delta(sk), m)\}$ and $\{\sigma' \leftarrow M_\Sigma(pp, vk, m, \sigma, \Delta(sk))\}$, where $\sigma \leftarrow$ Sign(pp, sk, m), and M_Σ denotes a deterministic algorithm which takes pp, a verification key vk, a message-signature pair (m, σ) and a "shifted" value $\Delta(sk)$ as input, outputs a "shifted" signature σ'.
- *Linearity of Verifications.* It requires Verify$(pp, M_{vk}(pp, vk, \Delta(sk)),$ $m, M_\Sigma(pp, vk, m, \sigma, \Delta(sk))) = 1$, and Verify$(pp, vk, m, \sigma) = 1$.

Matsuda et al. [33] have shown that the Schnorr signature scheme [42] satisfies the homomorphic properties regarding keys and signatures, which is useful in finding the connection between a transaction and its modified versions.

4 SYSTEM MODEL

In this section, we present the system model for accountable blockchain rewriting. Then, we present the definition and the security model for the policy-based chameleon hash with black-box accountability (PCHBA).

4.1 System Model

The system model for accountable blockchain rewriting involves three types of users: attribute authority (AA), transaction owner, and transaction modifier. In terms of decentralized setting, every user can play the role of an attribute authority and tag other users with attributes. We use transaction "owner" and "modifier" in presenting PCHBA. In particular, the number of transaction modifiers is assumed to be a small amount because rewriting in blockchains cannot be performed by the majority of system users. We let a set of transaction modifiers combine their rewriting privileges to generate an access device/blackbox.

In Figure 1, a transaction owner appends the hashed objects to the blockchain. Later, a transaction modifier whose rewriting privilege is granted by AA, is required to modify the hashed objects, while the link of hash-chain remains intact. If a dispute occurs on a modified transaction, AA can decide whether the modified transaction is indeed modified by that transaction modifier.

4.2 Definition

A policy-based chameleon hash with black-box accountability consists of the following algorithms.

- $\mathsf{Setup}(1^\lambda)$: It takes a security parameter λ as input, outputs a chameleon hash key pair $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{KeyGen}(\mathsf{sk}, \delta)$: It takes the chameleon secret key sk, and a set of attributes $\delta \in \mathcal{U}$ as input, outputs a secret key sk_δ, which is indexed by an identity ID'.
- $\mathsf{Hash}(\mathsf{pk}, m, \Lambda, ID)$: It takes the chameleon public key pk, a message $m \in \mathcal{M}$, a policy Λ, and an identity ID as input, outputs a chameleon hash h, randomness r, and signature σ. The chameleon hash h appears in the blockchain after this procedure. Note that $\mathcal{M} = \{0,1\}^*$ denote a general message space.
- $\mathsf{Verify}(\mathsf{pk}, m, h, r, \sigma)$: It takes the chameleon public key pk, a message m, chameleon hash h, randomness r, and signature σ as input, output a bit b.
- $\mathsf{Adapt}(\mathsf{sk}_\delta, m, m', h, r, \sigma, ID')$: It takes the secret key sk_δ, messages m and m', chameleon hash h, randomness r, signature σ, and identity ID' as input, outputs a randomness r' if $1 = \Lambda(\delta)$ and $ID \leq ID'$.
- $\mathsf{Judge}(\mathsf{sk}, \mathcal{T}, \mathcal{O})$: It takes the chameleon secret key sk, a set of transactions $\mathcal{T}$, and a set of accused identities from an access blackbox $\mathcal{O}$ as input, outputs a (or more) linked transaction-identity pair.

Correctness. The PCHBA is *correct* if for all security parameters λ, for all $\delta \in \mathcal{U}$, for all $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)$, for all $\delta \in \Lambda$, for all $ID \leq ID'$, for all $\mathsf{sk}_\delta \leftarrow \mathsf{KeyGen}(\mathsf{sk}, \delta)$, for all $m \in \mathcal{M}$, for all $(h, r, \sigma) \leftarrow \mathsf{Hash}(\mathsf{pk}, m, \Lambda, ID)$, for all $m' \in \mathcal{M}$, for all $r' \leftarrow \mathsf{Adapt}(\mathsf{sk}_\delta, m, m', h, r, \sigma, ID')$, we have $1 = \mathsf{Verify}(\mathsf{pk}, m, h, r, \sigma) = \mathsf{Verify}(\mathsf{pk}, m', h, r', \sigma')$.

4.3 Security Model

In the permisssioned blockchain, the transaction owner is an honest party. When he appends a transaction, it is his best interest that this happens correctly and that its transaction could be rewritten if required. We assume the attribute authority is an honest party in the system. We consider three security guarantees, including indistinguishability, collision-resistance, and accountability.

- Indistinguishability. Informally, adversary cannot decide whether for a chameleon hash its randomness was freshly generated using Hash algorithm or was created using Adapt algorithm. 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 left-or-right HashOrAdapt oracle, which ensures that the randomness does not reveal whether it was obtained from Hash or Adapt algorithm. The hashed messages are adaptively chosen from the same message space $\mathcal{M}$ by $\mathcal{A}$. The identity is chosen from an index space: $ID \leftarrow \{1, \cdots, k\}$, where k denotes the total number of users.
  We require $\mathsf{Verify}(\mathsf{pk}, m', h_0, r_0, \sigma_0) = \mathsf{Verify}(\mathsf{pk}, m, h_1, r_1, \sigma_1) = 1$, and we define the advantage of the adversary as
  $$\mathsf{Adv}^{\mathsf{IND}}_{\mathcal{A}}(\lambda) = |\Pr[\mathcal{S} \rightarrow 1] - 1/2|.$$
  Definition 4.1. A PCHBA scheme is indistinguishable if for any probabilistic polynomial-time (PPT) $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IND}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

Experiment $\mathsf{Exp}^{\mathsf{IND}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt}(\mathsf{sk}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, b)}(\mathsf{pk})$
  where $\mathsf{HashOrAdapt}(\mathsf{sk}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, b)$ on input m, m', Λ, δ, ID, ID' :
    $(h_0, r_0, \sigma_0) \leftarrow \mathsf{Hash}(\mathsf{pk}, m', \Lambda, ID')$
    $(h_1, r_1, \sigma_1) \leftarrow \mathsf{Hash}(\mathsf{pk}, m, \Lambda, ID)$
    $\mathsf{sk}_\delta \leftarrow \mathsf{KeyGen}(\mathsf{sk}, \delta)$
    $r_1 \leftarrow \mathsf{Adapt}(\mathsf{sk}_\delta, m, m', h_1, r_1, ID)$
    return (h_b, r_b, σ_b)
return 1, if $b' = b$; else, return 0.

Figure 2: Indistinguishability.

Experiment $\mathsf{Exp}^{\mathsf{CR}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $Q_1, Q_2, Q_3 \leftarrow \emptyset$
$(m^*, r^*, m^{*'}, r^{*'}, h^*, \sigma^*, \sigma^{*'}) \leftarrow \mathcal{A}^{O}(\mathsf{pk})$
  where $O \leftarrow \{\mathsf{KeyGen}, \mathsf{KeyGen}', \mathsf{Hash}, \mathsf{Adapt}\}$
  and $\mathsf{KeyGen}(\mathsf{sk}, \cdot)$ on input δ :
    $\mathsf{sk}_\delta \leftarrow \mathsf{KeyGen}(\mathsf{sk}, \delta)$
    $Q_1 \leftarrow Q_1 \cup \{\delta\}$
    return sk_δ
  and $\mathsf{KeyGen}'(\mathsf{sk}, \cdot)$ on input δ :
    $\mathsf{sk}_\delta \leftarrow \mathsf{KeyGen}(\mathsf{sk}, \delta)$
    $Q_2 \cup \{(i, \mathsf{sk}_\delta)\}$
    $i \leftarrow i + 1$
  and $\mathsf{Hash}(\mathsf{pk}, \cdot, \cdot, \cdot)$ on input m, Λ, ID :
    $(h, r, \sigma) \leftarrow \mathsf{Hash}(\mathsf{pk}, m, \Lambda, ID)$
    $Q_3 \leftarrow Q_3 \cup \{(h, m, \Lambda)\}$
    return (h, r, σ)
  and $\mathsf{Adapt}(\mathsf{pk}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot)$ on input m, m', h, r, j, σ, ID' :
    return $\perp$, if $(j, \mathsf{sk}_\delta) \notin Q_2$ for some sk_δ
    $r' \leftarrow \mathsf{Adapt}(\mathsf{pk}, \mathsf{sk}_\delta, m, m', h, r, \sigma, ID')$
    if $(h, m, \Lambda) \in Q_3$ for some Λ,
    let $Q_3 \leftarrow Q_3 \cup \{(h, m', \Lambda)\}$
    return r'
return 1, if
$1 = \mathsf{Verify}(\mathsf{pk}, m^*, h^*, r^*, \sigma^*) = \mathsf{Verify}(\mathsf{pk}, m^{*'}, h^*, r^{*'}, \sigma^{*'})$
$\wedge (h^*, \cdot, \Lambda) \in Q_3$, for some $\Lambda \wedge m^* \neq m^{*'} \wedge \Lambda \cap Q_1 = \emptyset \wedge (h^*, m^*, \cdot) \notin Q_3$
else, return 0.

Figure 3: Collision-Resistance.

- Collision-Resistance. Informally, an adversary can find collisions for a chameleon hash if she possesses a secret key satisfies a policy embedded in that 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 attributes (i.e., KeyGen' oracle and Adapt oracle).
  We define the advantage of the adversary as
  $$\mathsf{Adv}^{\mathsf{CR}}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A} \rightarrow 1].$$
  Definition 4.2. A PCHBA scheme is collision resistant if for any PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{CR}}_{\mathcal{A}}(\lambda)$ is negligible in λ.
- Accountability. Informally, an adversary cannot generate a bogus message-signature pair for a chameleon hash, in which the chameleon hash links to an accused identity. However, the message-signature pair has never been generated by the accused identity. We define a formal experiment in Figure 4. We allow $\mathcal{A}$ to see whether a chameleon hash links to an identity (i.e., Judge oracle). Let set Q record the chameleon hashes generated by the Judge oracle.
  We denote $T = (h, m, r, \sigma)$ and $T' = (h, m', r', \sigma')$ as original and modified transactions with respect to chameleon hash h, respectively. We also denote a linked transaction-identity pair as (T', ID'). We define the advantage of the adversary as
  $$\mathsf{Adv}^{\mathsf{ACT}}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A} \rightarrow 1].$$

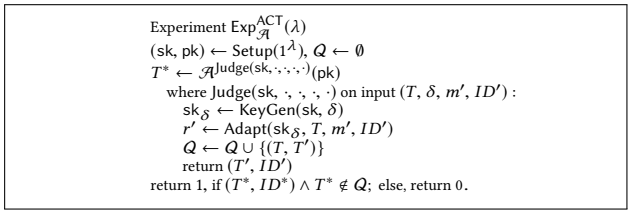

Figure 4: Accountability.

Definition 4.3. A PCHBA scheme is accountable if for any PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ACT}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

Remark. We compare our proposed security models with the closely related security models. First, the indistinguishability and collision-resistance models are derived from the strong indistinguishability and insider collision-resistance models defined in [23], respectively. The derived indistinguishability and collision-resistance models are nearly the same as defined in [23]. Second, our proposed accountability model is derived from the sanitizer accountability model defined in [41]. Note that our proposed accountability model grants a judge oracle to attackers, which determines whether or not a modified transaction links to a responsible modifier.

5 GENERIC CONSTRUCTION

The proposed generic construction consists of the following building blocks.

- An indistinguishable and collision-resistance secure chameleon hash with ephemeral trapdoor scheme CHET= (Setup, KeyGen, Hash, Verify, Adapt).
- An adaptive[1] secure ciphertext-policy attribute-based encryption with black-box traceability scheme ABET=(Setup, KeyGen, Enc, Dec, Trace).
- An existential unforgeability under chosen message attack EUF-CMA secure digital signature scheme Σ=(Setup, KeyGen, Sign, Verify).

We use an attribute authority AA, a transaction owner ID, and a transaction modifier ID' to present our proposed generic construction. We define a deterministic algorithm in Σ as $vk \leftarrow \mathsf{KeyGen}'(pp, 0, sk)$, which takes one public value and two secret values as input.

