ReTRACe: Revocable and Traceable Blockchain Rewrites using Attribute-based Cryptosystems

Gaurav Panwar[*]
New Mexico State University
Las Cruces, NM, USA
gpanwar@nmsu.edu

Roopa Vishwanathan[*]
New Mexico State University
Las Cruces, NM, USA
roopav@nmsu.edu

Satyajayant Misra
New Mexico State University
Las Cruces, NM, USA
misra@cs.nmsu.edu

ABSTRACT

In this paper, we study efficient and authorized rewriting of transactions already written to a blockchain. Mutable transactions will make a fraction of all blockchain transactions, but will be a necessity to meet the needs of privacy regulations, such as the General Data Protection Regulation (GDPR). The state-of-the-art rewriting approaches have several shortcomings, such as lack of user anonymity, inefficiency, and absence of revocation mechanisms. We present ReTRACe, an efficient framework for blockchain rewrites. ReTRACe is designed by composing a novel revocable chameleon hash with ephemeral trapdoor scheme, a novel revocable fast attribute based encryption scheme, and a dynamic group signature scheme. We discuss ReTRACe, and its constituent primitives in detail, along with their security analyses, and present experimental results to demonstrate the scalability of ReTRACe.

ACM Reference Format:
Gaurav Panwar, Roopa Vishwanathan, and Satyajayant Misra. 2021. ReTRACe: Revocable and Traceable Blockchain Rewrites using Attribute-based Cryptosystems. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies (SACMAT '21), June 16–18, 2021, Virtual Event, Spain.* ACM, New York, NY, USA, 28 pages. https://doi.org/10.1145/3450569.3463565

1 INTRODUCTION AND RELATED WORK

In access control techniques, revocation is a problem that is easily motivated from a practical standpoint. We present ReTRACe, a system for performing access control including revocation for transaction rewrites in blockchains. A blockchain is an append-only ledger to which entities can post messages in a decentralized manner. A message could be a financial transaction, a smart contract, or any data that needs to be shared among several users, but whose provenance needs to be verified. At a high level, a *block* is a collection of multiple messages (also called transactions), and their hash digests. Usually, once a message has been written to the blockchain, the message is considered immutable and cannot be edited.

While blockchain edits or rewrites are not required in all applications, there is an important class of applications where editing messages written onto a blockchain is essential. For instance, in the European Union (EU), the general data protection regulation (GDPR), Chapter 2, Article 17: *Right to erasure* and Article 19: *Notification obligation regarding rectification or erasure of personal data or restriction of processing*, gives users or *data subjects* the right to request that their personal data be erased/edited by the person or entity collecting or publishing their data, per their request. The U.S. state of California passed the California Consumer Privacy Act (CCPA) in 2018 [17], which has codified similar privacy rights, as described in Article 2. 999.308 (c) (2).

Blockchain technology has widespread applications in healthcare, regulatory compliance and audit, record management, Internet of Things, and more [16]. It is easy to envisage examples where a user's sensitive data is part of the committed blockchain transactions, and at a later point needs to be erased. For example, a global consortium of banks is currently using a blockchain platform (R3 [44]) to manage financial agreements, securities trading, etc. These transaction records will include clients' information and could potentially contain personally identifiable information.

As per the individual's "right to forget" (e.g., in GDPR), a user can request a purge of their identification data from the blockchain, (true even with encrypted data), and should be able to independently verify the said purge. Also, when a blockchain is used for record keeping and auditing the actions of a set of mutually untrusting parties, there may be situations where a non-monetary record needs to be expunged from the blockchain, e.g., offensive content, leaked personal information/encryption keys, etc., and companies have prototyped editable blockchains for addressing this [50]. The U.S. Department of Homeland Security in a recent report on the use of blockchains in government, has judged permissioned blockchains to be useful for maintaining government records, supply chain monitoring, and government approval chain processes [29], which will encourage blockchain adoption. In all such applications, there might arise need for correcting/updating transactions. *Motivated by this, we concentrate on a permissioned blockchain.*

Recently, some solutions have been proposed to enable modifications on data posted to a blockchain [27]. One method is a *hard fork*, which involves diverging the blockchain at the point where a message needs to be changed, creating a new forked blockchain, and invalidating all subsequent blocks in the old blockchain. Another technique is to modify a message m to m', post m' pointing to m on the blockchain. In addition to being inefficient, these techniques do not expunge the old message from the chain, and do not enable fine-grained control over who can modify messages.

Ateniese *et al.* [2, 50] proposed a solution for blockchain edits by rewriting entire blocks, using *chameleon hash functions.* Deuber *et al.* [28] and Reparo [46] proposed mechanisms for block-level rewrites where any user can propose a redaction or an edit of a

[*]Authors contributed equally.

SACMAT '21, June 16–18, 2021, Virtual Event, Spain
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8365-3/21/06…$15.00
https://doi.org/10.1145/3450569.3463565

block, which is then voted upon by other users, and the edit is accepted if it wins a majority vote. Coarse block-level solutions result in needlessly rewriting an entire block, when only say, a single transaction in a block needs to be expunged, nor do they provide fine-grained transaction level control by helping us set permissions about who can rewrite individual transactions. In this paper, we focus on *transaction-level rewriting*. Our intent is to build a system where a transaction can be updated, if needed, at which point only the updated transaction will be visible on the blockchain.

1.1 Related Work

Recently, chameleon hash functions have been proposed to enable blockchain rewrites. Chameleon hash functions [38] enable a user to find collisions in the domain of a hash function, such that several pre-images can be created that map to the same hash digest. The process of finding a collision on a given message, termed *adapting* the message, is done using a *trapdoor* associated with the digest. For increased security and flexibility, Camenisch *et al.* [18] proposed the idea of two trapdoors being associated with a digest, a permanent *long-term trapdoor*, and an *ephemeral trapdoor*, which is chosen per message; a message cannot be adapted without knowing *both* trapdoors. Derler *et al.* [27] presented an application of [18] to transaction-level blockchain rewrites, where the ephemeral trapdoor associated with a digest of a message posted on the blockchain is encrypted using ciphertext-policy attribute-based encryption (CPABE), and only users possessing enough attributes that satisfy the policy can decrypt the ephemeral trapdoor and adapt a message.

The problem with the above schemes is that access to the ephemeral trapdoor, once issued, cannot be revoked, i.e., once given out, the ephemeral trapdoor is accessible to users in perpetuity. Ideally, we would like to revoke users, such that upon revocation of a user u who possesses the ephemeral trapdoor, the ephemeral trapdoor is swiftly updated to a value unknown to u. We use chameleon hashes, attribute-based encryption (ABE) and a dynamic group signature scheme (GSS) to achieve our goal.

1.2 Challenges in the State-of-the-art

Motivated by the problem of performing transaction rewrites on a blockchain, we seek to answer the following questions: Who gets to erase or overwrite transactions/messages? What if a user is allowed to update a given message only for a short period of time? Once a message has been modified, should the identity of the modifier be revealed, and if yes, to whom? These questions need to be addressed to make editable transactions usable, which is our goal in this paper. We focus on three important challenges: 1) *Revoking access to ephemeral trapdoors*: a revoked user cannot update a message even if she has a local copy of the long-term and ephemeral trapdoors. 2) *Revoking attributes from users*: a user who no longer has access to an attribute secret key cannot update a message and/or trapdoor. 3) *Traceability*: a user who anonymously rewrote a particular message, but violated the rewriting policy can be identified. For addressing the first two challenges, we need to design new cryptographic primitives, since existing ones do not provide the properties we require.

1.3 Our Contributions

Our contributions in this paper include:
1) Design of a new *revocable chameleon hash with ephemeral trapdoor scheme*, RCHET, which guarantees that a revoked user cannot adapt a blockchain message or trapdoor.
2) Design of a new, efficient *revocable ABE scheme*, RFAME, to control access to the ephemeral trapdoor.
3) Design of a *revocable and traceable blockchain rewriting framework*, ReTRACe, using our novel RCHET and RFAME schemes, and a dynamic group signature scheme as building primitives. In ReTRACe, authorized users can adapt messages, using the ephemeral trapdoor of a message's chameleon hash digest. Access to the ephemeral trapdoor can be revoked instantly as needed. Authorized users can *anonymously* post to and adapt messages on a blockchain, but their identities can be unmasked by legitimate oversight authorities if needed.
4) Implementation of RCHET, RFAME, and ReTRACe to demonstrate scalability, and their security analyses.

We note that ReTRACe does not modify the way miners communicate with one another and how the blocks are mined in a permissioned blockchain adapted with ReTRACe's functionality (except transaction verification). Both the ReTRACe messages and non-ReTRACe messages can co-exist in the underlying permissioned blockchain, i.e., after integrating ReTRACe into a permissioned blockchain, the resultant chain can accept mutable (messages with trapdoors controlled with ABE policies) as well as regular immutable messages such as financial transactions and records of payments between different entities.
Organization: In Section 2, we discuss the constituents of our system model, and cover preliminaries and assumptions. In Section 3, we give a short technical overview of ReTRACe. In Sections 4 and 5, we introduce RCHET and RFAME, their definitions, security analyses, and constructions. In Section 6 we briefly discuss dynamic GSS schemes, and in Section 7, we give the definition, security analysis, and construction of ReTRACe. In Section 8 we present our experimental analysis, and Section 9 concludes the paper.
Since ReTRACe has several building blocks, we need to make careful choices on what is included in the main paper and the Appendix. We present our new ideas, constructions, and experiments in the main paper, and include formal definitions and proofs of our constructions in the Appendix.

2 SYSTEM MODEL AND THREAT MODEL

ReTRACe is constructed using three primitives: 1) a revocable chameleon hash scheme, RCHET, which provides a dynamic and mutable ephemeral trapdoor required for updating a message posted on the BC, 2) a revocable attribute-based encryption scheme, RFAME used to control access to the ephemeral trapdoor, and 3) a dynamic group signature scheme (DGSS), which helps legitimate users knowing the current ephemeral trapdoor, to sign message updates anonymously before posting them to the blockchain. Both, RFAME and DGSS are associated with policies. In this section, we discuss the parties involved in ReTRACe, and the policy structures that control their ability to update messages. Table 1 presents important notations used.

