Practical and Secure Policy-based Chameleon Hash for Redactable Blockchains

Abstract. Policy-based chameleon hash functions have been widely used in blockchain rewriting systems. They allow anyone to create a mutable transaction associated with an access policy, while an authorized user who possesses sufficient rewriting privileges from a trusted authority satisfying the access policy can rewrite the mutable transaction. However, existing chameleon hash functions lack certain fundamental security guarantees, including forward security and backward security. In this paper, we introduce a new primitive called forward/backward-secure policy-based chameleon hash (FB-PCH for short). We present a practical instantiation. We prove that the proposed scheme achieves forward/backward-secure collision-resistance, and show its practicality through implementation and evaluation analysis.

Keywords: Policy-based Chameleon Hash, Forward/Backward Security, Redactable Blockchain.

1 Introduction

A blockchain is a decentralized, distributed, and often public, digital ledger consisting of blocks designed to record transactions among all network nodes. A key feature of blockchain is that all transactions recorded in the digital ledger are available at each node, and the transactions cannot be modified once they appear in the blockchain. However, mutability is necessary for blockchain applications. First, it can be used to redact/delete illicit content recorded in the blockchain. For example, some users may record certain sensitive information or compromised secret keys in the blockchain. It is necessary to let some authorized users redact/delete illicit content. Second, it can enlarge blockchain's application domains. A user may create a mutable transaction that includes his personal data. Later, the user itself (or an authorized party) is allowed to manage the recorded data. In particular, the "Right to be Forgotten" imposed by GDPR in Europe allows people to have the right to ask for deleting their personal data [2], including those recorded in a blockchain.

Chameleon hash (CH) functions have been widely used in the blockchain rewriting systems. CH is a trapdoor-based hash function, and it enables a trapdoor holder to change any hashed message without changing its chameleon hash output [23]. The state-of-the-art is called policy-based chameleon hash (PCH) [17]. A PCH-based blockchain rewriting system allows anyone to create a mutable transaction with an access policy, and a trapdoor holder can rewrite the hashed message involved in the mutable transaction if her attribute-based trapdoor satisfies the access policy. Essentially, the idea of PCH is analogous to attribute-based access control. We use *modifier* to represent the trapdoor holder, and we use *rewriting privilege* to represent the modifier's attributes-based trapdoor.

Motivation. Blockchain rewriting aims to redact/delete illicit data recorded in the blockchain. The existing solutions require modifiers (e.g., trapdoor holders, regulators, even miners) to enable rewriting. However, the modifiers may abuse their rewriting privileges and keep illicit data in the blockchain. For example, a malicious modifier may refuse to redact the transactions containing illicit data, or she might delete users' data in which there is no illicit content. To address this issue, the first method is to set data regulation rules, such that the miners should pack the transactions that conform to the rule (e.g., GDPR). Nevertheless, it is infeasible for miners to detect certain misbehavior. For instance, a user may insert illicit data into multiple transactions, while the miner cannot effectively check for illicit data from incoming transactions before they are appended into the blockchain [18]. The second method is to provide revocation to the blockchain system [22]. It allows a (semi-)trusted authority (i.e., the party that manages modifiers' rewriting privileges) to revoke the modifiers' rewriting privileges in time due to their malicious behavior. The trusted authority can also redact/delete illicit data inside mutable transactions. But, the revocation mechanism used in [22] is linear in the number of modifiers.

PCH-based blockchain rewriting is desirable in practice. First, it ensures fine-grained access control. By attaching the preferred access policies to mutable transactions, the transaction owner can decide what modifiers might rewrite his generated transactions. Second, it enjoys logarithmic complexity in revocation such as [30]. The revocable PCH-based blockchain rewriting ensures backward security, allowing the trusted authority to periodically revoke some modifiers' rewriting privileges [30]. But, the modifiers' rewriting privileges can be compromised due to key leakage attacks [24]. Therefore, it is necessary to consider forward-secure PCH-based blockchain rewriting, where the attacker holding the present compromised key should not rewrite the mutable transactions generated in the past. The motivation of this work is to make PCH forward-secure and backward-secure simultaneously.

Our Contributions. We introduce a new primitive called forward/backward-secure policy-based chameleon hash (FB-PCH). Specifically, FB-PCH is built from a forward/backward-secure attribute-based encryption (FB-ABE) scheme and a chameleon hash (CH) function. We use FB-ABE to enforce access control, and use CH to ensure blockchain rewriting. The first contribution of this work is to formalize forward/backward-secure semantic security model and collision-resistance model, respectively. To the best of our knowledge, this is the first attempt to formalize the collision-resistance model that supports forward security and backward security altogether for policy-based chameleon hashes.

The second contribution of this work is to construct a *secure* and *practical* FB-ABE scheme. We show an envisioned method to construct FB-ABE, which includes two steps: 1) build a practical revocable ABE (RABE) scheme, and 2) make RABE forward-secure. Many RABE schemes are available in the literature.

For example, some RABE schemes enable non-revoked users to update their decryption keys periodically using the key update information broadcast by the key generation center (KGC) [15, 30]. In other words, KGC has the right to revoke any system users periodically (we call it epoch-based revocation).

The next step is to make the RABE scheme forward-secure. Similarly, several techniques are available to ensure non-interactive forward security, such as hierarchical identity-based encryption (HIBE) and puncturable encryption [12]. We construct FB-ABE from HIBE because: 1) it is suitable for the epoch-based revocation used in the RABE scheme; 2) it supports a non-interactive key update process (i.e., key holder updates its decryption key without any interaction with the KGC); and 3) it keeps ciphertext short (e.g., HIBE with constant-size ciphertexts [11]). Specifically, we apply the key update process used in HIBE to RABE's decryption keys. Thus, the key holder can privately derive new decryption keys from present keys. Such key derivation is a one-way process: a parent decryption key can derive child decryption keys.

The above envisioned method has several issues. First, it is not practical. Specifically, the updated decryption key of RABE is usually associated with a time-based hash function $\mathtt{H}(t)^{r_x}$ [15, 30], but the decryption key of HIBE is based on a q-strong assumption in the form of $F(t)^r$ [11], where $F(t) = g_0 \cdot \prod g_i^{t_i}$, $\{g_0, \cdots\}$ denotes public parameters, $t = t_1 || \cdots || t_i$ denotes a time string, and (r, r_x) denotes the randomness chosen by KGC. If a decryption key contains group elements $\mathtt{H}(t)^{r_x}$ and $F(t)^r$, the corresponding ciphertext should include group elements: $\mathtt{H}(t)^s$ and $F(t)^s$, where s denotes randomness chosen by an encryptor. The extra costs in generating decryption keys and ciphertext are not desirable for blockchain applications. Second, the non-interactive key update and the revocation cannot work together. For example, if we extend a concrete RABE scheme [30] to FB-ABE, the non-interactive key update cannot be achieved because the $\mathtt{H}(t)^{r_x}$-based decryption key is fixed by KGC using randomness r_x. Also, the revocation will not work if we replace the hash function $\mathtt{H}(t)^{r_x}$ involved in the revocation mechanism with $F(t)^{r_x}$. In RABE, the non-revoked key holders should update their decryption keys using the key update information from KGC. But, the $F(t)^{r_x}$-based decryption key enables (either revoked or non-revoked) key holders to update their keys without using the key update information. So, a natural question is: *Can we adapt an attribute-based encryption scheme to guarantee that non-interactive key update and revocation work together without causing heavy overhead?*

Our solution affirmatively answers this question. First, we replace the key update information $\mathtt{H}(t)^{r_x}$ with $F(t)^{r_x}$. That is, the resulting decryption key contains $F(t)^{r+r_x}$, and the resulting ciphertext contains $F(t)^s$ only. Second, we allow the encryption process to include key update information published by KGC. This design can guarantee that the non-interactive key update and the revocation work together. If the key holders derive new decryption keys without using the key update information, they cannot decrypt ciphertext because the randomness (e.g., r'_x) used to derive the new keys differs from the randomness r_x used to generate the key update information. The decryption works if the ciphertext and the non-revoked decryption keys are associated with the same key update information published by KGC. Third, we allow KGC to publish a "limited" key update information: $(g_{t+1}^{r_x}, \cdots, g_{t'}^{r_x})$, where t' denotes a desirable bounded time. This design allows the non-revoked key holders to update their decryption keys privately until the bounded time t' (namely, the "limited" non-interactive key update). We describe this feature as enhanced usability, compared to publishing key update information every period by KGC. Besides, KGC can disable it by withholding the "limited" key update information. To conclude, our solution can save computational and storage costs, ensure non-interactive key update and revocation work together, and increase the usability of FB-ABE. We stress that the proposed FB-ABE scheme is the technical contribution of this work. We summarize our major contributions as follows.

– *Security Models.* We formalize forward/backward-secure semantic model for attribute-based encryption schemes, as well as forward/backward-secure collision-resistance model for policy-based chameleon hash functions.
– *New Primitive.* We introduce a *new* primitive called forward/backward-secure policy-based chameleon hash (FB-PCH). The primitive's unique feature is that it enables the modifiers' trapdoors (or rewriting privileges) to be periodically updated or revoked.
– *Practical Constructions.* We present a *practical* instantiation, and show that our scheme is practical when integrating it into mutable blockchain. To ensure fine-grained, forward and backward secure rewritings in blockchains. We propose a *new* forward/backward-secure attribute-based encryption (FB-ABE) scheme.

1.1 Related Work

The concept of blockchain rewriting was introduced by Ateniese et al. [7]. It allows anyone to create a mutable transaction by hashing a message using a trapdoor-based chameleon hash (CH) function [23]. The trapdoor holder can replace the hashed/recorded message in a chain with a new one without changing the chameleon hash output and breaking the link of the hash-chain. Derler et al. [17] introduced policy-based CH (we call it PCH) to ensure fine-grained blockchain rewritings. Specifically, each mutable transaction is associated with an access policy. The trapdoor holder can rewrite it if her attribute set satisfies the access policy. Later, Tian et al. [29] introduced accountable PCH (we denote it as PCH'). It allows a trusted third party to trace the identity of the trapdoor holder if malicious blockchain rewritings occur.

Deuber et al. [18] take a different approach to ensure blockchain rewritings in the permissionless setting (where anyone can join the network as a node and access the blockchain) compared to the permissioned setting (where certain permission is required to access the blockchain). It allows anyone to broadcast a modification request, and a number of miners vote for it (we call it V-BR). The rewriting is executed in the blockchain if the modification request receives a threshold number of votes. This voting-based approach can be used in permissionless settings such as Bitcoin and Ethereum.

Collision resistance is a fundamental security guarantee for CH. Derler et al. [16] revisited the existing collision-resistance notions in the literature and introduced a new primitive called strongly collision-resistant CH (sCH). Specifically, a mutable transaction includes: 1) a chameleon hash output, which is a public-key encryption ciphertext; and 2) a check string, which is a zero-knowledge (ZK) proof. The decryption key holder can rewrite the mutable transaction by generating a new ZK proof.

Revocable CH has been recently investigated in the literature. Jia et al. [22] introduced a hierarchical revocable CH scheme (they call it sCHRS). Hierarchal revocation means that a semi-trusted party (which called regulator) holding a master trapdoor can revoke the user's rewriting privilege to a mutable transaction by creating a new sub-trapdoor. The user holding the old sub-trapdoor cannot perform rewriting on the mutable transaction. Note that the revocation described in [22] achieves both forward and backward security because the modifier cannot rewrite in both directions once being revoked. But, it is not a PCH-based solution. Recently, Xu et al [30] introduced revocable PCH (RPCH). Their revocation relies on a RABE scheme, such that the trapdoor holder's rewriting privileges can be periodically revoked by the KGC. Note that the revocation described in [30] achieves backward security only.

In this work, we focus on forward/backward-secure PCH. Our solution relies on a new FB-ABE scheme and the standard CH scheme for fine-grained blockchain rewriting. In particular, we adapt the ABE scheme [4] to ensure forward and backward security simultaneously. Since the redactable blockchain solutions are various in the literature, we summarize and compare our work with the aforementioned research works in Table 1. We emphasize that our proposed FB-PCH scheme can also be applied to sanitizable signatures [27]. The benefit is that attackers with the current compromised trapdoor cannot forge the signatures generated in the past.

Table 1. The comparison between various blockchain rewriting solutions.

	CH [7]	PCH [17]	V-BR [18]	sCH[16]	PCH' [29]	RPCH [30]	sCHRS [22]	Ours
CH-based	✓	✓	×	✓	✓	✓	✓	✓
Permissioned	both	perm	perm-less	both	perm	perm	both	perm
Fine-grained	×	✓	×	×	✓	✓	×	✓
Backward-Sec	×	×	×	×	×	✓	✓	✓
Forward-Sec	×	×	×	×	×	×	✓	✓

2 System Overview

In this section, we show how FB-PCH works in a permissioned blockchain rewriting system. The system should include three types of roles: user, modifier and attribute authority (AA). First, we assume at most k users and n modifiers in the system, where $n \leq k$. Second, we assume every user have a key pair $(\texttt{sk}, \texttt{pk})$, and their public keys are known to all other users. Each modifier possesses a set of attributes. Third, the roles can be intersected, such as a user can be a modifier, and/or AA. Every user can act as AA and tag other users with attributes. Last, AA is responsible for making the rule to protect users' privacy (e.g., GDPR) in the permissioned setting. He is also responsible for granting modifiers' rewriting privileges and revoking them in case of misbehave.

Consider a scenario where a user Bob creates a number of time-based mutable transactions: $(Tx_1, Tx_2, Tx_3, \cdots)$. For example, Tx_1 is created at time t_1, and Tx_2 is created at time t_2 with the condition that $t_1 < t_2$. A modifier Alice is granted a rewriting privilege associated with time t_2 from an attribute authority (AA) via a secure channel (horizontal solid arrow in Figure 1). Thus, Alice can rewrite the mutable transaction Tx_2 (dashed box in Figure 1). But, Alice cannot rewrite Tx_1. Later, AA may publicly revoke Alice's rewriting privilege at time t_3 (horizontal dashed arrow in Figure 1). FB-PCH is suitable in this scenario for the following reasons: 1) AA can revoke Alice's rewriting privilege at time t_3 so that she can no longer rewrite Bob's mutable transactions such as Tx_3. This results in backward-secure rewriting. 2) If an attacker compromised Alice's rewriting privilege associated with time t_2, the attacker cannot rewrite the mutable transaction Tx_1. This results in forward-secure rewriting. 3) Our solution ensures fine-grained and controlled rewriting in blockchain. Specifically, user Bob decides the specific time period that the modifier can rewrite his mutable transactions. AA decides whether Alice could rewrite Bob's mutable transactions or not via granting/revoking rewriting privileges at some time.