- $\mathsf{Setup}(1^\lambda)$: AA takes a security parameter λ as input, outputs a chameleon secret key $\mathsf{sk} \leftarrow (\mathsf{msk}_{\mathsf{ABET}}, \mathsf{sk}_{\mathsf{CHET}})$ and a chameleon public key $\mathsf{pk} \leftarrow (\mathsf{pk}_{\mathsf{CHET}}, \mathsf{mpk}_{\mathsf{ABET}}, pp)$, where $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \leftarrow \mathsf{KeyGen}_{\mathsf{CHET}}(\mathsf{PP}_{\mathsf{CHET}})$, $\mathsf{PP}_{\mathsf{CHET}} \leftarrow \mathsf{Setup}_{\mathsf{CHET}}(1^\lambda)$, $(\mathsf{msk}_{\mathsf{ABET}}, \mathsf{mpk}_{\mathsf{ABET}}) \leftarrow \mathsf{Setup}_{\mathsf{ABET}}(1^\lambda)$ and $pp \leftarrow \mathsf{Setup}_\Sigma(1^\lambda)$.
- $\mathsf{KeyGen}(\mathsf{sk}, \delta)$: AA takes the chameleon secret key sk, a set of attributes δ as input, outputs a secret key $\mathsf{sk}_\delta \leftarrow (\mathsf{sk}_{\mathsf{CHET}}, ssk)$, where $ssk \leftarrow \mathsf{KeyGen}_{\mathsf{ABET}}(\mathsf{msk}_{\mathsf{ABET}}, \delta)$, in which ssk is associated with a transaction modifier's identity ID'.
- $\mathsf{Hash}(\mathsf{pk}, m, \Lambda, ID)$: A transaction owner ID appends a policy-based chameleon hash with a message m into the blockchain, performs the following.
  (1) generate a hash h, a randomness r, and an ephemeral trapdoor R: $(h, \mathsf{r}, \mathsf{R}) \leftarrow \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m)$.
  (2) generate a ciphertext $C \leftarrow \mathsf{Enc}_{\mathsf{ABET}}(\mathsf{mpk}_{\mathsf{ABET}}, \mathsf{R}, \Lambda, ID)$. Note that ID denotes the transaction owner's identity, and Λ denotes a policy.
  (3) generate a pair $(sk, vk) \leftarrow \mathsf{KeyGen}_\Sigma(pp)$, and generate a digital signature $\sigma \leftarrow \mathsf{Sign}_\Sigma(sk, c)$, where the signed message is $c \leftarrow \mathsf{KeyGen}'_\Sigma(pp, sk, \mathsf{R})$.
  (4) output $(m, h, \mathsf{r}, C, vk, c, \sigma)$.
- $\mathsf{Verify}(\mathsf{pk}, m, h, \mathsf{r}, C, vk, c, \sigma)$: It outputs 1 if the following checks hold and 0 otherwise: $1 \leftarrow \mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, h, \mathsf{r})$ and $1 \leftarrow \mathsf{Verify}_\Sigma(vk, c, \sigma)$.
- $\mathsf{Adapt}(\mathsf{sk}_\delta, m, m', h, \mathsf{r}, C, vk, c, \sigma, ID')$: A transaction modifier ID' with secret key sk_δ and a new message m' performs the following
  (1) check $1 \stackrel{?}{=} \mathsf{Verify}(\mathsf{pk}, m, h, \mathsf{r}, C, vk, c, \sigma)$.
  (2) obtain $\mathsf{R} \leftarrow \mathsf{Dec}_{\mathsf{ABET}}(\mathsf{mpk}_{\mathsf{ABET}}, C, ssk)$.
  (3) compute a collision $\mathsf{r}' \leftarrow \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, m, m', h, \mathsf{r}, \mathsf{R})$.
  (4) generate a ciphertext $C' \leftarrow \mathsf{Enc}_{\mathsf{ABET}}(\mathsf{mpk}_{\mathsf{ABET}}, \mathsf{R}, \Lambda, ID')$.
  (5) generate a key pair $(sk', vk') \leftarrow \mathsf{KeyGen}_\Sigma(pp)$, and generate a digital signature $\sigma' \leftarrow \mathsf{Sign}_\Sigma(sk', c')$, where $c' \leftarrow \mathsf{KeyGen}'_\Sigma(pp, sk', \mathsf{R})$.
  (6) output $(m', h, \mathsf{r}', C', vk', c', \sigma')$.
- $\mathsf{Judge}(\mathsf{sk}, \mathcal{O}, \mathcal{T})$: Given a set of transactions $\{T'\} \in \mathcal{T}$, and a set of accused transaction modifiers $\{ID'\} \in \mathcal{O}$, AA outputs a transaction-identity pair (T', ID') if a transaction T' links to an accused modifier ID', where $T' = (m', h, \mathsf{r}', C', vk', c', \sigma')$.

Correctness. The Judge algorithm allows AA to identify a one-to-one relationship between a transaction and an accused transaction modifier. AA repeats this process until identifying all the one-to-one relationships between transactions in $\{T'\}$ and accused transaction modifiers in $\{ID'\}$. Now, we explain this process further. First, any public user verifies a connection between a transaction T and its modified version T'. We denote the connection as $T \leftrightarrow T'$. The connection can be established, since both message-signature pair (c, σ) in T and message-signature pair (c', σ') in T', are associated with the same ephemeral trapdoor R. In particular, the ephemeral trapdoor R is used in many modified versions of a transaction (i.e., $T \leftrightarrow \{T'\}$), as different transaction modifiers may modify the same transaction. Second, any public user obtains a set of accused transaction modifiers from interacting with an access blackbox O due to the property of black-box traceability in ABET, such that $\{ID'\} \leftarrow \mathsf{Trace}_{\mathsf{ABET}}(\mathsf{mpk}_{\mathsf{ABET}}, O, \epsilon)$ (ϵ is a security parameter). The access blackbox O will not return the transaction owner's identity ID because he is not supposed to have the rewriting privilege.

Eventually, AA links a modified transaction T' to an accused identity ID' using the chameleon secret key sk, and outputs an identity-transaction pair: (T', ID'), meaning that a transaction T' is indeed modified by a transaction modifier ID'. We note that the accused transaction modifiers returned from an access blackbox O may not modify the transactions in $\{T'\}$. In this case, AA outputs $\perp$, which means no relationship is established between a transaction in $\{T'\}$ and an accused transaction modifier in $\{ID'\}$. In other words, the accused transaction modifiers only reveal their rewriting privileges without rewriting the blockchain.

[1] Adaptive is an abbreviation for indistinguishability under the adaptive chosen ciphertext attack.

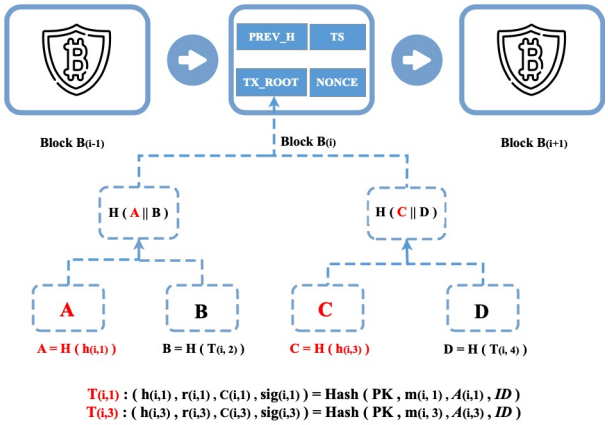

Figure 5: Using PCHBA for transaction-level blockchain rewriting.

Remark. One may notice that a transaction modifier ID' can assign a new policy Λ' to a modified transaction T' in the Adapt algorithm. The access policy embedded in the transaction can be dynamically updated to satisfy different security requirements in case of blockchain system evolves. Therefore, we remark that the proposed generic construction supports flexible access policies for blockchain rewriting. Besides, the transaction modifier uses her unique identity ID' (not random identity and we provide more details in the instantiation) to create T'. One may also notice that such flexibility could be misused. The modifier ID' may intend to rewrite transaction T with malicious content and change the rewriting privileges that preventing others from modifying the transaction. We first argue that AA can reset the transaction T's access policy if such malicious behavior happens. Second, we can let the transaction owner sign his access policy such that no modifier can change it.

5.1 Security Analysis

We show the security result of our proposed construction, and the detailed proofs are referred to Appendix B.

Theorem 5.1. *The PCHBA scheme achieves indistinguishability if the CHET is indistinguishable.*

Theorem 5.2. *The PCHBA scheme achieves collision-resistance if the ABET is adaptive secure, and the CHET is collision resistant.*

Theorem 5.3. *The PCHBA scheme achieves accountability if the digital signature scheme Σ is EUF-CMA secure.*

5.2 Application

We show an application of PCHBA for blockchain rewriting. On a high-level, the blockchain remains intact even if certain policy-based mutable transactions have been modified. We recall that each block stores a compact representation of a set of transactions. Specifically, TX_ROOT (i.e., the root hash of a Merkle tree in Figure 5) accumulates all transactions inside a block.

The party who engages in the role of an attribute authority includes chameleon public key pk using a transaction signed under the key corresponding to the public key of an address owner. The attribute authority issues secret keys for all transaction modifiers in the system. If a transaction owner ID wishes to append certain policy-based transactions into the blockchain, then these transactions must be hashed using $\mathsf{Hash}(\mathsf{pk}, m, \Lambda, ID)$. In Figure 5, a block B_i accumulates four transactions $T_{(i,1)}, T_{(i,2)}, T_{(i,3)}, T_{(i,4)}$, where $T_{(i,1)}$ and $T_{(i,3)}$ are policy-based transactions with different access policies and same identity $(\Lambda_{(i,1)}, ID), (\Lambda_{(i,3)}, ID)$, which are chosen by the transaction owner ID. The remaining transactions are processed as usual, i.e., $T_{(i,2)}$ and $T_{(i,4)}$ are hashed using traditional collision-resistance hash function H.

When the policy-based transactions need to be modified, a transaction modifier ID' with chameleon secret key $\mathsf{sk}_{\delta'}$ satisfying $(\Lambda_{(i,1)}, ID), (\Lambda_{(i,3)}, ID)$ (e.g., $1 = \Lambda_{(i,1)}(\delta')$ and $ID \leq ID'$) can compute valid collisions (or randomness) for hash values A and C, and provide the new randomness. Now, the modifier broadcasts the new randomness to the blockchain network. All participants verify the correctness of the new randomness (i.e., Verify algorithm) and update their local copy of the blockchain by replacing the old randomness with the new randomness. Since the randomness, signature, and ciphertext in a transaction are not included in the hash computation of the aggregation and are provided as non-hashed part of the transaction/block, the values $PREV\_H$ are never modified.

If certain transactions are created using PCHBA, we can ensure: 1) indistinguishability, meaning that given a randomness, any public user cannot decide whether the randomness associated with transaction $T_{(i,1)}$ (or $T_{(i,3)}$) was created using Hash or Adapt algorithm; 2) collision-resistance, such that only transaction modifier ID' with enough rewriting privileges can rewrite the blockchain; 3) accountability, which means that if there exists dispute over the modified transactions, attribute authority can link each modified transaction to a responsible transaction modifier.

6 INSTANTIATION AND IMPLEMENTATION

In this section, we start with discussing the choice of primitives to construct a PCHBA protocol. We then present the proposed instantiation, provide the implementation and evaluation analysis. First, to initiate CHET, we could use chameleon hash construction in [19], which is discrete logarithm (DL) based. Meanwhile, the RSA-based and DL-based chameleon hash equipped with bilinear map [19] should also be suitable. Second, we choose the CP-ABE scheme [9] and an anonymized HIBE scheme in [14] to instantiate ABET. The existing CP-ABE schemes with black-box traceability [32, 39] are alternatively applicable to instantiate ABET, and the accountable CP-ABE scheme [30] is also available if public traceability (see Section 7) is not required. However, they are not practical enough compared to our proposed instantiation (see Table 4).

6.1 The Proposed ABET Scheme

For constructing a practical ABET, we require the underlying ABE scheme is the most efficient one in the literature, while the size of the ciphertext is constant (i.e., independent of the number of users in the system). Therefore, we rely on a recent ABE scheme [9] and a HIBE scheme [14]. First, the fast attribute-based message

encryption FAME [9] can be viewed as the stepping stone to construct ABET. FAME supports unbounded ABE universes, has no restriction on the monotone policies used, has constant-time decryption, is adaptively secure under the standard DLIN assumption, and is based on the asymmetric prime-order Type-III pairing. The HIBE [14] has constant-size ciphertext. Specifically, the ciphertext has just three group elements (and one of them can be shared with FAME), and the decryption requires only two pairing operations.

The intertwined ABET scheme is not anonymous because its ciphertext reveals the transaction modifier's identity to the public. We convert the intertwined ABET scheme into an anonymous one by exploiting the asymmetric pairings (as described in [25]). At the same time, the ciphertext size remains short: we include one additional element in ciphertext to ensure the anonymity of the ciphertext. We prove the ABET scheme's semantic and anonymous security from two new assumptions, which were described in Section 3.1. The concrete construction of ABET is included in the proposed PCHBA instantiation. Below, we present Theorem 6.1 to show the proposed ABET scheme has semantic security and ciphertext anonymity. The security analysis is referred to Appendix C. We notice that we cannot directly apply anonymous HIBE (A-HIBE) with constant-size ciphertext [43] to construct ABET, as A-HIBE is not a prime-order based construction.