Naming Conventions: In our discussions, we refer to the blockchain as BC, we use *message* to mean a pre-image of the chameleon hash function, which is posted on the BC as a transaction, and when we say "trapdoor", we mean the ephemeral trapdoor of the chameleon hash, unless otherwise specified. A chameleon hash function's output has mutable and immutable parts, the immutable part contains the digest of the hash function, the mutable part includes the trapdoor needed for creating a message collision, among other things.

Table 1: Notations

Variable	Definition
λ	Security parameter
AIA	Attribute-issuing authority
Υ_{ABE}	Attribute Based Encryption (ABE) policy for regular users
Υ_{GS}	Group Signature Scheme (DGSS) policy for regular users
Υ_{info}	Policy consisting of $(\Upsilon_{\text{ABE}}, \Upsilon_{\text{GS}})$
$\Upsilon_{ABE\text{admin}}$	ABE policy for administrators
$\Upsilon_{GS\text{admin}}$	DGSS policy for administrators
Υ_{admin}	Policy consisting of $(\Upsilon_{ABE\text{admin}}, \Upsilon_{GS\text{admin}})$
ξ_{msg}	Set of signatures over a message
$\xi_{\Gamma_{\text{info}}}$	Set of signatures over an ephemeral trapdoor
M	Matrix representing a monotone span program
ρ	Mapping function from row numbers of M to attributes
$\text{mpk}_{\text{ABE}}, \text{msk}_{\text{ABE}}$	Public, secret key pair of an AIA
U	Universe of attributes
Γ_{info}	Ephemeral trapdoor for a chameleon hash
pk_{CH}, sk_{CH}	Public key and long term trapdoor of a chameleon hash
$DGSS$	Dynamic group signature scheme
GM, TM	Group manager and tracing manager
RCHET	Revocable chameleon hash with ephemeral trapdoor scheme
RFAME	Revocable fast attribute-based message encryption scheme
BC	Blockchain

2.1 Policies

ReTRACe has four kinds of policies, all represented as Boolean predicates: 1) A trapdoor associated with the digest of a given message is encrypted under an ABE policy, Υ_{ABE}. 2) The policy Υ_{GS}, spells out certain DGSS groups from which users can anonymously sign a message update, and post it on the BC. 3) For revoking access to the trapdoor, we define a policy, $\Upsilon_{ABE\text{admin}}$, that sets forth who can create an updated ephemeral trapdoor, and encrypt it under a new policy, Υ'_{ABE}. 4) Finally, $\Upsilon_{GS\text{admin}}$ governs who can change Υ_{GS}, and thus exclude users of certain DGSS groups from producing valid signatures.

For simplicity of exposition, we assume that the $\Upsilon_{ABE\text{admin}}$ and $\Upsilon_{GS\text{admin}}$ policies, once set, cannot be changed (which can be relaxed on an as-needed basis by setting up a higher level of control). We stress that it is the $\Upsilon_{ABE\text{admin}}$ and $\Upsilon_{GS\text{admin}}$ *policy clauses* that are immutable; our system allows for the *set of people* satisfying them to be dynamic and ever-changing.

Unlike ABE schemes, DGSS do not have the concept of policies built into them; we introduce this notion for ReTRACe. We define a DGSS policy as a publicly-known Boolean predicate linked to a message m that specifies which groups can produce valid signatures on m, e.g., $(m, \Upsilon_{\text{GS}} =$ "Admin" AND "Payroll"), indicates users belonging to groups "Admin" and "Payroll" can produce valid signatures on m. In case of Boolean predicates with conjunctive clauses, the signatures (σ) and corresponding group public keys (gpk) are collected into a set, ξ_m, where $\xi_m = (\sigma_{\text{Admin}}, gpk_{\text{Admin}}), (\sigma_{\text{Payroll}}, gpk_{\text{Payroll}})$. Any public verifier can check the validity of a set of signatures for a message w.r.t. a given policy.

2.2 Parties

The parties involved in ReTRACe are categorized into:
1) Originator: The originator creates a message, its digest and trapdoor, and sets the four policies, $\Upsilon_{\text{ABE}}, \Upsilon_{\text{GS}}, \Upsilon_{ABE\text{admin}}$, and $\Upsilon_{GS\text{admin}}$ that regulate future message updates.
2) Set of Authorized Users, AuthU: The set of users who can create a collision on a message posted on the BC by the originator (**AuthU** could include the originator) as long as they possess enough attributes that satisfy Υ_{ABE}, and are a member of a DGSS group satisfying Υ_{GS}.
3) Set of Authorized User Administrators, AuthUAdmins: This is the set of users who can modify the trapdoor, as well as create a collision on the message (**AuthUAdmins** could include the originator), as long as they possess enough attributes to satisfy $\Upsilon_{ABE\text{admin}}$ and are a member of a DGSS group satisfying $\Upsilon_{GS\text{admin}}$.
4) Attribute-issuing authority and Group Manager: The attribute-issuing authority (AIA) for the ABE scheme, and the Group Manager (GM) for the DGSS issue keys to their respective users. For presentation clarity, we use a single AIA and GM, but in practice, more can be used easily in ReTRACe if design warrants.

2.3 Blockchain Operations

A ReTRACe transaction consists of a mutable part (plaintext message, policies, trapdoor, ciphertexts, signatures, etc.) and an immutable part (digest). When a ReTRACe transaction is included in a block by the miners, only the immutable part of the ReTRACe transaction is used in the Merkle tree of the given block instead of the hash of the whole transaction. This makes it possible to update a ReTRACe transaction in the future as long as the mutable part of the updated transaction verifies with the immutable digest on the BC.

ReTRACe is independent of the kind of underlying BC system including the consensus mechanism, e.g., Proof of Work/Stake, as long as the security requirements for ReTRACe defined in Section 2.4 are met. For a user to be able to post any ReTRACe messages in the BC, the user needs to be previously onboarded with a AIA and GM in the given ReTRACe system.

To use ReTRACe with a given BC system, the hash verification function of the BC needs to be modified to perform the ReTRACe messages' verification (miner verification, and public verification of the ReTRACe messages). But ReTRACe does not leak sensitive information to the miners hence the miners are not onboarded by the AIA and GM. Adding or removing miners in the system using ReTRACe works in the same as any other BC system. The miners just have some extra steps when verifying updates to ReTRACe transactions, but none of those steps involve the miners learning privileged information about the messages posted on the BC.

Each block on a ReTRACe-modified BC could have mutable as well as immutable transactions and the resultant hash of the block will always be immutable as per the rules of traditional BCs. The verification algorithm can verify mutable as well as immutable transactions, thus ReTRACe can be implemented on a blockchain which supports both, mutable and immutable transactions. Note that once a transaction is written to the BC as immutable, it cannot be modified, only transactions which were originally posted as ReTRACe transactions with a trapdoor and access policy are updatable. The AIA and GM are publicly available to all users to sign up with, if any user needs to post mutable transactions on the BC, user do not need to sign up with the AIA/GM if they do not want to post immutable transactions.

2.4 Trust Assumptions and Threat Model

We make a few modest trust assumptions in ReTRACe. As in most BC-enabled systems, we assume the consensus protocol being used ensures honest operation by miners. We assume the AIA will honestly generate attributes and secret keys; one could relax this assumption by using multi-AIA techniques of [20, 39], where components of users' secret keys are generated by multiple AIAs. Similarly for the DGSS scheme, we assume that the GM for a group will issue signing keys properly, and trace users honestly. We could dilute this assumption by using techniques of Bootle *et al.* [15], by introducing a *tracing manager*, and separating the roles of group and tracing managers. These relaxations could be applied on as as-needed basis, we do not discuss them for narrative simplicity.

Threat Model and Security Goals: Our main goal is to protect against adversaries that have no access to the long-term trapdoor and/or current version of the ephemeral trapdoor for a given message, m, posted on the BC, do not satisfy the policies associated with m, yet who try to update m or its trapdoors. A second goal is to protect the privacy of, and provide anonymity to, the individuals who post a message or update a message and/or its trapdoor on the BC. The only way for an adversary to violate our goals is to break the security of our cryptographic constructs. We do not consider network attacks (e.g., eclipse attacks, traffic analysis, etc.) in this paper, prior works [9, 43] that focus on them can be used in conjunction with ReTRACe.

2.5 Computational Assumptions

The security of ReTRACe is derived from time-tested, well-regarded assumptions based on the Discrete Log problem, the Decision Linear problem (DLIN), and the Decisional Diffie-Hellman (DDH) problem. We model ABE access control policies as Boolean formulas with AND and OR gates, where each input is associated with an attribute, and the Boolean formulas are represented as monotone span programs, as is common in the literature.

3 ReTRACe SYSTEM OVERVIEW

In this section, we give a high-level, brief technical overview of ReTRACe. Without loss of generality, let us consider a single AIA and GM in the system. Let $[1..n]$ represent a set of users, the AIA issues sets of secret keys, $\mathbb{SK}_1, \ldots, \mathbb{SK}_n$, and the GM issues sets of signing keys, $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ to the n users. Let $\mathsf{mpk}_{\mathsf{ABE}}$ and gpk denote the public keys of the ABE and DGSS schemes respectively.