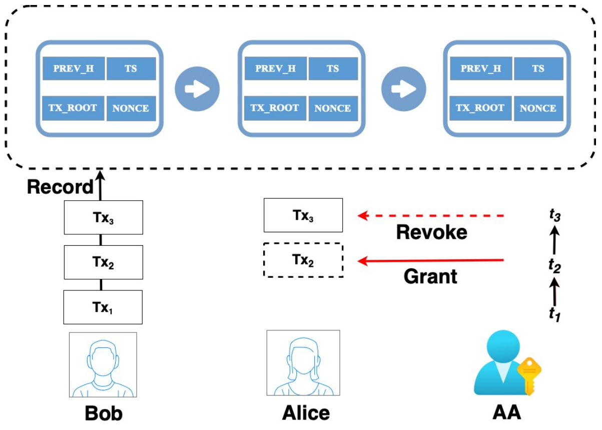

Fig. 1. The structure of our blockchain rewriting system.

3 Preliminaries

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

3.1 Complexity Assumptions

Bilinear Maps. We define the output of group generation as $(q, g, h, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \widehat{\mathsf{e}}) \leftarrow \mathsf{GroupGen}(1^\lambda)$, where q is a prime number, g/h is a generator of group $\mathbb{G}/\mathbb{H}$. Note that $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$ are cyclic groups of order q, and $\widehat{\mathsf{e}} : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$ is an asymmetric bilinear map. We introduce a new composite assumption, which is based on a decisional linear (DLIN) problem [4] and a weak bilinear Diffie-Hellman inversion (wBDHI) problem [11]. We use this assumption to prove the semantic security of the proposed FB-ABE scheme.

Definition 1. *Given a security parameter λ, we define the following distribution for $\mathtt{Adv}_{\mathcal{A}}^{Com}$:*

$$\begin{aligned}
&|\Pr[\mathtt{Adv}(1^\lambda, \mathsf{par}, D, T_0) = 1] - \\
&\Pr[\mathtt{Adv}(1^\lambda, \mathsf{par}, D, T_1) = 1]|, \textit{where} \\
&\mathsf{par} = (q, g, h, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \widehat{\mathsf{e}}) \leftarrow \mathsf{GroupGen}(1^\lambda), \\
&a_1, a_2, \alpha_1, \alpha_2 \leftarrow \mathbb{Z}_q^*, s \leftarrow \mathbb{Z}_q; D = (g^{a_1}, g^{a_2}, h^{a_1}, h^{a_2}, \\
&g^{a_1 \cdot \alpha_1^{\ell+1}}, g^{a_2 \cdot \alpha_2^{\ell+1}}, h^{a_1 \cdot \alpha_1^{\ell+1}}, h^{a_2 \cdot \alpha_2^{\ell+1}}, \\
&\{g^{\alpha_{1/2}}, g^{\alpha_{1/2}^2}, \cdots, g^{\alpha_{1/2}^\ell}\}, \{h^{\alpha_{1/2}}, h^{\alpha_{1/2}^2}, \cdots, h^{\alpha_{1/2}^\ell}\}); \\
&T_0 = (g^{\alpha_1^{\ell+1} + \alpha_2^{\ell+1}}, h^{\alpha_1^{\ell+1} + \alpha_2^{\ell+1}}), T_1 = (g^s, h^s).
\end{aligned}$$

The composite assumption is secure if $\mathtt{Adv}_{\mathcal{A}}^{Com}(\lambda)$ is negligible in λ.

We present the security analysis of the proposed composite assumption. We prove its security in group $\mathbb{G}$, and one can easily add group elements from $\mathbb{H}$ to the following theorem.

Theorem 1. *Let $(\epsilon_1, \epsilon_2, \epsilon_T) : \mathbb{Z}_q \to \{0,1\}^*$ be three random encodings (injective functions) where $\mathbb{Z}_q$ is a prime field, 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 $(a, b, c) \xleftarrow{R} \mathbb{Z}_q$ and encodings $\epsilon_1, \epsilon_2, \epsilon_T$ are randomly chosen, we define the advantage of the adversary in solving the composite assumption with at most $\mathcal{Q}$ queries to the group operation oracles $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_T$ and the bilinear pairing $\widehat{\mathsf{e}}$ as*

$$\begin{aligned}
|\mathtt{Adv}_{\mathcal{A}}^{Com}(\lambda) &= \Pr[\mathcal{A}(q, \epsilon_1(1), \epsilon_1(a_1), \epsilon_1(a_2), \epsilon_1(a_1 \cdot \alpha_1^{\ell+1}), \epsilon_1(a_2 \cdot \alpha_2^{\ell+1}), \\
&\{\epsilon_1(\alpha_1), \cdots, \epsilon_1(\alpha_1^\ell)\}, \{\epsilon_1(\alpha_2), \cdots, \epsilon_1(\alpha_2^\ell)\}, \epsilon_1(s), \epsilon_1(t_0), \epsilon_1(t_1), \epsilon_2(1), \epsilon_T(1)) \\
&= w : (a_1, a_2, \alpha_1, \alpha_2, s \xleftarrow{R} \mathbb{Z}_q, w \in (0,1), t_w = (\alpha_1^\ell + \alpha_2^\ell), t_{1-w} = s)] - 1/2| \\
&\leq \frac{2(\ell+2)(2\ell+9)^2}{q}
\end{aligned}$$

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 $p_i \in \mathbb{Z}_q[A_1, A_2, AL_1, AL_2, S, T_0, T_1]$ are 7-variate polynomials over $\mathbb{Z}_q$, such that $p_0 = 1, p_1 = A_1, p_2 = A_2, p_3 = A_1 \cdot AL_1^{\ell+1}, p_4 = A_2 \cdot AL_2^{\ell+1}, \{p_u = AL_1^u\}_{u=1}^{\ell}, \{p_v = AL_2^v\}_{v=1}^{\ell}, p_{2\ell+5} = T_0, p_{2\ell+6} = T_1$. $\mathcal{S}$ also generates $q_0 = 1, t_0 = 1$. Then, $\mathcal{S}$ sets three lists as $L_1 = \{(p_i, \epsilon_{1,i})\}_{i=0}^{2\ell+6}, L_2 = (q_0, \epsilon_{2,0}), L_T = (t_0, \epsilon_{T,0})$, where $(\{\epsilon_{1,i}\}_{i=0}^{2\ell+6} \in \{0,1\}^*, \{\epsilon_{2,0}\} \in \{0,1\}^*, \{\epsilon_{T,0}\} \in \{0,1\}^*)$ are arbitrary distinct strings.

At the beginning of the game, $\mathcal{S}$ sends the encoding strings $(\{\epsilon_{1,i}\}_{i=0,\cdots,2\ell+6}, \epsilon_{2,0}, \epsilon_{T,0})$ to $\mathcal{A}$. Next, $\mathcal{S}$ simulates the group operation oracles $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_T$ and the bilinear pairing $\widehat{\mathsf{e}}$ as follows. We assume that all requested operands are obtained from $\mathcal{S}$.

- $\mathcal{O}_1$: 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., $\mathcal{O}_2, \mathcal{O}_T$) are treated similarly.
- $\widehat{\mathsf{e}}$: 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 $\mathcal{Q}$ times of corresponding oracles, $\mathcal{A}$ terminates and outputs a guess $b' = \{0,1\}$. At this point, $\mathcal{S}$ chooses random $a_1, a_2, \alpha_1, \alpha_2, s \in \mathbb{Z}_q$, generates $t_b = (\alpha_1^{\ell+1} + \alpha_2^{\ell+1})$ and $t_{1-b} = s$. $\mathcal{S}$ sets $A_1 = a_1, A_2 = a_2, AL_1 = \alpha_1, AL_2 = \alpha_2, S = s, T_0 = t_b, T_1 = t_{1-b}$. The simulation is perfect unless the `abort` event happens. Thus, we bound the probability of event `abort` by analyzing the following cases:

1. $p_i(a_1, a_2, \alpha_1, \alpha_2, s, t_0, t_1) = p_j(a_1, a_2, \alpha_1, \alpha_2, s, t_0, t_1)$: The polynomial $p_i \neq p_j$ due to the construction method of L_1, and $(p_i - p_j)(a_1, \cdots)$ is a non-zero polynomial of degree $[0, 1, \cdots, \ell+2]$ (ℓ+2 is produced by $A_1 \cdot AL_1^{\ell+1}$). The maximum degree of $A_1 \cdot AL_1^{\ell+1}(p_i - p_j)(a_1, \cdots)$ is ℓ+2. By using Lemma 1 in [28], we have $\Pr[(p_i - p_j)(a_1, \cdots) = 0] \leq \frac{\ell+2}{q}$ and thus $\Pr[p_i(a_1, \cdots) = p_j(a_1, \cdots)] \leq \frac{\ell+2}{q}$. Therefore, we have the `abort` probability is $\Pr[\mathsf{abort}_1] \leq \frac{\ell+2}{q}$.
2. $q_i(a_1, \cdots) = q_j(a_1, \cdots)$: The polynomial $q_i \neq q_j$ due to the construction method of L_2, and $(q_i - q_j)(a_1, \cdots)$ is a non-zero polynomial of degree 0. The maximum degree is "0" since the list L_2 contains a single string $\epsilon_{(2,0)}$ only (note that we do not include group elements in $\mathbb{H}$). Therefore, the `abort` probability is "0".

3. $t_i(a_1, \cdots) = t_j(a_1, \cdots)$: The polynomial $t_i \neq t_j$ due to the construction method of L_3, and $(t_i - t_j)(a_1, \cdots)$ is a non-zero polynomial of degree $[0, 1, \cdots, \ell+2]$. The maximum degree of $(A_1 \cdot AL_1^{\ell+1}(t_i - t_j)(a_1, \cdots)$ is also $\ell+2$. Therefore, we have $\Pr[(t_i - t_j)(a_1, \cdots) = 0] \leq \frac{\ell+2}{q}$ and thus $\Pr[t_i(a_1, \cdots) = t_j(a_1, \cdots)] \leq \frac{\ell+2}{q}$.

By summing over all valid pairs (i, j) in each case (i.e., at most $2\binom{2\ell+9}{2}$ pairs), we have the abort probability is

$$\Pr[\mathsf{abort}] = \Pr[\mathsf{abort}_1] + \Pr[\mathsf{abort}_2] + \Pr[\mathsf{abort}_3] \leq 2\binom{2\ell+9}{2} \cdot (\frac{\ell+2}{q} + \frac{\ell+2}{q})$$
$$\leq \frac{2(\ell+2)(2\ell+9)^2}{q}.$$

3.2 Chameleon Hash Function

A chameleon hash function consists of the following algorithms [23].

– KeyGen: It takes a security parameter λ as input, outputs a chameleon key pair $(\mathtt{sk}, \mathtt{pk})$.
– Hash: It takes the chameleon public key $\mathtt{pk}$, and a message $m \in \mathcal{M}$ as input, outputs a chameleon hash h, randomness r. Note that $\mathcal{M} = \{0,1\}^*$ denotes a general message space.
– Verify: It takes the chameleon public key $\mathtt{pk}$, a message m, a chameleon hash h and randomness r as input, outputs a bit $b \in \{0,1\}$.
– Adapt: It takes the chameleon secret key $\mathtt{sk}$, messages m, m', chameleon hash h and randomness r as input, outputs a new randomness r'.

Chameleon hash function should ensure indistinguishability and collision-resistance. Indistinguishability means that the randomness associated with a chameleon hash does not reveal whether it was obtained from directly hashing or computed using the chameleon secret key. The collision-resistance means that adversaries cannot find two pairs (m, r) and (m', r') that map to the same chameleon hash h. Since the definition of indistinguishability and collision-resistance are various in the literature [7, 13, 16, 23], we focus on a standard one [13] to prove the security of the proposed FB-PCH. For constructing FB-PCH, we rely on the discrete logarithm (DL)-based CH. Specifically, CH is associated with a public key $\mathtt{pk} = g^{\mathtt{sk}}$. For hashing a message m, $\mathsf{Hash}(\mathtt{pk}, m)$ outputs (h, r), where $h = g^m \cdot \mathtt{pk}^r$. In Verify, the hash value h is verified via $h \overset{?}{=} g^m \cdot \mathtt{pk}^r$ given $(\mathtt{pk}, m, r)$. In Adapt, we compute $r' = r + (m - m')/\mathtt{sk}$.