Theorem 6.1. *The proposed ABET scheme achieves semantic security and ciphertext anonymity, if the DLE and eDDH assumptions are held in the asymmetric pairing groups.*

6.2 Instantiation

In ABET scheme, two types of inputs are given to a hash function H_1: input of the form (y, ℓ, t) or that of the form (v, ℓ, t), where y is an arbitrary string (e.g., attribute), v is a positive integer, $\ell \in \{1, 2, 3\}$ and $t \in \{1, 2\}$. We represent these two inputs as $y\ell t$ and $0v\ell t$ (appending "0" in the second format is used to differentiate with the first one), respectively. We denote user's identities as vectors, we assume a transaction modifier has identity $ID_i = (I_1, \cdots I_i) \in (\mathbb{Z}_q)^i$, and a transaction owner has identity $ID_j = (I_1, \cdots I_j) \in (\mathbb{Z}_q)^j$, where $j \leq i$. We define a hierarchy as follows: identity ID_i is close to the root node k, and identity ID_j is close to the leaf node. We allow the transaction modifier ID_i to decrypt a policy-based ciphertext generated by the transaction owner ID_j.

- Setup(1^λ): It takes a security parameter λ as input, outputs: 1) chameleon key pair $(\mathsf{sk}, \mathsf{pk}) = (x, h^x)$, where $x \in \mathbb{Z}_q^*$; 2) master public key $\mathtt{mpk} = (g, h, H_1, H_2, T_1, T_2, \{g_1, \cdots g_k\}, \{g_1^\alpha, \cdots g_k^\alpha\}, \{h_1, \cdots h_k\}, g^\alpha, h^{d/\alpha}, h^{1/\alpha}, h^{\beta/\alpha})$, and master secret key $\mathtt{msk} = (a_1, a_2, b_1, b_2, \alpha, \beta, g^{d_1}, g^{d_2}, g^{d_3}, \{z_1, \cdots, z_k\})$, where $(a_1, a_2, b_1, b_2, \alpha, \beta) \in \mathbb{Z}_q^*$, $(d_1, d_2, d_3) \in \mathbb{Z}_q$, $\{z_1, \cdots, z_k\} \in \mathbb{Z}_q$, $d = d_1 + d_2 + d_3$, g is generator of group $\mathbb{G}$, h is generator of group $\mathbb{H}$, $(\mathbb{G}, \mathbb{H}, \mathbb{G}_T)$ are groups of order q, $H_1 = h^{a_1}, H_2 = h^{a_2}$, $T_1 = \hat{\mathsf{e}}(g, h)^{d_1 \cdot a_1 + d_3/\alpha}, T_2 = \hat{\mathsf{e}}(g, h)^{d_2 \cdot a_2 + d_3/\alpha}$, $\{g_1, \cdots, g_k\} = \{g^{z_1}, \cdots, g^{z_k}\}$, and $\{h_1, \cdots, h_k\} = \{h^{z_1}, \cdots, h^{z_k}\}$.
  We denote an identity as $ID = \{I_1, \cdots, I_k\} \in (\mathbb{Z}_q)^k$. Let $\mathsf{G} : \mathbb{G}_T \to \{0,1\}^*$ be a pseudo-random generator. Let $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}, \mathsf{H}_2 : \{0,1\}^* \to \mathbb{Z}_\mathsf{q}$ be two hash functions, and the size of hash output H_2 is assumed to be l. Let $\hat{\mathsf{e}} : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$ be a bilinear pairing. Each user in the permissioned blockchain is assigned by a secret credential such as $\widehat{ID}_j^\alpha$, where $\widehat{ID}_j = h_k^{I_1} \cdots h_j^{I_j} \cdot h \in \mathbb{H}$.
- KeyGen(sk, δ): It inputs a chameleon secret key sk, and a set of attributes δ, outputs a secret key $\mathsf{sk}_{\delta_i} = (x, ssk_i)$. Specifically, it first picks $(r_1, r_2) \in \mathbb{Z}_q^*$, $R \in \mathbb{Z}_q$, $r = r_1 + r_2$, computes $\mathsf{sk}_0 = (h^{b_1 \cdot r_1}, h^{b_2 \cdot r_2}, h^{(r_1+r_2)/\alpha}, g^{1/\alpha}, g^{r/\alpha}, g^R)$. Second, it picks $\sigma_y \in \mathbb{Z}_q$, for all $y \in \delta$ and $t = \{1, 2\}$, computes $\mathsf{sk}_{y,t} = \mathsf{H}_1(\mathsf{y1t})^{\mathsf{b}_1 \cdot \mathsf{r}_1/\mathsf{a}_\mathsf{t}} \cdot \mathsf{H}_1(\mathsf{y2t})^{\mathsf{b}_2 \cdot \mathsf{r}_2/\mathsf{a}_\mathsf{t}} \cdot \mathsf{H}_1(\mathsf{y3t})^{\frac{\mathsf{r}_1+\mathsf{r}_2}{\alpha \cdot \mathsf{a}_\mathsf{t}}} \cdot \mathsf{g}^{\frac{\sigma_\mathsf{y}}{\alpha \cdot \mathsf{a}_\mathsf{t}}}$, and sets $\mathsf{sk}_y = (\mathsf{sk}_{y,1}, \mathsf{sk}_{y,2}, g^{-\sigma_y})$. Third, it picks $\sigma' \in \mathbb{Z}_q$, for $t = \{1, 2\}$, computes $\mathsf{sk}'_t = g^{d_t} \cdot \mathsf{H}_1(\mathtt{011t})^{\mathsf{b}_1 \cdot \mathsf{r}_1/\mathsf{a}_\mathsf{t}} \cdot \mathsf{H}_1(\mathtt{012t})^{\mathsf{b}_2 \cdot \mathsf{r}_2/\mathsf{a}_\mathsf{t}} \cdot \mathsf{H}_1(\mathtt{013t})^{\frac{\mathsf{r}_1+\mathsf{r}_2}{\alpha \cdot \mathsf{a}_\mathsf{t}}} \cdot \mathsf{g}^{\frac{\sigma'}{\alpha \cdot \mathsf{a}_\mathsf{t}}}$, and sets $\mathsf{sk}' = (\mathsf{sk}'_1, \mathsf{sk}'_2, g^{d_3} \cdot g^{-\sigma'})$. Last, it computes $\mathsf{sk}_1 = g^d \cdot \widehat{ID}_i^{\alpha \cdot r} \cdot g^{\beta \cdot R}, \mathsf{sk}_2 = \{g_{i-1}^{\alpha \cdot r}, \cdots g_1^{\alpha \cdot r}\}$. The decryption key is $ssk_i = (\mathsf{sk}_0, \{\mathsf{sk}_y\}_{y \in \delta}, \mathsf{sk}', \mathsf{sk}_1, \mathsf{sk}_2)$. We denote a transaction modifier's identity in the decryption key as $\widehat{ID}_i = g_k^{I_1} \cdots g_i^{I_i} \cdot g \in \mathbb{G}$.
- Hash($\mathsf{pk}, m, \mathbf{M}, ID_j$): To hash a message $m \in \mathbb{Z}_q$ under a policy $(\mathbf{M}, \pi)$, and an identity $ID_j = (I_1, \cdots I_j)$, a transaction owner performs the following
  (1) choose a randomness $\mathsf{r} \in \mathbb{Z}_q^*$, and compute $p = \mathsf{pk}^\mathsf{r}$.
  (2) choose an ephemeral trapdoor R, compute $e = \mathsf{H}_2(\mathsf{R}), \mathsf{h}' = \mathsf{h}^\mathsf{e}$. Note that R denotes a short bit-string.
  (3) compute a chameleon hash $b = p \cdot h'^m$.
  (4) generate a signing/verification key pair as (sk, vk), where $(sk, vk) = (s, \widehat{ID}_j^{\alpha \cdot s})$, $(s_1, s_2) \in \mathbb{Z}_q^*$, $s = s_1 + s_2$.
  (5) generate a ciphertext on the message $M = (\mathsf{r}, \mathsf{R})$ with the policy $(\mathbf{M}, \pi)$ ($\mathbf{M}$ has n_1 rows and n_2 columns) and identity ID_j. Specifically, it first computes $ct_0 = (H_1^{s_1}, H_2^{s_2}, h^{s/\alpha}, h^{\beta \cdot s/\alpha})$, for $u = \{1, \cdots, n_1\}$ and $\ell = \{1, 2, 3\}$, it computes $ct_{u,\ell} = \mathsf{H}_1(\pi(\mathsf{u})\ell 1)^{\mathsf{s}_1} \cdot \mathsf{H}_1(\pi(\mathsf{u})\ell 2)^{\mathsf{s}_2} \cdot \prod_{\mathsf{v}=1}^{\mathsf{n}_2} [\mathsf{H}_1(0\mathsf{v}\ell 1)^{\mathsf{s}_1} \cdot \mathsf{H}_1(0\mathsf{v}\ell 2)^{\mathsf{s}_2}]^{\mathbf{M}_{(\mathsf{u},\mathsf{v})}}$. Then, it computes $ct = \mathsf{r} \oplus \mathsf{G}(T_1^{s_1} \cdot T_2^{s_2})$, $ct' = (\mathsf{R}||0^{l-|\mathsf{R}|}) \oplus \mathsf{H}_2(\hat{\mathsf{e}}(\mathsf{g}, \mathsf{h}^{\mathsf{d}/\alpha})^\mathsf{s})$, $ct_1 = \widehat{ID}_j^{\alpha \cdot s}$ (i.e., vk), $ct_2 = \widehat{ID}_j^s$, $ct_3 = ct_1^s$ $\widehat{ID}_j = h_k^{I_1} \cdots h_j^{I_j} \cdot h \in \mathbb{H}$. Eventually, it outputs $C = (ct_0, \{ct_u\}_{u \in n_1}, ct, ct', ct_1, ct_2, ct_3)$.
  (6) generate a signature $epk = g^{esk}, \sigma = esk + sk \cdot \mathsf{H}_2(\mathsf{epk}||\mathsf{c})$, where (esk, epk) denotes an ephemeral key pair, and $c = h^{sk+(\mathsf{R}||0^{l-|\mathsf{R}|})}$ denotes the signed message.
  (7) output $(m, p, h', b, C, c, epk, \sigma)$.
- Verify($\mathsf{pk}, m, p, h', b, C, c, epk, \sigma$): Any public user can verify whether a given hash (b, h') is valid, it outputs 1 if $b = p \cdot h'^m$, and $\hat{\mathsf{e}}(g^\alpha, ct_2)^\sigma = \hat{\mathsf{e}}(epk, ct_1) \cdot \hat{\mathsf{e}}(g, ct_3)^{\mathsf{H}_2(\mathsf{epk}||\mathsf{c})}$, where $ct_1 = vk$.
- Adapt($\mathsf{sk}_{\delta_i}, m, m', p, h', b, C, c, epk, \sigma, ID_i$): The transaction modifier with a secret key sk_{δ_i}, a new message $m' \in \mathbb{Z}_q$, and an identity $ID_i = (I_1, \cdots I_i)$, performs the following
  (1) check $1 \stackrel{?}{=} \mathsf{Verify}(\mathsf{pk}, m, p, h', b, C, c, epk, \sigma)$.
  (2) run the following steps to retrieve the encrypted ephemeral trapdoor R:
    (a) generate a delegated decryption key with respect to an identity $ID_{i+1} = (I_1, \cdots I_{i+1})$. It picks $(z_1, z_2) \in \mathbb{Z}_q$, $z = z_1 + z_2$, and computes $\mathsf{sk}_0 = (h^{b_1 \cdot (r_1+z_1)}, h^{b_2 \cdot (r_2+z_2)}, h^{(r_1+r_2+z)/\alpha}, g^{1/\alpha}, g^{(r+z)/\alpha}, g^R)$, $\mathsf{sk}_{y,t} = \mathsf{H}_1(\mathsf{y1t})^{\mathsf{b}_1 \cdot (\mathsf{r}_1+\mathsf{z}_1)/\mathsf{a}_\mathsf{t}} \cdot \mathsf{H}_1(\mathsf{y2t})^{\mathsf{b}_2 \cdot (\mathsf{r}_2+\mathsf{z}_2)/\mathsf{a}_\mathsf{t}} \cdot \mathsf{H}_1(\mathsf{y3t})^{\frac{\mathsf{r}_1+\mathsf{r}_2+\mathsf{z}}{\alpha \cdot \mathsf{a}_\mathsf{t}}} \cdot \mathsf{g}^{\frac{\sigma_\mathsf{y}}{\alpha \cdot \mathsf{a}_\mathsf{t}}}$, $\mathsf{sk}'_t = g^{d_t} \cdot \mathsf{H}_1(\mathtt{011t})^{\mathsf{b}_1 \cdot (\mathsf{r}_1+\mathsf{z}_1)/\mathsf{a}_\mathsf{t}} \cdot \mathsf{H}_1(\mathtt{012t})^{\mathsf{b}_2 \cdot (\mathsf{r}_2+\mathsf{z}_2)/\mathsf{a}_\mathsf{t}} \cdot \mathsf{H}_1(\mathtt{013t})^{\frac{\mathsf{r}_1+\mathsf{r}_2+\mathsf{z}}{\alpha \cdot \mathsf{a}_\mathsf{t}}}$.

