StrongPCH: Strongly Accountable Policy-based Chameleon Hash for Blockchain Rewriting

ABSTRACT

Policy-based chameleon hash (PCH) is a useful primitive in blockchain rewriting. It allows a party to compute a chameleon hash associates with an access policy, and another party who possesses enough privileges satisfying the access policy can rewrite the hashed object. However, the PCH lacks strong accountability. The chameleon trapdoor holder may abuse his/her rewriting privilege and maliciously rewrite the hashed object without being identified. In this paper, we introduce a new primitive called strongly accountable policy-based chameleon hash StrongPCH. We first present a generic framework of StrongPCH. Then, we present a practical instantiation, and show its practicality through implementation and evaluation analysis.

KEYWORDS

Blockchain Rewriting, Chameleon Hash, Strong Accountability.

ACM Reference Format:
. 2021. StrongPCH: Strongly Accountable Policy-based Chameleon Hash for Blockchain Rewriting. In *Proceedings of ACM Conference (Conference'17).*

1 INTRODUCTION

Blockchain technologies have received tremendous attention recently. Blockchain was initially used in the context of Bitcoin networks, such that all payment transactions are appended to a public ledger (or blockchain). Each transaction is verified by network nodes in a peer-to-peer manner [23]. 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 [16]) executed by the network nodes. Blockchain can be viewed as a hash-chain, where the hash of a block links to the next block in the chain. Each block includes a set of transactions that are accumulated into a single hash value using the Merkle tree [21], and each transaction contains any object of interest that needs to be recorded.

The transactions appended in the blockchain are supposed to be immutable, the recorded object cannot be modified. However, rewriting the "immutable" transaction is necessary in practice, or even being legally obliged in data regulation laws such as GDPR [1]. Blockchain rewriting can be realized by replacing a standard hash function, used for generating transaction hash in the blockchain, by a trapdoor-based chameleon hash [17]. Then, the users who are given the trapdoor (we call them authorized modifiers) can modify mutable transactions. In other words, the same transaction can be modified by modifiers with the same privileges only. Nonetheless, for many real-life blockchain applications, blockchain rewriting with fine-grained access control is desired so that various rewriting privileges can be granted to different modifiers and that the same transaction can be modified by modifiers with different privileges. For blockchain rewriting with fine-grained access control, a user first associates his transaction with an access policy. Then, an authorized modifier possessing a chameleon trapdoor can modify the transaction if her rewriting privilege corresponding to the chameleon trapdoor satisfies the embedded access policy. It is possible that multiple modifiers with different rewriting privileges can modify the same transaction if their rewriting privileges satisfy the access policy associated with the transaction.

Motivation. Blockchain rewriting with fine-grained access control has been recently studied in the literature [12, 29]; however, the proposed solutions suffer from the *rewriting privilege abuse attacks*. The authorized modifiers may abuse (e.g., update or delegate) their given rewriting privileges and distribute them to some users who have no given rewriting privileges. As a result, the users (we call them unauthorized modifiers) can rewrite transactions successfully using the abused rewriting privileges.

Strong accountability against the rewriting privilege abuse attacks is critical to blockchain rewriting. We consider two blockchain rewriting cases: 1) an authorized modifier rewrites a transaction using her rewriting privilege granted from a trusted authority. 2) an unauthorized modifier has no rewriting privilege from the trusted authority; he may still rewrite transactions using an abused rewriting privilege. If a transaction is maliciously modified, the trusted authority should link the modified transaction to an authorized modifier (in case one) or an unauthorized modifier (in case two). In case two, the authority should also identify an original modifier from an abused rewriting privilege, meaning that this original modifier is also responsible for the modified transaction.

This work. We introduce strongly accountable policy-based chameleon hash (StrongPCH) for blockchain rewriting applications. Strong accountability means that, given a modified transaction, the trusted authority can link the modified transaction to responsible modifiers even if rewriting privileges are abused in generating the modified transaction. Overall, the major contributions of this work are summarized as follows.

- *Generic Framework.* We introduce the *first* generic framework of StrongPCH for blockchain rewriting. The framework's unique feature is that it enables the modifiers' public keys to be accountable for the modified transactions, regardless of whether the modifiers are authorized or not.
- *Practical Instantiation.* We present a *practical* instantiation, and validate its practicality. To this end, we propose a practical attribute-based encryption with traceability and an anonymized digital signature scheme, which are of independent interest.

2 OVERVIEW

In this section, we provide an overview of StrongPCH and its design rationale. The StrongPCH is constructed from attribute-based encryption with traceability (ABET), chameleon hash function (CH),

Conference'17, July 2017, Washington, DC, USA
2021. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and digital signature. Let us consider a three-party setting, including a user, a modifier, and an attribute authority (AA). If a user appends a transaction to the blockchain, he outputs a policy-based chameleon hash that includes a chameleon hash, a randomness, a ciphertext, and a signature. In this process, the user chooses a trapdoor and encrypts it. The fine-grained blockchain rewriting can be achieved by restricting who can decrypt the trapdoor. To generate a signature, the user randomizes the trapdoor using his signing key and signs the randomized trapdoor (note that the randomized trapdoor is the signed message). Later, a modifier who possesses enough rewriting privileges can rewrite the transaction because she can obtain the trapdoor by decrypting the ciphertext. To complete the process of rewriting, the modifier also randomizes the trapdoor using her signing key and signs the randomized version of the trapdoor. The intention is to use the same trapdoor to link message-signature pairs so as to achieve strong accountability.

We achieve strong accountability via the following steps. The first step is to find the linked transactions. Given a set of transactions, any public user (including AA) 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 trapdoor. The second step is to find the identity of the revealed rewriting privilege. Given a modifier's rewriting privilege (or decryption key), the modifier's identity can be retrieved by AA due to ABET's traceability. Note that the revealed rewriting privileges can be the abused ones. The third step is that AA links the modified transaction to the modifier's public key using its master secret key. We stress that only AA can link the message-signature pair's corresponding verification key involved in the modified transaction to the modifier's public key.

For constructing a practical StrongPCH, we start with the construction of ABET first. We rely on the most efficient ABE scheme [4], and a commitment scheme [15] to built an ABET scheme. The proposed ABET scheme works as follows: 1) the master key pair in ABE includes a binding key pair from the commitment scheme; 2) the modifier's decryption key includes a commitment on her identity (note that commitment can be regarded as encryption of her identity); 3) AA can extract the modifier's identity from a revealed decryption key using its master secret key. Second, We use a discrete logarithm (DL)-based CH [17] for blockchain rewriting. Third, we propose an anonymized signature scheme to ensure accountable and anonymous blockchain rewritings. Anonymity means that, given two transactions, the public neither know which one is modified nor which modifier is responsible for the modified transaction. To conclude, the proposed ABET scheme and the anonymized signature scheme can be regarded as the main technical contribution in this work.

3 PRELIMINARIES

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

3.1 Complexity Assumption

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 group generation as $(q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \hat{e}) \leftarrow \mathsf{GroupGen}(1^\lambda)$, where q is a prime number, $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$ are cyclic groups of order q, and $\hat{e} : \mathbb{G}\times\mathbb{H} \to \mathbb{G}_T$ is a bilinear map such that: (1) Bilinearity: $\forall g, h \in \mathbb{G}$ and $a, b \in \mathbb{Z}_q$, we have $\hat{e}(g^a, h^b) = \hat{e}(g, h)^{ab}$; (2) Non-degeneracy: $\hat{e}(g, h)$ has order q in $\mathbb{G}_T$. We assume that group operations in $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$ and bilinear map $\hat{e}$ 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 variant of the decisional linear assumption, which is used to prove the semantic security the proposed ABET scheme.

Definition 1 (Extended Decisional Linear Assumption). *Given group generators g and h, define the following distribution:*

$$\begin{aligned}
\mathsf{Adv}^{eDLIN}_{\mathcal{A}}(\lambda) \quad = \quad & |\Pr[\mathsf{Adv}(1^\lambda, \mathsf{par}, D, T_0) = 1] \\
& - \Pr[\mathsf{Adv}(1^\lambda, \mathsf{par}, D, T_1) = 1]|, \textit{where} \\
& \mathsf{par} = (q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \hat{e}, g, h) \leftarrow \mathsf{GroupGen}(1^\lambda) \\
& a_1, a_2 \leftarrow \mathbb{Z}_q^*, s_1, s_2, z, s \leftarrow \mathbb{Z}_q; w \in \{0,1\}; \\
& D = (g^{a_1}, g^{a_2}, \underline{g^{1/a_1}, g^{1/a_2}}, h^{a_1}, h^{a_2}, \\
& g^{a_1 \cdot s_1}, g^{a_2 \cdot s_2}, \underline{g^{s_1/a_1}, g^{s_2/a_2}, g^{a_1 \cdot s_2}, g^{a_2 \cdot s_1}}, \\
& h^{a_1 \cdot s_1}, h^{a_2 \cdot s_2}, \underline{g^z, h^z}); \\
& T_w = (g^{z(s_1+s_2)}, h^{z(s_1+s_2)}); T_{1-w} = (g^s, h^s).
\end{aligned}$$

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

The underline part is the key difference between DLIN and eDLIN. The eDLIN has additional terms from group $\mathbb{G}$, and one can add the similar terms from group $\mathbb{H}$ into the above assumption. We also introduce two variants of the bilinear Diffie-Hellman assumption, which are used to prove the unforgeability and anonymity of the proposed digital signature scheme.

Definition 2 (Computational Bilinear Diffie-Hellman). *Given group generators $g \in \mathbb{G}$ and $h \in \mathbb{H}$, define the following distribution:*

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

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

Definition 3 (Decisional Bilinear Diffie-Hellman). *Given group generators $g \in \mathbb{G}$ and $h \in \mathbb{H}$, define the following distribution:*

$$\begin{aligned}
\mathsf{Adv}^{DBDH}_{\mathcal{A}}(\lambda) \quad = \quad & |\Pr[\mathsf{Adv}(1^\lambda, \mathsf{par}, D, T_0) = 1] \\
& - \Pr[\mathsf{Adv}(1^\lambda, \mathsf{par}, D, T_1) = 1]|, \textit{where} \\
& \mathsf{par} = (q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \hat{e}, g, h) \leftarrow \\
& \mathsf{GroupGen}(1^\lambda), a, b, d, e, f \leftarrow \mathbb{Z}_q^*, \\
& \{c, l\} \in \mathbb{Z}_q, w \in \{0,1\}; \\
& D = (g^a, g^b, g^f, \underline{g^{ed}, g^{ec_i}, g^{el_i}, \forall i \in [q]}) \\
& T_w = g^{ab+edc_i}, T_{1-w} = g^{fb+edl_i}.
\end{aligned}$$

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

In the proposed DBDH assumption, an adversary is given instances either all from $\mathbb{G}$ or from $\mathbb{H}$. Since $(g^d, \{g^{c_i}, g^{l_i}\})$ or $(h^d, \{h^{c_i}, h^{l_i}\})$, are missing from D, the additional terms (underline part) are of no help in deciding T_w. We prove the proposed eDLIN, DBDH and CBDH assumptions in the generic group model [28]. The detailed theorems and proofs are referred to Appendix A.

3.2 Access Policy

Access Structure. Let $\mathcal{U}$ be an attribute universe. 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 Programs (MSPs)[25]. A secret-sharing scheme $\prod$ with domain of secrets $\mathbb{Z}_q$ 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 exists a matrix $\mathbf{M}$ with n_1 rows and n_2 columns called the share-generating matrix for $\prod$. For $i = 1, ..., n_1$, we define a function π labels row i of $\mathbf{M}$ with attribute $\pi(i)$ from the attribute universe $\mathcal{U}$. When we consider the column vector $\vec{v} = (s, r_2, ..., r_{n_2})$, 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 $(\mathbf{M}\vec{v})_j$ belongs to attribute $\pi(j)$, where $j \in [n_2]$.

According to [7], every linear secret-sharing scheme has the linear reconstruction property, which is defined as follows: we assume that $\prod$ is an MSP for the access structure Λ, $1 = \Lambda(\delta)$ is an authorized set and let $I \subset \{1, 2, ..., n_1\}$ be defined as $I = \{i \in [n_1] \wedge \pi(i) \in \delta\}$. There exist the constants $\{\gamma_i \in \mathbb{Z}_q\}_{i \in I}$ such that for any valid shares $\{\lambda_i = (\mathbf{M}\vec{v})_i\}_{i \in I}$ of a secret s according to $\prod$, $\sum_{i \in I} \gamma_i \lambda_i = s$. Meanwhile, these constants $\{\gamma_i\}_{i \in I}$ can be found in time polynomial in the size of the share-generating matrix $\mathbf{M}$. For any unauthorized set δ', no such $\{\gamma_i\}$ exist.

3.3 Building Blocks

(1) *Chameleon Hash (CH).* It consists of the following algorithm [17].
- KeyGen: It takes a security parameter λ as input, outputs a chameleon key pair $(\mathsf{sk}, \mathsf{pk})$.
- Hash: It takes the chameleon public key pk, and a message $m \in \mathcal{M}$ as input, outputs a chameleon hash b, a randomness r. Here $\mathcal{M} = \{0,1\}^*$ denotes a general message space.
- Verify: It takes the chameleon public key pk, message m, chameleon hash b and randomness r as input, outputs a bit $b \in \{0,1\}$.
- Adapt: It takes the chameleon secret key sk, messages m, m', chameleon hash b and randomness r as input, outputs a new randomness r'.

We assume that the Adapt algorithm always verifies if the chameleon hash b given is valid, or it outputs $\perp$. Correctness is straightforward and is given in [17]. For security, chameleon hash function needs to achieve collision-resistance and uniformity. Informally speaking, collision-resistance means that any malicious party cannot find two pairs (m, r) and (m', r') that map to the same chameleon hash b. Uniformity requires that the output of Hash is uniformly distributed. Besides, for a uniformly random value r, the new randomness r' is also a uniformly distributed random value. The formal definitions of collision-resistance and uniformity are given in [17].

```
Experiment $\mathsf{Exp}_{\mathcal{A}}^{IND\text{-}CCA2}(\lambda)$
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $id \in \mathcal{I}$, $b \leftarrow \{0,1\}$
$(m_0, m_1, \Lambda^*) \leftarrow \mathcal{A}^{O^{\mathsf{KeyGen}}, O^{\mathsf{Dec}}}(\mathsf{mpk})$
    where $dk \leftarrow O^{\mathsf{KeyGen}}(\mathsf{msk}, id, \delta)$, $m \leftarrow O^{\mathsf{Dec}}(\mathsf{msk}, C)$
$C^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, \Lambda^*, m_b)$
$b' = \mathcal{A}^{O^{\mathsf{KeyGen}}, O^{\mathsf{Dec}}}(C^*)$
return 1, if $b' = b$; else, return 0
```

Figure 1: IND-CCA2 Security.

(2) *Attribute-based Encryption with Traceability (ABET).* It consists of the following algorithms, and this definition is inspired by [20].
- Setup: It takes a security parameter λ as input, outputs a master key pair $(\mathsf{msk}, \mathsf{mpk})$.
- KeyGen: It takes the master secret key msk, a user's identity $id \in \mathcal{I}$, and an attribute set $\delta \in \mathcal{U}$ as input, outputs a decryption key dk, which is associated with user's identity. Here the identity space $\mathcal{I} \in \{0, 1, \cdots, I\}$ should be small such that it can be efficiently searched exhaustively, where I is a polynomial in λ.
- Enc: It takes the master public key mpk, a message m, and an access structure Λ as input, outputs a ciphertext C. The C implicitly contains Λ.
- Dec: It takes the master public key mpk, ciphertext C, and the decryption key dk as input, outputs message m if $1 = \Lambda(\delta)$.
- Trace: It takes the master secret key msk, and a decryption key dk as input, outputs an identity id to which the decryption key dk associates.