3.3 Forward Security

We present a non-interactive forward security technique, which is used in the hierarchical identity-based encryption scheme (HIBE)[11]. Essentially, forward security allows efficient key updating while preventing the derivation of past keys from present and future keys. We mainly show *how to generate and update keys* because they determine the forward security. Below, we present a secret key $\mathtt{sk}_t$ at time $t = t_1||t_2||\cdots||t_i \in \mathcal{T}$,

$$\mathtt{sk}_t = (h^r, \underline{g^{\mathtt{sk}} \cdot F(t)^r}, g_{i+1}^r, \cdots, g_\ell^r)$$

where the maximum number of time slots is $\mathcal{T} = 2^\ell - 1$, ℓ indicates a binary tree's depth, r is randomness, $F(t) = g_0 \prod g_i^{t_i}$ is a public function on time t, $(g, g_0, g_1, \cdots, g_\ell)$ denotes the public parameters, and $F(t)^r = (g_0 \cdot g_1^{t_1} \cdots g_i^{t_i})^r$. Note that the public key $h^{\mathtt{sk}}$ never changes. We use $\{1, 2\}$-string to represent time period. For example, the string $(\epsilon, 1, 11, 111, 112, 12, \cdots, 222)$ is corresponding to time period $(1, 2, 3, 4, 5, 6, \cdots, 15)$, where ϵ denotes the first time period. To derive a new key $\mathtt{sk}_{t'}$ at time $t' = t||t_{i+1}$ from $\mathtt{sk}_t$ (note that t is a prefix of t'), the secret key holder performs the following operation on the above underlined part

$$\begin{aligned} g^{\mathtt{sk}} \cdot F(t')^{r+r'} &= g^{\mathtt{sk}} \cdot (g_0 \prod g_{i+1}^{t_{i+1}})^{r+r'} \\ &= g^{\mathtt{sk}} \cdot (g_0 \prod g_i^{t_i})^r \cdot g_{i+1}^{r \cdot t_{i+1}} \cdot (g_0 \prod g_{i+1}^{t_{i+1}})^{r'} \end{aligned}$$

where r' is a new randomness. For the first and other elements of the new secret key $\mathtt{sk}_{t'}$, the key holder can easily update them by multiplying $g^{r'}, g_{i+2}^{r'}, \cdots, g_\ell^{r'}$, respectively. So, the new secret key is shown as follows

$$\mathtt{sk}_{t'} = (h^{r+r'}, g^{\mathtt{sk}} \cdot F(t')^{r+r'}, g_{i+2}^{r+r'}, \cdots, g_\ell^{r+r'})$$

After each key update, the key holder erases the old secret key to ensure forward security. The above approach shows that each key update requires a new randomness to break the link between the old and new secret keys. But, the complexity of the key update is linear in the number of time periods $\mathcal{T}$. To achieve forward security with logarithmic complexity $\mathcal{O}(\log(\mathcal{T})^2)$, one can use the binary tree-based approach described in [14, 19]. We skip the description of key update with logarithmic complexity as it is not our major contribution. We remark that punctual encryption [21] is an alternative approach to ensure non-interactive forward security. We leave the forward-secure constructions built from punctual encryption as our future work.

3.4 Epoch-based Revocation

We rely on an epoch-based revocation mechanism [10] to revoke the modifiers' rewriting privileges. The epoch-based revocation leverages a binary tree. It allows non-revoked users to update their secret keys (or credentials) simultaneously within a time period (i.e., epoch). By contrast, a trivial method would require the revocation authority to work linear in the number of non-revoked users.

But, this method does not scale well as the number of users grows. The epoch-based revocation mechanism is practical because: 1) the complexity of rewriting privilege updates is logarithmic $\mathcal{O}(\log(k))$, where k denotes the number of users. 2) the authority periodically publishes the updated information without any interaction. The epoch-based revocation relies on a node selection algorithm $\mathsf{KUNodes}$. It defines a binary tree with k leaves, where each leaf is associated with a user. It also denotes the root node of the tree BT as root. If θ is a leaf node, let $\mathsf{Path}(\theta)$ be the set of nodes on the path from θ to root (note that both θ and root are inclusive). The $\mathsf{KUNodes}$ algorithm takes the binary tree BT, the revocation list rl, and a revocation time t as input, performs the following operations: it first marks all ancestors of users that were revoked by revocation time t as revoked nodes. Then, it outputs all the non-revoked children of revoked nodes. We briefly show a description of $\mathsf{KUNodes}$ in Figure 2, the formal definition can be found in [10].

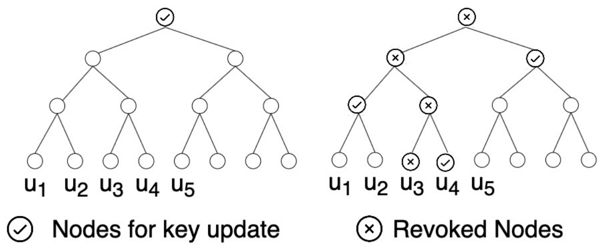

Fig. 2. A pictorial description of node selection algorithm $\mathsf{KUNodes}$. Left sub-figure has no revoked users, and right sub-figure shows a user u_3 is revoked.

4 The Proposed FB-ABE

In this section, we present the definition and security model for forward/backward-secure attribute-based encryption. Then, we show the proposed scheme and its security analysis.

4.1 Definition and Model

The forward/backward-secure attribute-based encryption (FB-ABE) consists of the following algorithms. This definition includes two types of key updates: One is controlled by the key generation center (KGC) for backward security, the other one is decided by the decryption key holder for forward security. Let $\mathcal{U}$ be an attribute universe, $\mathcal{I}$ be an identity space, and $\mathcal{T}$ be a time space.

– $\mathsf{Setup}(1^\lambda, k)$: KGC takes a security parameter λ and a number of users k as input, outputs a master key pair $(\mathtt{msk}, \mathtt{mpk})$, a revocation list rl (initially empty) and a state st.

- $\mathsf{KeyGen}(\mathtt{msk}, id, \delta, st)$: KGC takes the master secret key $\mathtt{msk}$, a user's identity $id \in \mathcal{I}$, a set of attributes $\delta \subseteq \mathcal{U}$, and a state st as input, outputs an initial decryption key $\mathtt{sk}_{(\delta,t)}$ for user id associated with the attribute set δ and a time $t \in \mathcal{T}$.
- $\mathsf{KeyUp_0}(\mathtt{msk}, t, rl, st)$: KGC takes the master secret key $\mathtt{msk}$, a key update time t, a revocation list rl, and a state st as input, outputs a key update information $\mathtt{sk}_t$.
- $\mathsf{KeyUp_1}(\mathtt{sk}_{(\delta,t)}, t')$: The user id takes its decryption key $\mathtt{sk}_{(\delta,t)}$ and a new time t' as input, outputs a new decryption key $\mathtt{sk}_{(\delta,t')}$, where $t \leq t'$.
- $\mathsf{DK}(\mathtt{sk}_{(\delta,t)}, \mathtt{sk}_t)$: The user id takes its decryption key $\mathtt{sk}_{(\delta,t)}$ and a key update $\mathtt{sk}_t$ as input, outputs a new decryption key $\mathtt{sk}_{(\delta,t)}$ or $\perp$ (i.e., the user id at time t was revoked).
- $\mathsf{Enc}(\mathtt{mpk}, \mathtt{sk}_t, m, \Lambda, t')$: It takes the master public key $\mathtt{mpk}$, the key update information $\mathtt{sk}_t$, a message m, an access policy Λ, and a time t' as input, outputs a ciphertext C.
- $\mathsf{Dec}(\mathtt{sk}_{(\delta,t)}, C)$: It takes a ciphertext C, and the decryption key $\mathtt{sk}_{(\delta,t)}$ as input, outputs the message m if $1 = \Lambda(\delta)$ (i.e., the attribute set δ satisfies the access policy Λ) and $t = t'$.
- $\mathsf{Revoke}(id, t, rl, st)$: KGC takes a user's identity $id \in \mathcal{I}$, a revocation time $t \in \mathcal{T}$, the present revocation list rl, and a state st as input, outputs an updated revocation list rl.

Correctness and Security. The FB-ABE is *correct* if for all security parameters λ, for all $(\mathtt{msk}, \mathtt{mpk}), rl, st \leftarrow \mathsf{Setup}(1^\lambda, k)$, for all $\delta \in \mathcal{U}$, for all non-revoked $id \in \mathcal{I}$, for all $\mathtt{sk}_{(\delta,t)} \leftarrow \mathsf{KeyGen}(\mathtt{msk}, id, \delta, st)$, for all $t \in \mathcal{T}$, for all $\mathtt{sk}_t \leftarrow \mathsf{KeyUp_0}(\mathtt{msk}, t, rl, st)$, for all $t' \in \mathcal{T}$, for all $\mathtt{sk}_{(\delta,t')} \leftarrow \mathsf{KeyUp_1}(\mathtt{sk}_{(\delta,t)}, t')$, for all $\mathtt{sk}_{(\delta,t)} \leftarrow \mathsf{DK}(\mathtt{sk}_{(\delta,t)}, \mathtt{sk}_t)$, for all $m \in \mathcal{M}$, for all $\delta \in \Lambda$, for all $C \leftarrow \mathsf{Enc}(\mathtt{mpk}, \mathtt{sk}_t, m, \Lambda, t')$, for all $t = t'$, we have $m \leftarrow \mathsf{Dec}(\mathtt{sk}_{(\delta,t)}, C)$.

Now, we focus on the forward/backward-secure semantic security of FB-ABE. Informally, an attacker cannot distinguish between encryption of m_0 and m_1 under an access policy Λ^* in the past time period such as t^*, even if the attacker is given the user's present decryption keys. The access policy Λ^* is chosen by the attacker during the challenge phase. The formal security game between a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ and a challenger $\mathcal{S}$ is shown as follows.

- $\mathcal{S}$ sets up the game by creating k users with the corresponding initial decryption keys $\{\mathtt{sk}_{(\delta,0)}\} \leftarrow \mathsf{KeyGen}(\mathtt{msk}, id, \delta, st)$, where $(\mathtt{msk}, \mathtt{mpk}), rl, st \leftarrow \mathsf{Setup}(1^\lambda, k)$. For each user id, $\mathcal{S}$ can update user's initial decryption key to $\{\mathtt{sk}_{(\delta,t)}\}_{t=1}^{\mathcal{T}}$. Eventually, $\mathcal{S}$ chooses a random bit b, and returns all public parameters $(\mathtt{mpk}, rl, st)$ to $\mathcal{A}$.
- $\mathcal{A}$ can make the following queries to $\mathcal{S}$ during the training phase.
  - KeyUp. If $\mathcal{A}$ issues a key update query with respect to a user id at time $t \in \mathcal{T}$, $\mathcal{S}$ updates the user's decryption key to $\mathtt{sk}_{(\delta,t+1)}$ by running $\mathsf{KeyUp_1}$ and increases t. $\mathcal{S}$ also publishes a key update information at time t by running $\mathsf{KeyUp_0}$.

- Revoke. If $\mathcal{A}$ issues a revoke query with respect to a user id at time t, $\mathcal{S}$ updates the revocation list rl by including user's id, and returns the updated state st at time t to $\mathcal{A}$.
- BreakIn. If $\mathcal{A}$ issues a break in query with respect to a user id at time $\bar{t}$, $\mathcal{S}$ returns the corresponding decryption key $\mathtt{sk}_{(\delta,\bar{t})}$ to $\mathcal{A}$. This query can be issued once for each user, and after this query, $\mathcal{A}$ cannot make further key update or decryption queries with respect to that user.
- Dec. If $\mathcal{A}$ issues a decryption query on a ciphertext C, $\mathcal{S}$ returns the encrypted message m to $\mathcal{A}$.

– In the challenge phase, $\mathcal{A}$ sends two challenge messages (m_0, m_1), an access policy Λ^*, and a time t^* to $\mathcal{S}$, $\mathcal{S}$ returns a challenge ciphertext $C^* \leftarrow \mathsf{Enc}(\mathtt{mpk}, \mathtt{sk}_{t^*}, m_b, \Lambda^*, t^*)$ to $\mathcal{A}$. In this game, $\mathcal{A}$ must follow certain restrictions for any user id^* whose attribute set satisfies the access policy Λ^*.

- The key update and revoke oracles should be queried in a non-decreasing order of time.
- If the break in oracle is queried on user id^* at or before time t^*, the revoke oracle must be queried on (id^*, t) for any $t \leq t^*$.
- If user id^* is not revoked at or before time t^*, the break in query cannot be previously queried on (id^*, t^*). But, the break in oracle can be queried on $(id^*, \bar{t})$, such that $t^* < \bar{t}$.
- The decryption oracle cannot be queried on C^*.

We define the advantage of $\mathcal{A}$ in the above game as

$$\mathtt{Adv}_{\mathcal{A}}(\lambda) = |\Pr[\mathcal{S} \to 1] - 1/2|.$$

Definition 2. *A FB-ABE scheme is semantically secure if for any PPT $\mathcal{A}$, $\mathtt{Adv}_{\mathcal{A}}(\lambda)$ is negligible in λ.*

4.2 The Proposed Construction

Design Rationale. Attribute-based encryption schemes that support key delegation can guarantee forward security [9, 20]. The decryption key holders in such schemes can generate new (restrictive) keys by re-randomizing their given decryption keys. Sahai et al. [26] introduced ciphertext delegation for ABE. It allows an (untrusted) server to update ciphertexts to prevent revoked users from decrypting data encrypted in the past. Essentially, they apply forward-secure encryption [14] to ciphertext management. Our construction applies the same forward-secure technique [14] to update users' decryption keys. Our construction also relies on an adaption of revocation mechanism [10], which was used to construct the existing RABE schemes [15, 30**?**].

We construct our FB-ABE scheme from an ABE scheme [4], a non-interactive forward security technique [11], and an epoch-based revocation mechanism [10]. We show three key steps to construct FB-ABE. The first step is to apply the non-interactive key update process described in [11] to ABE scheme' decryption keys. For example, a key holder can update a decryption key $\mathtt{sk}_{(\delta,t)}$ to $\mathtt{sk}_{(\delta,t')}$, where $t \leq t'$. The second step is to apply the same key update process to the epoch-based revocation mechanism. Specifically, KGC publishes key update information $\mathtt{sk}_t = (g_x \cdot F(t)^{r_x}, h^{r_x}, g_{t+1}^{r_x}, \cdots, g_{t'}^{r_x})$, where g_x is a secret value, r_x is randomness, t is revocation time and t' is the desirable bounded time. As a result, the key holders can privately update their keys until time t'. The third step is to include key update information h^{r_x} to the encryption process. For example, an encryptor generates a ciphertext including an element $h^{r_x \cdot s}$, where s is the randomness chosen by the encryptor. In this case, a non-revoked key holder should use the key update information $g_x \cdot F(t)^{r_x}$ to derive valid decryption keys for decrypting the ciphertext.