Let us consider a user, u who creates a message, m, to be posted on the BC. User u creates an *adaptable* (i.e., updatable) trapdoor τ, that enables future modifications of m (using our novel RCHET scheme), sets a policy, Υ_{ABE}, which defines the set of authorized users, **AuthU**. User u encrypts τ with $\mathsf{mpk}_{\mathsf{ABE}}$ (using our novel revocable ABE scheme, RFAME), $E_{\mathsf{mpk}_{\mathsf{ABE}}}(\tau, \Upsilon_{\mathsf{ABE}}) \rightarrow X$. For controlling who can update τ in the future, u picks an $r \leftarrow\!\!\$\ \mathbb{Z}_q$ (where $\mathbb{G}$ and $q = |\mathbb{G}|$ are part of the public parameters, and will be used in the cryptographic operations of ReTRACe). User u then sets the $\Upsilon_{ABE\mathrm{admin}}$ that defines the set of authorized user administrator(s), **AuthUAdmins**, and computes $E_{\mathsf{mpk}_{\mathsf{ABE}}}(r, \Upsilon_{ABE\mathrm{admin}}) \rightarrow X_r$. A user in **AuthU** updates m during a message adaptation, a user in **AuthUAdmins** updates m, X and X_r during a trapdoor update.

A user in **AuthUAdmins** that satisfies $\Upsilon_{ABE\mathrm{admin}}$ can obtain r, prove knowledge of r to the miner(s), and update the trapdoor τ. User u also sets the DGSS policies, Υ_{GS}, and $\Upsilon_{GS\mathrm{admin}}$ that stipulate only members of **AuthU** and **AuthUAdmins** are authorized to produce valid anonymous signatures on an updated message and updated trapdoor, respectively, before posting to the BC. Finally u posts tuple $t = (m, X, X_r, \Upsilon_{\mathrm{info}} = (\Upsilon_{\mathsf{ABE}}, \Upsilon_{\mathsf{GS}}), \Upsilon_{\mathrm{admin}} = (\Upsilon_{ABE\mathrm{admin}}, \Upsilon_{GS\mathrm{admin}}))$, along with a signature on t to the BC. We assume standard techniques such as nonces/timestamps to prevent replay attacks are used.

Any user $i \in [1..n]$, s.t. $i \in$ **AuthU** whose secret key set $\mathbb{SK}_i \in \{\mathbb{SK}_1, \ldots, \mathbb{SK}_n\}$ satisfies Υ_{ABE}, can decrypt X, obtain τ, and update m to m' (using RCHET). Note that being able to satisfy Υ_{ABE} only allows i to decrypt the trapdoor, τ, and update m, but not update τ. User i will produce a signature on m' using sk_i that satisfy Υ_{GS}, and post m' and the signature on the BC.

The hash of a given transaction only corresponds to the m contained in it. The miner's verification function in ReTRACe ensures that only members of **AuthU** can update m and C_1, and only members of **AuthUAdmins** have permission to update m, C_1 and C_2. The miner's in ReTRACe do not need any extra or privileged information other than what is already available to the rest of the system as part of the public parameters of ReTRACe, hence they *do not* need to be onboarded with the AIA or GM in the system.

Revocation of users from **AuthU** is handled by either a member of **AuthUAdmins** updating Υ_{ABE} to Υ'_{ABE} (RFAME), or by the AIA/GM revoking individual users. Any user $j \in [1..n]$, s.t. $j \in$ **AuthUAdmins** whose secret key set $\mathbb{SK}_j \in \{\mathbb{SK}_1, \ldots, \mathbb{SK}_n\}$ satisfies $\Upsilon_{ABE\mathrm{admin}}$, can decrypt X_r, obtain r, update τ to τ' (using RCHET), such that τ' will only be decryptable by non-revoked users (using RFAME for access control). User j will compute $E_{\mathsf{mpk}_{\mathsf{ABE}}}(\tau', \cdot) \rightarrow C'_1$, j will prove knowledge of r to the miner, thus proving it can satisfy $\Upsilon_{ABE\mathrm{admin}}$, and is a member of **AuthUAdmins**. Then j will sign and post m' and X' on the BC. The DGSS signature will be produced using j's set of signing keys, sk_j, that satisfy $\Upsilon_{GS\text{admin}}$.

We discuss the details and subtleties of ReTRACe in Section 7, including some corner cases in Remark 7.1; in what follows, we describe the building primitives. We also present two pertinent use cases of ReTRACe in Appendix 10.2—smart contracts and financial services.

4 REVOCABLE CHAMELEON HASH WITH EPHEMERAL TRAPDOORS

We envisioned a revocable CHET scheme, where the long-term trapdoor remains permanent, but access to the ephemeral trapdoor can be revoked at will. Intuitively, for performing revocation, we update the ephemeral trapdoor, and prevent the revoked user from accessing the updated trapdoor.

Definition 4.1. *(Revocable chameleon hash with ephemeral trapdoor (*RCHET*) scheme):*
(1) $\underline{\mathsf{RCHET.systemSetup}(1^\lambda) \rightarrow (\mathsf{pubpar})}$*: This algorithm on input a security parameter outputs the public parameters of the system. We assume* pubpar *is implicitly passed as input to all other algorithms.*
(2) $\underline{\mathsf{RCHET.userKeySetup}(1^\lambda) \rightarrow (sk_{ch}, pk_{ch})}$*: This algorithm on input the public parameters returns a long-term trapdoor, sk_{ch}, and a public key.*
(3) $\underline{\mathsf{RCHET.cHash}(sk_{ch}, pk_{ch}, m) \rightarrow \{(\mathsf{digest}, \mathsf{rand}, \Gamma_{\mathsf{pubinfo}}, \Gamma_{\mathsf{privinfo}}), \perp\}}$*: This algorithm takes as input the long-term trapdoor, the public key, and a message m. If successful, it outputs a* digest*, randomness,* rand*, which will be used in verifying the digest, and an ephemeral trapdoor consisting of public/private parts,* $(\Gamma_{\mathsf{pubinfo}}, \Gamma_{\mathsf{privinfo}})$.
(4) $\underline{\mathsf{RCHET.verifyTransaction}(pk_{ch}, m, \mathsf{digest}, \mathsf{rand}, \Gamma_{\mathsf{pubinfo}}) \rightarrow \{0,1\}}$*: This algorithm takes as input a public key (pk_{ch}), message m,* digest*,* rand*, public part of the ephemeral trapdoor, and returns 1 if the* digest *is correct.*
(5) $\underline{\mathsf{RCHET.adaptMessage}(sk_{ch}, m, m', \mathsf{digest}, \mathsf{rand}, \Gamma_{\mathsf{pubinfo}}, \Gamma_{\mathsf{privinfo}}) \rightarrow \{\mathsf{rand}', \perp\}}$*: This algorithm takes as input the long-term trapdoor, a message m that needs to be adapted to m', the* digest*,* rand*, and the ephemeral trapdoor* $(\Gamma_{\mathsf{pubinfo}}, \Gamma_{\mathsf{privinfo}})$*. If successful, it outputs an updated* rand'.
(6) $\underline{\mathsf{RCHET.adaptTrapdoor}(sk_{ch}, m, m', \mathsf{digest}, \mathsf{rand}, \Gamma_{\mathsf{pubinfo}}, \Gamma_{\mathsf{privinfo}}) \rightarrow \{(\mathsf{rand}', \Gamma'_{\mathsf{pubinfo}}, \Gamma'_{\mathsf{privinfo}}), \perp\}}$*: This algorithm takes in the long-term trapdoor, a message, m, a new message m',* digest*,* rand *and* $\Gamma_{\mathsf{pubinfo}}, \Gamma_{\mathsf{privinfo}}$*. If successful, it outputs an updated* rand' *and updated public/private parts of the ephemeral trapdoor,* $\Gamma'_{\mathsf{pubinfo}}, \Gamma'_{\mathsf{privinfo}}$.

We present our construction of an RCHET scheme in Figure 1. The security of our RCHET construction is based on the discrete log and DDH assumptions. At a high level, the idea is to hide the long-term and ephemeral trapdoors in the exponent of public parameters and use non-interactive zero knowledge proofs of knowledge (NIZKPoKs) to prove knowledge of them. In the construction, Π is an IND-CCA2 public key encryption scheme, which is used to encrypt randomness associated with the trapdoors. We overload the verification function for both, signatures and zero-knowledge proofs as verify, which will be clear from context. Since our construction uses an NIZKPoK, we require a common reference string (crs); we discuss ideas on how to generate the crs securely in Appendix 10.3.

4.1 RCHET Security Properties

The properties of indistinguishability, public and private collision resistance were introduced by Camenisch *et al.* [18] for CHET schemes. Derler *et al.* [27] retained the three properties, but strengthened their security definition by giving the adversary access to an oracle for adapting messages in the private collision resistance game, while [18] only gave adversary access to a hash oracle. We *further strengthen* the security properties by: 1) introducing the notion of *revocation collision resistance*, which any RCHET scheme must provide, and 2) giving the adversary access to oracles for *both*, adapting messages and adapting trapdoors.

Informally, *indistinguishability* requires that an outsider, given a random string, rand, cannot tell if rand was obtained by hashing the original message, a message update or a trapdoor update. *Public collision resistance* requires that a user who possesses neither the long-term nor ephemeral trapdoor, cannot find collisions by himself. *Private collision resistance* requires that even the holder of the long-term trapdoor cannot find collisions, as long as the ephemeral trapdoor is unknown to them. *Revocation collision resistance* requires that a user that knows both, the long-term trapdoor and ephemeral trapdoor, cannot find collisions after the ephemeral trapdoor has been updated, as long as the new ephemeral trapdoor is unknown to them. We formalize these security properties, and give the proof of the following theorem in Appendix 10.4.

Theorem 4.1. *If the discrete log assumption and DDH assumption hold in $\mathbb{G}$, H is collision resistant, Π is IND-CCA2 secure, and the NIZKPoKs satisfy completeness, simulation soundness, extractability and zero knowledge, then our revocable chameleon hash with ephemeral trapdoors scheme,* RCHET *shown in Figure 1 is secure.*

5 REVOCABLE ATTRIBUTE-BASED ENCRYPTION

In this section, we describe our revocable ciphertext policy attribute-based encryption (CPABE) scheme, RFAME, and discuss its efficiency and security properties. Most of the benchmark schemes in the ABE literature do not consider attribute revocation [33, 47–49]. FAME [1] is a state-of-the-art efficient ABE scheme that performs better in terms of qualitative and quantitative metrics, compared to prior works, and provides full IND-CPA security, although it does not discuss revocation. We use FAME as a starting point for designing an efficient *revocable CPABE scheme*, RFAME.