$\mathsf{g}^{\frac{\sigma'}{\alpha\cdot a_t}}$, $\mathsf{sk}' = (\mathsf{sk}'_1, \mathsf{sk}'_2, g^{d_3}\cdot g^{-\sigma'})$, $\mathsf{sk}_1 = g^d \cdot \widehat{ID}_i^{\alpha\cdot r} \cdot g^{\beta\cdot R}\cdot (g_{i-1}^{\alpha\cdot r})^{I_{i+1}}\cdot (g_k^{\alpha\cdot I_1}\cdots g_{i-1}^{\alpha\cdot I_{i+1}}\cdot g)^z$, $\mathsf{sk}_2 = \{g_{i-2}^{\alpha\cdot r}\cdot g_{i-2}^{\alpha\cdot z},\cdots, g_1^{\alpha\cdot r}\cdot g_1^{\alpha\cdot z}\}$. The delegated key is $ssk_{i+1} = (\mathsf{sk}_0, \{\mathsf{sk}_y\}_{y\in\delta}, \mathsf{sk}', \mathsf{sk}_1, \mathsf{sk}_2)$.

(b) check $(\mathsf{R}||0^{l-|\mathsf{R}|}) \stackrel{?}{=} ct' \oplus \mathsf{H}_2[\frac{\hat{\mathsf{e}}(\mathsf{g}^{\mathsf{d}}\cdot\widehat{\mathsf{ID}}_{\mathsf{i+1}}^{\alpha(\mathsf{r+z})}\cdot \mathsf{g}^{\beta\cdot\mathsf{R}}, \mathsf{h}^{\mathsf{s}/\alpha})}{\hat{\mathsf{e}}(\mathsf{g}^{(\mathsf{r+z})/\alpha}, \mathsf{ct}_1)\cdot\hat{\mathsf{e}}(\mathsf{g}^{\mathsf{R}}, \mathsf{h}^{\beta\cdot\mathsf{s}/\alpha})}]$. The format "$||0^{l-|\mathsf{R}|}$" is used to check when the value R is retrieved with certainty $1-2^{l-|\mathsf{R}|}$.[2] If the value is retrieved, then the delegation procedure terminates, and the delegated decryption key is ssk'_i.

(3) run the following steps to obtain the encrypted randomness r. Recall that if the set of attributes δ in ssk'_i satisfies the MSP $(\mathbf{M}, \pi)$, then there exist constants $\{\gamma_u\}_{u\in I}$ that satisfy the equation in Section 3.2. It computes $\mathsf{r} = ct \oplus \mathrm{G}(\frac{B}{A}) = ct \oplus \mathrm{G}(T_1^{s_1}\cdot T_2^{s_2})$ (the correctness of the decryption is referred to [9]), where A and B are described below.

$$A = \hat{\mathsf{e}}(\prod_{u\in I} ct_{(u,1)}^{\gamma_u}, \mathsf{sk}_{(0,1)})\cdot\hat{\mathsf{e}}(\prod_{u\in I} ct_{(u,2)}^{\gamma_u}, \mathsf{sk}_{(0,2)}) \cdot\hat{\mathsf{e}}(\prod_{u\in I} ct_{(u,3)}^{\gamma_u}, \mathsf{sk}_{(0,3)})$$

$$B = \hat{\mathsf{e}}(\mathsf{sk}'_1\cdot\prod_{u\in I}\mathsf{sk}_{(\pi(u),1)}^{\gamma_u}, ct_{(0,1)})\cdot\hat{\mathsf{e}}(\mathsf{sk}'_2\cdot\prod_{u\in I}\mathsf{sk}_{(\pi(u),2)}^{\gamma_u}, ct_{(0,2)}) \cdot\hat{\mathsf{e}}(\mathsf{sk}'_3\cdot\prod_{u\in I}\mathsf{sk}_{(\pi(u),3)}^{\gamma_u}, ct_{(0,3)})$$

where $\mathsf{sk}_{(0,1)}$, $\mathsf{sk}_{(0,2)}$, $\mathsf{sk}_{(0,3)}$ denote the first, second and third element of sk_0, and the same rule applies to ct_0.

(4) derive an adapted randomness as $\mathsf{r}' = \mathsf{r} + (m-m')\cdot e/x$, and compute $p' = \mathsf{pk}^{\mathsf{r}'}$, where $e = \mathsf{H}_2(\mathsf{R})$.

(5) generate a signing/verification key pair (sk', vk'), where $(sk', vk') = (s', \widehat{ID}_i^{\alpha\cdot s'})$, $(s'_1, s'_2) \in \mathbb{Z}_q^*$, and $s' = s'_1 + s'_2$.

(6) generate a ciphertext C' on message $M' = (\mathsf{r}', \mathsf{R})$ using randomness (s'_1, s'_2), under policy $(\mathbf{M}, \pi)$ and identity ID_i.

(7) generate a signature $epk' = g^{esk'}$, $\sigma' = esk' + sk'\cdot\mathsf{H}_2(\mathsf{epk}'||\mathsf{c}')$, where $c' = h^{sk'+(\mathsf{R}||0^{l-|\mathsf{R}|})}$.

(8) output $(m', p', h', b, C', c', epk', \sigma')$.

Correctness. We discuss the correctness of the Judge algorithm. First, any public user verifies the connection between a transaction and its modified version, and this connection is publicly verifiable. For example, given two chameleon hash outputs: $(m, m', b, p, p', h', C, C', c, c', epk, \sigma, epk', \sigma')$, the public performs the following

(1) verify chameleon hash $b = p\cdot h'^{m} = p'\cdot h'^{m'}$.

(2) verify message-signature pair (c, σ) under (epk, vk), and message-signature pair (c', σ') under (epk', vk').

(3) verify $ct'_{(0,3)} = ct_{(0,3)}\cdot\Delta(sk)$, where $\Delta(sk) = c'/c = h^{sk'-sk}$ (the meaning of $\Delta(sk)$ is referred to Section 3.3). Note that (c, c') are derived from the same secret R.

Second, any public user obtains a set of accused transaction modifiers from interacting with an access blackbox O. Note that the transaction modifier's delegated chameleon secret keys are disallowed to be used in generating O. If transaction modifiers use their delegated chameleon secret keys to generate O, the public cannot identify the accused transaction modifiers correctly because the delegated keys may share the same identity.

Eventually, AA identifies all the one-to-one relationships between modified transactions and accused transaction modifiers. For example, AA links an accused transaction modifier $ID_i = (I_1, \cdots, I_i)$ to a modified transaction T' via the following check using master secret key α.

$$T' : \hat{\mathsf{e}}(g, (ct_1)^{1/\alpha^2}) = \hat{\mathsf{e}}((g_k^{I_1}\cdots g_i^{I_i}\cdot g), h^{s/\alpha}) = \hat{\mathsf{e}}(\widehat{ID}_i, ct_{(0,3)}),$$

where $ct_1 = vk$. If the above check holds, then AA confirms that a modified transaction T' links to an accused transaction modifier ID_i. We remark that the pairing operations involved in the Verify algorithm are used to prevent a transaction modifier from using random identities for blockchain rewriting. As a result, the transaction modifier ID_i need to use her (unique) secret credential $\widehat{ID}_i^{\alpha}$ to rewrite transaction T.

Table 1: Average time taken by various operations on the MNT224 curve. All times are measured in milliseconds (ms).

Groups:	Multiplication	Exponentiation	Hash	Pairing
$\mathbb{Z}_q$	-	-	1.0×10^{-7}	-
$\mathbb{G}$:	1.29×10^{-6}	6.2×10^{-4}	3.51×10^{-5}	-
$\mathbb{H}$:	1.06×10^{-5}	4.21×10^{-3}	1.06×10^{-2}	-
$\mathbb{G}_T$:	3.45×10^{-6}	9.02×10^{-4}	-	3.2×10^{-3}

6.3 Implementation and Evaluation

We implement our proposed instantiation using Charm framework [10] and evaluate its performance on a PC with Intel Core i9 (3.6Ghz × 2) and 7.7GiB RAM. We specifically benchmark each algorithm of the instantiation with various parameters. In the implementation, we use MNT224 curve [37] for pairing, which is the best Type-III pairing in PBC [6], and it has around 100-bit security level. To achieve a higher security level with comparable performance, one may choose BLS12-381 curve [3], which has around 128-bit security level. We instantiate the hash function and the pseudo-random generator with the corresponding standard interfaces provided by the Charm framework. The implementation code is available on GitHub [2]. Table 1 lists the average time taken by various operations on MNT224 curve in milliseconds. One can see that operations on group $\mathbb{H}$ are significantly more expensive than on $\mathbb{G}$, ranging from 7 times for exponentiation to as much as 300 times for hashing.

First, we provide a performance comparison between PCHBA and PCH [23] in Table 2. For a fair comparison, we do not consider the encoding/decoding algorithms used in PCH, the pseudo-random generator used in PCHBA. We focus on expensive operations, including multiplication, exponentiation, and pairing. We assume transaction owner and modifier have the same identity without delegation $ID_k = (I_1, \cdots I_k) \in (\mathbb{Z}_q)^k$. Table 2 shows that the computational cost of each algorithm in PCHBA is higher than PCH. This is because the additional cryptographic primitives, including HIBE and Σ, are used in PCHBA. In particular, the running time

[2] This technique was used in digital lockers, which are computationally secure symmetric encryption schemes [20]. In a digital locker, obtaining any information about the plaintext from the ciphertext is as hard as guessing the symmetric key, and concatenation with $0^{l-\mathrm{R}}$ ensures that the wrong symmetric key can be recognized with high probability.

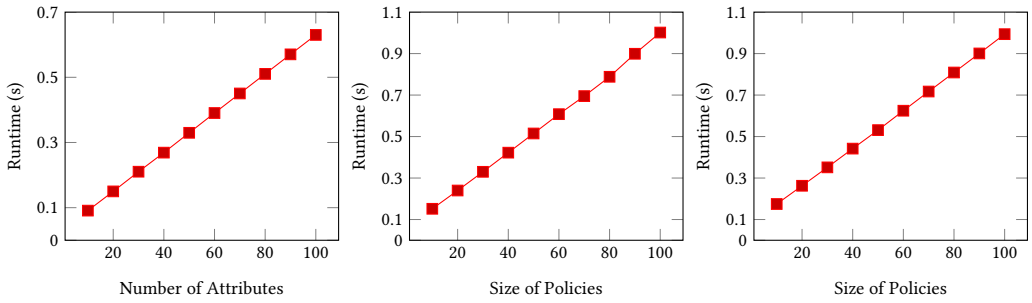

Figure 6: Running time of KeyGen (left), Hash (middle) and Adapt (right) algorithms.

Table 2: The Performance Comparison with PCH. T denotes the number of attributes to KeyGen, (n_1, n_2) **denotes the dimension of the MSP for encryption to** Hash, I **is the number of attributes for decryption to** Adapt, k **denotes the total number of users in the system. Let** $N = 12n_1n_2+6n_1$. **We set** $T = 3$, $n_1 = 3$, $k = 10$ **for timings.**

Operations:	Multiplication	Exponentiation	Pairing	Time (ms)
$\text{Setup}_{\text{PCH/PCHBA}}$:	1 / -	7 / 12+3k	2 / 2	0.012/0.060
$\text{KeyGen}_{\text{PCH/PCHBA}}$:	8T+9 / 8T+k+11	9T+15 /9T+k+18	- / -	0.026/0.034
Hash_{PCH}:	N+5	6n_1+7	-	0.105
$\text{Hash}_{\text{PCHBA}}$:	N+k+2	6n_1+k+14	1	0.176
$\text{Verify}_{\text{PCH/PCHBA}}$:	2 / 2	2 / 3	- / 3	0.001/0.011
$\text{Adapt}_{\text{PCH}}$:	N+6I+12	6n_1+11	6	0.141
$\text{Adapt}_{\text{PCHBA}}$:	N+k+6I+11	6n_1+k+15	9	0.209
$\text{Judge}_{\text{PCHBA}}$:	k+5	k+7	8	0.097

Table 3: The number of multiplications and exponentiations in $\mathbb{G}$, $\mathbb{H}$, and $\mathbb{G}_T$.