The ABET is *correct* if for all security parameters λ, all keys $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(1^\lambda)$, for all $\delta \in \mathcal{U}$, for all $id \in \mathcal{I}$, $dk \leftarrow \mathsf{KeyGen}(\mathsf{msk}, id, \delta)$, for all $m \in \mathcal{M}$, $C \leftarrow \mathsf{Enc}(\mathsf{mpk}, m, \Lambda)$, we have $m \leftarrow \mathsf{Dec}(\mathsf{mpk}, dk, C)$ if $1 = \Lambda(\delta)$. The ABET scheme includes two security guarantees: semantic security and traceability.
- 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 access structure Λ^* of an attacker's choice as long as no member of the group is authorized to decrypt on her own. The indistinguishability under adaptive chosen ciphertext attack (IND-CCA2) security is defined similarly except that the access structure Λ^* is chosen by attackers at the time of challenge stage.

Definition 4. *The IND-CCA2 experiment between a PPT adversary $\mathcal{A}$ and a simulator $\mathcal{S}$ is defined below.*
In the experiment, $\mathcal{A}$ is not allowed to query O^{Dec} on the challenge ciphertext C^*, and the challenge attributes set satisfies $1 \neq \Lambda^*(\delta)$ (otherwise, trivial attacks would occur). We define the advantage of the adversary as

$$\mathsf{Adv}_{\mathcal{A}}^{IND\text{-}CCA2}(\lambda) = |\Pr[\mathcal{S} \rightarrow 1] - 1/2|.$$

An ABET scheme is semantically secure if $\mathsf{Adv}_{\mathcal{A}}^{IND\text{-}CCA2}(\lambda)$ is negligible in λ.

- Traceability. Informally, given a decryption key (e.g., a malicious user may distribute or sell her decryption key to the public) and the master secret key, the user's identity can be extracted. We show a formal definition for traceability as follows.

  Definition 5. *An ABET scheme is traceable if for any PPT $\mathcal{A}$,*

  $$|\Pr[id \neq \mathsf{Trace}(\mathtt{msk}, dk) : (\mathtt{msk}, dk) \leftarrow \mathcal{A}(1^{\lambda})]|$$

  is negligible in λ.

(3) *Non-interactive Zero-knowledge Proof.* Let R be an efficiently computable binary relation, and L be an NP-language consisting of statements in R, such that $L = \{x | \exists w : R(x, w) = 1\}$, where x is the statement and w is the witness. A non-interactive zero knowledge proof system [15] for a relation R is shown below.
- $\mathsf{Setup}(1^{\lambda})$: It takes a security parameter λ as input, outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, x, w)$: It takes the common reference string crs, a statement x, and a witness w as input, outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi)$: It takes the common reference string crs, a statement x, and a proof π as input, outputs $1 \overset{?}{=} \mathsf{Verify}(\mathsf{crs}, x, \pi)$.

The non-interactive proof system for relation R must satisfy completeness and soundness, and some special properties that include proof of knowledge (PoK), zero-knowledge, and witness indistinguishability. Since we focus on traceability in this work, we present the definition of PoK only, other definitions are referred to [15]. Informally, a proof system is a proof of knowledge (PoK) if the witness can be extracted from the proof.

Definition 6. *A non-interactive proof system is PoK for R if there exists a PPT knowledge extractor $E = (E_1, E_2)$, such that E_1 returns a correctly distributed common reference string* crs *with an extraction key ξ that allows E_2 to extract a witness from a proof. For any PPT $\mathcal{A}$, we have*

$$\begin{aligned} & |\Pr[\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda}) : \mathcal{A}(\mathsf{crs}) = 1| \\ = \quad & |\Pr[(\mathsf{crs}, \xi) \leftarrow E_1(1^{\lambda}) : \mathcal{A}(\mathsf{crs}) = 1|, \end{aligned}$$

and

$$\begin{aligned} & |\Pr[(\mathsf{crs}, \xi) \leftarrow E_1(1^{\lambda}) : (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}); \\ & w \leftarrow E_2(\mathsf{crs}, \xi, x, \pi) : (x, w) \in R \\ & \textit{if} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1| = 1. \end{aligned}$$

In this work, we use a non-interactive commitment scheme to ensure the traceability of the proposed ABET scheme. The overview of proof commitment scheme based on DLIN assumption is shown below [15].
- Perfectly Binding Key Generation: It first generates public parameters $\mathsf{par} = (q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \hat{\mathsf{e}}, g, h) \leftarrow \mathsf{GroupGen}(1^{\lambda})$ (see Definition 1). Then, it chooses $a_1, a_2, z \leftarrow \mathbb{Z}_q^*$ and $r_0, s_0 \leftarrow \mathbb{Z}_q$, computes $(E, F) = (g^{a_1}, g^{a_2})$ and $(U, V, W) = (E^{r_0}, F^{s_0}, g^{r_0+s_0+z})$. Eventually, it outputs a public commitment key $ck = (\mathsf{par}, E, F, U, V, W)$, and a perfectly binding key $xk = (a_1, a_2, z)$.
- Commitment: It takes the public commitment key ck, and a user's identity $id \in \mathcal{I}$ as input, outputs a commitment $c = (c_1, c_2, c_3) = (U^{id} \cdot E^{w}, V^{id} \cdot F^{s}, W^{id} \cdot g^{w+s})$, where $w, s \leftarrow \mathbb{Z}_q$.
- Extraction: It takes the perfectly binding key ck, and a commitment c as input. Then, it computes $(g^{z})^{id} = c_3 \cdot c_1^{-1/a_1} \cdot c_2^{-1/a_2}$, exhaustively searches for id, and outputs the identity id.

The committed message is user's identity. Specifically, we insert user's identity id into the commitment, and use the perfectly binding key to extract the user's identity.

(4) *Digital Signature.* A digital signature scheme $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is homomorphic, if the following conditions are held.
- *Simple Key Generation.* It means $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(pp)$ and $pp \leftarrow \mathsf{Setup}(1^{\lambda})$, where pk is derived from sk via a key generation algorithm $\mathsf{pk} \leftarrow \mathsf{KeyGen}'(pp, \mathsf{sk})$.
- *Linearity of Keys.* It requires $\mathsf{KeyGen}'(pp, \mathsf{sk} + \Delta(\mathsf{sk})) = M_{\mathsf{pk}}(pp, \mathsf{KeyGen}'(pp, \mathsf{sk}), \Delta(\mathsf{sk}))$, where M_{pk} denotes a deterministic algorithm which takes pp, a public key pk and a "shifted" value $\Delta(\mathsf{sk})$ as input, outputs a "shifted" public key pk'. Δ denotes the difference or shift between two keys.
- *Linearity of Signatures.* Two distributions are identical: $\{\sigma' \leftarrow \mathsf{Sign}(pp, \mathsf{sk} + \Delta(\mathsf{sk}), m)\}$ and $\{\sigma' \leftarrow M_{\Sigma}(pp, \mathsf{pk}, m, \sigma, \Delta(\mathsf{sk}))\}$, where $\sigma \leftarrow \mathsf{Sign}(pp, \mathsf{sk}, m)$, and M_{Σ} denotes a deterministic algorithm which takes pp, a public key pk, a message-signature pair (m, σ) and a "shifted" value $\Delta(\mathsf{sk})$ as input, outputs a "shifted" signature σ'.
- *Linearity of Verifications.* It requires $\mathsf{Verify}(pp, M_{\mathsf{pk}}(pp, \mathsf{pk}, \Delta(\mathsf{sk})), m, M_{\Sigma}(pp, \mathsf{pk}, m, \sigma, \Delta(\mathsf{sk}))) = 1$, and $\mathsf{Verify}(pp, \mathsf{pk}, m, \sigma) = 1$.

The proposed signature scheme satisfies the homomorphic properties regarding keys and signatures. We use this property to find the connection between a transaction and its modified versions.

4 DEFINITION AND MODEL

In this section, we first present the system model for blockchain rewritings. Then, we show the definition and the security model of StrongPCH.

4.1 System Model

The system model includes the following entities: user, owner, modifier, and attribute authority (AA), in which the entities can intersect, such as a user can be an owner, and/or a modifier/authority. In terms of a decentralized setting, every user can play the role of an attribute authority and tag other users with attributes. In particular, the number of modifiers is assumed to be small because blockchain rewriting should not be performed by a majority of users in the system. We assume at most k users exist in the system.

In Figure 2, an owner appends a hashed object into the blockchain. Later, a modifier with enough rewriting privileges is allowed to modify the hashed object while the entire chain remains intact. If a dispute over the hashed object occurs, AA can decide whether the hashed object is modified by the modifier or even correct the modified object.

4.2 Definition

A strongly accountable policy-based chameleon hash (StrongPCH) consists of the following algorithms.

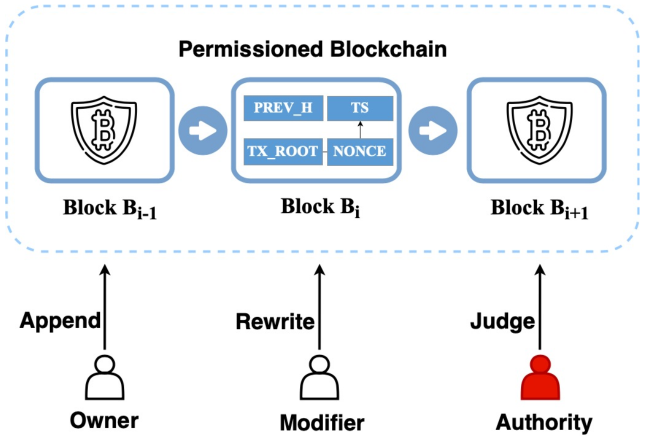

Figure 2: System Model under Consideration

- $\mathsf{Setup}(1^\lambda)$: It takes a security parameter λ as input, outputs a chameleon key pair $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{KeyGen}(\mathsf{sk}, id, \delta)$: It takes the chameleon secret key sk, an identity $id \in \mathcal{I}$, and a set of attributes $\delta \in \mathcal{U}$ as input, outputs a secret key sk_δ.
- $\mathsf{Hash}(\mathsf{pk}, m, \Lambda, \mathsf{sk}_i)$: It takes the chameleon public key pk, a message $m \in \mathcal{M}$, an access structure Λ, and a user's signing key sk_i as input, outputs a chameleon hash h, a randomness r and a signature σ_i.
- $\mathsf{Verify}(\mathsf{pk}, m, h, r, \sigma_i)$: It takes the chameleon public key pk, message m, hash value h, randomness r and signature σ as input, outputs a bit $b \in \{0, 1\}$.
- $\mathsf{Adapt}(\mathsf{sk}_\delta, m, m', h, r, \sigma, \mathsf{sk}_j)$: It takes the secret key sk_δ, messages m and m', hash value h, randomness r, signature σ, and a modifier's signing key sk_j as input, outputs a new randomness r' if $1 = \Lambda(\delta)$, and a new signature σ_j.
- $\mathsf{Judge}(\mathsf{sk}_\delta, T, T')$: It takes a secret key sk_δ, a transaction T and its modified version T' as input, outputs a transaction-identity pair (T', id) if the modified transaction T' links to an identity id, where $T = (h, m, r, \sigma_i)$ and $T' = (h, m', r', \sigma_j)$.

Correctness. We assume that the Adapt algorithm always verifies if the chameleon hash h and the signature σ given are valid, otherwise it outputs $\perp$. The StrongPCH is *correct* if for all security parameters λ, all keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)$, for all $\delta \in \Lambda$, for all $id \in \mathcal{I}$, for all $\mathsf{sk}_\delta \leftarrow \mathsf{KeyGen}(\mathsf{sk}, id, \delta)$, for all $m \in \mathcal{M}$, for all $(h, r, \sigma_i) \leftarrow \mathsf{Hash}(\mathsf{pk}, m, \Lambda, \mathsf{sk}_i)$, for all $m' \in \mathcal{M}$, for all $(r', \sigma_j) \leftarrow \mathsf{Adapt}(\mathsf{sk}_\delta, m, m', h, r, \sigma_i, \mathsf{sk}_j)$, we have $1 = \mathsf{Verify}(\mathsf{pk}, m, h, r, \sigma_i) = \mathsf{Verify}(\mathsf{pk}, m', h, r', \sigma_j)$.

4.3 Security Model

In the permisssioned blockchain, the transaction owner is an honest user. When he appends a transaction, it is his best interest that this happens correctly and that its transaction could be rewritten if required. We also assume the attribute authority is an honest user in the system.

Indistinguishability. Informally, adversary cannot decide whether for a chameleon hash its randomness (i.e., check string) 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 3. The security experiment allows $\mathcal{A}$ to access a left-or-right HashOrAdapt oracle, which ensures that the check string 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 $\mathcal{A}$.

```
Experiment $\mathsf{Exp}^{\mathrm{IND}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $w \leftarrow \{0, 1\}$
$w' \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt}(\mathsf{sk}, \cdots, w)}(\mathsf{pk})$
  where $\mathsf{HashOrAdapt}(\mathsf{sk}, \cdots, b)$ on input $m, m', \Lambda, id, \delta$ :
    $(h_0, r_0, \sigma_0) \leftarrow \mathsf{Hash}(\mathsf{pk}, m', \Lambda, \mathsf{sk}_0)$
    $(h_1, r_1, \sigma_1) \leftarrow \mathsf{Hash}(\mathsf{pk}, m, \Lambda, \mathsf{sk}_1)$
    $\mathsf{sk}_\delta \leftarrow \mathsf{KeyGen}(\mathsf{sk}, id, \delta)$
    $(r_1, \sigma_1) \leftarrow \mathsf{Adapt}(\mathsf{sk}_\delta, m, m', h_1, r_1, \sigma_1, \mathsf{sk}_1)$
    return $(h_w, r_w, \sigma_w)$
return 1, if $w' = w$; else, return 0.
```

Figure 3: Indistinguishability.

```
Experiment $\mathsf{Exp}^{\mathrm{CR}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $Q_1, Q_2, Q_3 \leftarrow \emptyset$
$(m^*, r^*, m'^*, r'^*, h^*, \sigma^*, \sigma'^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk})$
  where $\mathcal{O} \leftarrow \{\mathsf{KeyGen}', \mathsf{KeyGen}'', \mathsf{Hash}', \mathsf{Adapt}'\}$
  and $\mathsf{KeyGen}'(\mathsf{sk}, \cdots)$ on input $\delta, id$ :
    $\mathsf{sk}_\delta \leftarrow \mathsf{KeyGen}(\mathsf{sk}, id, \delta)$
    $Q_1 \leftarrow Q_1 \cup \{\delta\}$
    return $\mathsf{sk}_\delta$
  and $\mathsf{KeyGen}''(\mathsf{sk}, \cdots)$ on input $\delta, id$ :
    $\mathsf{sk}_\delta \leftarrow \mathsf{KeyGen}(\mathsf{sk}, id, \delta)$
    $Q_2 \cup \{(i, \mathsf{sk}_\delta)\}$
    $i \leftarrow i + 1$
  and $\mathsf{Hash}'(\mathsf{pk}, \cdots)$ on input $m, \Lambda$ :
    $(h, r, \sigma_i) \leftarrow \mathsf{Hash}(\mathsf{pk}, m, \Lambda, \mathsf{sk}_i)$
    $Q_3 \leftarrow Q_3 \cup \{(h, m, \Lambda)\}$
    return $(h, r, \sigma_i)$
  and $\mathsf{Adapt}'(\mathsf{pk}, \cdots)$ on input $m, m', h, r, \sigma_i, j$ :
    return $\perp$, if $(j, \mathsf{sk}_\delta) \notin Q_2$ for certain $\mathsf{sk}_\delta$
    $(r', \sigma_j) \leftarrow \mathsf{Adapt}(\mathsf{sk}_\delta, m, m', h, r, \sigma_i, \mathsf{sk}_j)$
    if $(h, m, \Lambda) \in Q_3$ for certain $\Lambda$,
    let $Q_3 \leftarrow Q_3 \cup \{(h, m', \Lambda)\}$
    return $(r', \sigma_j)$
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 certain $\Lambda \wedge m^* \neq m'^*$
$\wedge \Lambda \cap Q_1 = \emptyset \wedge (h^*, m^*, \cdot) \notin Q_3$
else, return 0.
```

Figure 4: Collision-Resistance.