Revocation model: In ABE, there are broadly speaking, two possible kinds of revocation. One is *policy-level revocation*, where revocation entails deleting a clause from an ABE policy, e.g., Υ = "CS students" and "EE staff" can be updated to a more restrictive policy, Υ' = "CS students". The other is other is the more fine-grained *user-level revocation* which calls for revoking decryption rights of individual users, e.g., if we do not wish to update Υ, but revoke access of a member of staff from EE. We consider user-level revocation in this paper (although our scheme also supports modifiable policies, if needed). In Table 2 we qualitatively compare our scheme, RFAME, with previous revocable ABE schemes (identified by first author

RCHET Algorithms

a) $\mathsf{RCHET.systemSetup}(1^\lambda) \rightarrow (\mathsf{pubpar})$: This algorithm generates the public parameters of the system:

1. $(\mathbb{G}, g, q) \leftarrow \mathsf{GGen}(1^\lambda)$. GGen generates prime-order cyclic group $\mathbb{G}$, $g \in \mathbb{G}$, $q = |\mathbb{G}|$.

2. Pick H's key, $k \in \mathcal{K}$, and $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$, where $\mathcal{K}$ is the key-space of H. Set and return $\mathsf{pubpar} = (k, \mathbb{G}, g, q, \mathsf{crs})$. We assume pubpar is implicitly passed as input to all other algorithms.

b) $\mathsf{RCHET.userKeySetup}(1^\lambda) \rightarrow (sk_{ch}, pk_{ch})$: This algorithm generates the long-term trapdoor and a public key:

1. Pick $x \leftarrow \mathbb{Z}_q^*$, $h \leftarrow g^x$, $\pi_{pk} \leftarrow \mathsf{NIZKPoK}\{x : h = g^x\}$, generate keys $(SK, PK) \leftarrow \Pi.\mathsf{KeyGen}(1^\lambda)$.

2. Set $pk_{ch} = (PK, h, \pi_{pk})$ and $sk_{ch} = (SK, x)$. Return (sk_{ch}, pk_{ch}).

c) $\mathsf{RCHET.cHash}(sk_{ch}, pk_{ch}, m) \rightarrow \{(\mathsf{digest}, \mathsf{rand}, \Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}), \perp\}$: Creates a chameleon hash for a message m:

1. Check $\mathsf{verify}(\pi_{pk}, h) \overset{?}{=} 1$, if not, return $\perp$. Pick $r, \mathsf{etd}, d, \delta \leftarrow \mathbb{Z}_q^*$.

2. Compute $h' \leftarrow g^{\mathsf{etd}}, D \leftarrow g^d$, and $\Delta \leftarrow g^\delta$. Do $\pi_t \leftarrow \mathsf{NIZKPoK}\{\mathsf{etd} : h' = g^{\mathsf{etd}}\}, \pi_D \leftarrow \mathsf{NIZKPoK}\{d : D = g^d\}, \pi_\Delta \leftarrow \mathsf{NIZKPoK}\{\delta : \Delta = g^\delta\}$.

3. Generate hash of message to be posted, $a \leftarrow H_k(m)$ and create chameleon hash parameters: $\beta \leftarrow (r + \frac{\delta}{x} + \frac{d}{x}), p \leftarrow h^r, b \leftarrow p \cdot h'^a$. Do $\pi_p \leftarrow \mathsf{NIZKPoK}\{r : p = h^r\}$. Do $C \leftarrow \Pi.\mathsf{Encrypt}(PK, r)$, $C' \leftarrow \Pi.\mathsf{Encrypt}(PK, a)$.

4. Return $\mathsf{digest} = (b, h', \pi_t, C, C')$, $\mathsf{rand} = (\beta, p, \pi_p)$, $\Gamma_{\text{pubinfo}} = (\Delta, \pi_\Delta, D, \pi_D)$, $\Gamma_{\text{privinfo}} = (\delta, d, \mathsf{etd})$.

d) $\mathsf{RCHET.verifyTransaction}(pk_{ch}, m, \mathsf{digest}, \mathsf{rand}, \Gamma_{\text{pubinfo}}) \rightarrow \{0, 1\}$: This algorithm verifies the digest for a message m.

1. Check $\mathsf{verify}(\pi_{pk}, h) \overset{?}{=} 1$, $\mathsf{verify}(\pi_p, p) \overset{?}{=} 1$, $\mathsf{verify}(\pi_t, h') \overset{?}{=} 1$, $\mathsf{verify}(\pi_D, D) \overset{?}{=} 1$, and $\mathsf{verify}(\pi_\Delta, \Delta) \overset{?}{=} 1$.

2. Check $b \overset{?}{=} \frac{h^\beta \cdot h'^a}{D \cdot \Delta}$, where $a \leftarrow H_k(m)$. If check passes, return 1, else return 0.

e) $\mathsf{RCHET.adaptMessage}(sk_{ch}, m, m', \mathsf{digest}, \mathsf{rand}, \Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}) \rightarrow \{\mathsf{rand}', \perp\}$: This algorithm updates a message m:

1. Decrypt $a \leftarrow \Pi.\mathsf{Decrypt}(SK, C')$, check $a \overset{?}{\leftarrow} H_k(m)$. Check $b \overset{?}{=} \frac{h^\beta \cdot h'^a}{D \cdot \Delta}$. If checks fail, return $\perp$.

2. Check $h \overset{?}{=} g^x$, $p \overset{?}{=} g^{xr}$, $h' \overset{?}{=} g^{etd}$, $D \overset{?}{=} g^d$, and $\Delta \overset{?}{=} g^\delta$. If checks fail, return $\perp$.

3. Decrypt $r \leftarrow \Pi.\mathsf{Decrypt}(SK, C)$, if $r = \perp$, return $\perp$.

4. Compute $a' \leftarrow H_k(m')$. Compute $r' \leftarrow (\frac{rx + a \cdot \mathsf{etd} - a' \cdot \mathsf{etd} + \delta}{x})$.

5. Set $p' = h^{r'}$ and do $\pi_{p'} \leftarrow \mathsf{NIZKPoK}\{r' : p' = h^{r'}\}$. Compute $\beta' \leftarrow (r' + \frac{d}{x})$. Set and output $\mathsf{rand}' = (\beta', p', \pi_{p'})$.

f) $\mathsf{RCHET.adaptTrapdoor}(sk_{ch}, m, m', \mathsf{digest}, \mathsf{rand}, \Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}) \rightarrow \{(\mathsf{rand}', \Gamma'_{\text{pubinfo}}, \Gamma'_{\text{privinfo}}), \perp\}$: This algorithm modifies the trapdoor to an existing chameleon hash for a message m as follows:

1. Decrypt $a \leftarrow \Pi.\mathsf{Decrypt}(SK, C')$, check $a \overset{?}{\leftarrow} H_k(m)$. Check $b \overset{?}{=} \frac{h^\beta \cdot h'^a}{D \cdot \Delta}$. If checks fail, return $\perp$.

2. Check $h \overset{?}{=} g^x$, $p \overset{?}{=} g^{xr}$, $h' \overset{?}{=} g^{etd}$, $D \overset{?}{=} g^d$, and $\Delta \overset{?}{=} g^\delta$. If checks fail, return $\perp$.

3. Decrypt $r \leftarrow \Pi.\mathsf{Decrypt}(SK, C)$, if $r = \perp$, return $\perp$. Compute $a' \leftarrow H_k(m')$.

4. Pick $d', \delta' \leftarrow \mathbb{Z}_q^*$. Compute $D' \leftarrow g^{d'}, \Delta' \leftarrow g^{\delta'}$, do $\pi_{D'} \leftarrow \mathsf{NIZKPoK}\{d' : D' = g^{d'}\}$, $\pi_{\Delta'} \leftarrow \mathsf{NIZKPoK}\{\delta' : \Delta' = g^{\delta'}\}$.

5. Set $r' \leftarrow (\frac{rx + a \cdot \mathsf{etd} - a' \cdot \mathsf{etd} + \delta'}{x}), p' \leftarrow h^{r'}$. Compute $\beta' \leftarrow (r' + \frac{d'}{x})$, $\pi_{p'} \leftarrow \mathsf{NIZKPoK}\{r' : p' = h^{r'}\}$.

6. Prove knowledge of DDH tuple $(g, g^\delta, g^d, g^{\delta d})$. Set and output $\mathsf{rand}' = (\beta', p', \pi_{p'})$, $\Gamma'_{\text{pubinfo}} = (\Delta', \pi_{\Delta'}, D', \pi_{D'})$, and $\Gamma'_{\text{privinfo}} = (\delta', d', \mathsf{etd})$.

Figure 1: Construction of Revocable Chameleon Hash with Ephemeral Trapdoors (RCHET)

names). The key difference between RFAME and [22] is RFAME's efficient rekeying.

Intuition tells us that for user-level revocation, non-revoked users must be provided some information that the revoked user is not privy to, which would entail transmitting some new key material to non-revoked users, and some ciphertexts would need to be re-encrypted, to prevent the revoked user from decrypting them using their old keys. In RFAME, we achieve user-level revocation with minimal rekeyings/re-encryptions.