Operations:	Multiplication	Exponentiation
$\text{KeyGen}_{\mathbb{G}/\mathbb{H}/\mathbb{G}_T}$:	8T+k+11 / - / -	9T+k+15 / $\underline{3}$ / -
$\text{Hash}_{\mathbb{G}/\mathbb{H}/\mathbb{G}_T}$:	12n_1n_2+6n_1 / $\underline{k+2}$ / -	6n_1+1 / $\underline{k+10}$ / 3
$\text{Adapt}_{\mathbb{G}/\mathbb{H}/\mathbb{G}_T}$:	12n_1n_2+6n_1+6I+3 / $\underline{k+1}$ / 8	6n_1+1 / $\underline{k+9}$ / 5

of KeyGen, Hash, and Adapt algorithms are measured and shown in Figure 6.3. The performance of these algorithms is linear to the number of attributes or the size of polices. The run-time of KeyGen is 0.630 seconds, even if the number of attributes is 100. The running time of Hash and Adapt algorithms is about one second for handling a policy of size 100.

Second, we evaluate the KeyGen, Hash, and Adapt algorithms in terms of the number of elements from $\mathbb{G}$, $\mathbb{H}$, and $\mathbb{G}_T$. We specifically highlight the number of operations on group $\mathbb{H}$ in Table 3, because the operations on $\mathbb{H}$ are more expensive than on $\mathbb{G}$ in the MNT224 curve (as shown in Table 1). Table 2 and 3 show that the computational cost of KeyGen, Hash, and Adapt algorithms is also linear to the number of system users k. However, we can pre-compute certain heavy operations during Setup, including $\widehat{ID}_i = g_k^{I_1} \cdots g_i^{I_i} \cdot g \in \mathbb{G}$, and $\widehat{ID}_j = h_k^{I_1} \cdots h_i^{I_j} \cdot h \in \mathbb{H}$.

Third, we evaluate the storage cost of the secret key, and hash/adapt output based on the number of elements from $\mathbb{G}$, $\mathbb{H}$ (note that the size of an element of $\mathbb{H}$ is three times the size of $\mathbb{G}$ in the MNT224 curve), $\mathbb{G}_T$, and $\mathbb{Z}_q$. Table 3 shows that the number of elements in $\mathbb{H}$ is independent of T and n_1, since the underlying FAME [9] supports a constant number of group elements in $\mathbb{H}$. For the Hash algorithm, the extra storage cost of PCHBA includes one element in $\mathbb{G}$, five elements in $\mathbb{H}$ (four elements for the ciphertext and one element for signature), one element in $\mathbb{G}_T$, and one element in $\mathbb{Z}_q$, compared to PCH.

Last, we show the impact of integrating the proposed PCHBA into a mutable blockchain. Figure 5 shows that the collision-resistant hash function H is used to accumulate transactions, and chain blocks. The randomness, ciphertext, and signature are stored in a non-hashed part of the transaction. The rewriting of mutable transaction incurs almost no overhead to Merkle tree generation time and chain validation time. Now, we show the storage cost to the mutable blockchain. We mention that the number of mutable transactions ranges from 2% to 10% inside a block according to [24]. Each mutable transaction needs to store $(m, h, r, C, vk, c, \sigma)$ (i.e., hash output). The storage cost of a mutable transaction includes: 1) $\mathcal{L}_{\mathbb{Z}_q} + 3\mathcal{L}_{\mathbb{H}}$ regarding DL-based chameleon hash; 2) $n_1 \times \mathcal{L}_{\mathbb{G}} + 7\mathcal{L}_{\mathbb{H}} + \mathcal{L}_{\mathbb{G}_T}$ regarding ABET; 3) $\mathcal{L}_{\mathbb{G}} + \mathcal{L}_{\mathbb{H}} + \mathcal{L}_{\mathbb{Z}_q}$ regarding digital signature.

7 RELATED WORK

Blockchain Rewriting. Blockchain rewriting was firstly introduced by Ateniese et al. [11]. The blockchain can be edited or corrected by an authorized party, which is achieved by using a chameleon hash CH [29]. The hashing of CH is parametrized by a public key pk, and the CH behaves like a collision-resistant hash function if the secret key sk (or trapdoor) is unknown. In contrast, the party who has a trapdoor can efficiently find valid collisions and output a new message-randomness pair without changing the hash output. The blockchain supports rewriting if CH replaces the traditional hash function, and the blockchain rewriting has been adopted by many applications such as Accenture [1].

Later, Camenisch et al. [19] introduced a new primitive: chameleon hash with ephemeral trapdoor CHET, which is extended from CH. The CHET requires that the authorized party has two trapdoors for rewriting: one is the long-term trapdoor sk associated with the public key pk; the other is an ephemeral trapdoor *etd* which is generated by the party who created the hash output. The CHET provides more usability in rewriting, such that the party who creates the hash value can decide, whether the holder of pk shall be able to rewrite the hash by providing or withholding the second trapdoor *etd*.

Derler et al. [23] proposed a policy-based chameleon hash PCH, which aims to provide a fine-grained and controlled blockchain rewriting. Essentially, the public key encryption scheme used in

CHET is replaced by a CP-ABE scheme [9]. Suppose the PCH is applied to the blockchain. The rewriting action can occur in a transaction-level, which is more practical than rewriting the entire block in [11].

Recently, Samelin and Slamanig proposed a policy-based sanitizable signatures P3S [41]. The sanitizing in P3S is also based on policies, such that a sanitizer can sanitize if it holds a trapdoor associated with attributes satisfying the policy associated with a signature. In particular, they formulated a set of security definitions, including signer and sanitizer accountability.

Deuber et al. [24] introduced an efficient redactable blockchain in the permissionless setting (where there is no trusted authority). The proposed protocol relies on a consensus-based e-voting system [28], such that the modification is executed in the chain if a modification request from any user gathers enough votes from miners (we call it V-CH for convenience). When integrated into the Bitcoin system, V-CH incurs only a small additional overhead.

In this work, we focus on chameleon hash-based blockchain rewriting. The proposed PCHBA achieves fine-grained transaction-level blockchain rewriting. It enjoys black-box accountability and works in a permissioned blockchain. This work can be regarded as a step forward from PCH [23] because PCHBA enables the modifiers of transactions to be held accountable for the modified transactions.

Traitor Tracing. Traitor tracing was introduced by Chor, Fiat, and Naor [22], which allows content distributers to identify pirates who violates copyright restrictions. The tracing algorithm takes tracing key tk as input and interacts a pirate decoder (behaves like a black-box oracle), outputs the index $i \in \{1, \cdots, k\}$, indicating a secret key sk_i was used to create the pirate decoder. According to the definition of tracing algorithm, traitor tracing constructions have two categories: secret tracing [15, 16, 30] (e.g., tracing algorithm plus fingerprinting code [17]), and public tracing [18, 32, 39] (e.g., tracing algorithm plus broadcast encryption [26]).

Boneh et al. [16] proposed a public-key based traitor tracing system with a sub-linear size ciphertext $O(\sqrt{k})$. They introduced a private linear broadcast encryption, which allows a broadcaster to generate a ciphertext from tracing key tk and index $\{i, i+1, \cdots, k\}$, where $i \in \{1, \cdots, k+1\}$. The ciphertext can be decrypted under keys $\{dk_i, dk_{i+1}, \cdots, dk_k\}$. Later, Boneh and Naor [15] proposed another public-key based traitor tracing system with constant-size ciphertext $O(1)$. The proposed system relies on a fingerprinting code, such as Boneh-Shaw code [17]. The fingerprinting code includes a code generator algorithm, and a tracing algorithm which is used for traitor tracing. Recently, Lai and Tang [30] introduced a traceable (or accountable) attribute-based encryption scheme with constant-size ciphertext, which is also based on the fingerprinting code. The proposed construction is built on top of an attribute-based encryption [40], and a tardos code [45]. Note that the tracing algorithm in [15, 16, 30] treats tracing key tk as a secret value.

Boneh and Waters [18] introduced an augmented broadcast encryption with public traitor tracing. The tracing algorithm can be executed by any public user, as it inputs (system) public key for traitor tracing. The proposed augmented broadcast encryption is public-key based broadcast system, and it has sub-linear size ciphertext $O(\sqrt{k})$. In CCS13, Liu et al. [32] proposed a traceable CP-ABE. The proposed construction supports public traceability in the attribute-based setting, and it has sub-linear size ciphertext $O(\sqrt{k})$. In particular, the public tracing is closely related to the traitor tracing in broadcast encryption [16, 18]. Later, Ning et al. [39] proposed a traceable CP-ABE with constant-size ciphertext $O(1)$. The construction is built on top of a CP-ABE [31] and an anonymous HIBE [43], in which the public tracing is based on the tracing algorithm used in [16, 18, 32].

Table 4: The Comparison between CP-ABE schemes with (public) traceability. † means that on which ABE scheme the traceable CP-ABE is based. k denotes the total number of users in the system. Hash function allows any bit-string to be mapped as an attribute, which is modelled as a random oracle in the security proof. The asymmetric prime-order Type-III pairing is the most efficient one.

	Scheme†	Group-order	Cipher-size	Public-trace	Hash	Type-III
[32]	ABE [31]	Composite	$O(\sqrt{k})$	✓	×	×
[39]	ABE [31]	Composite	$O(1)$	✓	×	×
[30]	ABE [40]	Prime	$O(1)$	×	×	×
Ours	FAME [9]	Prime	$O(1)$	✓	✓	✓

In this work, we propose a new ABET scheme, which is used to construct PCHBA for blockchain rewriting. We claim that it is a practical construction, and we present a comparison between the existing traceable CP-ABEs in Table 4. It shows that our proposed ABET is the first construction based on FAME, it inherits all the security and performance advantages from FAME and is based on prime-order asymmetric pairings.

8 CONCLUSION

In this paper, we proposed a generic framework of policy-based chameleon hash with black-box accountability for blockchain rewriting. The proposed framework can help thwart maliciously rewriting of blockchain by any chameleon trapdoor holder. We presented a practical instantiation, and shew that the proposed instantiation is suitable for blockchain applications. We leave the construction of policy-based chameleon hash for blockchain rewriting in the permissionless setting as our future work.

ACKNOWLEDGMENTS

This work was supported by the Ministry of Education, Singapore, under its MOE AcRF Tier 2 grant (MOE2018-T2-1-111). This research was also supported by the Natural Science Foundation of China (Grant No. 61872264, 62072371). Yingjiu Li was supported in part by the Ripple University Blockchain Research Initiative. We would like to thank anonymous ACSAC reviewers for their insightful suggestions and advice.

REFERENCES

[1] [n.d.]. Accenture. https://www.accenture.com/sg-en.
[2] [n.d.]. Accountable Blockchain Rewriting. https://github.com/SMC-SMU/Accountable-Blockchain-Rewriting.
[3] [n.d.]. BLS12-381. https://tools.ietf.org/html/draft-yonezawa-pairing-friendly-curves-02.
[4] [n.d.]. General Data Protection Regulation. https://eugdpr.org.
[5] [n.d.]. Hyperledger. https://www.hyperledger.org.
[6] [n.d.]. Pairing-Friendly Curves. https://tools.ietf.org/id/draft-yonezawa-pairing-friendly-curves-01.html.
[7] [n.d.]. Ripple. https://ripple.com.

[8] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. 2005. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In *CRYPTO*. 205–222.

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

[10] 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.

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

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

[13] John Bethencourt, Amit Sahai, and Brent Waters. 2007. Ciphertext-Policy Attribute-Based Encryption. In *IEEE S&P*. 321–334.

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

[15] Dan Boneh and Moni Naor. 2008. Traitor tracing with constant size ciphertext. In *CCS*. 501–510.

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

[17] Dan Boneh and James Shaw. 1998. Collusion-secure fingerprinting for digital data. *IEEE Transactions on Information Theory* 44, 5 (1998), 1897–1905.

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

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

[20] Ran Canetti, Benjamin Fuller, Omer Paneth, Leonid Reyzin, and Adam Smith. 2016. Reusable fuzzy extractors for low-entropy distributions. In *EUROCRYPT*. 117–146.

[21] Jie Chen, Romain Gay, and Hoeteck Wee. 2015. Improved dual system ABE in prime-order groups via predicate encodings. In *EUROCRYPT*. 595–624.

[22] Benny Chor, Amos Fiat, and Moni Naor. 1994. Tracing traitors. In *CRYPTO*. 257–270.