We require $1 \leftarrow \mathsf{Verify}(\mathsf{pk}, m', h_0, r_0, \sigma_0) = \mathsf{Verify}(\mathsf{pk}, m, h_1, r_1, \sigma_1)$, and we define the advantage of the adversary as

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

Definition 7. *A StrongPCH scheme is indistinguishable if for any PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathrm{IND}}_{\mathcal{A}}(\lambda)$ is negligible in λ.*

Collision-Resistance. Informally, an adversary can find collisions for a chameleon hash if she possesses a secret key satisfying the policy embedded in that chameleon hash (this condition is modelled by KeyGen' oracle). We define a formal experiment in Figure 4. 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

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

```
Experiment $\mathsf{Exp}^{\mathrm{ACT}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $Q \leftarrow \emptyset$
$\mathsf{sk}_\delta \leftarrow \mathsf{KeyGen}(\mathsf{sk}, id, \delta)$
$T^* \leftarrow \mathcal{A}^{\mathsf{Judge}'(\mathsf{sk},\cdots)}(\mathsf{pk})$
    where $\mathsf{Judge}'(\mathsf{sk},\cdots)$ on input $\mathsf{sk}_\delta, T, T'$ :
        $(T', id) \leftarrow \mathsf{Judge}(\mathsf{sk}_\delta, T, T')$
        $Q \leftarrow Q \cup \{(T, T')\}$
        return $(T', id)$
return 1, if $(T^*, id^*) \wedge T^* \notin Q$; else, return 0.
```

Figure 5: Accountability.

Definition 8. *A StrongPCH scheme is collision resistant if for any PPT $\mathcal{A}$, $\mathsf{Adv}^{CR}_{\mathcal{A}}(\lambda)$ 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 identity, but the message-signature pair has never been generated by the identity. We define a formal experiment in Figure 5. $\mathcal{A}$ is given a Judge' oracle to show whether a chameleon hash links to an identity. Let set Q record the chameleon hash generated by the Judge' oracle.
We denote $T = (h, m, r, \sigma_i)$ and $T' = (h, m', r', \sigma_j)$ 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

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

Definition 9. *A StrongPCH scheme is accountable if for any PPT $\mathcal{A}$, $\mathsf{Adv}^{ACT}_{\mathcal{A}}(\lambda)$ is negligible in λ.*

Remark 1. *We compare our proposed security models with the closely related security models [11, 12, 26]. First, our proposed indistinguishability and collision-resistance models are derived from the strong indistinguishability and standard collision-resistance models defined in [12]. The derived indistinguishability and collision-resistance models are nearly the same as defined in [12]. Here we do not consider a stronger collision-resistance model called full collision-resistance, which is defined in [11]. We argue that the standard collision-resistance provides sufficient security guarantee for PCH and StrongPCH. The detailed comparison across various collision-resistance models is referred to [11]. Second, our proposed accountability model is derived from the sanitizer accountability model defined in [26]. Note that our proposed accountability model grants a judge oracle to attackers, determing whether or not a modified transaction links to a modifier.*

5 THE PROPOSED CONSTRUCTION

The proposed construction consists of the following building blocks.

- A chameleon hash scheme CH=(KeyGen, Hash, Verify, Adapt).
- An attribute-based encryption with traceability scheme ABET=(Setup, KeyGen, Enc, Dec, Trace).
- A digital signature scheme Σ=(Setup, KeyGen, Sign, Verify).

We assume that every user has a secret/public key pair $(\mathsf{sk}, \mathsf{pk})$ and that public keys are known to all users in the system. Each modifier has a set of attributes δ and an identity id. We assume a user with pk_i creates a transaction T, while a modifier with pk_j rewrites the transaction T. We re-define an algorithm in Σ as $\mathsf{pk} \leftarrow \mathsf{KeyGen}'(pp, 0, \mathsf{sk})$, which takes one public value and two secret values as input. We present the proposed construction below.

- $\mathsf{Setup}(1^\lambda)$: An attribute authority (AA) takes a security parameter λ as input, outputs public parameters $\mathsf{PP} = (\mathsf{mpk}_{\mathsf{ABET}}, pp_\Sigma)$, where $(\mathsf{msk}_{\mathsf{ABET}}, \mathsf{mpk}_{\mathsf{ABET}}) \leftarrow \mathsf{Setup}_{\mathsf{ABET}}(1^\lambda)$, $pp_\Sigma \leftarrow \mathsf{Setup}_\Sigma(1^\lambda)$.
- $\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{ABET}}, id_j, \delta)$: AA takes the master secret key $\mathsf{msk}_{\mathsf{ABET}}$, a user pk_j's identity id_j and a set of attributes δ as input, outputs a secret key $\mathsf{sk}_{\delta_j} \leftarrow \mathsf{KeyGen}_{\mathsf{ABET}}(\mathsf{msk}_{\mathsf{ABET}}, id_j, \delta)$. Note that the user pk_j becomes a modifier after this procedure.
- $\mathsf{Hash}(\mathsf{mpk}_{\mathsf{ABET}}, m, \Lambda, \mathsf{sk}_i)$: A user pk_i records a message m to the blockchain, and performs the following operations
  1. generate a chameleon hash $(b, \mathsf{r}) \leftarrow \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m)$, where $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \leftarrow \mathsf{KeyGen}_{\mathsf{CH}}(1^\lambda)$.
  2. generate a ciphertext $C \leftarrow \mathsf{Enc}_{\mathsf{ABET}}(\mathsf{mpk}_{\mathsf{ABET}}, \mathsf{sk}_{\mathsf{CH}}, \Lambda)$.
  3. generate a signature $\sigma_i \leftarrow \mathsf{Sign}_\Sigma(\mathsf{sk}_i, c_i)$, where the signed message is $c_i \leftarrow \mathsf{KeyGen}'_\Sigma(pp_\Sigma, \mathsf{sk}_{\mathsf{CH}}, \mathsf{sk}_i)$, and the verification key is $apk_i \leftarrow \mathsf{KeyGen}'_\Sigma(pp_\Sigma, 0, \mathsf{sk}_i)$
  4. output $(\mathsf{pk}_{\mathsf{CH}}, m, \mathsf{r}, b, C, apk_i, c_i, \sigma_i)$.
- $\mathsf{Verify}(\mathsf{pk}_{\mathsf{CH}}, m, \mathsf{r}, b, C, apk_i, c_i, \sigma_i)$: It outputs 1 if the following checks hold: $1 = \mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, b, \mathsf{r})$ and $1 = \mathsf{Verify}_\Sigma(apk_i, c_i, \sigma_i)$, and 0 otherwise.
- $\mathsf{Adapt}(\mathsf{sk}_{\delta_j}, m, m', \mathsf{pk}_{\mathsf{CH}}, b, \mathsf{r}, C, apk_i, c_i, \sigma_i, \mathsf{sk}_j)$: The modifier pk_j with secret key sk_{δ_j} and a new message m' performs the following operations
  1. verify the policy-based chameleon hash $1 \overset{?}{=} \mathsf{Verify}(\mathsf{pk}_{\mathsf{CH}}, m, \mathsf{r}, b, C, apk_i, c_i, \sigma_i)$.
  2. decrypt the trapdoor $\mathsf{sk}_{\mathsf{CH}} \leftarrow \mathsf{Dec}_{\mathsf{ABET}}(\mathsf{sk}_{\delta_j}, C)$, and return 0 if $\mathsf{sk}_{\mathsf{CH}} = \perp$.
  3. generate an adapted randomness $\mathsf{r}' \leftarrow \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', b, \mathsf{r})$.
  4. generate a new signature $\sigma_j \leftarrow \mathsf{Sign}_\Sigma(\mathsf{sk}_j, c_j)$, where $c_j \leftarrow \mathsf{KeyGen}'_\Sigma(pp_\Sigma, \mathsf{sk}_{\mathsf{CH}}, \mathsf{sk}_j)$.
  5. output $(m', \mathsf{pk}_{\mathsf{CH}}, \mathsf{r}', b, C, apk_j, c_j, \sigma_j)$.
- $\mathsf{Judge}(\mathsf{sk}_{\delta_j}, T, T')$: Given two transactions (T, T'), and a revealed secret key sk_{δ_j}, AA performs the following operations
  1. verify the policy-based chameleon hashes $1 = \mathsf{Verify}(\mathsf{pk}_{\mathsf{CH}}, T) = \mathsf{Verify}(\mathsf{pk}_{\mathsf{CH}}, T')$, where $T = (m, b, \mathsf{r}, apk_i, c_i, \sigma_i)$ and $T' = (m', b, \mathsf{r}', apk_j, c_j, \sigma_j)$.
  2. extract the modifier's identity $id_j \leftarrow \mathsf{Trace}_{\mathsf{ABET}}(\mathsf{msk}_{\mathsf{ABET}}, \mathsf{sk}_{\delta_j})$.
  3. output the linked transaction-modifier pair (T', pk_j).

Correctness. The judge process allows AA to identify a relationship between a modified transaction and a responsible modifier. Now, we explain this process in detail. First, given two transactions (T, T'), the public verifies a connection between a transaction T and its modified version T'. Since the message-signature pair (c_i, σ_i) in T and message-signature pair (c_j, σ_j) in T' are associated with the same secret key (i.e., trapdoor) $\mathsf{sk}_{\mathsf{CH}}$, the connection can be established. In particular, the trapdoor $\mathsf{sk}_{\mathsf{CH}}$ is used in many modified versions of a transaction, as different modifiers may rewrite the same transaction.

Second, AA obtains an identity id_j from a revealed secret key sk_{δ_j} (e.g., a modifier may reveal her secret key to the public) due to ABET's traceability. Third, AA links the modified transaction T' to an authorized modifier's public key pk_j using the master secret key $\mathsf{msk}_{\mathsf{ABET}}$, and outputs a linked transaction-modifier pair: (T', pk_j), meaning that the transaction T' is indeed modified by the modifier

pk_j. Another scenario is that if an unauthorized modifier pk^* holds the abused secret key sk_{δ_j} and rewrites transaction T, AA outputs: $(T', \mathsf{pk}^*, \mathsf{pk}_j)$. This means that the transaction T' is modified by the unauthorized modifier pk^* whose rewriting privilege is given by the authorized modifier pk_j. In other words, the modifier pk_j reveals her secret key sk_{δ_j} to the unauthorized modifier pk^*, without rewriting the blockchain. Overall, our strong accountability can hold the modifiers' public keys accountable for the modified transactions, regardless of whether the modifiers are authorized or not.

5.1 Security Analysis

Theorem 10. *The StrongPCH scheme is indistinguishable if the CH scheme is indistinguishable, and the Σ scheme is anonymous.*

Theorem 11. *The StrongPCH scheme is collision-resistant if the ABET scheme is semantically secure, and the CH scheme is collision-resistant.*

Theorem 12. *The StrongPCH scheme is strongly accountable if the digital signature scheme Σ is EUF-CMA secure, and the ABET scheme is traceable.*

The detailed security analysis is deferred to Appendix B.

5.2 Application

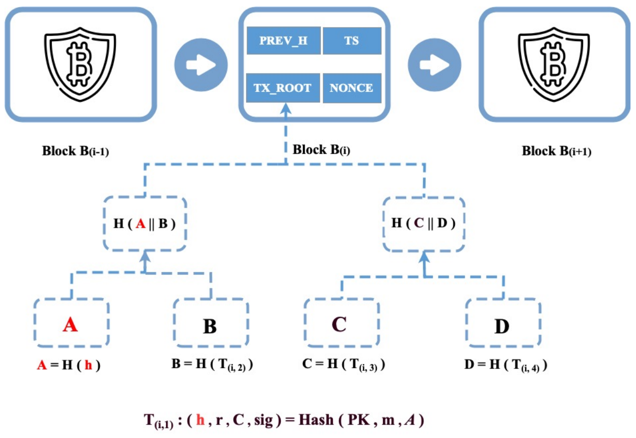

Figure 6: Blockchain Applications. The mutable transactions A is derived using the StrongPCH function. The normal transactions (B, C, D) are derived using the conventional collision-resistant hash function H (e.g., SHA-256).

We show a simple application of the proposed StrongPCH function for blockchain rewriting. Specifically, the blockchain remains intact but certain transactions in a block can be rewritten. In the blockchain, each block stores a compact representation of a set of transactions. That is, the root hash of a Merkle tree (i.e., *TX_ROOT*) that accumulates all transactions associates with a block.

If a user (or owner) appends a transaction $T_{(i,1)}$ to the blockchain, the recorded object/message m must be hashed using $\mathsf{Hash}(\mathsf{pk}, m, \Lambda, \mathsf{sk}_i)$. In Figure 6, a block B_i accumulates four transactions $T_{(i,1)}, T_{(i,2)}, T_{(i,3)}, T_{(i,4)}$, where transactions $T_{(i,1)}$ is a mutable one. It associated with an access policy Λ, which is chosen by the transaction owner. The remaining transactions are hashed using conventional collision-resistant hash function H. If the transaction $T_{(i,1)}$ needs to be modified, a modifier with a secret key satisfying Λ can find a collision r' for hash value A, and replace the randomness (i.e., check string) r by r'. Note that the randomness r, the ciphertext C, and signature σ are not included in the hash computation of the aggregation, and are provided as non-hashed part of the transaction/block. Besides, the hash function used to chain blocks is also H and the values *PREV_H* are never modified.

Table 1: The Comparison across various ABET schemes. † means that on which ABE scheme the ABET is based. k denotes the total number of users in the system. Secret/Public Tracing means the key generation center (i.e., KGC)/the public identifies misbehaving users. S-Trace means that KGC identifies the users whose decryption keys are revealed and/or abused.

	Scheme†	Group-order	Cipher-size	Tracing	S-Trace
[20]	ABE [19]	Composite	$O(\sqrt{k})$	Public	×
[24]	ABE [19]	Composite	$O(1)$	Secret	✓
[18]	ABE [25]	Prime	$O(1)$	Public	×
[29]	FAME [4]	Prime	$O(1)$	Public	×
Ours	FAME [4]	Prime	$O(1)$	Secret	✓

6 INSTANTIATION AND EVALUATION

In this section, we start with discussing the choice of primitives to construct an StrongPCH protocol in the blockchain setting. We then present the concrete construction, provide the implementation and evaluation analysis. First, we use the chameleon hash construction in [9] to initiate CH, which is based on DL. Meanwhile, the DL-based chameleon hash equipped with a bilinear map in [9] is also suitable. Second, we choose the CP-ABE scheme [4] and a commitment scheme [15] to instantiate ABET. We stress that the existing traceable CP-ABE schemes [18, 20, 24, 29] can be used to instantiate ABET. However, either they involve composite-order group operations (thus less practical) or they are vulnerable to rewriting privilege abuse attacks (e.g., [18, 29]). Table 1 summarizes the comparison between our proposed ABET and other ABET instantiations.

6.1 The proposed ABET Scheme

For constructing a practical ABET scheme, we rely on a recent work: Fast Attribute-based Message Encryption (FAME) [4]. FAME supports unbounded ABE universes, no restriction on the monotone policies, constant-time decryption, adaptive security under the standard DLIN assumption, and is based on the asymmetric prime-order Type-III pairing. Table 1 shows that our proposed ABET scheme is the first strongly traceable construction based on FAME. It inherits all the (semantic) security and performance advantages from FAME. On the other hand, the proposed ABET scheme does not support black-box traceability as described in [18, 20, 29].

The proposed ABET scheme is embedded into the instantiation. Below, we present Theorem 13 to show the proposed ABET achieves semantic security and traceability. The security analysis of the proposed ABET scheme is referred to Appendix C.