5.1 Definitions and Construction

Definition 5.1. *(Revocable ciphertext policy attribute-based encryption (*RFAME*) scheme)*
1) $\mathsf{RFAME.Setup}(1^\lambda, \mathbb{U}) \rightarrow (mpk, msk)$*: This algorithm takes as input the security parameter, the attributes in the universe,* $\mathbb{U}$*, and generates the master public and secret keys.*
2) $\mathsf{RFAME.KeyGen}(msk, \mathbb{S}) \rightarrow (sk_1, sk_2, \ldots, sk_{|\mathbb{S}|})$*: This algorithm takes in the master secret key, a set of attributes* $\mathbb{S}$*, and outputs secret keys for each attribute in* $\mathbb{S}$*.*
3) $\mathsf{RFAME.Encrypt}(mpk, m, \Upsilon) \rightarrow C$*: The encrypt algorithm takes in*

Table 2: Comparison of different revocable CPABE schemes

Scheme	Standard assumptions	Security	Application-specific	Policy/user-level revocation	Type-III pairings	Revocation list	Large universe
Bethencourt [10]	No (Generic group model)	Full	No	Poliy-level	No	No	Yes
Boldyreva [12], Attrapadung [3]	Yes (DBDH)	Selective	No	User-level	No	Yes	Yes
Yu [51]	Yes (DBDH)	Selective	Yes	User-level	No	Yes	Yes
Sahai [45]	No (Subgroups)	Full	No	Policy-level	No	Yes	Yes
Cui [23]	No (q-type)	Selective	Yes	User-level	No	Yes	No
Datta [26]	Yes (DLIN)	Full	No	User-level	No	Yes	Yes
Chow [22]	Yes (XDH)	Full	No	User-level	Yes	No	No
RFAME	Yes (DLIN)	Full	No	User-level	Yes	No	No

the master public key, a message to be encrypted, and an access policy, Υ. It outputs a ciphertext C.
4) RFAME.Decrypt$(sk_1, \ldots, sk_{|\mathbb{S}|}, C, \Upsilon) \rightarrow \{m, \perp\}$*: The decryption algorithm takes in the set of signing keys, a ciphertext tagged with a policy Υ, and outputs the message m if decryption is successful, else outputs $\perp$.*
5) RFAME.Revoke$(mpk, msk, uid, v) \rightarrow (mpk', msk', sk_v)$*: This algorithm takes in the master public and secret keys, a user uid with attribute v, who needs to be revoked. It returns the new master public and secret keys to AIA, and the new secret key, sk_v, for the non-revoked users possessing v.*

We give our construction of RFAME in Figure 2, whose security is based on the DLIN assumption. We walk the reader through a few initial steps of the decryption algorithm that will help in verifying correctness in Appendix 10.5. We now describe our intuitive ideas behind RFAME and its efficiency.

Efficient Rekeying in RFAME: Let us consider an AIA of an organization that issues four kinds of attributes, "Admin", "Payroll", "Benefits", and "Accounts". Let us assume there are y unique users possessing each attribute–a total of $4y$ users in the system, and that there are three messages in the system encrypted under different policies: *i)* Msg_1 is encrypted under $\Upsilon_{\mathsf{Msg}_1}$ = ("Admin" OR "Payroll"); *ii)* Msg_2 is encrypted under $\Upsilon_{\mathsf{Msg}_2}$ = ("Payroll" OR "Benefits"); *iii)* Msg_3 is encrypted under $\Upsilon_{\mathsf{Msg}_3}$ = ("Benefits" OR "Accounts").

Using current revocable CPABE schemes (e.g., [22]), if one user possessing the *Admin* attribute terminates employment, their secret key is revoked by the AIA, and the other $y-1$ *Admin* users get rekeyed. Msg_1 needs to be re-encrypted to prevent the revoked user from decrypting it. Since *Payroll* is part of the $\Upsilon_{\mathsf{Msg}_1}$ policy, all y users holding *Payroll* get rekeyed, as Msg_1 got re-encrypted. *Payroll* users getting rekeyed results in Msg_2 needing to be re-encrypted. Consequently, users holding *Benefits* and *Accounts* attributes need to be rekeyed, and Msg_3 needs to get re-encrypted. In total, we need to perform three re-encryptions and rekey $4y-1$ users, for a *single* user revocation. Our goal is to avoid such a domino effect.

At a high level, our idea is to associate each attribute, attr with some unique randomness, r, and embed r into the secret keys of all users who possess attr. When a user possessing attr needs to be revoked, we update the randomness to r' and reissue new secret keys with r' embedded in them only to the non-revoked users holding attr, and re-encrypt all ciphertexts whose policies involve attr. For facilitating this, the AIA can maintain a compact local table identifying which users possess a given attribute—a small storage cost in exchange for avoiding system-wide rekeying of users.

Furthermore, non-revoked users possessing attributes other than attr can still use their *current keys* to decrypt re-encrypted ciphertexts, which significantly reduces the number of users that need to be rekeyed, and the ciphertexts that need to be re-encrypted. Note that in [45], the authors propose a scheme that does not require re-encryptions, for *policy-level* revocation, with the restriction that a ciphertext can only be re-encrypted to a *more restrictive* policy. Our work is significantly more flexible, in that we perform *user-level* revocation, and do not impose any restrictions on policies. Also, [45] is proven only CPA-secure; ReTRACe requires a CCA-secure scheme.[1]

Thus, RFAME handles revocation more efficiently; when a user possessing *Admin* gets revoked, only the other $y-1$ users in *Admin* are rekeyed, and only Msg_1 needs to be re-encrypted to prevent the revoked user from decrypting it. The price we pay for this is that RFAME is a small-universe revocable CPABE scheme.

In summary, in the worst case, if x is the number of unique attributes in a system, y the number of users per attribute, then the overhead, using state-of-the-art revocation methods is $O(x)$ re-encryptions and $O(xy)$ rekeyings. In RFAME, the overhead is $\Theta(1)$ re-encryptions and $\Theta(y)$ rekeyings. We first prove RFAME CPA-secure, we later turn this into a CCA-secure scheme for ReTRACe. We give the IND-CPA game for RFAME and the proof of the following theorem in Appendix 10.6.

Theorem 5.1. RFAME *is fully IND-CPA secure under the DLIN assumption on Type III pairings in the random oracle model.*

6 DYNAMIC GROUP SIGNATURE SCHEMES

We use a group signature scheme for providing privacy and anonymity to users posting messages on the BC, yet retaining the ability to trace them if necessary. The group signature scheme can be easily

[1]CCA security in their scheme is essentially unachievable; since their scheme re-randomizes the ciphertext, an adversary can just submit the re-randomized challenge ciphertext back to the decryption oracle, thus trivially winning the CCA game.

RFAME Algorithms

a) RFAME.SetupABE$(1^\lambda, U) \rightarrow (\text{mpk}_{\text{ABE}}, \text{msk}_{\text{ABE}})$: The algorithm first generates the group parameters $(q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e, g, h)$, picks $a_1, a_2, b_1, b_2, p_1, p_2 \leftarrow_{\$} \mathbb{Z}_q^*$, $d_1, d_2, d_3 \leftarrow_{\$} \mathbb{Z}_q$. It picks $\alpha_y \leftarrow_{\$} \mathbb{Z}_q^*$, and computes h^{α_y} for each $y \in U$. It sets $\text{mpk}_{\text{ABE}} = (h, H_1 = h^{a_1}, H_2 = h^{a_2}, T_1 = e(g,h)^{p_1 d_1 a_1 + d_3}, T_2 = e(g,h)^{p_2 d_2 a_2 + d_3}, h^{\alpha_{y_1}}, \ldots, h^{\alpha_{y_{|U|}}})$, and sets $\text{msk}_{\text{ABE}} = (g, h, a_1, a_2, b_1, b_2, p_1, p_2, g^{d_1}, g^{d_2}, g^{d_3}, \alpha_{y_1} \ldots \alpha_{y_{|U|}})$.

b) RFAME.KeyGenABE$(\text{mpk}_{\text{ABE}}, \text{msk}_{\text{ABE}}, y_1, \ldots, y_{|U|}) \rightarrow SK$: The algorithm generates the secret keys for all attributes $y \in U$. Pick $r_1, r_2 \leftarrow_{\$} \mathbb{Z}_q$. Compute $\text{sk}_0 = (h^{b_1 r_1}, h^{b_2 r_2}, h^{r_1 + r_2})$.
For all $y \in U$ and $t \in \{1, 2\}$, pick $\sigma_y, \sigma' \leftarrow_{\$} \mathbb{Z}_q$, and compute:

$$\text{sk}_{y,t} = \mathcal{H}(y1t)^{\frac{b_1 r_1}{a_t + \alpha_y}} \cdot \mathcal{H}(y2t)^{\frac{b_2 r_2}{a_t + \alpha_y}} \cdot \mathcal{H}(y3t)^{\frac{r_1 + r_2}{a_t + \alpha_y}} \cdot g^{\frac{\sigma_y}{a_t + \alpha_y}} \cdot g^{\frac{\alpha_y}{a_t + \alpha_y}}; \text{sk}_{y,3} = (g^{-\alpha_y} \cdot g^{-\sigma_y})$$

$$\text{sk}'_t = \mathcal{H}(011t)^{\frac{b_1 r_1}{a_t}} \cdot \mathcal{H}(012t)^{\frac{b_2 r_2}{a_t}} \cdot \mathcal{H}(013t)^{\frac{r_1 + r_2}{a_t}} \cdot g^{\frac{\sigma'}{a_t}}; \text{sk}'_3 = (g^{d_3} \cdot g^{-\sigma'}), \text{sk}'' = g^{d_t p_t}$$

$$\text{Set and return } SK = (\text{sk}_0, \text{sk}_{y,1}, \text{sk}_{y,2}, \text{sk}_{y,3}, \text{sk}'_1, \text{sk}'_2, \text{sk}'_3, \text{sk}'')$$

c) RFAME.Encrypt$(\text{mpk}_{\text{ABE}}, msg, (\mathbf{M}, \rho)) \rightarrow C$. Pick $s_1, s_2 \leftarrow_{\$} \mathbb{Z}_q$. Let $\rho(i)$ denote a mapping to the attributes $i \in I$ that satisfy a given policy. Compute:

$$\text{ct}_{\rho(i),1} = H_1^{s_1} \cdot (h^{\alpha_{\rho(i)}})^{s_1} = h^{s_1(a_1 + \alpha_{\rho(i)})} \text{ and similarly } \text{ct}_{\rho(i),2} = h^{s_2(a_2 + \alpha_{\rho(i)})}, \text{ and set}$$

$$\text{ct}_0 = (\text{ct}_{0,1} = H_1^{s_1}, \text{ct}_{0,2} = H_2^{s_2}, \text{ct}_{\rho(i),1}, \text{ct}_{\rho(i),2}, \text{ct}_{0,3} = h^{s_1 + s_2})$$