Let $\mathsf{G} : \mathbb{G}_T \to \{0,1\}^*$ be a pseudo-random generator (PRG). Let $\mathtt{H} : \{0,1\}^* \to \mathbb{G}$ and $\mathtt{H}_q : \{0,1\}^* \to \mathbb{Z}_q$ be two hash functions. The hash function $\mathtt{H}$ considers two types of inputs: (y_1, y_2, y_3) and (y'_1, y_2, y_3), where y_1 is an arbitrary string (e.g., an attribute value), y'_1 is a positive integer, $y_2 \in \{1,2,3\}$ and $y_3 \in \{1,2\}$. We use $y_1 y_2 y_3$ and $0y'_1 y_2 y_3$ to represent two types of hash inputs. Let $\widehat{\mathsf{e}} : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$ be an asymmetric bilinear pairing.

– $\mathsf{Setup}(1^\lambda, k)$: KGC takes a security parameter λ as input, outputs a master public key $\mathtt{mpk} = (g, h, H_1, H_2, T_1, T_2, \{g_0, g_1, \cdots g_\ell\})$, and a master secret key $\mathtt{msk} = (a_1, a_2, b_1, b_2, g^{d_1}, g^{d_2}, g^{d_3})$, where $(a_1, a_2, b_1, b_2) \in \mathbb{Z}_q^*$, $(d_1, d_2, d_3) \in \mathbb{Z}_q$, $H_1 = h^{a_1}, H_2 = h^{a_2}$, $T_1 = \widehat{\mathsf{e}}(g,h)^{d_1 \cdot a_1 + d_3}, T_2 = \widehat{\mathsf{e}}(g,h)^{d_2 \cdot a_2 + d_3}$. KGC also publishes a revocation list $rl = \emptyset$ and a binary tree with depth ℓ and state $st = \mathsf{BT}$. We define $F(t) = g_0 \cdot \prod g_i^{t_i} \in \mathbb{G}$ for time $t = t_1 || t_2 || \cdots || t_i$, and define $F(t, id) = g_0 \cdot \prod g_i^{t_i} \cdot g_\ell^{\mathtt{H}_q(id)} \in \mathbb{G}$ for time t and identity id.
– $\mathsf{KeyGen}(\mathtt{msk}, id, \delta, st)$: KGC takes a master secret key $\mathtt{msk}$, a user's identity id, a set of attributes δ, and a state st as input, outputs an initial decryption key $\mathtt{sk}_{(\delta,t)}$ for user id at time t. Specifically, KGC performs the following operations
  1. pick $(r_1, r_2) \in \mathbb{Z}_q^*$, and compute $\mathtt{sk}_0 = (h^{b_1 \cdot r_1}, h^{b_2 \cdot r_2}, h^r)$, where $r = r_1 + r_2$.
  2. pick $\sigma_y \in \mathbb{Z}_q$, for all $y \in \delta$ and $y_3 = \{1,2\}$, compute $\mathtt{sk}_{(y,y_3)} = \mathtt{H}(y1y_3)^{b_1 \cdot r_1 / a_{y_3}} \cdot \mathtt{H}(y2y_3)^{b_2 \cdot r_2 / a_{y_3}} \cdot \mathtt{H}(y3y_3)^{\frac{r}{a_{y_3}}} \cdot g^{\sigma_y / a_{y_3}}$, $\mathtt{sk}_{(y,3)} = g^{-\sigma_y}$, and set $\mathtt{sk}_y = (\mathtt{sk}_{(y,1)}, \mathtt{sk}_{(y,2)}, \mathtt{sk}_{(y,3)})$.
  3. pick $\sigma' \in \mathbb{Z}_q$, for $y_3 = \{1,2\}$, compute $\mathtt{sk}'_{y_3} = g^{d_{y_3}} \cdot \mathtt{H}(011y_3)^{b_1 \cdot r_1 / a_{y_3}} \cdot \mathtt{H}(012y_3)^{b_2 \cdot r_2 / a_{y_3}} \cdot \mathtt{H}(013y_3)^{\frac{r}{a_{y_3}}} \cdot g^{\sigma' / a_{y_3}}$. KGC chooses an unassigned leaf θ and stores the user's identity id at node θ. For each node $x \in \mathsf{Path}(id)$, KGC performs the following operations to compute $\mathtt{sk}'_3$.
     (a) fetch a secret value g_x from the node x. If x is not defined, KGC chooses $g_x \in \mathbb{G}$, and stores the secret value g_x in the node x.
     (b) compute $\mathtt{sk}'_3 = \frac{g^{d_3}}{g_x} \cdot g^{-\sigma'} \cdot F(t, id)^r$.

     KGC computes $\mathtt{sk}'_4 = \{g_{i+1}^r, \cdots, g_\ell^r\}$, and sets $\mathtt{sk}' = (\mathtt{sk}'_1, \mathtt{sk}'_2, \{x, \mathtt{sk}'_3\}_{x \in \mathsf{Path}(id)}, \mathtt{sk}'_4)$.

  Eventually, KGC outputs the initial decryption key $\mathtt{sk}_{(\delta,t)} = (\mathtt{sk}_0, \{\mathtt{sk}_y\}_{y \in \delta}, \mathtt{sk}')$. KGC also updates the binary tree BT and publishes an updated state st.
– $\mathsf{KeyUp}_0(\mathtt{msk}, \mathsf{t}, \mathsf{rl}, \mathsf{st})$: KGC takes a master secret key $\mathtt{msk}$, a revocation time t, a revocation list rl, and a state $st = \mathsf{BT}$ as input. For each node $x \in \mathsf{KUNodes}(\mathsf{BT}, rl, t)$, KGC performs the following operations

1. fetch a secret value g_x from the node x. If x is not defined, KGC chooses $g_x \in \mathbb{G}$ and stores the secret value g_x in the node x.
2. pick $r_x \in \mathbb{Z}_q$, and compute $\mathtt{sk}_{t_x} = (\mathtt{sk}_{(t,0)}, \mathtt{sk}_{(t,1)}, \mathtt{sk}_{(t,2)}) = (g_x \cdot F(t)^{r_x}, h^{r_x}, g_{i+1}^{r_x}, \cdots, g_\ell^{r_x})$.
3. publish key update information $\mathtt{sk}_t = \{x, \mathtt{sk}_{t_x}\}_{x \in \mathsf{KUNodes(BT,rl,t)}}$.

KGC controls the "limited" key update information via withholding or releasing the value $\mathtt{sk}_{(t,2)}$. KGC can choose any index $i \in \{1, \cdots, \ell\}$, and publish $\mathtt{sk}_{(t,2)} = (g_{i+1}^{r_x}, \cdots, g_\ell^{r_x})$.

– $\mathsf{KeyUp_1}(\mathtt{sk}_{(\delta,t)}, t')$: The user with $\mathtt{sk}_{(\delta,t)}$ performs the following operations
1. pick $r' \in \mathbb{Z}_q$, and update $\mathtt{sk}_0 = (h^{b_1 \cdot r_1}, h^{b_2 \cdot r_2}, h^r, h^{r+r'})$.
2. update $\mathtt{sk}'_3$ for time $t' = t || t_{i+1}$ from $\mathtt{sk}'_4 = \{g_{i+1}^r, \cdots, g_\ell^r\}$ as follows

$$\begin{aligned}\mathtt{sk}'_3 &= \frac{g^{d_3}}{g_x} \cdot g^{-\sigma'} \cdot F(t)^r \cdot g_{i+1}^{r \cdot t_{i+1}} \cdot F(t')^{r'} \\ &= \frac{g^{d_3}}{g_x} \cdot g^{-\sigma'} \cdot F(t')^r \cdot F(t')^{r'} \\ &= \frac{g^{d_3}}{g_x} \cdot g^{-\sigma'} \cdot F(t')^{r+r'},\end{aligned}$$

where $F(t') = g_0 \cdot \prod g_{i+1}^{t_{i+1}} \in \mathbb{G}$, and t is a prefix of t'. The user id can modify the $F(t, id)^r$-based $\mathtt{sk}'_3$ to a $F(t)^r$-based one using $g_\ell^{r \cdot \mathtt{H}_q(id)}$. For each $x \in \mathsf{Path}(id)$, the user updates $\mathtt{sk}'_3$ using the same method described above with randomness r'.
3. update $\mathtt{sk}'_4 = \{g_{i+2}^{r+r'}, \cdots, g_\ell^{r+r'}\}$, and set $\mathtt{sk}' = (\mathtt{sk}'_1, \mathtt{sk}'_2, \{x, \mathtt{sk}'_3\}_{x \in \mathsf{Path}(id)}, \mathtt{sk}'_4)$.

We use $\mathtt{sk}_{(\delta,t')}$ to represent the resulting decryption key at time t' after key update.

– $\mathsf{DK}(\mathtt{sk}_{(\delta,\mathrm{t})}, \mathtt{sk}_\mathrm{t})$: The user with $\mathtt{sk}_{(\delta,t)}$ and a key update information $\mathtt{sk}_t$ performs the following operations. The user parses $\mathtt{sk}_{(\delta,t)} = (\mathtt{sk}_0, \{\mathtt{sk}_y\}_{y \in \delta}, \mathtt{sk}')$, where $\{i, \mathtt{sk}'_3\}_{i \in \mathsf{I}} \in \mathtt{sk}'$, and $\mathtt{sk}_t = \{j, \mathtt{sk}_{t_j}\}_{j \in \mathsf{J}}$, for some set of nodes I, J.
1. if $i = j$, updates $\mathtt{sk}'_3$ from $\mathtt{sk}_{(t,0)}$ as follows

$$\begin{aligned}\mathtt{sk}'_3 &= \frac{g^{d_3}}{g_x} \cdot g^{-\sigma'} \cdot F(t)^r \cdot g_x \cdot F(t)^{r_x} \\ &= g^{d_3} \cdot g^{-\sigma'} \cdot F(t)^{r+r_x}\end{aligned}$$

where $\frac{g^{d_3}}{g_x} \cdot g^{-\sigma'} \cdot F(t)^r$ is from the decryption key $\mathtt{sk}_{(\delta,t)}$, and $g_x \cdot F(t)^{r_x}$ is from the key update information $\mathtt{sk}_{(t,0)}$.
2. else, returns $\perp$ (i.e., $\mathtt{sk}_{(\delta,t)}$ and $\mathtt{sk}_t$ have no shared tree node in BT).

We use $\mathtt{sk}_{(\delta,t)}$ to represent the resulting decryption key at time t after seeing key update $\mathtt{sk}_t$ from KGC. Similarly, the user can derive a new key $\mathtt{sk}_{(\delta,t')}$ from $\mathtt{sk}_{t'}$ at time t'.

– $\mathsf{Enc}(\mathtt{mpk}, \mathtt{sk}_t, m, (\mathbf{M}, \pi), t)$: To encrypt a message $m \in \mathbb{Z}_q$ under a policy $(\mathbf{M}, \pi)$ (note that $\mathbf{M}$ has n_1 rows and n_2 columns) at time t, a user first computes $ct_0 = (H_1^{s_1}, H_2^{s_2}, h^s, F(t)^s, \mathtt{sk}_{(t,1)}^s)$, where $s = s_1 + s_2$, and $(s_1, s_2) \in \mathbb{Z}_q^*$.

For $i \in \{1, \cdots, n_1\}$ and $y_2 = \{1, 2, 3\}$, it computes $ct_{(i,y_2)} = \mathtt{H}(\pi(i) y_2 1)^{s_1} \cdot \mathtt{H}(\pi(i) y_2 2)^{s_2} \cdot \prod_{j=1}^{n_2} [\mathtt{H}(0 j y_2 1)^{s_1} \cdot \mathtt{H}(0 j y_2 2)^{s_2}]^{\mathbf{M}_{(i,j)}}$, where $\mathbf{M}_{(i,j)}$ denotes the (i,j)-th element of matrix $\mathbf{M}$, and $\pi : \{1, \cdots, n_1\} \rightarrow \mathcal{U}$ denotes a mapping function. Then, it computes $ct = m \oplus \mathsf{G}(T_1^{s_1} \cdot T_2^{s_2})$. Let $C = (t, ct_0, \{ct_i\}_{i \in n_1}, ct)$ be the resulting ciphertext.

– $\mathsf{Dec}(\mathtt{sk}_{(\delta,t)}, C)$: The user with a decryption key $\mathtt{sk}_{(\delta,t)}$ can get m by decrypting C. The idea is, if the attribute set δ in $\mathtt{sk}_{(\delta,t)}$ satisfies the policy $(\mathbf{M}, \pi)$, there exist constants $\{\gamma_i\}_{i \in I}$ such that $\sum_{i \in I} \gamma_i \mathbf{M}_i = (1, 0, \cdots, 0)$, where $I = \{i | i \in \{1, \cdots, n_1\}, \pi(i) \in \delta\}$ means that a set of rows in $\mathbf{M}$ belongs to δ.
– $\mathsf{Revoke}(id, t, rl, st)$: For all nodes x associated with identity id, KGC adds (x, id) to rl, publishes the updated rl and st at time t.

Correctness and Security. We show the techniques involved in building our practical and secure FB-ABE scheme. First, we replace the hash function $\mathtt{H}(t)$ by $F(t)$ for key update $\mathsf{KeyUp_0}$. The intention is to save computational and storage costs. Suppose we trivially combine the forward-secure technique and the revocation mechanism, where the key update information from KGC involves $\mathtt{H}(t)^{r_x}$. The resulting ciphertext should include a value $\mathtt{H}(t)^s$, the decryption key will include a value $\mathtt{H}(t)^r$, and the decryption process will require an additional pairing $\widehat{\mathsf{e}}(\mathtt{H}(t)^{r+r_x}, h^s) = \widehat{\mathsf{e}}(\mathtt{H}(t)^s, h^{r+r_x})$. Second, we allow the encryption process to include the key update information $ct_{(0,5)} = \mathtt{sk}^s_{(t,1)} = h^{r_x \cdot s}$ so that the revoked users cannot decrypt ciphertext using their self-generated decryption keys. For example, if a revoked user generates a $F(t)^{r'_x}$-based decryption key, the decryption process will not work because the randomnesses (r_x, r'_x) are different in the decryption process. Note that r_x is chosen by KGC, and r'_x is chosen by the revoked user. The correctness of the decryption process and the security analysis are referred to the full version of the paper [3].