Theorem 13. *The proposed ABET scheme achieves semantic security and traceability if the eDLIN assumption is held in the asymmetric pairing groups and the underlying proof commitment scheme is perfectly extractable.*

6.2 The proposed Signature Scheme

We propose an anonymous digital signature scheme Σ, such that the signature does not reveal the signer's identity to the public. The construction is based on a pairing-based signature scheme [14]. Note that the forward-security defined in [14] is beyond the scope of this work. Now, we show the overview of Σ, and the detailed scheme is shown in the instantiation. A signature σ on a message m under public key $\mathsf{pk} = h^{\mathsf{sk}}$ is of the form:

$$\sigma = (\sigma', \sigma'') = (g^{\beta \cdot \mathsf{sk}} \cdot \mathsf{H}(m)^{esk}, h^{\alpha \cdot esk}) \in \mathbb{G} \times \mathbb{H}$$

where H denotes a collision-resistant hash function, (α, β) are secret values chosen by the authority, and esk denotes an ephemeral secret key chosen by a signer. Note that the signer generates an element $\hat{e}(g,h)^{\alpha \cdot \beta \cdot \mathsf{sk}}$ (underline part) as his verification key:

$$\hat{e}(\sigma', h^{\alpha}) \overset{?}{=} \underline{\hat{e}(g,h)^{\alpha \cdot \beta \cdot \mathsf{sk}}} \cdot \hat{e}(\mathsf{H}(m), \sigma'')$$

The proposed Σ scheme satisfies the homomorphic properties as described in Section 3.3. Below, we present Theorem 14 to show the proposed Σ scheme is of EUF-CMA security and anonymity. Anonymity means that any third party cannot link a valid signature σ to a specific signer. However, it can be linked by the attribute authority. The security analysis is referred to Appendix D.

Theorem 14. *The proposed Σ scheme achieves EUF-CMA security and anonymity if the proposed CBDH and DBDH assumptions are held in the asymmetric pairing groups, respectively.*

6.3 Instantiation

In the ABET scheme, two types of inputs are given to a hash function H: inputs of the form (y, ℓ, t) or those of the form (j, ℓ, t), where y is an arbitrary string (e.g., attribute), j is a positive integer, $\ell \in \{1,2,3\}$ and $t \in \{1,2\}$. We represent these two inputs as $y\ell t$ and $0j\ell t$ (appending "0" in the second format is used to differentiate with the first one), respectively. Let $\mathsf{G} : \mathbb{G}_T \rightarrow \{0,1\}^*$ be a pseudo-random generator. Let $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{G}$ be a hash function. Let $\hat{e} : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$ be a bilinear pairing. Each user holds a key pair $(\mathsf{sk}, \mathsf{pk})$, where $\mathsf{pk} = h^{\mathsf{sk}}$.

- $\mathsf{Setup}(1^{\lambda})$: Attribute authority (AA) takes a security parameter λ as input, outputs a master public key $\mathsf{mpk} = (g, h, H_1, H_2, T_1, T_2, E, F, U, V, W, Z_1, Z_2, g^{\beta}, h^{1/\alpha}, h^{\beta/\alpha})$ and a master secret key $\mathsf{msk} = (a_1, a_2, b_1, b_2, g^{d_1}, g^{d_2}, g^{d_3}, r_0, s_0, z, \alpha, \beta)$, where 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, $(a_1, a_2, b_1, b_2, \alpha, \beta) \in \mathbb{Z}_q^*$, $(r_0, s_0, d_1, d_2, d_3, z) \in \mathbb{Z}_q$, $H_1 = h^{a_1}, H_2 = h^{a_2}, T_1 = \hat{e}(g,h)^{d_1 \cdot a_1 + z \cdot d_3}, T_2 = \hat{e}(g,h)^{d_2 \cdot a_2 + z \cdot d_3}$, $(E, F) = (g^{1/a_1}, g^{1/a_2})$, $(U, V, W) = (E^{r_0}, F^{s_0}, g^{r_0 + s_0})$ and $(Z_1, Z_2) = (g^z, h^z)$.
- $\mathsf{KeyGen}(\mathsf{msk}, id_j, \delta)$: AA takes a user pk_j's identity id_j, and a set of attributes δ as input, outputs a decryption key sk_{δ_j} for user pk_j. Specifically, $\mathsf{sk}_j = (\mathsf{sk}_0, \{\mathsf{sk}_y\}_{y \in \delta}, \mathsf{sk}', \mathsf{sk}_1, \mathsf{sk}_2, \mathsf{sk}_3, \mathsf{sk}_4)$, where $(r_1, r_2) \in \mathbb{Z}_q^*$, $(\sigma_y, \sigma', w, s) \in \mathbb{Z}_q$, $\mathsf{sk}_0 = (h^{b_1 \cdot r_1}, h^{b_2 \cdot r_2}, Z_2^{r_1 + r_2})$, $\forall y \in \delta$ and $t = \{1,2\}$: $\mathsf{sk}_{y,t} = \mathsf{H}(y1t)^{b_1 \cdot r_1 / a_t} \cdot \mathsf{H}(y2t)^{b_2 \cdot r_2 / a_t} \cdot \mathsf{H}(y3t)^{z \cdot (r_1 + r_2)/a_t} \cdot g^{z \cdot \sigma_y / a_t}$, $\mathsf{sk}'_t = g^{d_t} \cdot \mathsf{H}(011t)^{b_1 \cdot r_1 / a_t} \cdot \mathsf{H}(012t)^{b_2 \cdot r_2 / a_t} \cdot \mathsf{H}(013t)^{z \cdot (r_1 + r_2)/a_t} \cdot g^{z \cdot \sigma' / a_t}$, $\mathsf{sk}'_3 = g^{d_3} \cdot g^{-\sigma'}$, $\forall t = \{1,2\} : \mathsf{sk}' = (\mathsf{sk}'_1, \mathsf{sk}'_2, \mathsf{sk}'_3)$, $\mathsf{sk}_1 = (U^{id_j} \cdot E^{w})^z, \mathsf{sk}_2 = (V^{id_j} \cdot F^{s})^z, \mathsf{sk}_3 = W^{id_j} \cdot g^{w+s}, \mathsf{sk}_4 = \mathsf{sk}_3 \cdot g^{id_j}$.
- $\mathsf{Hash}(\mathsf{mpk}, m, \Lambda, \mathsf{sk}_i)$: To hash a message m with a policy Λ, an owner pk_i performs the following operations
  1. choose a randomness (i.e., check string) $\mathsf{r} \in \mathbb{Z}_q^*$, and a trapdoor R, compute a chameleon hash $b = g^m \cdot \mathsf{h}^{\mathsf{r}}$, where $\mathsf{h} = g^{\mathsf{R}}$.
  2. generate a ciphertext on message R under policy $(\mathbf{M}, \pi)$: $C = (ct_0, ct_1, \cdots, ct_{n_1}, ct')$, where $(s_1, s_2) \in \mathbb{Z}_q^*$, $ct_0 = (H_1^{s_1}, H_2^{s_2}, H_1^{s_2}, H_2^{s_1}, Z_2^{s_1 + s_2})$, $\forall i = \{1, \cdots, n_1\}$ and $\ell = \{1,2,3\}$: $ct_{(i,\ell)} = \mathsf{H}(\pi(i)\ell 1)^{s_1} \cdot \mathsf{H}(\pi(i)\ell 2)^{s_2} \cdot \prod_{j=1}^{n_2} [\mathsf{H}(0j\ell 1)^{s_1} \cdot \mathsf{H}(0j\ell 2)^{s_2}]^{\mathbf{M}_{(i,j)}}$, $ct' = \mathsf{R} \oplus \mathsf{G}(T_1^{s_1} \cdot T_2^{s_2})$.
  3. generate a signature $\sigma_i = (\sigma'_i, \sigma''_i) = (g^{\beta \cdot \mathsf{sk}_i} \cdot \mathsf{H}(c_i)^{esk_i}, h^{esk_i/\alpha})$, where $esk_i \in \mathbb{Z}_q$, $c_i = h^{\mathsf{sk}_i + \mathsf{R}}$, and $apk_i = \hat{e}(g, h^{\beta/\alpha})^{\mathsf{sk}_i}$.
  4. output a policy-based chameleon hash $(m, \mathsf{h}, \mathsf{r}, b, C, apk_i, c_i, \sigma_i)$.
- $\mathsf{Verify}(\mathsf{mpk}, m, \mathsf{h}, \mathsf{r}, b, C, apk_i, c_i, \sigma_i)$: Any user can verify whether a chameleon hash (b, r) is valid, it outputs 1 if $b = g^m \cdot \mathsf{h}^{\mathsf{r}}$ and $\hat{e}(\sigma'_i, h^{\alpha}) \overset{?}{=} apk_i \cdot \hat{e}(\mathsf{H}(c_i), \sigma''_i)$.
- $\mathsf{Adapt}(\mathsf{mpk}, m, \mathsf{h}, \mathsf{r}, b, C, apk_i, c_i, \sigma_i, \mathsf{sk}_j)$: A modifier pk_j with secret key sk_{δ_j} and a new message m', performs the following operations
  1. verify the chameleon hash, and the message-signature pair as described above.
  2. run the following steps to obtain the encrypted trapdoor R. If the attribute set δ in sk_{δ_j} satisfies the MSP $(\mathbf{M}, \pi)$, then there exist constants $\{\gamma_i\}_{i \in I}$ that satisfy the equation in Section 3.2. It computes $\mathsf{R} = ct' \oplus \mathsf{G}(B/A)$, where A and B are described below (note that the detailed decryption is referred to Appendix C).

$$\begin{aligned} A &= \hat{e}(\prod_{i \in I} ct_{(i,1)}^{\gamma_i}, \mathsf{sk}_{(0,1)}) \cdot \hat{e}(\prod_{i \in I} ct_{(i,2)}^{\gamma_i}, \mathsf{sk}_{(0,2)}) \\ &\cdot \hat{e}(\prod_{i \in I} ct_{(i,3)}^{\gamma_i}, \mathsf{sk}_{(0,3)}) \cdot \hat{e}(ct_{(0,1)} \cdot ct_{(0,3)}, \mathsf{sk}_1) \\ &\cdot \hat{e}(ct_{(0,2)} \cdot ct_{(0,4)}, \mathsf{sk}_2) \\ B &= \hat{e}(\mathsf{sk}'_1 \cdot \prod_{i \in I} \mathsf{sk}_{(\pi(i),1)}^{\gamma_i}, ct_{(0,1)}) \\ &\cdot \hat{e}(\mathsf{sk}'_2 \cdot \prod_{i \in I} \mathsf{sk}_{(\pi(i),2)}^{\gamma_i}, ct_{(0,2)}) \\ &\cdot \hat{e}(\mathsf{sk}'_3 \cdot \prod_{i \in I} \mathsf{sk}_{(\pi(i),3)}^{\gamma_i}, ct_{(0,5)}) \cdot \hat{e}(\mathsf{sk}_3, ct_{(0,5)}) \end{aligned}$$

     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 is applied to ct_0.
  3. computes an adapted randomness as $\mathsf{r}' = \mathsf{r} + (m - m')/\mathsf{R}$.
  4. generate a signature $\sigma_j = (\sigma', \sigma'') = (g^{\beta \cdot \mathsf{sk}_j} \cdot \mathsf{H}(c_j)^{esk_j}, h^{esk_j/\alpha})$, where $esk_j \in \mathbb{Z}_q$, $c_j = h^{\mathsf{sk}_j + \mathsf{R}}$, and $apk_j = \hat{e}(g, h^{\beta/\alpha})^{\mathsf{sk}_j}$.
  5. output a policy-based chameleon hash $(m', \mathsf{h}, \mathsf{r}', b, C, apk_j, c_j, \sigma_j)$.
- $\mathsf{Judge}(\mathsf{sk}_{\delta_j}, T, T')$: Given a revealed decryption key sk_{δ_j} and two transactions (T, T'), AA performs the following operations
  1. verify the transactions' chameleon hashes and the message-signature pairs.
  2. extract a modifier's identity id_j from the revealed decryption key $\mathsf{sk}_{\delta_j} : \{\mathsf{sk}_0, \cdots, \mathsf{sk}_3, \mathsf{sk}_4\}$.
  3. links apk_j to the modifier pk_j using master secret keys (α, β).

Correctness. We explain the judge process with more details. First, the public verifies the connection between a transaction and its modified version. For example, given two transactions: $T = (m, \mathsf{h}, \mathsf{r}, b, C, apk_i, c_i, \sigma_i,)$ and $T' = (m', \mathsf{h}, \mathsf{r}', b, C, apk_j, c_j, \sigma_j)$, the public performs the following operations

(1) verify chameleon hash $b = g^m \cdot \mathsf{h}^{\mathsf{r}} = g^{m'} \cdot \mathsf{h}^{\mathsf{r}'}$.
(2) verify message-signature pair (c_i, σ_i) under apk_i, and message-signature pair (c_j, σ_j) under apk_j.
(3) verify $apk_j = apk_i \cdot \Delta(\mathsf{sk})$, where $\Delta(\mathsf{sk}) = c_j / c_i = h^{\mathsf{sk}' - \mathsf{sk}}$ (the meaning of $\Delta(\mathsf{sk})$ is referred to Section 3.3). Note that (c_i, c_j) are derived from the same value R.

Second, AA extracts a modifier's identity id_j via the following equation

$$\begin{aligned} Z_1^{id_j} &= \mathsf{sk}_4^z \cdot \mathsf{sk}_1^{-a_1} \cdot \mathsf{sk}_2^{-a_2} \\ &= (W^{id_j} \cdot g^{w+s} \cdot g^{id_j})^z \cdot (U^{id_j} \cdot E^w)^{-a_1 \cdot z} \\ &\quad \cdot (V^{id_j} \cdot F^s)^{-a_2 \cdot z} \\ &= g^{z[id_j \cdot (r_0 + s_0 + 1) + (w+s)]} \cdot g^{-a_1 \cdot z \cdot (id_j \cdot r_0 + w) \cdot / a_1} \\ &\quad \cdot g^{-a_2 \cdot z \cdot (id_j \cdot s_0 + s) / a_2} = g^{z \cdot id_j}. \end{aligned}$$

AA can extract the discrete log id_j from $Z_1^{id_j}$ using Pollard's lambda method [8]. Third, AA links apk_j to the modifier's public key pk_j using its master secret key since $apk_j = \hat{e}(g, \mathsf{pk}_j)^{\beta/\alpha}$.

6.4 Implementation and Evaluation

To show the practicality of our proposed StrongPCH scheme, we implemented it in the Charm framework [5] and evaluate its performance on a PC with Intel Core i9 (3.6Ghz × 2) and 7.7GiB RAM. We benchmark each algorithm of the instantiation with various parameters. We specifically benchmark the algorithms of our scheme with various parameters. We implement the pairing groups based on MNT224 curve [22], and instantiate the hash functions and the pseudo-random generators with the corresponding standard interfaces provided by the Charm framework. The implementation code is available on GitHub [2].

First, we provide a performance comparison between StrongPCH and PCH' [29] (i.e., we denote the existing accountable PCH as PCH'). The running time of KeyGen, Hash, and Adapt algorithms are shown in Figure 7. In StrongPCH, the performance of KeyGen and Hash algorithms are linear to the number of attributes and the size of policies (with all AND operations), respectively. The runtime of KeyGen is 0.62 seconds, even if the number of attributes is 100. The running time of Hash algorithms is about 0.93 seconds for handling a policy of size 100. Figure 7 shows that the computational cost of KeyGen and Hash algorithms in StrongPCH are similar to PCH'. The major difference is that the run-time of Adapt algorithm in StrongPCH is constant (and fast). This is because the adapt process in StrongPCH does not require an encryption procedure (but it is required in PCH'). Besides, the run-time of Verify algorithm in StrongPCH is slightly lower than PCH' (i.e., the saved time is ≈ 0.006 seconds). Similarly, the Judge algorithm is also more efficient compared to PCH' (i.e., the saved time is ≈ 0.027 seconds).

Second, we evaluate the computational cost of the key generation, and hash/adapt outputs based on the number of group elements from $\mathbb{G}$, $\mathbb{H}$ (note that operations on group $\mathbb{H}$ are significantly more expensive than on $\mathbb{H}$) and $\mathbb{G}_T$. Table 2 shows that the number of elements used by StrongPCH in $\mathbb{H}$ is independent of T and n_1, as the underlying ABE scheme [4] supports a constant number of group elements in $\mathbb{H}$. Besides, Hash requires one paring operation, Verify requires three paring operations, and Adapt requires thirteen pairing operations. We note that the proposed ABET scheme requires three additional pairing operations during the decryption process in comparison to the ABE scheme [4].