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

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

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

[26] Amos Fiat and Moni Naor. 1993. Broadcast encryption. In *CRYPTO*. 480–491.

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

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

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

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

[31] Allison Lewko and Brent Waters. 2012. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *CRYPTO*. 180–198.

[32] 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.

[33] Takahiro Matsuda, Kenta Takahashi, Takao Murakami, and Goichiro Hanaoka. 2016. Fuzzy signatures: relaxing requirements and a new construction. In *ACNS*. 97–116.

[34] 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.

[35] 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.

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

[37] Atsuko Miyaji, Masaki Nakabayashi, and Shunzo Takano. 2000. Characterization of elliptic curve traces under FR-reduction. In *ICISC*. 90–108.

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

[39] 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.

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

[41] Kai Samelin and Daniel Slamanig. 2020. Policy-Based Sanitizable Signatures. In *CT-RSA*. 538–563.

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

[43] Jae Hong Seo and Jung Hee Cheon. 2011. Fully Secure Anonymous Hierarchical Identity-Based Encryption with Constant Size Ciphertexts. *Iacr Cryptology Eprint Archive* 2011 (2011), 21.

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

[45] Gábor Tardos. 2008. Optimal probabilistic fingerprint codes. *Journal of the ACM (JACM)* 55, 2 (2008), 1–24.

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

A THEOREMS IN SECTION 3.1

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 $\mathbb{G}_\mathbb{T}$. If $(a_1, a_2, s_1, s_2) \xleftarrow{R} \mathbb{Z}_q$ and encodings $\epsilon_1, \epsilon_2, \epsilon_T$ are randomly chosen, then we define the advantage of the adversary in solving the DLE with at most Q queries to the group operation oracles O_1, O_2, O_T and the bilinear pairing $\hat{e}$ as*

$$
\begin{aligned}
|\mathsf{Adv}^{DLE}_{\mathcal{A}}(\lambda) \quad = \quad & \Pr[\mathcal{A}(q, \epsilon_1(1), \epsilon_1(a_1), \epsilon_1(a_2), \\
& \epsilon_1(a_1 \cdot s_1^k), \epsilon_1(a_2 \cdot s_2^k), \epsilon_2(1), \\
& \{\epsilon_1(s_1), \cdots, \epsilon_1(s_1^{k-1}), \epsilon_1(s_1^{k+1}), \cdots, \epsilon_1(s_1^{2k})\}, \\
& \{\epsilon_1(s_2), \cdots, \epsilon_1(s_2^{k-1}), \epsilon_1(s_2^{k+1}), \cdots, \epsilon_1(s_2^{2k})\}) \\
= \quad & b : (a_1, a_2, s_1, s_2, s \xleftarrow{R} \mathbb{Z}_q, b \in (0,1), \\
& t_b = s_1^k + s_2^k, t_{1-b} = s)] - 1/2| \leq \frac{4k(Q + 2k + 2)^2}{q}
\end{aligned}
$$

Proof. Let $\mathcal{S}$ play the following game with $\mathcal{A}$. $\mathcal{S}$ 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 $p_i \in \mathbb{Z}_q[A_1, A_2, S_1, S_2, T_0, T_1]$ are 6-variate polynomials over $\mathbb{Z}_q$, such that $p_0 = 1, p_1 = A_1, p_2 = A_2, p_3 = A_1 \cdot S_1^k, p_4 = A_2 \cdot S_2^k, \{p_u = S_1^u\}_{u=1, u\neq k}^{2k}, \{p_v = S_2^v\}_{v=1, v\neq k}^{2k}, q_0 = 1, t_0 = 1$, and $(\{\epsilon_{1,i}\}_{i=0}^{2\cdot k+2} \in \{0,1\}^*, \{\epsilon_{2,0}\} \in \{0,1\}^*, \{\epsilon_{T,0}\} \in \{0,1\}^*)$ are arbitrary distinct strings, $\mathcal{S}$ then sets those pairs $(p_i, \epsilon_{1,i})$ as L_1. Therefore, the three lists are initialized as $L_1 = \{(p_i, \epsilon_{1,i})\}_{i=0}^{2\cdot k+2}$, $L_2 = (q_0, \epsilon_{2,0}), L_T = (t_0, \epsilon_{T,0})$.

At the beginning of the game, $\mathcal{S}$ sends the encoding strings $(\{\epsilon_{1,i}\}_{i=0,\cdots,2\cdot k+2}, \epsilon_{2,0}, \epsilon_{T,0})$ to $\mathcal{A}$. After this, $\mathcal{S}$ simulates the group operation oracles O_1, O_2, O_T and the bilinear pairing $\hat{e}$ as follows. 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 $\mathcal{S}$ perform 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 $\mathcal{S}$ perform 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 $a_1, a_2, s_1, s_2, s \in \mathbb{Z}_q$ and $t_b = s_1^k + s_2^k$ and $t_{1-b} = s$. $\mathcal{S}$ sets $A_1 = a_1, A_2 = a_2, S_1 = s_1, S_2 = s_2, S = s, T_0 = t_b, T_1 = t_{1-b}$. 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) $p_i(a_1, a_2, s_1, s_2, t_0, t_1) = p_j(a_1, a_2, s_1, s_2, t_0, t_1)$: The polynomial $p_i \neq p_j$ due to the construction method of L_1, and $(p_i - p_j)(a_1, a_2, s_1, s_2, t_0, t_1)$ is a non-zero polynomial of degree $0, 1, k+1$ ($k+1$ is produced by $A_1 \cdot S_1^k$), or $2k$ ($2k$ is produced by S_1^{2k}). The maximum degree of $(p_i - p_j)(a_1, a_2, s_1, s_2, t_0, t_1)$ is $2k$. By using Lemma 1 in [44], we have $\Pr[(p_i - p_j)(a_1, a_2, s_1, s_2, t_0, t_1) = 0] \leq \frac{2k}{q}$ and thus $\Pr[p_i(a_1, a_2, s_1, s_2, t_0, t_1) = p_j(a_1, a_2, s_1, s_2, t_0, t_1)] \leq \frac{2k}{q}$. So, we have the abort probability is $\Pr[\text{abort}_1] \leq \frac{2k}{q}$.

(2) $q_i(a_1, a_2, s_1, s_2, t_0, t_1) = q_j(a_1, a_2, s_1, s_2, t_0, t_1)$: The polynomial $q_i \neq q_j$ due to the construction method of L_2, and $(q_i - q_j)(a_1, a_2, s_1, s_2, t_0, t_1)$ is a non-zero polynomial of degree 0. The abort probability is "0" (i.e., the maximum degree is "0" since the list L_2 contains a single string $\epsilon_{(2,0)}$ only). Recall that we do not include elements from group $\mathbb{H}$ here.

(3) $t_i(a_1, a_2, s_1, s_2, t_0, t_1) = t_j(a_1, a_2, s_1, s_2, t_0, t_1)$: The polynomial $t_i \neq t_j$ due to the construction method of L_3, and $(t_i - t_j)(a_1, a_2, s_1, s_2, t_0, t_1)$ is a non-zero polynomial of degree $0, k+1$, or $2k$. The maximum degree of $(t_i - t_j)(a_1, a_2, s_1, s_2, t_0, t_1)$ is $2k$. So, we have $\Pr[(t_i - t_j)(a_1, a_2, s_1, s_2, t_0, t_1) = 0] \leq \frac{2k}{q}$ and thus $\Pr[\text{abort}_3] \leq \frac{2k}{q}$.

By summing over all valid pairs (i, j) in each case (i.e., at most $2\binom{Q+2k+2}{2}$ pairs), we have the abort probability is

$$\Pr[\text{abort}] = \Pr[\text{abort}_1] + \Pr[\text{abort}_2] + \Pr[\text{abort}_3] \leq 2\binom{Q+2k+2}{2} \cdot \left(\frac{2k}{q} + \frac{2k}{q}\right) \leq \frac{4k(Q+2k+2)^2}{q}.$$

□

Theorem A.2. *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 $(a, b, c) \xleftarrow{R} \mathbb{Z}_q$ and encodings $\epsilon_1, \epsilon_2, \epsilon_T$ are randomly chosen, then we define the advantage of the adversary in solving the eDDH with at most Q queries to the group operation oracles O_1, O_2, O_T and the bilinear pairing $\hat{e}$ as*

$$\begin{aligned} |\mathsf{Adv}_{\mathcal{A}}^{eDDH}(\lambda)| &= \Pr[\mathcal{A}(q, \epsilon_1(1), \epsilon_1(a), \epsilon_1(b), \epsilon_1(ab), \epsilon_2(1), \\ &\quad \epsilon_2(c), \epsilon_2(ab), \epsilon_2(1/ab), \epsilon_2(abc) \\ &= b : (a, b, c, s \xleftarrow{R} \mathbb{Z}_q, b \in (0,1), t_b = c/ab, t_{1-b} = s)] \\ &\quad - 1/2| \leq \frac{8(Q+12)^2}{q} \end{aligned}$$

Proof. Let $\mathcal{S}$ play the following game with $\mathcal{A}$. $\mathcal{S}$ 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 $p_i \in \mathbb{Z}_q[A, B, C, T_0, T_1]$ are 5-variate polynomials over $\mathbb{Z}_q$, such that $p_0 = 1, p_1 = A, p_2 = B, p_3 = AB, q_0 = 1, q_1 = c, q_2 = ab, q_3 = 1/ab, q_4 = abc, q_5 = T_0, q_6 = T_1, t_0 = 1$, and $(\{\epsilon_{1,i}\}_{i=0}^3 \in \{0,1\}^*, \{\epsilon_{2,i}\}_{i=0}^5 \in \{0,1\}^*, \{\epsilon_{T,0}\} \in \{0,1\}^*)$ are arbitrary distinct strings. Therefore, the three lists are initialised as $L_1 = \{(p_i, \epsilon_{1,i})\}_{i=0}^3, L_2 = \{(q_i, \epsilon_{2,i})\}_{i=0}^6, L_T = (t_0, \epsilon_{T,0})$.

At the beginning of the game, $\mathcal{S}$ sends the encoding strings $(\{\epsilon_{1,i}\}_{i=0,\cdots,3}, \{\epsilon_{2,i}\}_{i=0,\cdots,6}, \epsilon_{T,0})$ to $\mathcal{A}$, which includes 12 strings. After this, $\mathcal{S}$ simulates the group operation oracles O_1, O_2, O_T and the bilinear pairing $\hat{e}$ using the same method as described in Theorem A.1.

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 $a, b, c, s \in \mathbb{Z}_q$ and $t_b = c/ab$ and $t_{1-b} = s$. $\mathcal{S}$ sets $A = a, B = b, C = c, T_0 = t_b, T_1 = t_{1-b}$. 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) $p_i(a, b, c, t_0, t_1) = p_j(a, b, c, t_0, t_1)$: The polynomial $p_i \neq p_j$ due to the construction method of L_1, and $(p_i - p_j)(a, b, c, t_0, t_1)$ is a non-zero polynomial of degree $[0, 2]$. By using Lemma 1 in [44], we have $\Pr[(p_i - p_j)(a, b, c, t_0, t_1) = 0] \leq \frac{2}{q}$ because the maximum degree of $A \cdot B(p_i - p_j)(a, b, c, t_0, t_1)$ is 2. So, we have $\Pr[p_i(a, b, c, t_0, t_1) = p_j(a, b, c, t_0, t_1)] \leq \frac{2}{q}$, and the abort probability is $\Pr[\text{abort}_1] \leq \frac{2}{q}$.

(2) $q_i(a, b, c, t_0, t_1) = q_j(a, b, c, t_0, t_1)$: The polynomial $q_i \neq q_j$ due to the construction method of L_2, and $(q_i - q_j)(a, b, c, t_0, t_1)$ is a non-zero polynomial of degree $[0, 3]$, or q-2 (q-2 is produced by $(AB)^{q-2}$). Since $AB \cdot (AB)^{q-2} = (AB)^{q-1} \equiv 1 \pmod q$, we have $CAB \cdot (AB)^{q-2} \equiv CAB \pmod q$. The maximum degree of $C/AB(q_i - q_j)(a, b, c, t_0, t_1)$ is 3, so the abort probability is $\Pr[\text{abort}_2] \leq \frac{3}{q}$.

(3) $t_i(a, b, c, t_0, t_1) = t_j(a, b, c, t_0, t_1)$: The polynomial $t_i \neq t_j$ due to the construction method of L_3, and $(t_i - t_j)(a, b, c, t_0, t_1)$ is a non-zero polynomial of degree $[0, 3]$, or q-2. Since $C/AB(t_i - t_j)(a, b, c, t_0, t_1)$ has degree 3, we have $\Pr[(t_i - t_j)(a, b, c, t_0, t_1) = 0] \leq \frac{3}{q}$ and thus $\Pr[t_i(a, b, c, t_0, t_1) = t_j(a, b, c, t_0, t_1)] \leq \frac{3}{q}$.