We provide a sketch of the semantic security of the proposed FB-ABE. Our security proof is based on the proofs in [19] and [4]. We construct a challenger $\mathcal{S}$ to simulate the security game for $\mathcal{A}$. The key insight is that $\mathcal{S}$ must simulate the public parameters $\mathtt{mpk}$ using the same method described in [19]. As a result, $\mathcal{S}$ can simulate users' decryption keys to answer decryption and break in queries. If $\mathcal{S}$ guesses the challenge time and the challenge identity correctly, the simulation is perfect. Finally, $\mathcal{S}$ returns a challenge ciphertext to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.

The proposed construction only ensures CPA security (i.e., security against chosen plaintext attacks). Below we discuss how to construct a CCA-secure (i.e., security against chosen ciphertext attacks) FB-ABE scheme from Boneh-Katz conversion [**?**]. Most algorithms in such construction remain the same, except the Setup, Enc and Dec algorithms. Specifically, the Setup process chooses two hash functions (h_1, h_2). For example, $h_{1/2} = \{0,1\}^{448} \rightarrow \{0,1\}^{128}$. The encryptor performs the following operations: 1) runs the encapsulation scheme to obtain $(k = h_1(x), com = h_2(x), dec = x)$; 2) chooses a key $k' \in \mathbb{Z}_q$ and runs the Enc algorithm in the CPA-secure FB-ABE to obtain a ciphertext C on k'; 3) generates another ciphertext $C' = \mathsf{G}(k') \oplus (m||x)$, where G is a PRG; 4) computes a MAC tag on (C, C') using key k. The decryptor performs the following operations: 1) obtains k' by decrypting C via the Dec algorithm in the CPA-secure FB-ABE; 2) recovers $(m||x)$ from C' using k' and computes $k = h_1(x)$; 3) outputs m iff $com = h_2(x)$ and the verification of MAC tag on (C, C') under key k is valid.

5 The Proposed FB-PCH

In this section, we present the definition and security models for forward/backward-secure policy-based chameleon hash first. Then, we show the proposed scheme and its security analysis.

5.1 Definition and Models

A forward/backward-secure policy-based chameleon hash (FB-PCH) consists of the following algorithms.

– The Setup, KeyGen, KeyUp_0, KeyUp_1, Revoke algorithms remain same as FB-ABE. Here, we use AA to represent KGC.
– $\mathsf{Hash}(\mathtt{mpk}, \mathtt{sk}_t, m, \Lambda, t')$: It takes the master public key $\mathtt{mpk}$, the key update information $\mathtt{sk}_t$, a message $m \in \mathcal{M}$, a policy Λ and a time t' as input, outputs a chameleon hash h, and randomness r.
– $\mathsf{Verify}(\mathtt{mpk}, h, m, r)$: It takes the master public key $\mathtt{mpk}$, chameleon hash h, message m, and randomness r as input, outputs $b \in \{0, 1\}$.
– $\mathsf{Adapt}(\mathtt{sk}_{(\delta,t)}, h, m, r, m')$: It takes the chameleon secret key $\mathtt{sk}_{(\delta,t)}$, messages m and m', chameleon hash h, and randomness r as input, outputs an adapted randomness r' if $1 = \Lambda(\delta)$ and $t = t'$.

Correctness. The FB-PCH is *correct* if for all security parameters λ, for all $(\mathtt{msk}, \mathtt{mpk}), rl, st \leftarrow \mathsf{Setup}(1^\lambda, k)$, for all $\delta \in \mathcal{U}$, for all non-revoked $id \in \mathcal{I}$, for all $\mathtt{sk}_{(\delta,t)} \leftarrow \mathsf{KeyGen}(\mathtt{msk}, id, \delta, st)$, for all $t \in \mathcal{T}$, for all $\mathtt{sk}_t \leftarrow \mathsf{KeyUp}_0(\mathtt{msk}, t, rl, st)$, for all $t' \in \mathcal{T}$, for all $\mathtt{sk}_{(\delta,t')} \leftarrow \mathsf{KeyUp}_1(\mathtt{sk}_{(\delta,t)}, t')$, for all $\mathtt{sk}_{(\delta,t)} \leftarrow \mathsf{DK}(\mathtt{sk}_{(\delta,t)}, \mathtt{sk}_t)$, for all $m \in \mathcal{M}$, for all $\delta \in \Lambda$, for all $(h, r) \leftarrow \mathsf{Hash}(\mathtt{mpk}, \mathtt{sk}_t, m, \Lambda, t')$, for all $m' \in \mathcal{M}$, for all $t = t'$, for all $r' \leftarrow \mathsf{Adapt}(\mathtt{sk}_{(\delta,t)}, h, m, r, m')$, we have $1 = \mathsf{Verify}(\mathtt{mpk}, h, m, r) = \mathsf{Verify}(\mathtt{mpk}, h, m', r')$. Now, we define two security properties: forward/backward-secure collision-resistance and strong indistinguishability.

Forward/backward-secure Collision-Resistance. Informally, an attacker cannot find valid collisions for the chameleon hashes generated in the past such as time t^*, even if the attacker is given the present chameleon secret key. We allow attackers to access chameleon secret keys (i.e., KeyGen), and see collisions for arbitrary attributes (i.e., KeyGen' and Adapt). We define a formal game between an adversary $\mathcal{A}$ and a challenger $\mathcal{S}$ below.

– $\mathcal{S}$ sets up the game by creating k users with the corresponding identities, and generating system parameters $(\mathtt{msk}, \mathtt{mpk}), rl, st \leftarrow \mathsf{Setup}(1^\lambda, k)$. Eventually, $\mathcal{S}$ creates three initially empty sets $(\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3)$, and returns all public parameters $(\mathtt{mpk}, rl, st)$ to $\mathcal{A}$.

– $\mathcal{A}$ can make the following queries to $\mathcal{S}$ during the training phase.
  - KeyGen. If $\mathcal{A}$ issues a key generation query on a user id and an attribute set δ, $\mathcal{S}$ returns an initial chameleon secret key $\{\mathtt{sk}_{(\delta,0)}\} \leftarrow \mathsf{KeyGen}(\mathtt{msk}, id, \delta, \mathrm{st})$ to $\mathcal{A}$, and records (id, δ) to $\mathcal{Q}_1$.
  - KeyGen'. If $\mathcal{A}$ issues a key generation query on a user id and an attribute set δ, $\mathcal{S}$ records $(id, i, \mathtt{sk}_{(\delta,0)})$ to $\mathcal{Q}_2$, and increases index i.
  - Hash. If $\mathcal{A}$ issues a hash query on a message m, an access policy Λ, and a time t, $\mathcal{S}$ returns a chameleon hash value h and randomness r to $\mathcal{A}$, and records (h, m, Λ) to $\mathcal{Q}_3$.
  - Adapt. If $\mathcal{A}$ issues an adapt query on messages (m, m'), hash value h, randomness r, and index i, $\mathcal{S}$ returns a new randomness r' to $\mathcal{A}$, and records (h, m', Λ) to $\mathcal{Q}_3$. Note that $\mathcal{S}$ returns $\perp$ if the record $(id, i, \mathtt{sk}_{(\delta,0)}) \notin \mathcal{Q}_2$.
  - The KeyUp, Revoke, and $\mathsf{BreakIn}$ oracles behave the same as FB-ABE.

– At some point, $\mathcal{A}$ outputs a tuple $(id^*, h^*, m^*, r^*, m^{*'}, r^{*'}, t^*)$. $\mathcal{A}$ wins the game if the following conditions are held
  - The chameleon hashes are valid, such that $1 = \mathsf{Verify}(\mathtt{mpk}, h^*, m^*, r^*) = \mathsf{Verify}(\mathtt{mpk}, h^*, m^{*'}, r^{*'})$, and $m^* \neq m^{*'}$.
  - There is no attribute set (id^*, δ^*) in $\mathcal{Q}_1$ satisfies the policy Λ^*.
  - The hash oracle must be queried on the chameleon hash h^* and the access policy Λ^* at time t^*. But, the adapt oracle cannot be queried on the chameleon hash h^* and message m^*.

  The restrictions on KeyUp, Revoke and $\mathsf{BreakIn}$ oracles remain the same as FB-ABE. We define the advantage of $\mathcal{A}$ in the above game as

$$\mathtt{Adv}_{\mathcal{A}}^{\mathrm{CR}}(\lambda) = \Pr[\mathcal{A}\ wins].$$

Definition 3. *A FB-PCH scheme is collision resistant if for any PPT $\mathcal{A}$, $\mathtt{Adv}_{\mathcal{A}}^{CR}(\lambda)$ is negligible in λ.*

Indistinguishability against Full Key Exposure. Informally, an attacker $\mathcal{A}$ cannot decide whether for a chameleon hash its randomness was freshly generated using Hash algorithm or was created using Adapt algorithm, even if $\mathcal{A}$ gets access to all randomness that were used to generate/update users' chameleon secret keys (i.e., full key exposure [8]). We define a formal game between an adversary $\mathcal{A}$ and a challenger $\mathcal{S}$ as follows.

– $\mathcal{S}$ sets up the game and answers $\mathcal{A}$'s key generation, key update, revoke, break in, hash and adapt queries using the same method described in the above collision-resistance game except the following differences. First, $\mathcal{S}$ simulates $\mathtt{sk}_{(\delta,0)} \leftarrow \mathsf{KeyGen}(\mathtt{msk}, id, \delta, \mathrm{st}, w)$, where w denotes the randomness used in generating user id's chameleon secret key. Second, $\mathcal{S}$ records the randomness used in updating users' secret keys.

– In the challenge phase, if $\mathcal{A}$ sends two message (m, m'), an identity id, an access policy Λ, and a time t to $\mathcal{S}$, then $\mathcal{S}$ chooses a random bit b, returns (h_b, r_b) and all randomness $\{w\}$ to $\mathcal{A}$, where

$$(h_0, r_0) = \mathsf{Hash}(\mathtt{mpk}, \mathtt{sk}_t, m, \Lambda, t)$$
$$(h_1, r_1') = \mathsf{Hash}(\mathtt{mpk}, \mathtt{sk}_t, m', \Lambda, t)$$

and $r_1 = \mathsf{Adapt}(\mathtt{sk}_{(\delta,t)}, h_1, m', r_1', m)$. We require $1 \leftarrow \mathsf{Verify}(\mathtt{mpk}, m, h_0, r_0) = \mathsf{Verify}(\mathtt{mpk}, m', h_1, r_1)$, and no restrictions on KeyUp, Revoke and $\mathsf{BreakIn}$ oracles. We define the advantage of $\mathcal{A}$ as

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

Definition 4. *A FB-PCH scheme is indistinguishable if for any PPT $\mathcal{A}$, $\mathtt{Adv}_{\mathcal{A}}^{IND}(\lambda)$ is negligible in λ.*

5.2 The Proposed Construction

In this subsection, we present the concrete construction and its security analysis. The proposed FB-PCH includes two building blocks: the modified DL-based CH scheme and the proposed FB-ABE scheme. Below we only focus on hash, verify and adapt processes.

- $\mathsf{Hash}(\mathtt{mpk}, \mathtt{sk}_t, m, (\mathbf{M}, \pi), t)$: To hash a message $m \in \mathbb{Z}_q$ under a policy $(\mathbf{M}, \pi)$ at time t, a user performs the following operations
  1. choose an ephemeral trapdoor $tr \in \mathbb{Z}_q^*$, and randomness $\mathsf{r} \in \mathbb{Z}_q^*$, compute a chameleon hash $h = \mathtt{pk}^m \cdot R$ and $\xi = (R, \pi)$, where $\mathtt{pk} = g^{tr}$, $R = g^{\mathsf{r}}$, and π is a NIZK for $\log(R)$.
  2. generate a ciphertext C on trapdoor tr under policy $(\mathbf{M}, \pi)$ and time t.
  3. output (m, h, ξ, C).
- $\mathsf{Verify}(\mathtt{mpk}, m, h, \xi)$: Anyone can verify whether a given hash (m, h, ξ) is valid, it outputs 1 if $h = \mathtt{pk}^m \cdot R$ and π is valid.
- $\mathsf{Adapt}(\mathtt{sk}_{(\delta,t)}, m, m', h, \xi, C)$: A modifier with a secret key $\mathtt{sk}_{(\delta,t)}$, a new message $m' \in \mathbb{Z}_q$, performs the following operations
  1. check $1 \overset{?}{=} \mathsf{Verify}(\mathtt{mpk}, m, h, \xi)$.
  2. obtain the encrypted trapdoor tr by decrypting ciphertext C.
  3. compute a new randomness $\mathsf{r}' = \mathsf{r} + (m - m') \cdot tr$, and $\xi' = (R', \pi')$, where $R' = g^{\mathsf{r}'}$, and π' is a NIZK for $\log(R')$.
  4. output (m', h, ξ', C).

Correctness and Security. We use a non-interactive zero-knowledge (NIZK) proof to prevent key exposure attacks [6]. For example, the user computes $Y = g^y$, $e = \mathtt{H}_q(Y)$ and $s = y + \mathsf{r} \cdot e$, where y is randomness. Anyone can verify $g^s = Y \cdot R^{\mathtt{H}_q(Y)}$. If we directly use the DL-based chameleon hash described in Section 3.2, the value r can be trivially recovered given two message-randomness pairs. We provide a detailed comparison against PCH in Section 6 to show the performance of the proposed FB-PCH scheme. The proposed FB-PCH scheme ensures forward/backward-secure collision-resistance and indistinguishability. Our proofs are similar to the proofs described in [17]. Specifically, the security of forward/backward-secure collision-resistance (or indistinguishability) can be reduced to the collision-resistance (or indistinguishability) of the underlying CH scheme and the semantic security of FB-ABE. The detailed security analysis is deferred to Appendix C.

6 Evaluation and Application in a Blockchain

In this section, we evaluate our proposed FB-PCH. Next, we show how to apply the proposed solution to blockchain rewriting systems.

6.1 Evaluation

We implement our proposed FB-PCH scheme and evaluate its performance on a PC with Intel Core i9 and 7.7GiB RAM. We benchmark each algorithm of FB-PCH with various parameters. We implement the pairing groups based on MNT224 curve, and instantiate the hash functions with the corresponding standard interfaces provided by the Charm framework [5]. Our implementation code is given in GitHub [3].