Table 2: The number of multiplications and exponentiations used by StrongPCH in terms of $\mathbb{G}$, $\mathbb{H}$, and $\mathbb{G}_T$. T denotes the number of attributes in KeyGen, (n_1, n_2) denotes the dimension of the MSP in encryption. I is the number of attributes used in decryption.

Operations:	Multiplication	Exponentiation
KeyGen$_{\mathbb{G}/\mathbb{H}/\mathbb{G}_T}$:	$8T$+ 13 / - / -	$9T$+19 / 3 / -
Hash$_{\mathbb{G}/\mathbb{H}/\mathbb{G}_T}$:	$12n_1n_2$+$6n_1$+2 / - / 1	$6n_1$+5 / 7 / 3
Adapt$_{\mathbb{G}/\mathbb{H}/\mathbb{G}_T}$:	$6I$+4 / - / 1	$3I$+3 / 1 / 1

Third, we show the impact of integrating the proposed StrongPCH into a mutable blockchain. The first point is that the rewriting of mutable transactions incurs almost no overhead to Merkle tree generation and chain validation. This is because the randomness (i.e., check string), ciphertext, and signature are stored in the *non-hashed* part of a mutable transaction. We recall that the standard collision-resistant hash function (e.g., SHA256) is used to accumulate transactions and chain blocks (which is shown in Figure 6). The second point focuses on the overall storage cost of mutable blockchains. We note that the number of mutable transactions ranges from 2% to 10% inside a block according to [13]. The storage cost of each mutable transaction includes: 1) $2\mathcal{L}_{\mathbb{G}} + \mathbb{Z}_q$ for chameleon hash; 2) $n_1\mathcal{L}_{\mathbb{G}} + 5\mathcal{L}_{\mathbb{H}} + \mathbb{Z}_q$ for ABET; 3) $\mathcal{L}_{\mathbb{G}} + 2\mathcal{L}_{\mathbb{H}} + \mathcal{L}_{\mathbb{G}_T}$ for digital signature. Because we use MNT224 curve for pairing ($|\mathbb{G}| = 225, |\mathbb{Z}_q| = 224$), the storage overhead is about $225n_1$+7198 bits when changing an immutable transaction to a mutable one. Note that the size of an element in $\mathbb{H}/\mathbb{G}_T$ is 3/6 times of that of $\mathbb{G}$, a bitcoin transaction (usually) requires 256 bytes, and the maximum transaction size is 100 kilobytes [3].

7 RELATED WORK

Blockchain Rewriting. Blockchain rewriting was introduced by Ateniese et al. [6]. The blockchain is mutable if the traditional hash functions are replaced by chameleon hash (CH) [17] functions. 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 user who possesses the trapdoor can find collisions without changing the hash output.

Camenisch et al. [9] introduced a new primitive called chameleon hash with ephemeral trapdoor (CHET). A CHET requires that the modifier must have two trapdoors for rewriting: one is the trapdoor sk associated with the public key pk; the other is an ephemeral trapdoor *etd* chosen by the party who computed the chameleon hash value. The CHET provides enhanced usability to blockchain rewriting. The party who computes the hash value can decide

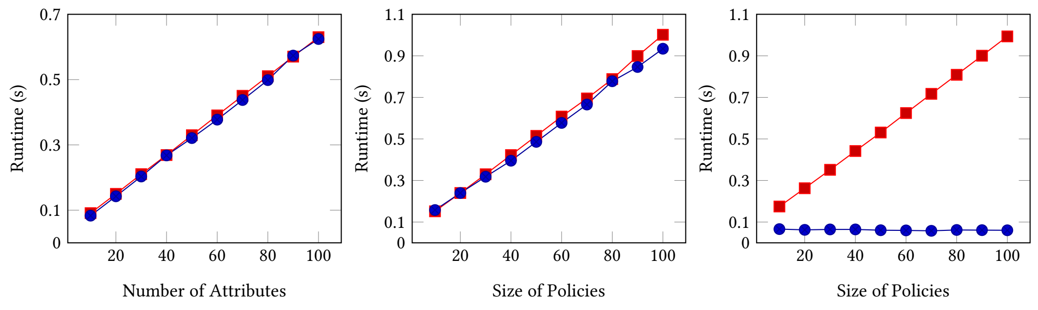

Figure 7: Running time of KeyGen **(left),** Hash **(middle) and** Adapt **(right) algorithms. Red line (with solid square) is for PCH', while blue line (with solid dot) is for StrongPCH.**

whether the holder of sk shall be able to rewrite the hashed message by providing or withholding the second trapdoor *etd*.

Derler et al. [12] introduced policy-based chameleon hash (PCH) to achieve fine-grained and controlled blockchain rewriting. Essentially, the public key encryption scheme used in CHET is replaced by a ciphertext-policy attribute-based encryption (CP-ABE) scheme [4]. If PCH is used in the blockchain, the blockchain rewriting can occur in a transaction-level, which is more practical than rewriting the entire block in [6]. Tian et al. [29] proposed an accountable PCH (PCH') for blockchain rewriting. The proposed PCH' enables the modifiers of transactions to be held accountable for the modified transactions. In particular, PCH' allows a third party (e.g., key generation center) to resolve any dispute over modified transactions. However, both PCH and PCH' suffer the rewriting privilege abuse attacks (i.e., a modifier may re-randomize/delegate her rewriting privilege and distribute it to other unauthorized modifiers without being detected).

Derler et al. [11] introduced a new CH function with full collision-resistance (we call it sCH). The full collision-resistance is a stronger collision-resistance notion than the existing ones in the literature. The basic idea of sCH is that the chameleon hash is a ciphertext C on message m, the randomness r of the chameleon hash is a NIZK proof π. The generation of π requires the party to know either the secret randomness used in generating ciphertext C on message m under a public key pk or the corresponding trapdoor sk. This construction's benefit is to ensure both ciphertext C and NIZK proof π are non-malleable together.

In anther work, Deuber et al. [13] introduced an efficient rewritable blockchain in the permissionless setting (we call it V-BR). The proposed scheme relies on a consensus-based e-voting system, such that the modification is executed in the chain if a modification request from any public user gathers enough votes from miners. When integrated into the existing Bitcoin system, V-BR incurs only a small additional overhead.

In this work, we introduce strongly accountable PCH (StrongPCH) that allows the authorized/unauthorized modifiers' public keys to be held accountable for the modified transactions. Compared to PCH', StrongPCH can trace the original modifiers' identities from the revealed rewriting privileges (or decryption keys), regardless of whether they are abused or not. This is because the users' identities embedded in the decryption keys are fixed by the attribute authority during key generation. In particular, the embedded identities remain intact even if rewriting privilege abuse attacks are performed by the authorized modifiers. In PCH', if an unauthorized modifier rewrites transactions using some delegated decryption keys, AA cannot trace the original modifiers' identities because the identities embedded in the decryption keys are changed due to a key delegation process existing in the ABE scheme [4]. Another difference is that we use an anonymous digital signature to ensure anonymous (or indistinuishable) blockchain rewriting. However, PCH' doesnot provide this privacy guarantee because PCH' relies on a conventional digital signature scheme [27].

Table 3: The comparison between various blockchain rewriting solutions. CH-based indicates CH-based blockchain rewriting. Blockchain rewriting can be performed in a permissioned/permissionless (perm/perm-less) setting, or both of them. Fine-grained means that each mutable transaction is associated with an access policy such that the transaction can be modified by anyone whose rewriting privilege satisfy the policy. S-ACT means Strong Accountability.

	CH [6]	PCH [12]	V-BR [13]	sCH[11]	PCH' [29]	Ours
CH-based	✓	✓	×	✓	✓	✓
Setting	both	perm	perm-less	both	perm	perm
Fine-grained	×	✓	×	×	✓	✓
Indistinguishable	✓	✓	×	✓	×	✓
ACT	✓	×	✓	✓	✓	✓
S-ACT	-	×	-	-	×	✓

8 CONCLUSION

In this paper, we proposed a generic framework of strongly accountable policy-based chameleon hash StrongPCH for blockchain rewriting. The proposed solution can help thwart maliciously rewriting of blockchain by any authorized or unauthorized modifiers. We presented a concrete construction of StrongPCH. The implementation and evaluation analysis have shown that the proposed construction is practical in blockchain applications.

REFERENCES

[1] [n.d.]. General Data Protection Regulation. https://eugdpr.org.
[2] [n.d.]. Source Code. https://github.com/SMC-SMU/Strong-Accountable-PCH.
[3] [n.d.]. Transaction Size. https://bitcoin.stackexchange.com/questions/1823/what-is-the-maximum-size-of-a-transaction.

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

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

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

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

[8] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. 2005. Evaluating 2-DNF formulas on ciphertexts. In *TCC*. 325–341.

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

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

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

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

[13] Dominic Deuber, Bernardo Magri, and Sri Aravinda Krishnan Thyagarajan. 2019. Redactable blockchain in the permissionless setting. *arXiv preprint arXiv:1901.03206* (2019).

[14] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. 2020. Pixel: Multi-signatures for consensus. In *USENIX*. 2093–2110.

[15] Jens Groth, Rafail Ostrovsky, and Amit Sahai. 2012. New techniques for noninteractive zero-knowledge. *Journal of the ACM (JACM)* 59, 3 (2012), 11.

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

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

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

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

[20] 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.

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

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

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

[24] Jianting Ning, Zhenfu Cao, Xiaolei Dong, and Lifei Wei. 2016. White-box traceable CP-ABE for cloud storage service: how to catch people leaking their access credentials effectively. *IEEE TDSC* 15, 5 (2016), 883–897.

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

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

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

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

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

A SECURITY ANALYSIS OF NEW ASSUMPTIONS

In this section, we present the security analysis of the proposed new assumptions, including the extended DLIN (eDLIN), Computational Bilinear Diffie-Hellman (CBDH), and Decisional Bilinear Diffie-Hellman (DBDH). We prove the security in group $\mathbb{G}$, along with the bilinear maps $\hat{e} : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$. For simplicity, we do not consider the isomorphism $\varphi : \mathbb{H} \rightarrow \mathbb{G}$ (i.e., $\mathbb{G} \neq \mathbb{H}$), and we do not include the group elements from group $\mathbb{H}$ in the following theorems. One can easily add these two functions into the following theorems.

THEOREM 15. *Let $(\epsilon_1, \epsilon_2, \epsilon_T) : \mathbb{Z}_q \rightarrow \{0,1\}^*$ be three random encodings (injective functions) where $\mathbb{Z}_q$ is a prime field, and the encoding of group elements are $\mathbb{G} = \{\epsilon_1(a) | a \in \mathbb{Z}_q\}, \mathbb{H} = \{\epsilon_2(b) | b \in \mathbb{Z}_q\}, \mathbb{G}_T = \{\epsilon_T(c) | c \in \mathbb{Z}_q\}$. If $(a,b,c) \overset{R}{\leftarrow} \mathbb{Z}_q$ and encodings $\epsilon_1, \epsilon_2, \epsilon_T$ are randomly chosen, then we define the advantage of the adversary in solving the eDLIN with at most Q queries to the group operation oracles O_1, O_2, O_T and the bilinear pairing $\hat{e}$ as*

$$\begin{aligned}
|\mathsf{Adv}^{eDLIN}_{\mathcal{A}}(\lambda) &= \Pr[\mathcal{A}(q, \epsilon_1(1), \epsilon_1(a_1), \epsilon_1(a_2), \\
&\quad \epsilon_1(1/a_1), \epsilon_1(1/a_2), \epsilon_1(a_1 s_1), \epsilon_1(a_1 s_2), \\
&\quad \epsilon_1(a_2 s_1), \epsilon_1(a_2 s_2), \epsilon_1(s_1/a_1), \epsilon_1(s_2/a_2), \\
&\quad \epsilon_1(z), \epsilon_1(t_0), \epsilon_1(t_1), \epsilon_2(1), \epsilon_T(1)) \\
&= w : (a_1, a_2, s_1, s_2, z, s \overset{R}{\leftarrow} \mathbb{Z}_q, w \in (0,1), \\
&\quad t_w = z(s_1 + s_2), t_{1-w} = s)] \\
&\quad -1/2| \leq \frac{10(Q+15)^2}{q}
\end{aligned}$$

PROOF. Let $\mathcal{S}$ play the following game for $\mathcal{A}$. $\mathcal{S}$ maintains three polynomial sized dynamic lists: $L_1 = \{(p_i, \epsilon_{1,i})\}, L_2 = \{(q_i, \epsilon_{2,i})\}, L_T = \{(t_i, \epsilon_{T,i})\}$. The $p_i \in \mathbb{Z}_q[A_1, A_2, S_1, S_2, Z, S, T_0, T_1]$ are 8-variate polynomials over $\mathbb{Z}_q$, such that $p_0 = 1, p_1 = A_1, p_2 = A_2, p_3 = A_1^{q-2}, p_4 = A_2^{q-2}, p_5 = A_1 \cdot S_1, p_6 = A_1 \cdot S_2, p_7 = A_2 \cdot S_1, p_8 = A_2 \cdot S_2, p_9 = A_1^{q-2} \cdot S_1, p_{10} = A_2^{q-2} \cdot S_2, p_{11} = T_0, p_{12} = T_1$. $\mathcal{S}$ also generates $q_0 = 1, t_0 = 1$. Besides, $(\{\epsilon_{1,i}\}_{i=0}^{12} \in \{0,1\}^*, \{\epsilon_{2,0}\} \in \{0,1\}^*, \{\epsilon_{T,0}\} \in \{0,1\}^*)$ are arbitrary distinct strings, $\mathcal{S}$ then sets those pairs $(p_i, \epsilon_{1,i})$ as L_1. Therefore, the three lists are initialised as $L_1 = \{(p_i, \epsilon_{1,i})\}_{i=0}^{12}, L_2 = (q_0, \epsilon_{2,0}), L_T = (t_0, \epsilon_{T,0})$.

At the beginning of the game, $\mathcal{S}$ sends the encoding strings $(\{\epsilon_{1,i}\}_{i=0,\cdots,12}, \epsilon_{2,0}, \epsilon_{T,0})$ 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 $\mathbb{H}, \mathbb{G}_T$ (i.e., O_2, O_T) is 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 $a_1, a_2, s_1, s_2, z, s \in \mathbb{Z}_q$, generates $t_b = z(s_1 + s_2)$ and $t_{1-b} = s$. $\mathcal{S}$ sets $A_1 = a_1, A_2 = a_2, S_1 = s_1, S_2 = s_2, Z = z, S = s, T_0 = t_b, T_1 = t_{1-b}$. 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) $p_i(a_1, a_2, s_1, s_2, z, s, t_0, t_1) = p_j(a_1, a_2, s_1, s_2, z, s, t_0, t_1)$: The polynomial $p_i \neq p_j$ due to the construction method of L_1, and $(p_i - p_j)(a_1, a_2, s_1, s_2, z, s, t_0, t_1)$ is a non-zero polynomial of degree $[0, 2]$ or $q-2$ ($q-2$ is produced by A_1^{q-2}). We have $A_1 \cdot A_1^{q-2} = A_1^{q-1} \equiv 1(\mod q)$, so $A_1 \cdot A_1^{q-2} \cdot S_1 \equiv A_1 \cdot S_1(\mod q)$ and the maximum degree of $A_1 \cdot S_1(p_i - p_j)(a_1, a_2, s_1, s_2, z, s, t_0, t_1)$ is 4. By using Lemma 1 in [28], we have $\Pr[(p_i - p_j)(a_1, a_2, s_1, s_2, z, s, t_0, t_1) = 0] \leq \frac{4}{q}$ and thus $\Pr[p_i(a_1, a_2, s_1, s_2, z, s, t_0, t_1) = p_j(a_1, a_2, s_1, s_2, z, s, t_0, t_1)] \leq \frac{4}{q}$. Therefore, we have the abort probability is $\Pr[\mathsf{abort}_1] \leq \frac{4}{q}$.