Assume $\mathbf{M}$ has n_1 rows and n_2 columns. Then, for each row, $i \in [1..n_1]$ and $l = 1, 2, 3$, compute:

$$\text{ct}_{i,l} = \mathcal{H}(\rho(i)l1)^{s_1} \cdot \mathcal{H}(\rho(i)l2)^{s_2} \cdot \prod_{j=1}^{n_2} [\mathcal{H}(0jl1)^{s_1} \cdot \mathcal{H}(0jl2)^{s_2}]^{(\mathbf{M})_{i,j}}; \text{Set } \text{ct}' = (T_1^{s_1} \cdot T_2^{s_2} \cdot msg)$$

$$\text{Set and output } C = (\text{ct}_0, \text{ct}_{i,l} \, \forall \, i \in [1..n_1], l \in \{1, 2, 3\}, \text{ct}')$$

d) RFAME.Decrypt$(SK, C, (\mathbf{M}, \rho)) \rightarrow \{msg, \perp\}$: Parse C as $(\text{ct}_0, \text{ct}_{i,l} \, \forall \, i \in [1..n_1], l \in \{1, 2, 3\}, \text{ct}')$. For each row $\text{ct}_{i,l} \in \mathbf{M}$, pick coefficients $\gamma_i \in \{0, 1\}$ such that $\sum_{i \in I} \gamma_i (\mathbf{M})_i = [1, 0, \ldots, 0]$.

$$\text{num} = \text{ct}' \cdot e(\prod_{i \in I} \text{ct}_{i,1}^{\gamma_i}, \text{sk}_{0,1}) \cdot e(\prod_{i \in I} \text{ct}_{i,2}^{\gamma_i}, \text{sk}_{0,2}) \cdot e(\prod_{i \in I} \text{ct}_{i,3}^{\gamma_i}, \text{sk}_{0,3})$$

$$\text{den} = \prod_{i \in I} e(\text{sk}_{\rho(i),1}^{\gamma_i}, \text{ct}_{\rho(i),1}) \cdot \prod_{i \in I} e(\text{sk}_{\rho(i),2}^{\gamma_i}, \text{ct}_{\rho(i),2}) \cdot e(\text{sk}'_3 \cdot \prod_{i \in I} \text{sk}_{\rho(i),3}^{\gamma_i}, \text{ct}_{0,3}) \cdot \prod_{t \in \{1,2\}} e(\text{sk}'_t \cdot \text{sk}''_t, \text{ct}_{0,t})$$

e) RFAME.Revoke$(\text{mpk}_{\text{ABE}}, \text{msk}_{\text{ABE}}, v) \rightarrow (\text{mpk}_{\text{ABE}}', \text{msk}_{\text{ABE}}', SK')$: Let a user holding attribute $v \in U$ be revoked by the *AIA*. This algorithm is run by the *AIA* which generates new parameters for the non-revoked users of attribute group v, and updates its mpk_{ABE} and msk_{ABE}. It picks $\beta_v \leftarrow \mathbb{Z}_q^*$, and computes h^{β_v}. It updates $\text{mpk}_{\text{ABE}}' = (h, H_1, H_2, T_1, T_2, h^{\alpha_{y_1}}, \ldots, h^{\alpha_{y_{|U|-1}}}, h^{\beta_v})$. The msk_{ABE} remains the same except the α_v gets replaced with β_v. It then generates (a component of) the secret key for all *non-revoked* users possessing attribute v as follows:

$$\text{sk}_{v,t} = \mathcal{H}(v1t)^{\frac{b_1 r_1}{a_t + \beta_v}} \cdot \mathcal{H}(v2t)^{\frac{b_2 r_2}{a_t + \beta_v}} \cdot \mathcal{H}(v3t)^{\frac{r_1 + r_2}{a_t + \beta_v}} \cdot g^{\frac{\sigma_v}{a_t + \beta_v}} \cdot g^{\frac{\beta_v}{a_t + \beta_v}}; \text{sk}_{v,3} = (g^{-\beta_v} \cdot g^{-\sigma_v})$$

where $t \in \{1, 2\}$, and all other variables are as defined in the SetupABE and KeyGenABE algorithms. Set $SK' = (\text{sk}_0, \text{sk}_{v,t}, \text{sk}_{v,3}, \text{sk}'_1, \text{sk}'_2, \text{sk}'_3, \text{sk}'')$. SK' is distributed only to the non-revoked users who possess attribute v.

Figure 2: Construction of Revocable Fast Attribute Based Encryption (RFAME)

replaced with a regular signature scheme in ReTRACe if anonymity is not required in the system. Group signature schemes are based on three kinds of groups: static, semi-dynamic, and dynamic groups. Static groups do not support user addition or revocation [6], semi-dynamic groups support addition but not revocation [7], and dynamic groups allow addition and revocation [15]. We use a dynamic group signature scheme (DGSS) in ReTRACe, we do not construct a DGSS, as existing constructions [15, 40] provide the properties we need. ReTRACe is independent of the specifics of any DGSS construction.

We define a DGSS in Appendix 10.7. At a high level, there is a group manager (GM), who administers group membership, and a set of users who get their signing keys from the GM. There are eight algorithms: the probabilistic GSetup, GKGen, UpdateGroup, Sign, UserTrace are used for setting up group parameters, GM's keys, users' keys, signing a message, and tracing the signer of a message, respectively. The deterministic IsActive, VerifySignature, Judge are used

to check if a user is an active group member, to verify signatures, and to judge if the tracing procedure has been run correctly by the GM. Additonally, a DGSS also has an interactive Join protocol run between the GM and users.
Security of DGSS: A DGSS is said to be secure if it is: 1) mathematically *correct*, 2) provides *anonymity* (does not reveal the identity of a group member who produced signatures), 3) provides *traceability* (a group manager can trace all valid signatures to corresponding active members of the group), and 4) provides *non-frameability* (even if the rest of the group collude, they cannot generate a signature that is falsely attributed to a honest user who did not produce it).

7 THE ReTRACe FRAMEWORK

We first present the definition of ReTRACe, which describes, at an abstract level, the purpose of each constituent algorithm.

Definition 7.1. *(*ReTRACe *scheme)*
(1) ReTRACe.Keygen(1^λ) → (SecPar, PubPar)*: This algorithm takes in a security parameter, and outputs the secret and public parameters of the* ReTRACe *system.*
(2) ReTRACe.UserSetup(SecPar, PubPar) → key*: This algorithm takes as input the secret/public parameters, and initializes each user with a tuple,* key*, consisting of their group signing keys,* RFAME *keys, and* RCHET *long-term trapdoor.*
(3) ReTRACe.Sign($\mathbb{GSK}$, m, Υ) → ξ_m*: This algorithm takes in a set of signing keys,* $\mathbb{GSK}$*, a message, a* DGSS *policy,* Υ*, and outputs a set of signatures,* ξ_m *satisfying* Υ*.*
(4) ReTRACe.CreateMessage(key, PubPar, m) → (msg, ξ_{msg})*: Is run by the originator, takes in a tuple,* key*, public parameters, a message, and outputs a tuple, msg, consisting of* RCHET, RFAME *and* DGSS *parameters for m, and a set of signatures on msg,* ξ_{msg}*.*
(5) ReTRACe.AdaptMessage(key, PubPar, m', msg, ξ_{msg}) → (msg', $\xi_{msg'}$)*: This algorithm, run by members of* ***AuthU*** *takes as input a tuple,* key*, the public parameters, a message, a tuple, msg, and a set of signatures on msg,* ξ_{msg}*. If the* RCHET, RFAME, DGSS *parameters contained in msg pass verification, it updates the m contained in msg to* m'*, returns an updated message tuple,* msg' *with new* RCHET, RFAME *and* DGSS *parameters, and a set of signatures,* $\xi_{msg'}$ *on* msg'*.*
(6) ReTRACe.Verify(PubPar, msg, ξ_{msg}) → $\{0, 1\}$*: This algorithm takes in* PubPar*, a tuple, msg, and and set of signatures on msg,* ξ_{msg}*. If the* RCHET, RFAME*, and* DGSS *parameters contained in msg pass verification, it returns 1.*
(7) ReTRACe.VerifyMiner(PubPar, msg, ξ_{msg}, ς) → $\{0, 1\}$*: This algorithm is run by the miners, and takes in the public parameters, a msg tuple, a set of signatures on msg,* ξ_{msg}*, and a variable* ς*, which conveys whether the person that submitted msg was a member of* ***AuthU*** *or* ***AuthUAdmins****. If the* RCHET, RFAME *and DGSS parameters in msg pass verification, the algorithm returns 1, and the msg will be posted on the BC.*
(8) ReTRACe.RevokeUser(key, PubPar, m', msg, ξ_{msg}) → (msg', $\xi_{msg'}$)*: This algorithm is run by members of* ***AuthUAdmins*** *who want to revoke users either by updating the Boolean policies associated with* RFAME/DGSS *contained in msg, or in response to the AIA/GM revoking users. It outputs an updated tuple,* msg' *containing new* RCHET, RFAME, DGSS *parameters and set of signatures,* $\xi_{msg'}$ *on* msg'*.*

```
ReTRACe.Keygen($1^\lambda$)
1 : GSetup($1^\lambda$) → pubpar
2 : GKGen(pubpar) → ($out_{GM}$, $st_{GM}$),
    where $out_{GM}$ = ($mpk$, info$_0$)
3 : Set $gpk$ = (pubpar, $mpk$), $st_{GM}$ is GM's state
4 : RFAME.SetupABE($1^\lambda$, $U$) → (mpk$_{ABE}$, msk$_{ABE}$)
5 : RCHET.cSetup($1^\lambda$) → param
6 : RCHET.userKeySetup(param) → ($sk_{ch}$, $pk_{ch}$)
7 : PubPar = ($\mathbb{G}$, $g$, $q$, $pk_{ch}$, mpk$_{ABE}$, $gpk$)
8 : SecPar = (msk$_{ABE}$, $st_{GM}$)
9 : return (SecPar, PubPar, $sk_{ch}$)
```