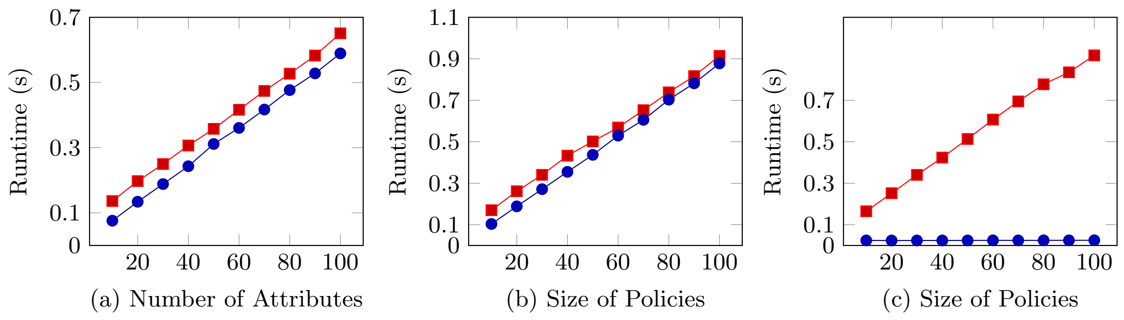

Fig. 3. Running time of KeyGen (left), Hash (middle) and Adapt (right) algorithms. Red line (with solid square) is for PCH, while blue line (with solid dot) is for FB-PCH.

First, we provide a performance comparison between FB-PCH and PCH [17]. The execution time of KeyGen, Hash, and Adapt algorithms are shown in Figure 3. In FB-PCH, the execution time of KeyGen and Hash algorithms are linear in the number of attributes and the size of policies, respectively. The execution time of KeyGen is 0.59 seconds given the number of attributes is 100. The execution time of Hash algorithms is about 0.88 seconds for handling a policy of size 100. Figure 3 (a-b) show that the execution time of KeyGen, and Hash algorithms in FB-PCH are similar to PCH. Figure 3 (c) shows that the Adapt algorithm of FB-PCH is more efficient than PCH because our adapt algorithm does not require an encryption process. Besides, the execution time of Verify algorithm in FB-PCH is also similar to PCH, which requires ≈ 0.002 seconds. For key update processes, the execution time of KeyUp_0 and KeyUp_1 algorithms are about 0.005 seconds, respectively.

Second, we evaluate the storage cost of key generation and hash/adapt processes based on the number of various group elements. Table 2 shows that our

FB-PCH incurs a reasonably higher overhead in group $\mathbb{G}$, but only one more group element in $\mathbb{H}$ than PCH. Note that the size of an element of $\mathbb{H}$ is three times the size of $\mathbb{G}$ in the MNT224 curve. To conclude, our scheme is practical in the sense that its computational complexity of achieving backward security is logarithmic in the number of system users. The storage cost of the user's chameleon secret keys and ciphertexts are logarithmic and constant in the number of time periods, respectively.

Table 2. The storage cost between FB-PCH and PCH. T denotes the number of attributes to KeyGen, (n_1, n_2) denotes the dimension of monotone span program (MSP) for encryption to Hash, ℓ denotes the binary tree's depth. RSA-based CH with public key $(N = pq, e)$ and trapdoor d, where $ed = 1 \mod (p-1)(q-1)$.

Operations:	$\mathbb{G}$	$\mathbb{H}$	$\mathbb{G}_T$	$\mathbb{Z}_q$	N	d
$\mathsf{KeyGen}_{\text{FB-PCH/PCH}}$:	$3T+2+\ell/3T+3$	4/3	-	-	-	-/1
$\mathsf{Hash}_{\text{FB-PCH/PCH}}$:	$4+n_1/n_1$	4/3	-/1	4/1	-/6	-/-

6.2 Application in a Blockchain

In this subsection, we show how to construct a distributed mutable blockchain by using FB-PCH. First, we explain the basic idea of the FB-PCH-based mutable blockchain. Second, we present the structure of a mutable transaction. Third, we show the detailed modification process.

Basic idea The mutable blockchain remains intact but some mutable transactions in a block can be rewritten. In blockchain, each block stores a compact representation of a set of transactions. The root hash of a Merkle tree (i.e., TX_ROOT) accumulates all transactions associated with a block. If a user appends a mutable transaction T_1 to the blockchain, the message m must be hashed via $(h, r, C) = \mathsf{Hash}(\mathtt{mpk}, m, \Lambda, t')$. The ciphertext C includes an access policy Λ, which is chosen by the user who created T_1. The other immutable transactions are processed using conventional collision-resistant hash function $\mathtt{H}$. If the transaction T_1 needs to be modified, a modifier with a secret key $\mathtt{sk}_{(\delta,t)}$ (given by attribute authority AA) satisfying Λ and $t = t'$ can replace the message-randomness pair (m, r) by (m', r'). The modifier will broadcast (m', r') to the blockchain network, and all participants verify the correctness of (m', r') and update their local copy of the blockchain accordingly.

Structure Compared to the immutable transaction, the key changes of a mutable transaction are mainly reflected in the existence of the auxiliary data. The auxiliary data includes (h, r, C), which is the output of FB-PCH on $(\mathtt{mpk}, m, \Lambda, t')$. The auxiliary data will be updated after each modification. If a validator receives a mutable transaction, he needs to verify the transaction (i.e., the verification

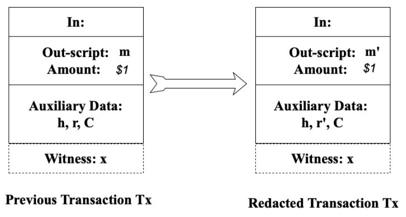

Fig. 4. The structure of a mutable transaction.

of FB-PCH) and witness x, where x denotes a signature computed on transaction using the corresponding secret key `sk` [1]. Note that in the permissioned blockchain, we require some pre-defined parties to validate transactions (we call it validator). The permissioned blockchain is usually based on Byzantine Fault Tolerance (BFT) consensus instead of "proof of work" (thus, no miner is required). If they are valid, the validator will take the FB-PCH hash value h as a leaf node of Merkle tree.

Form Figure 4, the hashed message and the corresponding randomness are updated from (m, r) to (m', r'). The modifier should broadcast (m', r'), an identifier id and her public key `pk` at time t to the blockchain network, and all system users are supposed to verify the correctness of (m', r') and update their local copy of blockchain. Note that the identifier id will help system users to identify which transaction needs to be updated, the public key `pk` will help AA to identify the corresponding modifier in case they misbehave. We argue that it is difficult to enforce that each user in the blockchain updates its local copy, but the whole system works if a majority of users follow its protocol as in the original blockchain system. Nonetheless, a malicious user can always keep separate copies of the original transactions. However, we note that all existing schemes, that rewrite blockchains, suffer from this limitation. We stress that the transaction amount cannot be changed. Otherwise, it will cause transaction inconsistency [18, 25].

Modification process We regard the message m in Figure 4 as illicit data. We take a mutable transaction T_1 recording message m in Figure 5 to explain the modification process. Since the message m is hashed under `mpk`, it can be modified by the key holder AA. But, AA will distribute various rewriting privileges to different modifiers based on their attribute sets. Thus, a modifier with enough rewriting privilege at specific time can redact transaction T_1 by replacing (m, r) with (m', r'). After modification, the modifier broadcasts (m', r'), id and `pk` at time t in the blockchain network, all users will perform the following checks.

- whether the modifier `pk` is non-revoked, i.e., $\texttt{pk} \notin rl$.
- whether the message m needs to be modified as m' according to the rule, like GDPR.
- whether the message-randomness pairs (m, r) and (m, r') are mapping to the same chameleon hash value h.

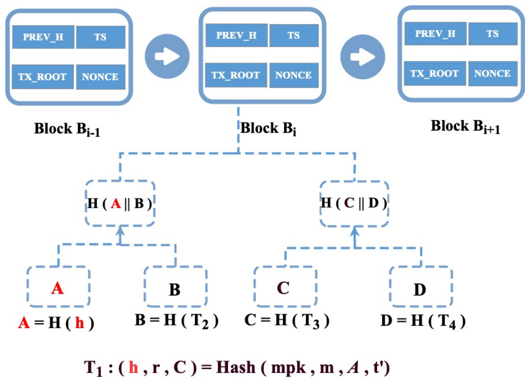

Fig. 5. FB-PCH for mutable blockchains. The mutable transaction A is created using the FB-PCH function. The normal transactions (B, C, D) are derived using the conventional collision-resistant hash function H (e.g., SHA-256).

If all the above requirements are met, all users will update their local copy of the blockchain by replacing m (in out-script) and r (in auxiliary data) with m' and r', respectively. Besides, we consider a scenario where the modified transaction contains illicit data after modification. This is because: 1) the modifier refuses to redact/delete message m in T_1; 2) the modifier chooses a message m' containing a different illicit data in T_1. It either of these cases occurs, we allow AA to redact/delete illicit data and revoke the corresponding modifier in time by updating the revocation list rl.

We show the impact of integrating the proposed FB-PCH into the mutable blockchain. First, the rewriting of mutable transactions incurs almost no overhead to Merkle tree generation and chain validation. This is because the randomness r and the ciphertext C are not included in the hash computation of the aggregation. Nevertheless, they are provided as the non-hashed part of the transaction (i.e., auxiliary data). Second, the overall storage cost of mutable transactions to the mutable blockchains is acceptable. This is because the number of mutable transactions ranges from 2% to 10% inside a block according to [18]. Third, we focus on the security benefit of applying FB-PCH to mutable blockchains. The forward and backward security ensure that the compromised chameleon trapdoors only affect the security of mutable transactions generated for present time. In other words, attackers shall modify neither the previously generated mutable transactions nor the future transactions.

7 Conclusion

In this paper, we introduced the notion of forward/backward-secure policy-based chameleon hash (FB-PCH). Applying FB-PCH to redactable blockchains enables the modifiers' rewriting privileges (or attribute-based trapdoors) to be periodically updated or revoked due to trapdoor leakage or misuse. We also proposed a practical and secure forward/backward-secure attribute-based encryption (FB-ABE) scheme. Our proposed FB-ABE scheme ensures that the non-interactive key update [11] and the epoch-based revocation mechanism [10] can work together efficiently. Our evaluation validated its practicality [3] and showed that our solution could be easily used to construct redactable blockchain systems.

Bibliography

[1] Bitcoin Script. https://en.bitcoin.it/wiki/Script.

[2] General Data Protection Regulation. https://gdpr-info.eu.

[3] Our Source Code. https://github.com/SMC-SMU/forward-backward-secure-PCH.

[4] S. Agrawal and M. Chase. Fame: fast attribute-based message encryption. In *CCS*, pages 665–682, 2017.

[5] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.

[6] G. Ateniese and B. de Medeiros. On the key exposure problem in chameleon hashes. In *SCN*, pages 165–179, 2004.

[7] G. Ateniese, B. Magri, D. Venturi, and E. Andrade. Redactable blockchain–or–rewriting history in bitcoin and friends. In *EuroS&P*, pages 111–126, 2017.

[8] A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In *TCC*, pages 60–79, 2006.

[9] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE S&P*, pages 321–334, 2007.

[10] A. Boldyreva, V. Goyal, and V. Kumar. Identity-based encryption with efficient revocation. In *ACM CCS*, pages 417–426, 2008.

[11] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.

[12] C. Boyd and K. Gellert. A modern view on forward security. *The Computer Journal*, 64(4):639–652, 2021.

[13] J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Chameleon-hashes with ephemeral trapdoors. In *PKC*, pages 152–182, 2017.

[14] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.

[15] H. Cui, R. H. Deng, Y. Li, and B. Qin. Server-aided revocable attribute-based encryption. In *ESORICS*, pages 570–587, 2016.

[16] D. Derler, K. Samelin, and D. Slamanig. Bringing order to chaos: The case of collision-resistant chameleon-hashes. In *PKC*, pages 462–492, 2020.

[17] D. Derler, K. Samelin, D. Slamanig, and C. Striecks. Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. In *NDSS*, 2019.

[18] D. Deuber, B. Magri, and S. A. K. Thyagarajan. Redactable blockchain in the permissionless setting. In *IEEE S&P*, pages 124–138, 2019.

[19] M. Drijvers, S. Gorbunov, G. Neven, and H. Wee. Pixel: Multi-signatures for consensus. In *USENIX*, pages 2093–2110, 2020.

[20] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS*, pages 89–98, 2006.

[21] M. D. Green and I. Miers. Forward secure asynchronous messaging from puncturable encryption. In *IEEE S&P*, pages 305–320, 2015.

[22] Y. Jia, S.-F. Sun, Y. Zhang, Z. Liu, and D. Gu. Redactable blockchain supporting supervision and self-management. In *ACM AsiaCCS*, pages 844–858, 2021.

[23] H. Krawczyk and T. Rabin. Chameleon signatures. In *NDSS*, 2000.

[24] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[25] I. Puddu, A. Dmitrienko, and S. Capkun. μchain: How to forget without hard forks. *IACR Cryptology ePrint Archive 2017/106*, 2017.

[26] A. Sahai, H. Seyalioglu, and B. Waters. Dynamic credentials and ciphertext delegation for attribute-based encryption. In *CRYPTO*, pages 199–217, 2012.

[27] K. Samelin and D. Slamanig. Policy-based sanitizable signatures. In *CT-RSA*, pages 538–563, 2020.

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

[29] Y. Tian, N. Li, Y. Li, P. Szalachowski, and J. Zhou. Policy-based chameleon hash for blockchain rewriting with black-box accountability. In *ACSAC*, pages 813–828, 2020.

[30] S. Xu, J. Ning, J. Ma, G. Xu, J. Yuan, and R. H. Deng. Revocable policy-based chameleon hash. In *ESORICS*, pages 327–347, 2021.

A Correctness of the Decryption.