(2) $q_i(a_1, a_2, s_1, s_2, z, s, t_0, t_1) = q_j(a_1, a_2, s_1, s_2, z, s, t_0, t_1)$: The polynomial $q_i \neq q_j$ due to the construction method of L_2, and $(q_i - q_j)(a_1, a_2, s_1, s_2, z, s, t_0, t_1)$ 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) $t_i(a_1, a_2, s_1, s_2, z, s, t_0, t_1) = t_j(a_1, a_2, s_1, s_2, z, s, t_0, t_1)$: The polynomial $t_i \neq t_j$ due to the construction method of L_3, and $(t_i - t_j)(a_1, a_2, s_1, s_2, z, s, t_0, t_1)$ is a non-zero polynomial of degree 0, 1, 2, or $2q - 4$ ($2q - 4$ is produced by $A_1^{q-2} \cdot A_2^{q-2} \cdot S_1 \cdot S_2$, and we denote it as $A^{2q-4} \cdot S^2$). Since $A^2 \cdot A^{2q-4} = (A^{q-1})^2 \equiv 1(\mod q)$, $A^2 \cdot A^{2q-4} \cdot S^2 \equiv (A \cdot S)^2(\mod q)$, so the maximum degree of $(A_1 \cdot S_1)^2(t_i - t_j)(a_1, a_2, s_1, s_2, z, s, t_0, t_1)$ is 6. Therefore, we have $\Pr[(t_i - t_j)(a_1, a_2, s_1, s_2, z, s, t_0, t_1) = 0] \leq \frac{6}{q}$ and thus $\Pr[t_i(a_1, a_2, s_1, s_2, z, s, t_0, t_1) = t_j(a_1, a_2, s_1, s_2, z, s, t_0, t_1)] \leq \frac{6}{q}$.

By summing over all valid pairs (i, j) in each case (i.e., at most $2\binom{Q+15}{2}$ pairs), we have the abort probability is

$$\Pr[\mathsf{abort}] = \Pr[\mathsf{abort}_1] + \Pr[\mathsf{abort}_2] + \Pr[\mathsf{abort}_3] \leq 2\binom{Q + 15}{2} \cdot \left(\frac{4}{q} + \frac{6}{q}\right) \leq \frac{10(Q + 15)^2}{q}.$$

□

Theorem 16. *Let $(\epsilon_1, \epsilon_2, \epsilon_T) : \mathbb{Z}_q \rightarrow \{0, 1\}^*$ 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 $(a, b, c) \xleftarrow{R} \mathbb{Z}_q$ and encodings $\epsilon_1, \epsilon_2, \epsilon_T$ are randomly chosen, we define the advantage of the adversary in solving the CBDH with at most Q queries to the group operation oracles O_1, O_2, O_T and the bilinear pairing $\hat{e}$ as*

$$\begin{aligned} |\mathsf{Adv}_{\mathcal{A}}^{CBDH}(\lambda) &= \Pr[\mathcal{A}(q, \epsilon_1(1), \epsilon_1(a), \epsilon_1(b), \epsilon_2(1) \\ &\quad \epsilon_2(a), \epsilon_2(b), \epsilon_2(c), \epsilon_2(ab), \epsilon_2(1/ab)) \\ &\quad = \epsilon_T(c/ab)]| \leq \frac{12(Q + 10)^2}{q} \end{aligned}$$

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 $p_i \in \mathbb{Z}_q[A, B]$ are 2-variate polynomials over $\mathbb{Z}_q$, such that $p_0 = 1, p_1 = A, p_2 = B$. $q_i \in \mathbb{Z}_q[A, B]$ are 2-variate polynomials over $\mathbb{Z}_q$, such that $q_0 = 1, q_1 = A, q_2 = B, q_3 = C, q_4 = AB, q_5 = AB^{q-2}$. $t_i \in \mathbb{Z}_q[A, B, C]$ are 3-variate polynomials over $\mathbb{Z}_q$, such that $t_0 = C(AB)^{q-2}$. Besides, $\{\epsilon_{1,i}\}_{i=0}^{2} \in \{0, 1\}^*, \{\epsilon_{2,i}\}_{i=0}^{5} \in \{0, 1\}^*, \{\epsilon_{T,0}\} \in \{0, 1\}^*$ are arbitrary distinct strings. Therefore, the three lists are initialised as $L_1 = \{(p_i, \epsilon_{1,i})\}_{i=0}^{2}, L_2 = \{(q_i, \epsilon_{2,i})\}_{i=0}^{5}, L_T = (t_0, \epsilon_{T,0})$.

At the beginning of the game, $\mathcal{S}$ sends the encoding strings $(\{\epsilon_{1,i}\}_{i=0,\cdots,2}, \{\epsilon_{2,i}\}_{i=0,\cdots,5}, \epsilon_{T,0})$ to $\mathcal{A}$. After this, $\mathcal{S}$ simulates the group operation oracles O_1, O_2, O_T and the bilinear pairing $\hat{e}$ using the same method as described in Theorem 15. After querying at most Q times of corresponding oracles, $\mathcal{A}$ terminates and outputs $\epsilon_T(c/ab)$. At this point, $\mathcal{S}$ chooses random $a, b, c \in \mathbb{Z}_q$. $\mathcal{S}$ sets $A = a, B = b, C = c$. The simulation by $\mathcal{S}$ is perfect unless the abort event happens. Thus, we bound the probability of event abort by analyzing the following cases:

(1) $p_i(a, b, c) = p_j(a, b, c)$: The polynomial $p_i \neq p_j$ due to the construction method of L_1, and $(p_i - p_j)(a, b, c)$ is a non-zero polynomial of degree $[0, 1]$. The maximum degree of $(p_i - p_j)(a, b, c)$ is 1. By using Lemma 1 in [28], we have $\Pr[(p_i - p_j)(a, b, c) = 0] \leq \frac{1}{q}$ and thus $\Pr[p_i(a, b, c) = p_j(a, b, c)] \leq \frac{1}{q}$. So, we have the abort probability is $\Pr[\mathsf{abort}_1] \leq \frac{1}{q}$.

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

(3) $t_i(a, b, c) = c/ab$: The degree of p_i is $[0, 1]$, and the the degree of q_i is $[0, 4]$. Since $CAB \cdot (AB)^{q-2} \equiv CAB(\mod q)$, we have $CAB(t_i - t_j)(a, b, c)$ is non-zero polynomial of degree $[0, 7]$, So, we have $\Pr[(t_i - t_j)(a, b, c) = 0] \leq \frac{7}{q}$ and thus $\Pr[\mathsf{abort}_3] \leq \frac{7}{q}$.

By summing over all valid pairs (i, j) in each case (i.e., at most $\binom{Q_{\epsilon_1}+3}{2} + \binom{Q_{\epsilon_2}+6}{2} + \binom{Q_{\epsilon_T}+1}{2}$ pairs), and $Q_{\epsilon_1} + Q_{\epsilon_2} + Q_{\epsilon_T} = Q + 10$, we have the abort probability is

$$\begin{aligned} \Pr[\mathsf{abort}] &= \Pr[\mathsf{abort}_1] + \Pr[\mathsf{abort}_2] + \Pr[\mathsf{abort}_3] \\ &\leq \left[\binom{Q_{\epsilon_1} + 3}{2} + \binom{Q_{\epsilon_2} + 6}{2} + \binom{Q_{\epsilon_T} + 1}{2}\right] \\ &\quad \cdot \left(\frac{1}{q} + \frac{4}{q} + \frac{7}{q}\right) \leq \frac{12(Q + 10)^2}{q}. \end{aligned}$$

□

Theorem 17. *Let $(\epsilon_1, \epsilon_2, \epsilon_T) : \mathbb{Z}_q \rightarrow \{0, 1\}^*$ 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 $(a, b, d, e, f, \{c_i, l_i\}) \xleftarrow{R} \mathbb{Z}_q$ and encodings $\epsilon_1, \epsilon_2, \epsilon_T$ are randomly chosen, we define the advantage of the adversary in solving the DBDH with at most Q queries to the group operation oracles O_1, O_2, O_T and the bilinear pairing $\hat{e}$ as*

$$\begin{aligned} |\mathsf{Adv}_{\mathcal{A}}^{DBDH}(\lambda) &= \Pr[\mathcal{A}(q, \epsilon_1(1), \epsilon_1(a), \epsilon_1(b), \epsilon_1(f), \\ &\quad \epsilon_1(ed), \{\epsilon_1(ec_i), \epsilon_1(el_i)\} \\ &= w : (a, b, d, e, f \xleftarrow{R} \mathbb{Z}_q, \{c_i, l_i\} \in \mathbb{Z}_q, \\ &\quad w \in (0, 1), t_w = ab + edc_i, \\ &\quad t_{1-w} = fb + edl_i)] - 1/2| \\ &\leq \frac{6(Q + 11)^2}{q} \end{aligned}$$

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 $p_i \in \mathbb{Z}_q[A,B,C,D,E,F,L,T_0,T_1]$ are 9-variate polynomials over $\mathbb{Z}_q$, such that $p_0 = 1, p_1 = A, p_2 = B, p_3 = F, p_4 = ED, p_5 = EC_i, p_6 = EL_i, p_7 = T_w, p_8 = T_{1-w}$. $\mathcal{S}$ also generates $q_0 = 1, t_0 = 1$. Besides, $\{\epsilon_{1,i}\}_{i=0}^{8} \in \{0,1\}^*, \epsilon_{2,0} \in \{0,1\}^*, \epsilon_{T,0} \in \{0,1\}^*$ are arbitrary distinct strings. Therefore, the three lists are initialised as $L_1 = \{(p_i, \epsilon_{1,i})\}_{i=0}^{8}, L_2 = (q_0, \epsilon_{2,0}), L_T = (t_0, \epsilon_{T,0})$.

At the beginning of the game, $\mathcal{S}$ sends the encoding strings $(\{\epsilon_{1,i}\}_{i=0,\cdots,8}, \epsilon_{2,0}, \epsilon_{T,0})$ to $\mathcal{A}$. After this, $\mathcal{S}$ simulates the group operation oracles O_1, O_2, O_T and the bilinear pairing ê using the same method as described in Theorem 15. After querying at most Q times of corresponding oracles, $\mathcal{A}$ terminates and outputs a guess $w' = \{0,1\}$. At this point, $\mathcal{S}$ chooses random $a, b, d, f, c_i, l_i \in \mathbb{Z}_q$, generates $t_w = ab + edc_i$ and $t_{1-w} = fb + edl_i$. $\mathcal{S}$ sets $A = a, B = b, D = d, E = e, F = f, C_i = c_i, L_i = l_i, T_0 = t_w, T_1 = t_{1-w}$. The simulation by $\mathcal{S}$ is perfect unless the abort event happens. Thus, we bound the probability of event abort by analyzing the following cases:

(1) $p_i(a,b,c,\cdots) = p_j(a,b,c,\cdots)$: The polynomial $p_i \neq p_j$ due to the construction method of L_1, and $(p_i - p_j)(a,b,c,\cdots)$ is a non-zero polynomial of degree $[0,3]$. The maximum degree of $(p_i - p_j)(a,b,c,\cdots)$ is 3. By using Lemma 1 in [28], we have $\Pr[(p_i - p_j)(a,b,c,\cdots) = 0] \leq \frac{3}{q}$ and thus $\Pr[p_i(a,b,c,\cdots) = p_j(a,b,c,\cdots)] \leq \frac{3}{q}$. So, we have the abort probability $\Pr[\text{abort}_1] \leq \frac{3}{q}$.

(2) $q_i(a,b,c,\cdots) = q_j(a,b,c,\cdots)$: The polynomial $q_i \neq q_j$ due to the construction method of L_2, and $(q_i - q_j)(a,b,c,\cdots)$ is a non-zero polynomial of degree 0. The abort probability is "0" (i.e., the maximum degree is "0" as L_2 contains a single string $\epsilon_{(2,0)}$ only).

(3) $t_i(a,b,c,\cdots) = t_j(a,b,c,\cdots)$: Since the degree of p_i is $[0,3]$, we have $\Pr[(t_i - t_j)(a,b,c,\cdots) = 0] \leq \frac{3}{q}$ and $\Pr[\text{abort}_3] \leq \frac{3}{q}$.

By summing over all valid pairs (i,j) in each case (i.e., at most $\binom{Q_{\epsilon_1}+9}{2} + \binom{Q_{\epsilon_2}+1}{2} + \binom{Q_{\epsilon_T}+1}{2}$ pairs), and $Q_{\epsilon_1} + Q_{\epsilon_2} + Q_{\epsilon_T} = Q + 11$, we have the abort probability is

$$\begin{aligned}\Pr[\text{abort}] &= \Pr[\text{abort}_1] + \Pr[\text{abort}_2] + \Pr[\text{abort}_3] \\ &\leq [\binom{Q_{\epsilon_1}+9}{2} + \binom{Q_{\epsilon_2}+1}{2} + \binom{Q_{\epsilon_T}+1}{2}] \\ &\cdot (\frac{3}{q} + \frac{3}{q}) \leq \frac{6(Q+11)^2}{q}.\end{aligned}$$

□

B SECURITY ANALYSIS OF STRONGPCH

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

B.1 Indistinguishability

Theorem 18. *The StrongPCH scheme is indistinguishable if the CH is indistinguishable and the Σ scheme is anonymous.*

Proof. We define a sequence of games $\mathbb{G}_i$, $i = 0, \cdots, 3$ and let $\mathsf{Adv}_i^{\text{SPCH}}$ denote the advantage of the adversary in game $\mathbb{G}_i$. Assume that $\mathcal{A}$ issues at most $n(\lambda)$ hash queries, which include a HashOrAdapt query (we call it g-th query).

- $\mathbb{G}_0$: This is original game for indistinguishability.
- $\mathbb{G}_1$: This game is identical to game $\mathbb{G}_0$ except that in the g-th hash query, challenger $\mathcal{S}$ directly hashes a message $(b, \mathsf{r}) \leftarrow \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}^*, m)$, instead of calculating the chameleon hash and randomness (b, r) using the Adapt algorithm. Below we show the difference between $\mathbb{G}_0$ and $\mathbb{G}_1$ is negligible if CH scheme is indistinguishable.
Let $\mathcal{S}$ denote an attacker against CH, who is given a chameleon public key pk^* and a HashOrAdapt oracle, aims to break the CH's indistinguishability. $\mathcal{S}$ generates the master key pairs and user's key pairs honestly. In particular, $\mathcal{S}$ sets the chameleon public key of the g-th hash query as pk^*. If $\mathcal{A}$ submits two messages (m_0, m_1, Λ) to $\mathcal{S}$ in the g-th query, $\mathcal{S}$ first obtains a chameleon hash (b_w, r_w) from his HashOrAdapt oracle on messages (m_0, m_1). Then, $\mathcal{S}$ honestly generates a signature $\sigma \leftarrow \mathsf{Sign}_{\Sigma}(\mathsf{sk}, c)$, and a ciphertext $C \leftarrow \mathsf{Enc}_{\mathsf{ABET}}(\mathsf{mpk}_{\mathsf{ABET}}, \perp, \Lambda)$. Note that the signed message c and the verification key apk can be generated using pk^* and user's secret key sk. Eventually, $\mathcal{S}$ returns $(m_w, b_w, \mathsf{r}_w, C, apk, c, \sigma)$ to $\mathcal{A}$. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. If $\mathcal{A}$ guesses the random bit correctly, then $\mathcal{S}$ can break the CH's indistinguishability. Hence, we have
$$\left|\mathsf{Adv}_0^{\text{SPCH}} - \mathsf{Adv}_1^{\text{SPCH}}\right| \leq \mathsf{Adv}_{\mathcal{S}}^{\text{CH}}(\lambda). \tag{1}$$
- $\mathbb{G}_2$: This game is identical to game $\mathbb{G}_1$ except that $\mathcal{S}$ replaces the encrypted secret key sk in C by an empty value $\perp$. Below we show the difference between $\mathbb{G}_1$ and $\mathbb{G}_2$ is negligible if the ABET scheme is semantically secure.
Let $\mathcal{S}$ denotes an attacker against ABET, who is given a public key pk^*, a key generation oracle and a decryption oracle, aims to break ABET's semantic security. $\mathcal{S}$ sets the game for $\mathcal{A}$ by creating users with key pairs $\{(\mathsf{sk}, \mathsf{pk})\}$. $\mathcal{S}$ randomly chooses a user as attribute authority, and sets his public key as pk^*.
$\mathcal{S}$ simulates the g-th hash query as follows. $\mathcal{S}$ first sends a pair of messages $(M_0, M_1) = (\mathsf{sk}, \perp)$ to his challenger, and obtains a ciphertext $C^* \leftarrow \mathsf{Enc}(\mathsf{pk}^*, M_w, \Lambda)$. Second, $\mathcal{S}$ simulates a hash-randomness (b, r) and a message-signature (c, σ) honestly according to the protocol specification. Eventually, $\mathcal{S}$ returns a tuple $(m, b, \mathsf{r}, C^*, apk, c, \sigma)$ to $\mathcal{A}$. Note that $\mathcal{S}$ can honestly answer $\mathcal{A}$'s key generation and decryption queries.
If the encrypted message is the secret key sk, 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 ABET's semantic security.
$$\left|\mathsf{Adv}_1^{\text{SPCH}} - \mathsf{Adv}_2^{\text{SPCH}}\right| \leq \mathsf{Adv}_{\mathcal{S}}^{\text{ABET}}(\lambda). \tag{2}$$
- $\mathbb{G}_3$: This game is identical to game $\mathbb{G}_2$ except that $\mathcal{S}$ outputs a random bit if a **Link** event happens where $\mathcal{A}$ links a message-signature (c, σ) to an honest signer. Since the underlying signature Σ is anonymous, the difference between $\mathbb{G}_2$ and $\mathbb{G}_3$ is negligible, we have
$$\left|\mathsf{Adv}_2^{\text{SPCH}} - \mathsf{Adv}_3^{\text{SPCH}}\right| \leq \mathsf{Adv}_{\mathcal{S}}^{\Sigma}(\lambda). \tag{3}$$