By summing over all valid pairs (i, j) in each case (i.e., at most $\binom{Q_{\epsilon_1}+4}{2} + \binom{Q_{\epsilon_2}+7}{2} + \binom{Q_{\epsilon_T}+1}{2}$ pairs), and $Q_{\epsilon_1} + Q_{\epsilon_2} + Q_{\epsilon_T} = Q + 12$, we have the abort probability is

$$\begin{aligned} \Pr[\text{abort}] &= \Pr[\text{abort}_1] + \Pr[\text{abort}_2] + \Pr[\text{abort}_3] \\ &\leq \left[\binom{Q_{\epsilon_1}+4}{2} + \binom{Q_{\epsilon_2}+7}{2} + \binom{Q_{\epsilon_T}+1}{2}\right] \cdot \left(\frac{2}{q} + 2\frac{3}{q}\right) \\ &\leq \frac{8(Q+12)^2}{q}. \end{aligned}$$

□

B SECURITY ANALYSIS OF PCHBA

In this section, we present the security analysis of the proposed PCHBA, including indistinguishability, collision-resistance, and accountability.

B.1 Proof of Theorem B.1

Theorem B.1. *The PCHBA achieves indistinguishability if the CHET 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 CHET, who is given a chameleon public key pk^* and a HashOrAdapt oracle, aims to break the indistinguishability of CHET. $\mathcal{S}$ randomly chooses $g \in [1, n(\lambda)]$ as a guess for the index of the HashOrAdapt query. In the g-th query, the distinguisher $\mathcal{S}$'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 creating k users with the corresponding identities $\{ID_i\}$. $\mathcal{S}$ randomly chooses a user as attribute authority and sets its chameleon public key as pk^*. $\mathcal{S}$ can honestly generate decryption keys for any transaction modifier associated with her identity ID_i and attribute set δ. If $\mathcal{A}$ submits a tuple (m_0, m_1, Λ) in the g-th query, then $\mathcal{S}$ first obtains a chameleon hash (h_b, r_b) from his HashOrAdapt oracle on messages (m_0, m_1). Then, $\mathcal{S}$ simulates a message-signature pair (c, σ) and a ciphertext C according to the protocol specification. Eventually, $\mathcal{S}$ returns $(h_b, \mathsf{r}_b, C, c, vk, \sigma)$ 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 CHET.

□

B.2 Proof of Theorem B.2

Theorem B.2. *The PCHBA achieves collision-resistance if the ABET is adaptive secure, and the CHET is collision-resistance.*

Proof. We define a sequence of games $\mathbb{G}_i$, $i = 0, \cdots, 3$ and let Adv_i^{PCHBA} 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 original game for collision-resistance.
- $\mathbb{G}_1$: This game is identical to game $\mathbb{G}_0$ 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 $(h^*, m^*, \mathsf{r}^*, C^*, c^*, vk^*, \sigma^*)$. $\mathcal{S}$ will output a random bit if $\mathcal{A}$'s attacking query does not occur in the g-th query. Therefore, we have
$$\mathsf{Adv}_0^{PCHBA} = n(\lambda) \cdot \mathsf{Adv}_1^{PCHBA} \tag{1}$$
- $\mathbb{G}_2$: This game is identical to game $\mathbb{G}_1$ except that in the g-th query, the encrypted message R^* in C^* is replaced by "$\perp$" ($\perp$ denotes an empty value). Below we show that the difference between $\mathbb{G}_1$ and $\mathbb{G}_2$ is negligible if ABET scheme is adaptive secure.
  Let $\mathcal{S}$ denote an attacker against ABET with semantic security, who is given a public key pk^*, a key generation oracle and a decryption oracle, aims to distinguish between encryptions of M_0 and M_1 associated with an identity and an access structure Λ. $\mathcal{S}$ simulates the game for $\mathcal{A}$ as follows.
  – $\mathcal{S}$ sets up $\mathsf{mpk}_{ABET} = \mathsf{pk}^*$ and completes the remainder of Setup honestly. $\mathcal{S}$ returns all public information, as well as chameleon secret key to $\mathcal{A}$.
  – $\mathcal{S}$ can honestly answer the queries made by $\mathcal{A}$ regarding decryption key generations and decryptions using his given oracles. In the g-th query, upon a Hash query w.r.t., a hashed message m^* from $\mathcal{A}$. $\mathcal{S}$ first submits two messages, an identity and a challenge policy (i.e., $[M_0 = \mathsf{R}^*, M_1 = \perp, ID, \Lambda]$) to his challenger, and obtains a challenge ciphertext C^*. Then, $\mathcal{S}$ returns the tuple $(h^*, m^*, \mathsf{r}^*, C^*, c^*, vk^*, \sigma^*)$ to $\mathcal{A}$. Note that $\mathcal{S}$ can simulate the message-signature pair (c^*, σ^*) honestly using the self-generated key pair (sk^*, vk^*). Besides, $\mathcal{S}$ can simulate the adapt query successfully using the self-generated chameleon key pair in CHET.
  If the encrypted message in C^* is real randomness R^*, then the simulation is consistent with $\mathbb{G}_1$; Otherwise, the simulation is consistent with $\mathbb{G}_2$. Therefore, if the advantage of $\mathcal{A}$ is significantly different in $\mathbb{G}_1$ and $\mathbb{G}_2$, $\mathcal{S}$ can break the semantic security of the ABET. Hence, we have
$$\left|\mathsf{Adv}_1^{PCHBA} - \mathsf{Adv}_2^{PCHBA}\right| \leq \mathsf{Adv}_{\mathcal{S}}^{ABET}(\lambda). \tag{2}$$
- $\mathbb{G}_3$: This game is identical to game $\mathbb{G}_2$ except that in the g-th query, $\mathcal{S}$ outputs a random bit if $\mathcal{A}$ outputs a valid collision $(h^*, m^{*'}, \mathsf{r}^{*'}, C^{*'}, c^{*'}, vk^{*'}, \sigma^{*'})$, and it was not previously returned by the Adapt oracle. Below we show that the difference between $\mathbb{G}_2$ and $\mathbb{G}_3$ is negligible if the CHET is collision-resistance.
  Let $\mathcal{S}$ denote an attacker against CHET with collision-resistance, who is given chameleon public key pk^*, 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 $\mathsf{pk}_{CHET} = \mathsf{pk}^*$ for the g-th hash query, and completes the remainder of Setup honestly. $\mathcal{S}$ returns all public information to $\mathcal{A}$.
  – $\mathcal{S}$ can simulate all queries made by $\mathcal{A}$ except the adapt queries. $\mathcal{S}$ simulates all user's decryption keys honestly using self-generated master key pair. If $\mathcal{A}$ submits an adapt query in the form of $(h, m, \mathsf{r}, C, c, \sigma, vk, m')$, then $\mathcal{S}$ obtains a randomness r' from his Adapt oracle, and returns $(h, m', \mathsf{r}', C', c', vk', \sigma')$ to $\mathcal{A}$. In particular, $\mathcal{S}$ simulates the g-th hash query as $(h^*, m^*, \mathsf{r}^*, C^*, c^*, vk^*, \sigma^*)$ with respect to the hashed message m^*, where $C^* \leftarrow \mathsf{Enc}_{ABET}(mpk_{ABET}, \perp, \Lambda^*, ID^*), c^* \leftarrow \mathsf{KeyGen}'(pp, sk^*, \perp)$, and key pair (sk^*, vk^*) is chosen by $\mathcal{S}$.
  – If $\mathcal{A}$ outputs a collision $(h^*, m^*, \mathsf{r}^*, C^*, c^*, vk^*, \sigma^*, m^{*'}, \mathsf{r}^{*'}, C^{*'}, c^{*'}, vk^{*'}, \sigma^{*'})$ with respect to the g-th query, and all the verifications defined in the collision-resistance model hold, then $\mathcal{S}$ output $(h^*, m^{*'}, \mathsf{r}^{*'}, C^{*'}, c^{*'}, vk^{*'}, \sigma^{*'})$ as a valid collision to CHET; Otherwise, $\mathcal{S}$ aborts the game. Therefore, we have
$$\left|\mathsf{Adv}_2^{PCHBA} - \mathsf{Adv}_3^{PCHBA}\right| \leq \mathsf{Adv}_{\mathcal{S}}^{CHET}(\lambda). \tag{3}$$
  Combining the above results together, we have
$$\mathsf{Adv}_{\mathcal{A}}^{PCHBA}(\lambda) \quad \leq \quad n(\lambda) \cdot (\mathsf{Adv}_{\mathcal{S}}^{ABET}(\lambda) + \mathsf{Adv}_{\mathcal{S}}^{CHET}(\lambda)).$$

□

B.3 Proof of Theorem B.3

Theorem B.3. *The PCHBA achieves accountability if the digital signature scheme Σ is EUF-CMA secure.*

Proof. The reduction is executed between an adversary $\mathcal{A}$ and a simulator $\mathcal{F}$. 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 $n(\lambda)$ chameleon hashes. $\mathcal{F}$ randomly chooses a chameleon hash, and sets up its

verification key as pk^*. $\mathcal{F}$ randomly chooses $g \in [1, n(\lambda)]$ as a guess for the index of the forgery with respect to that chameleon hash. $\mathcal{F}$ completes the remainder of Setup honestly. 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, and returns $(m, h, \mathsf{r}, C, c, vk, \sigma)$ to $\mathcal{A}$. The message-signature pairs and its verification keys can be perfectly simulated by $\mathcal{F}$ for any adapt query, due to the homomorphic property of Σ (regarding keys and signatures) and the secret R is chosen by $\mathcal{F}$. $\mathcal{F}$ records all the simulated chameleon hashes by including them to a set Q.
When forging attack occurs, i.e., $\mathcal{A}$ outputs $(m^*, h^*, \mathsf{r}^*, C^*, c^*, vk^*, \sigma^*)$, $\mathcal{F}$ checks whether:

- the forging attack happens at g-th session;
- the ciphertext C^* encrypts the ephemeral trapdoor R;
- the message-signature pair (c^*, σ^*) links to pk^* (due to the ephemeral trapdoor R);
- the message-signature pair $(c^*, \sigma^*) \notin Q$ (i.e., it was not previously simulated by $\mathcal{F}$);
- $1 \leftarrow \Sigma.\mathsf{Verify}(vk^*, c^*, \sigma^*)$ and $1 \leftarrow \mathsf{Verify}(h^*, m^*, \mathsf{r}^*)$ (i.e., verify algorithm in CHET).

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 $\sigma \leftarrow M_\Sigma(pp, \mathsf{pk}^*, \sigma^*, \Delta(\mathsf{sk}))$ due to the homomorphic property of Σ, 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.

□

C SECURITY ANALYSIS OF ABET

In this section, we present the security analysis of the proposed ABET scheme, including semantic security and ciphertext anonymity.

C.1 Semantic Security

Informally, an ABET scheme is secure against chosen plaintext attacks if no group of collude users can distinguish between encryption of M_0 and M_1 under an identity and an access structure Λ^* of an attacker's choice as long as no member of the group is authorized to decrypt on her own. The adaptive security is defined similarly except that the identity and access structure Λ^* are chosen by attackers at the time of challenge stage. The adaptive security model used here is based on the semantic security models defined in [9, 14].