(a) ReTRACe**: AIA/GM setup**

```
ReTRACe.UserSetup(SecPar, PubPar)
1 : Get $\mathbb{L}$, the list of all groups in DGSS
    that current user needs to join, set $\mathbb{GSK} = \varnothing$
2 : For each group in $\mathbb{L}$, Run DGSS.Join get
    $gsk$, $\mathbb{GSK} = \mathbb{GSK} \cup gsk$
3 : RFAME.KeyGenABE(mpk$_{ABE}$, msk$_{ABE}$, $y_1, \ldots,$
    $y_{|U|}$) → $SK$
4 : Retrieve $sk_{ch}$
5 : return key = ($\mathbb{GSK}$, $SK$, $sk_{ch}$)
```

(b) ReTRACe**: System setup for user**

```
ReTRACe.Sign($\mathbb{GSK}$, $m$, $\Upsilon_{GS}$)
1 : Pick $\mathbb{K}$ $s.t.$, $\mathbb{K} \subseteq \mathbb{GSK}$, $\Upsilon_{GS}(\mathbb{K}) = 1$
2 : for $gsk^i \in \mathbb{K}$, where $i \in 1 \cdots |\mathbb{GSK}|$ do
3 :     DGSS.Sign($gsk^i$, $i$.info, $m$) → $\{\sigma_i, \perp\}$
4 : $\xi = (\sigma_i, i.gpk) \cup \xi$
5 : return $\xi$
```

(c) ReTRACe**: Signing a message**

```
ReTRACe.Verify(PubPar, $msg$, $\xi_{msg}$)
1 : for $(\sigma_l, l.gpk) \in \xi_{msg}$ do
2 :     if DGSS.VerifySignature($l.gpk$, $l$.info, $msg$,
            $\sigma_l) \overset{?}{=} 0$, return 0
3 : for $(\sigma_l, l.gpk) \in \xi_{\Upsilon_{info}}$ do
4 :     if DGSS.VerifySignature($l.gpk$, $l$.info, $\Upsilon_{info}$,
            $\sigma_l) \overset{?}{=} 0$, return 0
5 : if RCHET.verifyTransaction($pk_{ch}$, $m$, digest,
    rand, $\Gamma_{pubinfo}) \overset{?}{=} 1$
6 :     return 1.
```

(d) ReTRACe**: Verifying a message**

Figure 3: ReTRACe **algorithms for system setup and signing/verifying messages**

REMARK 7.1. *An* originator *of a message could possibly create malformed policies, e.g., policies containing bogus or non-existent attributes. We assume the miner has knowledge of all the (public) attributes in the universe and in this case would reject malformed policies. An originator of a message, m, could create a bogus trapdoor, which would not be discovered until someone attempts to update m. Note that the miner cannot check if the encrypted trapdoor is correct or not, since the miner likely will not be part of the **AuthU**, or **AuthUAdmins** sets. Solutions to this problem include having the originator do a verifiable encryption of the trapdoor, while submitting m to the miner, or have the originator include an NIZK proof along with the message. We leave the construction of a scheme that incorporates these ideas as future work.*

```
ReTRACe.CreateMessage(key, PubPar, $m$)
1 :  RCHET.cHash($sk_{ch}$, $pk_{ch}$, $m$) $\rightarrow$
     (digest, rand, $\Gamma_{\text{pubinfo}}$, $\Gamma_{\text{privinfo}}$)
2 :  RFAME.Encrypt($\text{mpk}_{\text{ABE}}$, $\Gamma_{\text{privinfo}}$, ($\mathbf{M}_{\Upsilon_{\text{ABE}}}$, $\rho_{\Upsilon_{\text{ABE}}}$)) $\rightarrow X$
3 :  Create $\Upsilon_{\text{GS}}$. Set $\Upsilon_{\text{info}} = (\Upsilon_{\text{ABE}}, \Upsilon_{\text{GS}})$
4 :  $r \leftarrow\!\!\$\ \mathbb{Z}_q^*$, $\omega = g^r$, $\pi_\omega \leftarrow$ NIZKPoK$\{r : \omega = g^r\}$
5 :  RFAME.Encrypt($\text{mpk}_{\text{ABE}}$, $r$, ($\mathbf{M}_{\Upsilon_{ABE\text{admin}}}$, $\rho_{\Upsilon_{ABE\text{admin}}}$)) $\rightarrow X_r$
6 :  Create $\Upsilon_{GS\text{admin}}$
7 :  Set$\Upsilon_{\text{admin}} = (\Upsilon_{ABE\text{admin}}, \Upsilon_{GS\text{admin}}, X_r, \omega, \pi_\omega)$
8 :  ReTRACe.Sign($\mathbb{GSK}$, $\Upsilon_{\text{info}}$, $\Upsilon_{GS\text{admin}}$) $\rightarrow \xi_{\Upsilon_{\text{info}}}$
9 :  $msg = (m$, digest, rand, $\Gamma_{\text{pubinfo}}$, $X$, $\Upsilon_{\text{info}}$,
     $\Upsilon_{\text{admin}}$, $\xi_{\Upsilon_{\text{info}}})$
10 : ReTRACe.Sign($\mathbb{GSK}$, $msg$, $\Upsilon_{\text{GS}}$) $\rightarrow \xi_{msg}$
11 : Call ReTRACe.VerifyMiner(PubPar, $msg$, $\xi_{msg}$, $\pi_\omega$)
12 : return ($msg$, $\xi_{msg}$)
```

(a) ReTRACe**: Creating a message**

```
ReTRACe.AdaptMessage(key, PubPar, $m'$, $msg$, $\xi_{msg}$)
1 :  if ReTRACe.Verify(PubPar, $msg$, $\xi_{msg}$) $\overset{?}{=} 0$
         return $\perp$
2 :  RFAME.Decrypt($SK$, $X$, ($\mathbf{M}_{\Upsilon_{\text{ABE}}}$, $\rho_{\Upsilon_{\text{ABE}}}$)) $\rightarrow \Gamma_{\text{privinfo}}$
3 :  RCHET.adaptMessage($sk_{ch}$, $m$, $m'$, digest, rand,
     $\Gamma_{\text{pubinfo}}$, $\Gamma_{\text{privinfo}}$) $\rightarrow$ rand$'$
4 :  $msg' = (m'$, digest, rand$'$, $\Gamma_{\text{pubinfo}}$, $X$, $\Upsilon_{\text{info}}$,
     $\Upsilon_{\text{admin}}$, $\xi_{\Upsilon_{\text{info}}})$
5 :  ReTRACe.Sign($\mathbb{GSK}$, $msg'$, $\Upsilon_{\text{GS}}$) $\rightarrow \xi_{msg'}$
6 :  if ReTRACe.VerifyMiner(PubPar, $msg'$,
     $\xi_{msg'}$, $\perp$) $\overset{?}{=} 0$
         return $\perp$
7 :  return ($msg'$, $\xi_{msg'}$)
```

(b) ReTRACe**: Updating a message**

Figure 4: ReTRACe **algorithms for creating and updating a message**

7.1 ReTRACe Construction

We now give the detailed construction of the ReTRACe framework comprising of eight algorithms in Figure 3, Figure 4, Figure 5, and Figure 6. In the algorithms, $\mathbf{M}$ denotes the monotone span program representing an ABE policy, and ρ represents a mapping function that maps rows of $\mathbf{M}$ onto attributes. We use BC.write to denote a blockchain write operation. The Keygen, UserSetup, Sign, Verify, AdaptMessage algorithms are fairly self-explanatory. We now describe some of the salient features of the CreateMessage, VerifyMiner, and RevokeUser algorithms, which are more involved. We assume a given implementation will use standard techniques like nonces and timestamps to prevent against replay attacks.

ReTRACe.CreateMessage: This algorithm (Figure 4a), is run by the originator who first runs RCHET to create a digest and trapdoors for a message m. The originator sets Υ_{ABE}, Υ_{GS} (for members of **AuthU**), and $\Upsilon_{ABE\text{admin}}$ and $\Upsilon_{GS\text{admin}}$ (for members of **AuthUAdmins**). The ephemeral trapdoor is encrypted under Υ_{ABE} to obtain X. The originator then picks an $r \leftarrow\!\!\$\ \mathbb{Z}_q^*$ and encrypts it under $\Upsilon_{ABE\text{admin}}$ to obtain X_r. This ensures that only members of **AuthUAdmins** can decrypt r, and modify Υ_{ABE} and Υ_{GS}. The originator creates a tuple, msg, with X and **AuthU**, **AuthUAdmins** policy information, signs msg using her signing key(s) that satisfy Υ_{GS}, and creates a set of signatures, $\xi_{\Upsilon_{\text{info}}}$, with each signature bundled with its corresponding verification key. Finally, the originator signs Υ_{info} and sends the signature, along with msg, ξ_{msg}, $\xi_{\Upsilon_{\text{info}}}$ to the miner.

ReTRACe.VerifyMiner: This algorithm (Figure 5) is run only by miners to verify a message before posting it on the BC. If a message is being adapted ($\varsigma = \perp$), the miner does not do NIZK verifications. If a trapdoor is being adapted ($\varsigma = \pi_\omega$), the tuple submitted to the miner is an update to a pre-existing msg on the BC, and the ω used to verify the NIZK π_ω is obtained from the current msg on the BC. If a new message msg is being created ($\varsigma = \pi_\omega$), then the ω used to verify the NIZK π_ω is obtained from the msg tuple itself. In all cases, the miner checks if all signatures in ξ_{msg} pass verification w.r.t. Υ_{GS} contained in the tuple msg, checks if all signatures in $\xi_{\Upsilon_{\text{info}}}$ pass verification w.r.t. $\Upsilon_{GS\text{admin}}$, and the digest of m is checked. If all checks pass, the msg tuple, along with the list of signatures on it is written to the BC. Note that if ReTRACe is deployed in a BC that hosts both mutable and immutable transactions, then for immutable transactions, the miner verification process is the same as in current BC systems.