Combining the above results together, we have
$$\mathsf{Adv}_{\mathcal{A}}^{\text{SPCH}}(\lambda) \quad \leq \quad n(\lambda)(\mathsf{Adv}_{\mathcal{S}}^{\text{CH}}(\lambda) + \mathsf{Adv}_{\mathcal{S}}^{\text{ABET}}(\lambda) + \mathsf{Adv}_{\mathcal{S}}^{\Sigma}(\lambda)).$$

□

B.2 Collision Resistant

Theorem 19. *The StrongPCH scheme is collision-resistant if the ABET is semantically secure, and the CH scheme is collision-resistant.*

Proof. We define a sequence of games $\mathbb{G}_i$, $i = 0, \cdots, 3$ and let $\mathsf{Adv}_i^{\mathrm{SPCH}}$ denote the advantage of the adversary in game $\mathbb{G}_i$. Assume that $\mathcal{A}$ issues at most q queries to the $\mathsf{Hash}'_{\mathsf{SPCH}}$ 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 $g \in [1, q]$ as a guess for the index of the Hash' oracle which returns the chameleon hash $(\mathsf{pk}_{\mathsf{CH}}, m^*, b^*, \mathsf{r}^*, \cdots)$. $\mathcal{S}$ will output a random bit if $\mathcal{A}$'s attacking query does not occur in the g-th query. Therefore, we have

$$\mathsf{Adv}_0^{\mathrm{SPCH}} = q \cdot \mathsf{Adv}_1^{\mathrm{SPCH}} \tag{4}$$

- $\mathbb{G}_2$: This game is identical to game $\mathbb{G}_1$ except that in the g-th query, the encrypted message $\mathsf{sk}_{\mathsf{CH}}$ is replaced by $\perp$ (i.e., empty value). The difference between $\mathbb{G}_1$ and $\mathbb{G}_2$ is negligible if the ABET scheme is semantically secure. The reduction is performed using the same method described in previous game $\mathbb{G}_2$. Hence, we have

$$\left|\mathsf{Adv}_1^{\mathrm{SPCH}} - \mathsf{Adv}_2^{\mathrm{SPCH}}\right| \le \mathsf{Adv}_{\mathcal{S}}^{\mathrm{ABET}}(\lambda). \tag{5}$$

- $\mathbb{G}_3$: 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 $(\mathsf{pk}_{\mathsf{CH}}, m^{'*}, b^*, \mathsf{r}'^*, \cdots)$. Below we show that the difference between $\mathbb{G}_2$ and $\mathbb{G}_3$ is negligible if the CH is collision-resistant.
  Let $\mathcal{S}$ denote an attacker against CH with collision-resistance, who is given chameleon public key pk^*, a Hash' oracle, and an Adapt' oracle, aims to find a collision which was not simulated by the Adapt' oracle. $\mathcal{S}$ simulates the game for $\mathcal{A}$ as follows.
  - $\mathcal{S}$ sets up $\mathsf{pk}_{\mathsf{CH}} = \mathsf{pk}^*$ and completes the remainder of **Setup** honestly.
  - $\mathcal{S}$ simulates the response to the Hash' query as $(m, b, \mathsf{r}, C, apk, c, \sigma)$, where (m, b, r) is returned from his Hash' oracle, ciphertext C encrypts $\perp$, and the message-signature (c, σ) is generated honestly according to the protocol specification. Similarly, $\mathcal{S}$ can simulate a chameleon hash $(m^*, b^*, \mathsf{r}^*, C^*, apk^*, c^*, \sigma^*)$ at the g-th query. For the adapt query, $\mathcal{S}$ obtains a new randomness r' from his Adapt' oracle and returns $(m', b, \mathsf{r}', C, apk', c', \sigma')$ to $\mathcal{A}$.
  - At some point, if $\mathcal{A}$ outputs a collision $(\mathsf{pk}_{\mathsf{CH}}, m^*, m^{'*}, b^*, \mathsf{r}^*, \mathsf{r}'^*, C^*, apk^*, c^*, \sigma^*, apk^{'*}, c^{'*}, \sigma^{'*})$ with all the verifications are held, $\mathcal{S}$ outputs $(\mathsf{pk}_{\mathsf{CH}}, m^{'*}, b^*, \mathsf{r}'^*, \cdots)$ as a valid collision to the CH scheme; otherwise, $\mathcal{S}$ aborts the game. Therefore, we have

$$\left|\mathsf{Adv}_2^{\mathrm{SPCH}} - \mathsf{Adv}_3^{\mathrm{SPCH}}\right| \le \mathsf{Adv}_{\mathcal{S}}^{\mathrm{CH}}(\lambda). \tag{6}$$

Combining the above results together, we have

$$\mathsf{Adv}_{\mathcal{A}}^{\mathrm{SPCH}}(\lambda) \quad \le \quad q(\mathsf{Adv}_{\mathcal{S}}^{\mathrm{ABET}}(\lambda) + \mathsf{Adv}_{\mathcal{S}}^{\mathrm{CH}}(\lambda)).$$

□

B.3 Accountability

Theorem 20. *The StrongPCH scheme is strongly accountable if the Σ scheme is EUF-CMA secure, and the ABET scheme is traceable.*

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

- $\mathbb{G}_0$: This is original game for strong accountability.
- $\mathbb{G}_1$: This game is identical to game $\mathbb{G}_0$ except that $\mathcal{S}$ outputs a random bit if a **Forge** event happens where $\mathcal{A}$ outputs $(m, b, \mathsf{r}, C, apk^*, c^*, \sigma^*)$, such that σ^* is a valid signature under $apk^* = \mathsf{KeyGen}'_{\Sigma}(pp_{\Sigma}, 0, \mathsf{sk}^*)$ and c^*, and σ^* is not previously simulated by $\mathcal{S}$.
  Let $\mathcal{S}$ denote a forger against Σ, who is given a public key pk^* and a signing oracle $\mathcal{O}^{\mathsf{Sign}}$, aims to break the EUF-CMA security of Σ. $\mathcal{S}$ sets the game for $\mathcal{A}$ by creating k users with the corresponding key pairs. $\mathcal{S}$ randomly selects a user and guesses that the **Forge** event will happen to the user. $\mathcal{S}$ sets the user's public key as pk^*. Note that the corresponding verification key apk^* can be computed by $\mathcal{S}$ using the master key pair. $\mathcal{S}$ randomly chooses a user as attribute authority, and generates a master key pair $(\mathsf{sk}_{\mathsf{ABET}}, \mathsf{pk}_{\mathsf{ABET}})$ for the attribute authority.
  $\mathcal{S}$ simulates a hash query as follows. First, $\mathcal{S}$ chooses a chameleon secret key $\mathsf{sk}_{\mathsf{CH}}$ and generates a ciphertext as $C \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{ABET}}, \mathsf{sk}_{\mathsf{CH}}, \Lambda)$. Second, $\mathcal{S}$ sends a message $c \leftarrow \mathsf{KeyGen}_{\sigma}(pp, Dlog(\mathsf{pk}^*), \mathsf{sk}_{\mathsf{CH}})$ to his signing oracle $\mathcal{O}^{\mathsf{Sign}}$, and obtains a signature σ. Note that the signed message c can be perfectly simulated by $\mathcal{S}$ due to the homomorphic property of Σ regarding keys and signatures (e.g., $c = \mathsf{pk}^* \cdot h^{\mathsf{sk}_{\mathsf{CH}}}$ in Section 6). Eventually, $\mathcal{S}$ simulates a chameleon hash (b, r) honestly according to the protocol specification, and returns a tuple $(m, b, \mathsf{r}, C, \mathsf{apk}, c, \sigma)$ to $\mathcal{A}$. $\mathcal{S}$ records all the simulated message-signature pairs by including them to a set Q. $\mathcal{S}$ also simulates an adapt query honestly using the chameleon secret key $\mathsf{sk}_{\mathsf{CH}}$.
  When **Forge** event occurs, i.e., $\mathcal{A}$ outputs $(m, b, \mathsf{r}, C, apk^*, c^*, \sigma^*)$, $\mathcal{S}$ checks whether:
  - the verification key apk^* involves the challenge user pk^*;
  - the message-signature pair $(c^*, \sigma^*) \notin Q$ (i.e., it was not previously simulated by $\mathcal{S}$);
  - $1 \overset{?}{=} \mathsf{Verify}(\mathsf{pk}_{\mathsf{CH}}, m, b, \mathsf{r}, C, apk^*, c^*, \sigma^*)$.

  If all the above conditions hold, $\mathcal{S}$ confirms that it as a successful forgery from $\mathcal{A}$, and outputs σ^* as its own forgery; Otherwise, $\mathcal{S}$ aborts the game. Since at most k users involved in the game, we have

$$\left|\mathsf{Adv}_0^{\mathrm{SPCH}} - \mathsf{Adv}_1^{\mathrm{SPCH}}\right| \le k \cdot \mathsf{Adv}_{\mathcal{S}}^{\Sigma}(\lambda). \tag{7}$$

- $\mathbb{G}_2$: This game is identical to game $\mathbb{G}_0$ except that $\mathcal{S}$ outputs a random bit if a **Forge**$'$ event happens where $\mathcal{A}$ outputs $(m', b, \mathsf{r}', C, apk^{*'}, c^{*'}, \sigma^{*'})$, such that $\sigma^{*'}$ is a valid signature under apk^* and c^*, and $\sigma^{*'}$ is not previously simulated by $\mathcal{S}$. The reduction is performed using the same method described as above, we have

$$\left|\mathsf{Adv}_1^{\mathrm{SPCH}} - \mathsf{Adv}_2^{\mathrm{SPCH}}\right| \le k \cdot \mathsf{Adv}_{\mathcal{S}}^{\Sigma}(\lambda). \tag{8}$$

To this end, $\mathcal{A}$ has no advantage in game $\mathbb{G}_2$. The adversary $\mathcal{A}$ becomes a passive one after the first two games. Since the ABET scheme is traceable, we have

$$\mathsf{Adv}_2^{\mathrm{SPCH}} \le \mathsf{Adv}_{\mathcal{S}}^{\mathrm{ABET}}(\lambda). \tag{9}$$

Combining the above results together, we have

$$\mathsf{Adv}_{\mathcal{A}}^{\mathrm{PCH}^*}(\lambda) \quad \leq \quad 2k \cdot \mathsf{Adv}_{\mathcal{S}}^{\Sigma}(\lambda) + \mathsf{Adv}_{\mathcal{S}}^{\mathrm{ABET}}(\lambda)$$

$\square$

C CORRECTNESS AND SECURITY ANALYSIS OF ABET

In this section, we show the correctness of ABET, and its security analysis that includes semantic security and traceability.

C.1 Correctness of the proposed ABET

The detailed decryption is shown below.

- The ciphertext associates with access policy $(\mathbf{M}, \pi)$ in A is calculated as

$$\begin{aligned}
\prod_{i\in I} ct_{(i,\ell)}^{\gamma_i} &= \prod_{i\in I} (\mathsf{H}_1(\pi(i)\ell 1)^{s_1} \cdot \mathsf{H}_1(\pi(i)\ell 2)^{s_2} \\
&\quad \cdot \prod_{j=1}^{n_2} [\mathsf{H}_1(0j\ell 1)^{s_1} \cdot \mathsf{H}_1(0j\ell 2)^{s_2}]^{(\mathbf{M})_{(i,j)}})^{\gamma_i} \\
&= \mathsf{H}_1(01\ell 1)^{s_1} \cdot \mathsf{H}_1(01\ell 2)^{s_2} \\
&\quad \cdot \prod_{i\in I} \mathsf{H}_1(\pi(i)\ell 1)^{\gamma_i \cdot s_1} \cdot \mathsf{H}_1(\pi(i)\ell 2)^{\gamma_i \cdot s_2} \\
&\quad (\forall \ell \in [1,2,3])
\end{aligned}$$

- The first three pairings in A are calculated as

$$\begin{aligned}
&\prod_{t=[1,2]} [\hat{\mathsf{e}}(\mathsf{H}_1(011t), h)^{b_1\cdot r_1\cdot s_t} \cdot \hat{\mathsf{e}}(\mathsf{H}_1(012t), h)^{b_2\cdot r_2\cdot s_t} \\
&\cdot \hat{\mathsf{e}}(\mathsf{H}_1(013t), h)^{z(r_1+r_2)\cdot s_t} \\
&\cdot \prod_{i\in I} (\hat{\mathsf{e}}(\mathsf{H}_1(\pi(i)1t)^{\gamma_i}, h)^{b_1\cdot r_1\cdot s_t} \cdot \hat{\mathsf{e}}(\mathsf{H}_1(\pi(i)2t)^{\gamma_i}, h)^{b_2\cdot r_2\cdot s_t} \\
&\cdot \hat{\mathsf{e}}(\mathsf{H}_1(\pi(i)3t)^{\gamma_i}, h)^{z(r_1+r_2)\cdot s_t})]
\end{aligned}$$

- The last two pairings in A are calculated as

$$\begin{aligned}
&\hat{\mathsf{e}}(ct_{(0,1)} \cdot ct_{(0,3)}, \mathsf{sk}_1) \cdot \hat{\mathsf{e}}(ct_{(0,2)} \cdot ct_{(0,4)}, \mathsf{sk}_2) \\
=\ & \hat{\mathsf{e}}(\mathbb{H}_1^{s_1} \cdot \mathbb{H}_1^{s_2}, (U^{id_i} \cdot E^{w})^{z}) \\
&\cdot \hat{\mathsf{e}}(\mathbb{H}_2^{s_1} \cdot \mathbb{H}_2^{s_2}, (V^{id_i} \cdot F^{s})^{z}) \\
=\ & \hat{\mathsf{e}}(h^{a_1\cdot(s_1+s_2)}, g^{z\cdot(id_i\cdot r_0+w)/a_1}) \\
&\cdot \hat{\mathsf{e}}(h^{a_2\cdot(s_1+s_2)}, g^{z\cdot(id_i\cdot s_0+s)/a_2}) \\
=\ & \hat{\mathsf{e}}(g,h)^{(s_1+s_2)\cdot z\cdot(id_i\cdot r_0+w)} \\
&\cdot \hat{\mathsf{e}}(g,h)^{(s_1+s_2)\cdot z\cdot(id_i\cdot s_0+s)} \\
=\ & \hat{\mathsf{e}}(g,h)^{z\cdot(s_1+s_2)\cdot[id_i\cdot(r_0+s_0)+(w+s)]}
\end{aligned}$$