Theorem C.1. *The proposed ABET scheme achieves the adaptive semantic security in the random oracle model if the DLE assumption is held in the asymmetric pairing groups.*

Compact Group Representation [21]. We use $[a]_1, [b]_2$ to denote g^a, h^b respectively, for $g \in \mathbb{G}$ and $h \in \mathbb{H}$. If $\mathbf{v}$ is a vector given by $(v_1, v_2, \cdots, v_n)^{\mathrm{T}}$, then $[\mathbf{v}]_1$ means $(g^{v_1}, g^{v_2}, \cdots, g^{v_n})^{\mathrm{T}}$. $[\mathbf{M}]_1$ for a matrix $\mathbf{M}$ is defined similarly. $\hat{e}([\mathbf{A}]_1, [\mathbf{B}]_2)$ for two matrices $\mathbf{A}, \mathbf{B}$ is defined as $[\mathbf{A}^{\mathrm{T}} \cdot \mathbf{B}]_T$. If we define $\mathbf{A}, \mathbf{s}, \mathbf{s'}$ $\mathbf{s_1}^*$, and $\mathbf{s_2}^*$ as

$$\begin{bmatrix} a_1 & 0 \\ 0 & a_2 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} s_1^k \\ s_2^k \end{bmatrix}, \begin{bmatrix} s_1^k \\ s_2^k \\ s \end{bmatrix}, \begin{bmatrix} s_1 \\ \cdots \\ s_1^{k-1} \\ s_1^{k+1} \\ \cdots \\ s_1^{2k} \end{bmatrix}, and \begin{bmatrix} s_2 \\ \cdots \\ s_2^{k-1} \\ s_2^{k+1} \\ \cdots \\ s_2^{2k} \end{bmatrix},$$

then we can state the DLE assumption (with terms from group $\mathbb{G}$ only) as

$$([\mathbf{A}]_1, [\mathbf{As}]_1, [\mathbf{s}_1^*]_1, [\mathbf{s}_2^*]_1) \approx ([\mathbf{A}]_1, [\mathbf{s'}]_1, [\mathbf{s}_1^*]_1, [\mathbf{s}_2^*]_1)$$

where $\approx$ denotes the computational indistinguishability. If we include terms from both $\mathbb{G}$ and $\mathbb{H}$, then we state the DLE assumption as

$$\begin{aligned}&([\mathbf{A}]_1, [\mathbf{B}]_2, [\mathbf{s}_1^*]_1, [\mathbf{s}_2^*]_1, [\mathbf{s}_1^*]_2, [\mathbf{s}_2^*]_2)\\ &\approx ([\mathbf{A}]_1, [\mathbf{B}]_2, [\mathbf{s}_1^*]_1, [\mathbf{s}_2^*]_1, [\mathbf{s}_1^*]_2, [\mathbf{s}_2^*]_2, [\mathbf{s'}]_1, [\mathbf{s'}]_2)\end{aligned}$$

Proof of Sketch. The first step in the security analysis is to rewrite ABET scheme in a compact form by interpreting the outputs of random oracle appropriately and using the above notation to represent group elements. This compact form is the first hybrid, Hyb_1. Let Samp be an algorithm that takes security parameter λ as input, outputs

$$\mathbf{A} = \begin{bmatrix} a_1 & 0 \\ 0 & a_2 \\ 1 & 1 \end{bmatrix}, \mathbf{a}^\perp = \begin{bmatrix} a_1^{-1} \\ a_2^{-1} \\ -1 \end{bmatrix}$$

where $a_1, a_2 \in \mathbb{Z}_q^*$. The simulator $\mathcal{S}$ simulates the ABET scheme as follows.

- Setup: $\mathcal{S}$ sets up the group elements honestly $(\mathbf{A}, \mathbf{a}^\perp), (\mathbf{B}, \mathbf{b}^\perp) \leftarrow \mathsf{Samp}(1^\lambda)$, and chooses $(d_1, d_2, d_3, \alpha, \beta, \{z_1, \cdots, z_k\}) \leftarrow \mathbb{Z}_q$. Let $\mathbf{d} = (d_1, d_2, d_3)^{\mathrm{T}}, \mathbf{d'} = (d_1, d_2, d_3/\alpha)^{\mathrm{T}}, \mathbf{a} = (a_1, a_2)^{\mathrm{T}}, \mathbf{b} = (b_1, b_2)^{\mathrm{T}}$, $\mathbf{z} = (z_1, \cdots, z_k)^{\mathrm{T}}$. $\mathcal{S}$ also generates some variants

$$\mathbf{A'} = \begin{bmatrix} a_1 & 0 \\ 0 & a_2 \\ \alpha^{-1} & \alpha^{-1} \end{bmatrix}, \mathbf{a}^{\perp'} = \begin{bmatrix} a_1/\alpha \\ a_2/\alpha \\ -1 \end{bmatrix}$$

  , sets $\mathsf{mpk} = ([\mathbf{A}]_2, [\mathbf{d'}^{\mathrm{T}}\mathbf{A}]_T, [\mathbf{z}]_1, [\mathbf{z}]_2, [\alpha]_1, [1/\alpha]_2, [d/\alpha]_2, [\beta/\alpha]_2)$, and $\mathsf{msk} = ([\mathbf{d}]_1, \mathbf{a}, \mathbf{b}, \alpha, \beta, [z]_1)$, where $d = d_1 + d_2 + d_3$.
  To simulate the random oracle, $\mathcal{S}$ maintains two lists L and Q. The list L has entries of the form $(y, \mathbf{W}_y)$ or $(v, \mathbf{U}_v)$ where y is an arbitrary binary string, v is a positive integer, and $\mathbf{W}_y, \mathbf{U}_v$ are 3×3 matrices over $\mathbb{Z}_q$. The list Q has entries of the form (q, r) where q is either $y\ell t$ or $0v\ell t$ (for $\ell \in \{1, 2, 3\}$ and $t \in \{1, 2\}$) or something else, and hash output $r \in \mathbb{G}$. $\mathcal{S}$ answers the hash oracle queries using the same method described in [9].
- KeyGen: If $\mathcal{A}$ issues a key query in the form of (ID_i, δ), then $\mathcal{S}$ retrieves $\mathbf{W_y}$ for every $y \in \delta$ and $\mathbf{U_1}$ from the list L (If one of them is not found, then a random 3×3 matrix is generated using the same method described in [9]. The list L is also updated accordingly). Now, $\mathcal{S}$ chooses $r_1, r_2, \sigma' \in \mathbb{Z}_q$ and $\sigma_y, R \in \mathbb{Z}_q$ for $y \in \delta$. Let $\mathbf{r} = (r_1, r_2)^{\mathrm{T}}, \mathbf{r'} = (r_1/\alpha, r_2/\alpha)^{\mathrm{T}}$ and computes the decryption key as $\mathsf{sk}_0 = ([\mathbf{B'r}]_2, [r]_1, [r/\alpha]_1, [R]_1), \mathsf{sk}_y = [\mathbf{W_y B'r} + \sigma_y \mathbf{a}^{\perp'}]_1, \mathsf{sk'} = [\mathbf{d} + \mathbf{U_1 B'r} + \sigma' \mathbf{a}^{\perp'}]_1, \mathsf{sk}_1 = [d +$

$(\Sigma_1^i zI)\alpha r + \beta R]_1, \mathsf{sk}_2 = [(\Sigma_{i+1}^k zI)\alpha(r+1)]_1$, where the randomness is $r = r_1 + r_2$, and $ID_i = \{I_1, \cdots, I_i\}$. Eventually, $\mathcal{S}$ returns $(\mathsf{sk}_0, \{\mathsf{sk}_y\}_{y\in\delta}, \mathsf{sk}', \mathsf{sk}_1, \mathsf{sk}_2)$ to $\mathcal{A}$.

- Enc: If $\mathcal{A}$ sends messages (M_0, M_1), an identity ID_j, and a policy $(\mathbf{M}, \pi)$ to $\mathcal{S}$, $\mathcal{S}$ retrieves hash output $r = [(\mathbf{W}_{\pi(i)}^{\mathrm{T}}\mathbf{A})_{\ell,t}]_1$ and $r = [(\mathbf{U}_v^{\mathrm{T}}\mathbf{A})_{\ell,t}]_1$ for all $i \in [1, \cdots, n_1], v \in [1, \cdots, n_2], \ell, t$ from the list $\mathcal{Q}$ (if a $\pi(i)\ell t$ or $0v\ell t$ is not found in $\mathcal{Q}$, then it follows the same process as $\mathcal{S}$ simulates the random oracles previously). Now, $\mathcal{S}$ chooses $(s_1, s_2) \in \mathbb{Z}_q$, sets $\mathbf{s} = (s_1, s_2)$ and computes the challenge ciphertext as $ct_0 = ([\mathbf{A's}]_2, [\beta s/\alpha]_2)$, $ct_i = [\mathbf{W}_{\pi(i)}^{\mathrm{T}}\mathbf{As} + \sum_{v=1}^{n_2}(\mathbf{M})_{i,v}\mathbf{U}_v^{\mathrm{T}}\mathbf{As}]_1$, $ct = [\mathbf{d'^{T}As}]_T \cdot M_b$, $ct' = [ds/\alpha]_{\mathrm{T}} \cdot M_b$, $ct_1 = [(\Sigma_1^j zI)\alpha s]_2, ct_2 = [(\Sigma_1^j zI)s]_2$, $ct_3 = [(\Sigma_1^j zI)\alpha s^2]_2$, in which the shared secret is $s = s_1 + s_2$, and $ID_j = \{I_1, \cdots, I_j\}$.

Description of Hybrids. The security proof consists of a set of hybrids from Hyb_0 to Hyb_5. A hybrid describes how the simulator $\mathcal{S}$ interacts with an adversary $\mathcal{A}$. The Hyb_0 is the one where $\mathcal{S}$ and $\mathcal{A}$ interacts according to the (formal) semantic security game as described in [9]. The Hyb_1 has already been discussed above. In the subsequent hybrids, the indistinguishability between two hybrids can be either computationally-close or statistically-close. We stress that the security proof here uses the same proof approach described in [9], except the indistinguishability between two hybrids with computationally-close is reduced to the proposed DLE assumption.

C.2 Ciphertext Anonymity

Informally, ciphertext anonymity requires that any third party cannot distinguish the encryption of a chosen message for a first chosen identity from the encryption of the same message for a second chosen identity. In other words, the attacker cannot decide whether a ciphertext was encrypted for a chosen identity, or a random identity [8]. The identity can be chosen either at the setup stage (i.e., selective anonymity) or at the challenge stage (i.e., adaptive anonymity) in the security game. We prove the ABET scheme secures against selective-ID ciphertext anonymity because we require short ciphertext and secret keys.

Theorem C.2. *The proposed ABET scheme achieves selective-ID ciphertext anonymity if the eDDH assumption is held in the asymmetric pairing groups.*

Proof. Let $\mathcal{S}$ denote an eDDH problem distinguisher, who is given $(g, h, g^a, g^b, g^{ab}, h^{ab}, h^{1/ab}, h^{abc}, h^c)$, and aims to distinguish $h^{c/ab}$ and h^s. We add additional group elements from $\mathbb{G}$ for simulating user's decryption keys, and these extra group elements are no help in distinguishing $h^{c/ab}$ from random. The reduction is performed as follows.

- $\mathcal{S}$ sets up the master public key as $\mathsf{mpk} = (g, h, H_1, H_2, T_1, T_2, \{g_1, \cdots, g_k\}, \{g_1^{ab}, \cdots, g_k^{ab}\}, \{h_1, \cdots, h_k\}, g^{ab}, h^{d/ab}, h^{1/ab}, h^{\beta/ab})$, while the remaining master secret keys are honestly chosen by $\mathcal{S}$, including $(\cdots, d, \beta, \{z_i\})$. $\mathcal{S}$ implicitly simulates $\alpha = ab$. $\mathcal{A}$ selects a challenge user $ID^* = \{I_1, \cdots, I_j\}$ and submits it to $\mathcal{S}$.
- $\mathcal{S}$ simulates a user ID_i's decryption key as $ssk_i = (\mathsf{sk}_0, \mathsf{sk}_1, \cdots)$, where $\mathsf{sk}_0 = (\cdots, g^r, g^{\beta\cdot R} \cdot g^{-\Sigma(z_i I_i)r\cdot b})$, $\mathsf{sk}_1 = g^d \cdot g^{a\cdot R}, ID_i = (g_1^{I_1} \cdots g_i^{I_i} \cdot g), (r, R) \in \mathbb{Z}_q$. In particular, the simulated components $(g^r, g^{\beta\cdot R} \cdot g^{-\Sigma(z_i I_i)r\cdot b}, g^d \cdot g^{a\cdot R})$ are correctly distributed, because $g^d \cdot g^{a\cdot R} = g^d \cdot ID_i^{a\cdot b\cdot r} \cdot g^{a[\beta\cdot R - \Sigma(z_i I_i)r\cdot b]} = g^d \cdot ID_i^{\alpha\cdot r} \cdot g^{a\cdot\overline{R}}$, where $\overline{R} = \beta \cdot R - \Sigma(z_i I_i)r \cdot b$. So, the simulated components are $(g^r, g^{\overline{R}}, g^d \cdot ID_i^{\alpha\cdot r} \cdot g^{a\cdot\overline{R}})$, which match the real distribution.
- $\mathcal{S}$ simulates the challenge ciphertext w.r.t. user ID^* as $C = (ct_0, \cdots, ct_1)$: $ct_0 = (H_1^c, H_2^{s_2}, h^{(c+s_2)/ab}, h^{\beta\cdot(c+s_2)/ab})$, where $s_2 \in \mathbb{Z}_q$, and $ct_1 = h_k^{abI_1(c+s_2)} \cdots h_j^{abI_j(c+s_2)} \cdot h^{c+s_2}$. $\mathcal{S}$ implicitly simulates $s_1 = c$, and the involved $h^{c/ab}$ could be 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. □