We provide the correctness of the decryption process. Suppose that the user holds a decryption key in the form of $\mathtt{sk}_{(\delta,t)} = (\mathtt{sk}_0, \{\mathtt{sk}_y\}_{y\in\delta}, \mathtt{sk}')$, where $\mathtt{sk}_0 = (h^{b_1\cdot r_1}, h^{b_2\cdot r_2}, h^r, h^{r+r'})$, and $\mathtt{sk}'_3 = g^{d_3}\cdot g^{-\sigma'}\cdot F(t)^{r+r_x+r'}$ (i.e., the user is not revoked at time t).

– The ciphertext associated with $y_1 \in \delta$ and $y_2 = \{1,2,3\}$ is calculated as

$$\begin{aligned} A = &\ \widehat{\mathtt{e}}(\mathtt{H}(y_1 11)^{s_1\cdot\gamma_i}\cdot \mathtt{H}(y_1 12)^{s_2\cdot\gamma_i}, h^{b_1\cdot r_1})\cdot \widehat{\mathtt{e}}(\mathtt{H}(y_1 21)^{s_1\cdot\gamma_i}\cdot \mathtt{H}(y_1 22)^{s_2\cdot\gamma_i}, h^{b_2\cdot r_2}) \\ &\cdot \widehat{\mathtt{e}}(\mathtt{H}(y_1 31)^{s_1\cdot\gamma_i}\cdot \mathtt{H}(y_1 32)^{s_2\cdot\gamma_i}, h^{r})\cdot \underline{\widehat{\mathtt{e}}(F(t)^s, h^{r+r'})\cdot \widehat{\mathtt{e}}(F(t), h^{r_x\cdot s})} \\ = &\ \widehat{\mathtt{e}}(\mathtt{H}(y_1 11)^{s_1\cdot b_1\cdot r_1\cdot\gamma_i}, h)\cdot \widehat{\mathtt{e}}(\mathtt{H}(y_1 12)^{s_2\cdot b_1\cdot r_1\cdot\gamma_i}, h)\cdot \widehat{\mathtt{e}}(\mathtt{H}(y_1 21)^{s_1\cdot b_1\cdot r_2\cdot\gamma_i}, h) \\ &\cdot \widehat{\mathtt{e}}(\mathtt{H}(y_1 22)^{s_2\cdot b_2\cdot r_2\cdot\gamma_i}, h)\cdot \widehat{\mathtt{e}}(\mathtt{H}(y_1 31)^{s_1\cdot r\cdot\gamma_i}, h)\cdot \widehat{\mathtt{e}}(\mathtt{H}(y_1 32)^{s_2\cdot r\cdot\gamma_i}, h)\cdot \underline{\widehat{\mathtt{e}}(F(t)^s, h^{r_x+r+r'})} \end{aligned}$$

– The first pairing in B is calculated as

$$\begin{aligned} &\widehat{\mathtt{e}}(\mathtt{H}(\pi(i)11)^{b_1\cdot r_1\cdot\gamma_i/a_1}\cdot \mathtt{H}(\pi(i)21)^{b_2\cdot r_2\cdot\gamma_i/a_1}\cdot \mathtt{H}(\pi(i)31)^{r\gamma_i/a_1}\cdot g^{\sigma_i\cdot\gamma_i/a_1}\cdot (g^{d_1})^{\mathbf{M}_{(i,1)}\cdot\gamma_i} \\ &\cdot \prod_{j=2}^{n_2}[\mathtt{H}(0j11)^{b_1\cdot r_1/a_1}\cdot \mathtt{H}(0j21)^{b_2\cdot r_2/a_1}\cdot \mathtt{H}(0j31)^{r/a_1}\cdot g^{\sigma'_j/a_1}]^{\mathbf{M}_{(i,j)}\cdot\gamma_i}, h^{a_1\cdot s_1}) \end{aligned}$$

– The second pairing in B is calculated as

$$\widehat{\mathsf{e}}(\mathtt{H}(\pi(i)12)^{b_1\cdot r_1\cdot\gamma_i/a_2}\cdot\mathtt{H}(\pi(i)22)^{b_2\cdot r_2\cdot\gamma_i/a_2}\cdot\mathtt{H}(\pi(i)32)^{r\cdot\gamma_i/a_2}\cdot g^{\sigma_i\cdot\gamma_i/a_2}\cdot(g^{d_2})^{\mathbf{M}_{(i,1)}\cdot\gamma_i}$$
$$\cdot\prod_{j=2}^{n_2}[\mathtt{H}(0j12)^{b_1\cdot r_1/a_2}\cdot\mathtt{H}(0j22)^{b_2\cdot r_2/a_2}\cdot\mathtt{H}(0j32)^{r/a_2}\cdot g^{\sigma'_j/a_2}]^{\mathbf{M}_{(i,j)}\cdot\gamma_i},h^{a_2\cdot s_2})$$

– The third pairing in B is calculated as

$$\widehat{\mathsf{e}}(\underline{g^{d_3}\cdot g^{-\sigma'}\cdot F(t)^{r+r_x+r'}}\cdot\prod_{j=2}^{n_2}(g^{-\sigma'_j})^{\mathbf{M}_{(i,j)}\cdot\gamma_i},h^s)$$

– By multiplying above three parings in B, we have

$$\begin{aligned}B=&\widehat{\mathsf{e}}(\mathtt{H}(\pi(i)11)^{b_1\cdot r_1\cdot s_1\cdot\gamma_i},h)\cdot\widehat{\mathsf{e}}(\mathtt{H}(\pi(i)21)^{b_2\cdot r_2\cdot s_1\cdot\gamma_i},h)\cdot\widehat{\mathsf{e}}(\mathtt{H}(\pi(i)31)^{r\cdot s_1\cdot\gamma_i},h)\\&\cdot\widehat{\mathsf{e}}(\mathtt{H}(\pi(i)12)^{b_1\cdot r_1\cdot s_2\cdot\gamma_i},h)\cdot\widehat{\mathsf{e}}(\mathtt{H}(\pi(i)22)^{b_2\cdot r_2\cdot s_2\cdot\gamma_i},h)\cdot\widehat{\mathsf{e}}(\mathtt{H}(\pi(i)32)^{r\cdot s_2\cdot\gamma_i},h)\\&\cdot\widehat{\mathsf{e}}(g^{d_1\cdot a_1\cdot s_1},h)\cdot\widehat{\mathsf{e}}(g^{d_2\cdot a_2\cdot s_2},h)\cdot\widehat{\mathsf{e}}(g^{d_3\cdot s},h)\cdot\widehat{\mathsf{e}}(F(t)^{r+r_x+r'},h^s)\end{aligned}$$

where σ_i and σ'_j in B were cancelled out when multiplying, and recall that $\sum_{i\in I}\gamma_i\cdot\mathbf{M}_i=(1,0,\cdots,0)$.

– Eventually, B/A is calculated as below

$$\begin{aligned}B/A&=\widehat{\mathsf{e}}(g^{d_1\cdot a_1\cdot s_1},h)\cdot\widehat{\mathsf{e}}(g^{d_2\cdot a_2\cdot s_2},h)\cdot\widehat{\mathsf{e}}(g^{d_3\cdot(s_1+s_2)},h)\\&=\widehat{\mathsf{e}}(g,h)^{s_1\cdot(d_1\cdot a_1+d_3)}\cdot\widehat{\mathsf{e}}(g,h)^{s_2\cdot(d_2\cdot a_2+d_3)}\\&=T_1^{s_1}\cdot T_2^{s_2}.\end{aligned}$$

B Security Analysis of FB-ABE

Theorem 2. *The proposed FB-ABE scheme is semantically secure if the proposed composite assumption is held in the asymmetric pairing groups.*

Proof. We assume a simulator $\mathcal{S}$ whose goal is to the break the security of the composite assumption. $\mathcal{S}$ chooses a challenge time $t^*=t_1^*||t_2^*||\cdots||t_\ell^*$ and a challenge identity id^*.

– $\mathcal{S}$ simulates the public parameters as $g_c=\bar{g}^\gamma\cdot g^{\alpha_1^\ell}$, $\bar{g}=g^{z_c}$, $T_{1/2}=\widehat{\mathsf{e}}(g,h)^{d_{1/2}\cdot a_{1/2}}\cdot\widehat{\mathsf{e}}(g_c,h^{\alpha_1})$, $g_0=\bar{g}^\delta\cdot g^{\alpha_1^\ell\cdot t_1^*}\cdots\cdot g^{\alpha_1\cdot t_\ell^*}\cdot g^{\alpha_1\cdot\mathtt{H}_q(id^*)}$, $g_1=\bar{g}^{\gamma_1}/g^{\alpha_1^\ell},\cdots,g_\ell=\bar{g}^{\gamma_\ell}/g^{\alpha_1}$, where $d_{1/2},\gamma,\gamma_1,\cdots\gamma_\ell,\delta,z_c\in\mathbb{Z}_q$ are chosen by $\mathcal{S}$. Since $\mathcal{S}$ can easily break the composite problem if $g^{\alpha_1^{\ell+1}}$ is unknown, we use it to simulate the updated decryption keys.
$\mathcal{S}$ also generates other parameters honestly according to the scheme's description. By setting up the parameters as such, $\mathcal{S}$ implicitly sets $d_3=\alpha_1$, and $g_c^{d_3}=\bar{g}^{\alpha_1\cdot\gamma}g^{\alpha_1^{\ell+1}}$. Below, we mainly focus on the $g_c^{d_3}$-related simulations.

– $\mathcal{S}$ simulates the key update, break in, and decryption queries as follows.
  1. KeyUp. $\mathcal{S}$ just keep track of the present time t.

2. We consider two cases in simulating users' decryption keys for decryption queries. The first case is $t \neq t^*$. We show $\mathcal{S}$ can simulate a decryption key at time $t = t_1 || \cdots || t_{\bar{k}} || \cdots || t_\ell$, where $\bar{k} \in [1, \ell]$. Note that $t_{\bar{k}} \neq t^*_{\bar{k}}$, which implies that $\bar{k}$ is not prefix of t^*, and $\bar{k}$ is the smallest index at time t.

   In this case, $\mathcal{S}$ first chooses $z \in \mathbb{Z}_q$, and sets $r = \frac{\alpha_1^{\bar{k}}}{t_{\bar{k}} - t^*_{\bar{k}}} + z$. Then, $\mathcal{S}$ computes key components $(\mathtt{sk}_{(0,3)}, \mathtt{sk}'_3)$ as

$$\left(h^r, \frac{g_c^{d_3}}{g_x} \cdot g^{-\sigma'} \cdot \underline{(g_0 \cdot g_1^{t_1} \cdots g_{\bar{k}}^{t_{\bar{k}}} \cdot g_\ell^{\mathtt{H}_q(id)})^r}, g^r_{\bar{k}+1}, \cdots, g^r_\ell\right) \tag{1}$$

   This is a well-formed key at time $t = t_1 || \cdots || t_{\bar{k}}$, where g_x, σ' are chosen by $\mathcal{S}$. We show how to calculate the underlined (U) term in (1).

$$\begin{aligned} U &= [\bar{g}^\delta \cdot g^{\alpha_1^\ell \cdot t_1^*} \cdots g^{\alpha_1 \cdot t_\ell^*} \cdot g^{\alpha_1 \cdot \mathtt{H}_q(id^*)} (\bar{g}^{\gamma_1}/g^{\alpha_1^\ell})^{t_1} \cdots (\bar{g}^{\gamma_{\bar{k}}}/g^{\alpha_{\ell-\bar{k}+1}})^{t_{\bar{k}}} \cdot (\bar{g}^{\gamma_\ell}/g^{\alpha_1})^{\mathtt{H}_q(id)}]^r \\ &= [\bar{g}^{\delta + \Sigma_{i=1}^{\bar{k}} t_i \cdot \gamma_i + \gamma_\ell \cdot \mathtt{H}_q(id)} \cdot \prod_{i=1}^{\bar{k}-1} g_{\ell-i+1}^{t_i^* - t_i} \cdot g_{\ell-\bar{k}+1}^{t^*_{\bar{k}} - t_{\bar{k}}} \cdot \prod_{i=\bar{k}+1}^{\ell} g_{\ell-i+1}^{t_i^*} \cdot g^{\alpha_1 \cdot [\mathtt{H}_q(id^*) - \mathtt{H}_q(id)]}]^r \\ &= Z \cdot g_{\ell-\bar{k}+1}^{r(t^*_{\bar{k}} - t_{\bar{k}})} \end{aligned}$$

   where Z is shown as follows

$$Z = [\bar{g}^{\delta + \Sigma_{i=1}^{\bar{k}} t_i \cdot \gamma_i + \gamma_\ell \cdot \mathtt{H}_q(id)} \cdot \underline{\prod_{i=1}^{\bar{k}-1} g_{\ell-i+1}^{t_i^* - t_i}} \cdot \prod_{i=\bar{k}+1}^{\ell} g_{\ell-i+1}^{t_i^*} \cdot g^{\alpha_1 \cdot (\mathtt{H}_q(id^*) - \mathtt{H}_q(id))}]^r$$

   $\mathcal{S}$ can compute all the terms in Z, and the underlined term in Z is equal to 1 because $t_i = t_i^*$ for all $i < \bar{k}$. The remaining term in $(g_0 \cdot g_1^{t_1} \cdots g_{\bar{k}}^{t_{\bar{k}}})^r$ is $g_{\ell-\bar{k}+1}^{r(t^*_{\bar{k}} - t_{\bar{k}})}$. Since $r = \frac{\alpha_1^{\bar{k}}}{t_{\bar{k}} - t^*_{\bar{k}}} + z$, we can rewrite it as follows

$$g_{\ell-\bar{k}+1}^{r \cdot (t^*_{\bar{k}} - t_{\bar{k}})} = g_{\ell-\bar{k}+1}^{z(t^*_{\bar{k}} - t_{\bar{k}})} \cdot g_{\ell-\bar{k}+1}^{(t^*_{\bar{k}} - t_{\bar{k}}) \frac{\alpha_1^{\bar{k}}}{t_{\bar{k}} - t^*_{\bar{k}}}} = \frac{g_{\ell-\bar{k}+1}^{z(t^*_{\bar{k}} - t_{\bar{k}})}}{g^{\alpha_1^{\ell+1}}}$$