- The first three pairings in B are calculated as

$$\begin{aligned}
&\hat{\mathsf{e}}(g^{d_1} \cdot \mathsf{H}_1(0111)^{b_1\cdot r_1/a_1} \cdot \mathsf{H}_1(0121)^{b_2\cdot r_2/a_1} \\
&\cdot \mathsf{H}_1(0131)^{z\cdot(r_1+r_2)/a_1} \cdot g^{z\cdot\sigma'/a_1}, h^{a_1\cdot s_1}) \\
\cdot\ & \hat{\mathsf{e}}(\prod_{i\in I}(\mathsf{H}_1(y11)^{b_1\cdot r_1/a_1} \cdot \mathsf{H}_1(y21)^{b_2\cdot r_2/a_1} \\
&\cdot \mathsf{H}_1(y31)^{z\cdot(r_1+r_2)/a_1} \cdot g^{z\cdot\sigma_y/a_1})^{\gamma_i}, h^{a_1\cdot s_1}) \\
\cdot\ & \hat{\mathsf{e}}(g^{d_2} \cdot \mathsf{H}_1(0112)^{b_1\cdot r_1/a_2} \cdot \mathsf{H}_1(0122)^{b_2\cdot r_2/a_2} \\
&\cdot \mathsf{H}_1(0132)^{z\cdot(r_1+r_2)/a_2} \cdot g^{z\cdot\sigma'/a_2}, h^{a_2\cdot s_2}) \\
\cdot\ & \hat{\mathsf{e}}(\prod_{i\in I}(\mathsf{H}_1(y12)^{b_1\cdot r_1/a_2} \cdot \mathsf{H}_1(y22)^{b_2\cdot r_2/a_2} \\
&\cdot \mathsf{H}_1(y32)^{z\cdot(r_1+r_2)/a_2} \cdot g^{z\cdot\sigma_y/a_2})^{\gamma_i}, h^{a_2\cdot s_2}) \\
\cdot\ & \hat{\mathsf{e}}(g^{d_3} \cdot g^{-\sigma'} \cdot \prod_{i\in I}(g^{-\sigma_y})^{\gamma_i}, h^{z\cdot(s_1+s_2)}).
\end{aligned}$$

where $y = \pi(i)$

- The last pairing in B is calculated as

$$\begin{aligned}
\hat{\mathsf{e}}(\mathsf{sk}_3, ct_{0,5}) &= \hat{\mathsf{e}}(W^{id_i} \cdot g^{w+s}, h^{z\cdot(s_1+s_2)}) \\
&= \hat{\mathsf{e}}(g^{id_i\cdot(r_0+s_0)+(w+s)}, h^{z\cdot(s_1+s_2)}) \\
&= \hat{\mathsf{e}}(g,h)^{z\cdot(s_1+s_2)\cdot[id_i\cdot(r_0+s_0)+(w+s)]}
\end{aligned}$$

- If B/A, then we left the following equation

$$\begin{aligned}
&\prod_{t\in[1,2]} \hat{\mathsf{e}}(g^{d_t} \cdot g^{z\cdot\sigma'/a_t} \cdot \prod_{i\in I} g^{\gamma_i\cdot z\cdot\sigma_{\pi(i)}/a_t}, h^{a_t\cdot s_t}) \\
&\cdot \hat{\mathsf{e}}(g^{d_t} \cdot g^{-\sigma'} \cdot \prod_{i\in I} g^{-\gamma_i\sigma_{\pi(i)}}, h^{z\cdot(s_1+s_2)}) \\
=\ & \hat{\mathsf{e}}(g,h)^{d_1\cdot a_1\cdot s_1+d_2\cdot a_2\cdot s_2+z\cdot d_3\cdot(s_1+s_2)} = \underline{T_1^{s_1} \cdot T_2^{s_2}}.
\end{aligned}$$

C.2 Semantic Security

Theorem 21. *The proposed ABET scheme is semantically secure in the random oracle model if the eDLIN assumption holds in the asymmetric pairing groups.*

Compact Group Representation [10]. We use $[a]_1, [b]_2$ and $[z]_T$ to denote g^a, h^b and $\hat{\mathsf{e}}(g,h)^z$, respectively. 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. $\hat{\mathsf{e}}([\mathbf{A}]_1, [\mathbf{B}]_2)$ for two matrices $\mathbf{A}, \mathbf{B}$ is defined as $[\mathbf{A}^{\mathrm{T}} \cdot \mathbf{B}]_T$.

If we define $\mathbf{A}, \mathbf{A}^{-1}, \mathbf{s}, \mathbf{s}^*$ and $\mathbf{s}'$ as

$$\begin{bmatrix} a_1 & 0 \\ 0 & a_2 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} a_1^{-1} & 0 \\ 0 & a_2^{-1} \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}, \begin{bmatrix} s_2 \\ s_1 \end{bmatrix}, and \begin{bmatrix} zs_1 \\ zs_2 \\ s \end{bmatrix},$$

we can state the eDLIN assumption as

$$([\mathrm{A}]_1, [\mathrm{A}]_1, [\mathbf{B}]_2, [\mathrm{B}]_2, [z]_1, [z]_2) \approx ([\mathrm{A}]_1, [\mathrm{B}]_2, [\mathbf{s}']_1, [\mathbf{s}']_2)$$

where $\approx$ denotes the computational indistinguishability, and $[\mathrm{A}]_1 \leftarrow ([\mathbf{A}]_1, [\mathbf{A}^{-1}]_1, \mathbf{s}, \mathbf{s}^*)$ denotes the set of any combinations between matrixes and vectors in group $\mathbb{G}$. The same rule applies to $[\mathrm{B}]_2$.

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

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

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

- Setup: $\mathcal{S}$ sets up the group elements honestly, and obtains $(\mathbf{A}, \mathbf{A}^{-1}, \mathbf{a}^{\perp}), (\mathbf{B}, \mathbf{b}^{\perp}) \leftarrow \mathsf{Samp}(\lambda)$ and $(d_1, d_2, d_3, r_0, s_0, z) \leftarrow \mathbb{Z}_q$. Let $\mathbf{d} = (d_1, d_2, d_3)^{\mathrm{T}}, \mathbf{d'} = (d_1, d_2, zd_3)^{\mathrm{T}}, \mathbf{a} = (a_1, a_2)^{\mathrm{T}}, \mathbf{b} = (b_1, b_2)^{\mathrm{T}}, \mathbf{e} = (r_0, s_0)^{\mathrm{T}}$. $\mathcal{S}$ also generates some variants

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

  , sets $\mathsf{mpk} = ([\mathbf{A}]_2, [\mathbf{d}'^{\mathrm{T}}\mathbf{A}]_T, \underline{[\mathbf{A}^{-1}]_1, [\mathbf{A}^{-1}\mathbf{e}]_1, [z]_1, [z]_2})$ and $\mathsf{msk} = ([\mathbf{d}]_1, \mathbf{a}, \mathbf{b}, \underline{\mathbf{e}, z})$.
  To simulate the random oracle, we use the same method described in [4]. Specifically, $\mathcal{S}$ maintains two lists L and Q. The list L has entries of the form $(x, \mathbf{W}_x)$ or $(j, \mathbf{U}_j)$ where x is an arbitrary binary string, j is a positive integer, and $\mathbf{W}_x, \mathbf{U}_j$ are 3×3 matrices over $\mathbb{Z}_q$. The list Q has entries of the form (q, r) where q is either $x\ell t$ or $0j\ell t$ (for $\ell \in \{1,2,3\}$ and $t \in \{1,2\}$) or something else, and $r \in \mathbb{G}$. $\mathcal{S}$ answers the hash oracle queries using the same method described in [4].
- KeyGen: If $\mathcal{A}$ issues a key query in the form of (id, 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 [4]. The list L is also updated accordingly). Now, $\mathcal{S}$ chooses $r_1, r_2, \sigma' \in \mathbb{Z}_q$ and $\sigma_y \in \mathbb{Z}_q$ for $y \in \delta$. Let $\mathbf{r} = (r_1, r_2)^{\mathrm{T}}, \mathbf{r}' = (zr_1, zr_2)^{\mathrm{T}}, \mathbf{f} = (r, s)^{\mathrm{T}}$ and computes the decryption key as $\mathsf{sk}_0 = [\mathbf{B}'\mathbf{r}]_2, \underline{\mathsf{sk}_{(1,2,3)} = [z\mathbf{A}^{-1'} id(\mathbf{e} + \mathbf{f})]_1}, \mathsf{sk}_y = [\mathbf{W_y}\mathbf{B}\mathbf{r}' + \sigma_y \mathbf{a}^{\perp'}]_1, \mathsf{sk}' = [\mathbf{d} + \mathbf{U_1}\mathbf{B}\mathbf{r}' + \sigma'\mathbf{a}^{\perp'}]_1$. Eventually, $\mathcal{S}$ returns $(\mathsf{sk}_0, \{\mathsf{sk}_y\}_{y \in \delta}, \mathsf{sk}', \mathsf{sk}_{(1,2,3)})$ to $\mathcal{A}$.
- Enc: If $\mathcal{A}$ sends messages (M_0, M_1) and a policy $(\mathbf{M}, \pi)$, $\mathcal{S}$ retrieves $r = [(\mathbf{W}_{\pi(i)}^{\mathrm{T}}\mathbf{A})_{\ell,t}]_1$ and $r = [(\mathbf{U}_j^{\mathrm{T}}\mathbf{A})_{\ell,t}]_1$ for all $i \in [1, \cdots, n_1], j \in [1, \cdots, n_2], \ell, t$ from the list Q (if a $\pi(i)\ell t$ or $0j\ell t$ is not found in 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 $\mathbf{s}^* = (s_1, s_2)$, $\mathbf{s}' = (s_2, s_1)$ and computes the challenge ciphertext as $\underline{ct_0 = [\mathbf{A}'\mathbf{s}^*]_2, [\mathbf{a}\mathbf{s}']_2}$, $ct_i = [\mathbf{W}_{\pi(i)}^{\mathrm{T}}\mathbf{A}\mathbf{s}^* + \sum_{j=1}^{n_2} (\mathbf{M})_{i,j}\mathbf{U}_j^{\mathrm{T}}\mathbf{A}\mathbf{s}^*]_1$ and $ct' = [\mathbf{d}'^{\mathrm{T}}\mathbf{A}\mathbf{s}^*]_T \cdot M_b$.

Description of Hybrids. The security proof consists of a set of hybrids from Hyb_0 to Hyb_5. A hybrid describes how the challenger $\mathcal{S}$ 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. 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 [4], except the indistinguishability between two hybrids with computationally-close is reduced to the proposed eDLIN assumption.

C.3 Extractability

THEOREM 22. *The proposed ABET scheme is extractable if the underlying proof commitment scheme is perfectly extractable.*

The security reduction is straightforward. If an adversary $\mathcal{A}$ extracts a message *id* from a decryption key, then there exists an extractor E who can use the extracted message to hold the extractability of the proof commitment scheme.

D SECURITY ANALYSIS OF SIGNATURE Σ

In this section, we present the security analysis of the proposed Σ scheme, including unforgeability, and anonymity.

D.1 Unforgeability

THEOREM 23. *The proposed Σ scheme achieves selective EUF-CMA security if the proposed CBDH assumption is held in the asymmetric pairing groups.*

PROOF. Let $\mathcal{S}$ denote a CBDH problem solver, who is given $(g^a, g^b, h^a, h^b, h^c, h^{ab}, h^{1/ab})$, aims to compute $\hat{e}(g,h)^{c/ab}$. $\mathcal{S}$ simulates the game for $\mathcal{A}$ as follows.

- $\mathcal{S}$ sets up the game for $\mathcal{A}$ by creating system users with corresponding key pairs $\{(\mathsf{sk}, \mathsf{pk})\}$, where $\mathsf{pk} = h^{\mathsf{sk}}$. $\mathcal{S}$ randomly selects a user and sets up its verification key as $\mathsf{pk} = h^a$. In particular, $\mathcal{S}$ selects an index g and guesses that the forge event will happen to the g-th query on a challenge message m^* and an ephemeral public key $h^{esk} = h^c$. $\mathcal{S}$ also sets up a master public key as $g^{\beta} = g^b$, and completes the remainder of the setup honestly.
- $\mathcal{S}$ simulates hash queries as follows. If $\mathcal{A}$ queries the challenge message m^*, $\mathcal{S}$ returns $g^r \cdot g^{b_i}$ to $\mathcal{A}$, where $(r, b_i) \in \mathbb{Z}_q$ are randomly chosen by $\mathcal{S}$. Otherwise, $\mathcal{S}$ returns g^{b_i} to $\mathcal{A}$ as the response to a hash query on message M.
  If $\mathcal{A}$ issues a signing query on a message m, $\mathcal{S}$ simulates a signature as $\sigma = (\sigma', \sigma'') = (g^{b_i \cdot r_i}, h^{\alpha(b_i \cdot r_i - a \cdot b)/b_i})$, and the corresponding verification key is simulated as $\hat{e}(g^b, \mathsf{pk})^{\alpha}$. Note that the exponents α, b_i are randomly chosen by $\mathcal{S}$.
- When forge event occurs, i.e., $\mathcal{A}$ outputs $\sigma^* = (\sigma^{*'}, \sigma^{*''})$, where $\sigma^{*'} = g^{ab} \cdot g^{c(r+b_i)}$ and $\sigma^{*''} = h^{\alpha \cdot c}$, $\mathcal{S}$ checks whether:
  - the forging event happens at g-th query;
  - the message-signature pair (m^*, σ^*) was not previously simulated by $\mathcal{S}$.
  - the signature is valid $\hat{e}(\sigma^{*'}, h^{\alpha}) = \hat{e}(g^{\beta}, \mathsf{pk})^{\alpha} \cdot \hat{e}(g^r \cdot g^{b_i}, \sigma^{*''})$.

  If all the above conditions hold, $\mathcal{S}$ confirms that it as a successful forgery from $\mathcal{A}$, and extracts the solution $\hat{e}(g,h)^{c/ab} = [\hat{e}(\sigma^{*'}, h^{1/ab})/\hat{e}(g,h)]^{1/(r+b_i)}$ to the CBDH assumption.

□

D.2 Anonymity

THEOREM 24. *The proposed Σ scheme achieves anonymity if the proposed DBDH assumption is held in the asymmetric pairing groups.*

PROOF. Let $\mathcal{S}$ denote a DBDH problem distinguisher, who is given $(g^a, g^b, g^f, g^{ed}, g^{ec_i}, g^{el_i}, \forall i \in [q], h^a, h^b, h^f, h^{ed}, h^{ec_i}, h^{el_i}, \forall i \in [q])$, and aims to distinguish $T_w = g^{ab} \cdot g^{edc_i}$ and $T_{1-w} = g^{fb} \cdot g^{edl_i}$. We add the corresponding instances from group $\mathbb{H}$ for simulating message-signature pairs, and these extra instances are no help in solving the DBDH problem. $\mathcal{S}$ simulates the game for $\mathcal{A}$ as follows.

- Setup: $\mathcal{S}$ sets up the game for $\mathcal{A}$ by creating system users. $\mathcal{S}$ randomly selects two challenge users and sets $\mathsf{pk}_0 = h^a, \mathsf{pk}_1 = h^f$ and generates key pair for other users honestly. $\mathcal{S}$ also sets $g^\beta = g^b$, and computes the remainder of the setup honestly.
- $\mathcal{S}$ simulates user pk_0's signatures as follows.
  - $\mathcal{S}$ simulates a signature as $\sigma = (\sigma', \sigma'') = (T_0, h^{ec_i\alpha})$ on a message m; Note that the randomness esk is implicitly sets as c_i, and α is chosen by $\mathcal{S}$.
  - $\mathcal{S}$ simulates a verification key as $\hat{\mathrm{e}}(g^b, h^a)^\alpha$, and sets $g^d = \mathsf{H}(m)$.
  - $\mathcal{S}$ returns $(m, \sigma, \hat{\mathrm{e}}(g, h)^{ab\alpha})$ to $\mathcal{A}$.

  $\mathcal{S}$ can simulate user pk_1's signature using the same method described above.

Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. If $\mathcal{A}$ guesses the random bit correctly, then $\mathcal{S}$ can break the proposed DBDH assumption. $\square$