   Hence, the second component in (1) is equal to

$$\frac{g_c^{d_3}}{g_x} \cdot g^{-\sigma'} \cdot \underline{(g_0 \cdot g_1^{t_1} \cdots g_{\bar{k}}^{t_{\bar{k}}})^r} = g^{\alpha_1^{\ell+1}} \cdot \frac{\bar{g}^{\alpha_1 \cdot \gamma}}{g_x} \cdot Z \cdot \frac{g_{\ell-\bar{k}+1}^{z(t^*_{\bar{k}} - t_{\bar{k}})}}{g^{\alpha_1^{\ell+1}}} = \frac{\bar{g}^{\alpha_1 \cdot \gamma}}{g_x} \cdot Z \cdot g_{\ell-\bar{k}+1}^{z(t^*_{\bar{k}} - t_{\bar{k}})}$$

   To this end, $\mathcal{S}$ can simulate the second component in (1) because the unknown value $g^{\alpha_1^{\ell+1}}$ is cancelled out. The first component h^r in (1), and other components $(g^r_{\bar{k}+1}, \cdots, g^r_\ell)$ can be easily computed by $\mathcal{S}$ since they do not involve $g^{\alpha_1^{\ell+1}}$. This completes the simulation of $g_c^{d_3}$-related key components at time $t \neq t^*$. $\mathcal{S}$ simulates other key components using the same approach described in FAME [4]. Specifically, $\mathcal{S}$ first obtains the composite assumption challenge $(h^{a_1 \cdot \alpha_1^{\ell+1}}, h^{a_2 \cdot \alpha_2^{\ell+1}}, h^{\alpha_1^{\ell+1} + \alpha_2^{\ell+1}})$. Then, $\mathcal{S}$

derives $(h^{a_1 \cdot \alpha_1^{\ell+1} \cdot \bar{r}}, h^{a_2 \cdot \alpha_2^{\ell+1} \cdot \bar{r}}, h^{(\alpha_1^{\ell+1}+\alpha_2^{\ell+1}) \cdot \bar{r}})$ to simulate other key components, where $\bar{r}$ is a randomly chosen value from $\mathbb{Z}_q$.
The second case is $t = t^*$, but $id \neq id^*$. If $\mathcal{A}$ issues a decryption key query on an attribute set δ (i.e., $1 \neq \Lambda^*(\delta)$, $\mathcal{S}$ simulates a decryption key using the similar approach as above, but using the fact that $id \neq id^*$ instead of $t_{\bar{k}} \neq t^*_{\bar{k}}$. Recall that user's decryption key $\mathtt{sk}'_3$ involves $F(t, id) = g_0 \cdot \prod g_i^{t_i} \cdot g_\ell^{\mathtt{H}_q(id)} \in \mathbb{G}$. $\mathcal{S}$ sets $r = \frac{\alpha_1^\ell}{\mathtt{H}_q(id) - \mathtt{H}_q(id^*)} + z$ in equation (1) for $z \in \mathbb{Z}_q$, and the simulation follows the similar approach as above.
3. Break in. $\mathcal{S}$ simulates a decryption key at break in time $\bar{t}$ using the same method described above (i.e., the first case in simulating users' decryption keys). $\mathcal{S}$ can simulate it since $\bar{t}$ is not prefix of t^*.

Last, $\mathcal{S}$ can easily answer decryption queries and revoke queries.
– In the challenge phase, $\mathcal{S}$ returns a challenge ciphertext $C^* \leftarrow \mathsf{Enc}(\mathtt{mpk}, m_b, \Lambda^*, t^*)$ to $\mathcal{A}$, where ct_0 can be

$$b = 0 : (h^{a_1 \cdot \alpha_1^{\ell+1} \cdot \bar{r'}}, h^{a_2 \cdot \alpha_2^{\ell+1} \cdot \bar{r'}}, h^{(\alpha_1^{\ell+1}+\alpha_2^{\ell+1}) \cdot \bar{r'}}, g^{z_c \cdot (\delta + \sum_{i=1}^{\ell} \gamma_i \cdot t_i^*) \cdot (\alpha_1^{\ell+1}+\alpha_2^{\ell+1})})$$
$$b = 1 : (h^{a_1 \cdot \alpha_1^{\ell+1} \cdot \bar{r'}}, h^{a_2 \cdot \alpha_2^{\ell+1} \cdot \bar{r'}}, h^{s \cdot \bar{r'}}, g^{z_c \cdot (\delta + \sum_{i=1}^{\ell} \gamma_i \cdot t_i^*) \cdot s})$$

Note that $\bar{r'}$ is a randomly chosen value from $\mathbb{Z}_q$.

Since there are at most k system users and $\mathcal{T}$ times, we have

$$\mathtt{Adv}_{\mathcal{A}}^{\text{FB-ABE}}(\lambda) \leq k \cdot \mathcal{T} \cdot \mathtt{Adv}_{\mathcal{S}}^{Com}(\lambda)$$

C Security Analysis of FB-PCH

Theorem 3. *The FB-PCH scheme is forward/backward-secure collision resistant if the CH scheme is collision resistant, and the FB-ABE scheme is semantically secure.*

Proof. We define a sequence of games $\mathbb{G}_i$, $i = 0, \cdots, 3$ and let $\mathtt{Adv}_i^{\text{FB-PCH}}$ be the advantage of the adversary in game $\mathbb{G}_i$. We assume that $\mathcal{A}$ issues at most q queries to the Hash oracle.

– $\mathbb{G}_0$: This is original game for forward/backward-secure 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 query to the Hash oracle at time t^* which returns the chameleon hash $(m^*, \mathsf{h}^*, \mathsf{r}^*, b^*, C^*, t^*)$. $\mathcal{S}$ will output a random bit if $\mathcal{A}$'s attacking query does not occur in the g-th query. Because we assume the upper-bound of time is $\mathcal{T}$, we have

$$\mathtt{Adv}_0^{\text{FB-PCH}} = q \cdot \mathcal{T} \cdot \mathtt{Adv}_1^{\text{FB-PCH}} \tag{2}$$

– $\mathbb{G}_2$: This game is identical to game $\mathbb{G}_1$ except that in the g-th query, $\mathcal{S}$ replaces the encrypted trapdoor tr^* in C^* by $\perp$ (i.e., empty value). Below we show that

the difference between $\mathbb{G}_1$ and $\mathbb{G}_2$ is negligible if the FB-ABE is semantically secure.
Let $\mathcal{S}$ be an attacker against the FB-ABE scheme with semantic security, who is given a public key $\mathtt{pk}^*$ and a set of oracles, aims to distinguish between encryption of M_0 and M_1 under an access structure Λ at time t^*. $\mathcal{S}$ simulates the game for $\mathcal{A}$ as follows.

- $\mathcal{S}$ sets up $\mathtt{mpk} = \mathtt{pk}^*$ and completes the remainder of Setup honestly.
- $\mathcal{S}$ can honestly answer all queries made by $\mathcal{A}$ using the given set of oracles. In the g-th query, $\mathcal{S}$ submits an identity id, two messages $(tr, 0)$, an access structure Λ^* and a time t^* to his challenger and obtains a challenge ciphertext C^*. Eventually, $\mathcal{S}$ returns $(m^*, \mathsf{h}^*, \mathsf{r}^*, b^*, C^*)$ to $\mathcal{A}$, where $b^* = g^{m^*} \cdot \mathsf{h}^{*\mathsf{r}}, \mathsf{h}^* = h^{tr}$. Since the trapdoor tr and the randomness r are randomly chosen by $\mathcal{S}$, $\mathcal{S}$ can simulate the adapt queries successfully.

If the encrypted message in C^* is tr, then the simulation is consistent with $\mathbb{G}_1$; Otherwise, the simulation is consistent with $\mathbb{G}_2$. Therefore, if the advantage of $\mathcal{A}$ is significantly different in $\mathbb{G}_1$ and $\mathbb{G}_2$, $\mathcal{S}$ can break the semantic security of the FB-ABE scheme. Hence, we have

$$\left|\mathtt{Adv}_1^{\text{FB-PCH}} - \mathtt{Adv}_2^{\text{FB-PCH}}\right| \leq \mathtt{Adv}_{\mathcal{S}}^{\text{FB-ABE}}(\lambda). \tag{3}$$

– $\mathbb{G}_3$: 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 $(id^*, \mathsf{h}^*, m^*, \mathsf{r}^*, m'^*, \mathsf{r'}^*, C^*, t^*)$. Below we show that the difference between $\mathbb{G}_2$ and $\mathbb{G}_3$ is negligible if the CH scheme is collision resistant.
Let $\mathcal{S}$ denote an attacker against CH, who is given a chameleon public key $\mathtt{pk}^*$, 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 the chameleon public key of the g-th query as $\mathsf{h}^* = \mathtt{pk}^*$, and completes the remainder of Setup honestly.
- $\mathcal{S}$ can answer all adapt queries at different time made by $\mathcal{A}$ by choosing different trapdoors. For the g-th hash query, $\mathcal{S}$ simulates $(\mathsf{h}^*, \mathsf{r}, b, C)$ as the response to the hash oracle on the hashed message m, where chameleon hash value $b = g^m \cdot \mathsf{h}^{*\mathsf{r}}$, and ciphertext C encrypts "0". For the g-th adapt query, $\mathcal{S}$ obtains a new randomness r' from his adapt oracle and returns $(m', \mathsf{h}^*, \mathsf{r}', b, C)$ to $\mathcal{A}$.
- If $\mathcal{A}$ outputs a collision $(id^*, m^*, \mathsf{r}^*, m'^*, \mathsf{r'}^*, \mathsf{h}^*, b^*, C^*, t^*)$, and the chameleon hash is verified as valid (i.e., $b^* = g^{m^*} \cdot \mathsf{h}^{*\mathsf{r}^*} = g^{m'^*} \cdot \mathsf{h}^{*\mathsf{r'}^*}$), $\mathcal{S}$ outputs $(m^*, \mathsf{r}^*, m'^*, \mathsf{r'}^*, \mathsf{h}^*)$ as a collision to the CH scheme; otherwise, $\mathcal{S}$ aborts the game. Therefore, we have

$$\left|\mathtt{Adv}^{\text{FB-PCH}} - \mathtt{Adv}^{\text{FB-PCH}}\right| \leq \mathtt{Adv}_{\mathcal{S}}^{\text{CH}}(\lambda). \tag{4}$$

Combining the above results together, we have

$$\mathtt{Adv}_{\mathcal{A}}^{\text{FB-PCH}}(\lambda) \leq q \cdot \mathcal{T} \cdot (\mathtt{Adv}_{\mathcal{S}}^{\text{FB-ABE}}(\lambda) + \mathtt{Adv}_{\mathcal{S}}^{\text{CH}}(\lambda)).$$

Theorem 4. *The FB-PCH is indistinguishable if the DL-based CH scheme is indistinguishable, and the FB-ABE scheme is semantically secure.*

Proof. We define a sequence of games $\mathbb{G}_i$, $i = 0, \cdots, 3$ and let $\mathtt{Adv}_i^{\text{FB-PCH}}$ be the advantage of the adversary in game $\mathbb{G}_i$. We assume that $\mathcal{A}$ issues at most q hash queries at each game.

- $\mathbb{G}_0$: This is original game for indistinguishability.
- $\mathbb{G}_1$: 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 challenge query at time t^* in the challenge phase. $\mathcal{S}$ will output a random bit if $\mathcal{A}$'s attacking query does not occur in the g-th query. Since the upper-bound of time is $\mathcal{T}$, we have
$$\mathtt{Adv}_0^{\text{FB-PCH}} = q \cdot \mathcal{T} \cdot \mathtt{Adv}_1^{\text{FB-PCH}} \tag{5}$$
- $\mathbb{G}_2$: This game is identical to game $\mathbb{G}_1$ except that in the g-th query, $\mathcal{S}$ replaces the encrypted trapdoor tr^* in C^* by $\perp$ (i.e., empty value). By using the same security analysis as described in the above game $\mathbb{G}_2$, we have
$$\left|\mathtt{Adv}_1^{\text{FB-PCH}} - \mathtt{Adv}_2^{\text{FB-PCH}}\right| \leq \mathtt{Adv}_{\mathcal{S}}^{\text{FB-ABE}}(\lambda). \tag{6}$$
- $\mathbb{G}_3$: This game is identical to game $\mathbb{G}_2$ except that in the g-th query, $\mathcal{S}$ directly hashes a message m, instead of calculating the chameleon hash and randomness (h, r) using the trapdoor tr. Below we show the difference between $\mathbb{G}_2$ and $\mathbb{G}_3$ is negligible if the CH scheme is indistinguishable.
Let $\mathcal{S}$ denote an attacker against CH, who is given a chameleon public key $\mathtt{pk}^*$ and a $\mathsf{HashOrAdapt}$ oracle, aims to break the CH's indistinguishability. $\mathcal{S}$ generates master key pair honestly. $\mathcal{S}$ sets the chameleon public key of the g-th query as $\mathsf{h} = \mathtt{pk}^*$.
In the g-th query, if $\mathcal{A}$ submits $(id, m_0, m_1, \Lambda, \delta, t)$ to $\mathcal{S}$, $\mathcal{S}$ obtains a chameleon hash $(\mathsf{h}_w, \mathsf{r}_w)$ from his $\mathsf{HashOrAdapt}$ oracle on messages (m_0, m_1). Then, $\mathcal{S}$ honestly generates a ciphertext C according the protocol's description (note that $\mathcal{S}$ sets the encrypted trapdoor as $\perp$). Eventually, $\mathcal{S}$ returns $(m_w, \mathsf{h}_w, \mathsf{r}_w, b, C)$ 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
$$\mathtt{Adv}_{\mathcal{A}}^{\text{FB-PCH}}(\lambda) \leq \mathtt{Adv}_{\mathcal{S}}^{\text{CH}}(\lambda)).$$

Combining the above results together, we have
$$\mathtt{Adv}_{\mathcal{A}}^{\text{FB-PCH}}(\lambda) \leq q \cdot \mathcal{T} \cdot (\mathtt{Adv}_{\mathcal{S}}^{\text{FB-ABE}}(\lambda) + \mathtt{Adv}_{\mathcal{S}}^{\text{CH}}(\lambda)).$$