ReTRACe.RevokeUser: This algorithm (Figure 6a, Figure 6b) is called by a member of **AuthUAdmins** either when they want to revoke clauses from the ABE policy, Υ_{ABE}, or when the ephemeral trapdoor, Γ_{privinfo}, needs to be re-encrypted in response to the AIA revoking a user. Both cases are handled differently:

Case 1: Revoking a clause from Υ_{ABE}: This algorithm (Figure 6a) is run by an user $v \in$ **AuthUAdmins** who wants to modify an Υ_{ABE} associated with msg. The AIA/GM are not involved, and no algorithm from RFAME is called. User v first decrypts the trapdoor, Γ_{privinfo}, using her RFAME secret keys, v picks an r', and encrypts r' under $\Upsilon_{ABE\text{admin}}$. This is to ensure that only non-revoked members of **AuthUAdmins** can decrypt r' and adapt the ephemeral trapdoor in the future. Next, v adapts the ephemeral trapdoor. The new message and trapdoor are encrypted under a new policy, Υ'_{ABE}, which is a low cost operation and involves no re-keying operations. We have not depicted the Υ_{GS} getting updated, for clarity of presentation. There are four cases:

1) If Υ_{ABE} changes to a more inclusive Υ'_{ABE}, the new user groups need to be present in the Υ_{GS} as well.

2) If Υ_{ABE} changes to a more restrictive Υ'_{ABE}, the revoked users cannot decrypt the trapdoor and successfully adapt the message, so Υ_{GS} does not need to change.

3) If Υ_{GS} changes to a more restrictive Υ'_{GS}, such that the users satisfying Υ_{GS} were also part of Υ_{ABE}, Υ_{ABE} needs to change too, revoking the said users from the ABE scheme.

4) If Υ_{GS} changes to a more inclusive Υ'_{GS}, such that the users satisfying Υ'_{GS} are not part of Υ_{ABE}, the new users cannot decrypt the trapdoor and successfully adapt the message, so Υ_{ABE} does not need to change. User v then signs the new Υ'_{info} using their signing keys that satisfy $\Upsilon_{GS\text{admin}}$; the signature set is denoted as $\xi_{\Upsilon'_{\text{info}}}$. A new msg' is created and signed using a set of keys that satisfy Υ_{GS}, and the resulting signature set is denoted by $\xi_{msg'}$. Finally, msg' and $\xi_{msg'}$ are given to the miner who verifies it before posting on the BC.

Case 2: AIA revoking a user: This algorithm (Figure 6b) is run by a user $v \in$ **AuthUAdmins** as soon as the AIA revokes a user holding attribute y (which appears in either Υ_{ABE} or $\Upsilon_{ABE\text{admin}}$). First, the AIA updates its own public key from mpk_{ABE} to mpk_{ABE}' (which results in PubPar getting updated to PubPar$'$), and then issues new signing keys, SK', only to the *non-revoked* users holding attribute y. User v then proceeds to adapt the ephemeral trapdoor, Γ_{privinfo}, to prevent the revoked user from being able to perform any future message adaptations. User v then generates a new r', encrypts it, etc., the rest of the steps are similar to Case 1.

```
ReTRACe.VerifyMiner(PubPar, $msg$, $\xi_{msg}$, $\varsigma$)
for ($\sigma_l$, $l.gpk$) $\in \xi_{msg}$ do
    if DGSS.VerifySignature($l.gpk$, $l$.info, $msg$, $\sigma_l$) $\overset{?}{=} 0$
        return 0
for ($\sigma_l$, $l.gpk$) $\in \xi_{\Upsilon_{\text{info}}}$ do
    if DGSS.VerifySignature($l.gpk$, $l$.info, $\Upsilon_{\text{info}}$, $\sigma_l$) $\overset{?}{=} 0$
  return 0
if $\varsigma = \pi_\omega$
    if verify($\omega$, $\pi_\omega$) $\neq 1$, return 0
if RCHET.verifyTransaction($pk_{ch}$, $m$, digest, rand,
  $\Gamma_{\text{pubinfo}}$) $\overset{?}{=} 1$
        BC.write($msg$, $\xi_{msg}$, ) return 1
 return 0
```

Figure 5: ReTRACe**: Miner verifying a message**

7.2 ReTRACe **Security Properties**

We now informally discuss the security properties of ReTRACe: indistinguishability, public, private, and revocation collision resistance. The first three properties were first introduced by Derler

```
ReTRACe.RevokeUser(key, PubPar, $m'$, $msg$, $\xi_{msg}$)
if ReTRACe.Verify(PubPar, $msg$, $\xi_{msg}$) $\overset{?}{=} 0$
    return $\bot$
RFAME.Decrypt($SK$, $X_r$, ($\mathbf{M}_{\Upsilon_{ABE\text{admin}}}$,
    $\rho_{\Upsilon_{ABE\text{admin}}}$)) $\rightarrow r$
$r' \leftarrow\!\$\, \mathbb{Z}_q^*$, $\omega' = g^{r'}$.Set $\pi_{\omega'} \leftarrow$ NIZKPoK$\{r' : \omega' = g^{r'}\}$
RFAME.Encrypt($\text{mpk}_{\text{ABE}}$, $r'$, ($\mathbf{M}_{\Upsilon_{ABE\text{admin}}}$,
    $\rho_{\Upsilon_{ABE\text{admin}}}$)) $\rightarrow X_{r'}$
$\Upsilon'_{\text{admin}} = (\Upsilon_{ABE\text{admin}}, \Upsilon_{GS\text{admin}}, X_{r'}, \omega', \pi_{\omega'})$
RFAME.Decrypt($SK$, $X$, ($\mathbf{M}_{\Upsilon_{\text{ABE}}}$, $\rho_{\Upsilon_{\text{ABE}}}$)) $\rightarrow \Gamma_{\text{privinfo}}$
RCHET.adaptTrapdoor($sk_{ch}$, $m$, $m'$, digest, rand,
    $\Gamma_{\text{pubinfo}}$, $\Gamma_{\text{privinfo}}$) $\rightarrow$ (rand$'$, $\Gamma'_{\text{pubinfo}}$, $\Gamma'_{\text{privinfo}}$)
RFAME.Encrypt($\text{mpk}_{\text{ABE}}$, $\Gamma'_{\text{privinfo}}$, ($\mathbf{M}_{\Upsilon'_{\text{ABE}}}$,
    $\rho_{\Upsilon'_{\text{ABE}}}$)) $\rightarrow X'$
$\Upsilon'_{\text{info}} = (\Upsilon'_{\text{ABE}}, \Upsilon_{\text{GS}})$
 ReTRACe.Sign($\mathbb{GSK}$, $\Upsilon'_{\text{info}}$, $\Upsilon_{GS\text{admin}}$) $\rightarrow \xi_{\Upsilon'_{\text{info}}}$
 $msg' = (m'$, digest, rand$'$, $\Gamma'_{\text{pubinfo}}$, $X'$,
    $\Upsilon'_{\text{info}}$, $\Upsilon'_{\text{admin}}$, $\xi_{\Upsilon'_{\text{info}}})$
 ReTRACe.Sign($\mathbb{GSK}$, $msg'$, $\Upsilon_{\text{GS}}$) $\rightarrow \xi_{msg'}$
 Call VerifyMiner(PubPar, $msg'$, $\xi_{msg'}$, $\pi_{\omega'}$)
   return ($msg'$, $\xi_{msg'}$)
```

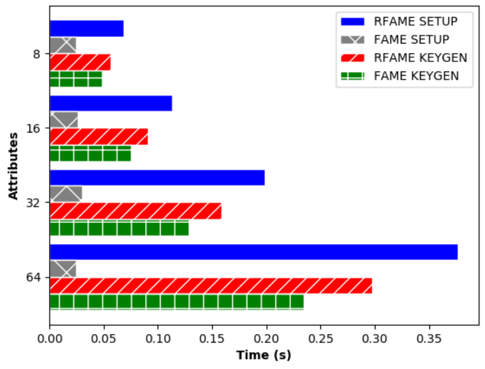
(a) ReTRACe**: Revoke Case 1: Revoke users by updating policies**

```
ReTRACe.RevokeUser(key, PubPar$'$, $m'$, $msg$, $\xi_{msg}$)
if ReTRACe.Verify(PubPar$'$, $msg$, $\xi_{msg}$) $\overset{?}{=} 0$
    return $\bot$
RFAME.Decrypt($SK$, $X_r$, ($\mathbf{M}_{\Upsilon_{ABE\text{admin}}}$,
    $\rho_{\Upsilon_{ABE\text{admin}}}$)) $\rightarrow r$
$r' \leftarrow\!\$\, \mathbb{Z}_q^*$, $\omega' = g^{r'}$.Set $\pi_{\omega'} \leftarrow$ NIZKPoK$\{r' : \omega' = g^{r'}\}$
RFAME.Encrypt($\text{mpk}_{\text{ABE}}'$, $r'$, ($\mathbf{M}_{\Upsilon_{ABE\text{admin}}}$,
    $\rho_{\Upsilon_{ABE\text{admin}}}$)) $\rightarrow X_{r'}$
$\Upsilon'_{\text{admin}} = (\Upsilon_{ABE\text{admin}}, \Upsilon_{GS\text{admin}}, X_{r'}, \omega', \pi_{\omega'})$
RFAME.Decrypt($SK$, $X$, ($\mathbf{M}_{\Upsilon_{\text{ABE}}}$, $\rho_{\Upsilon_{\text{ABE}}}$)) $\rightarrow \Gamma_{\text{privinfo}}$
RCHET.adaptTrapdoor($sk_{ch}$, $m$, $m'$, digest,
    rand, $\Gamma_{\text{pubinfo}}$, $\Gamma_{\text{privinfo}}$) $\rightarrow$ (rand$'$, $\Gamma'_{\text{pubinfo}}$, $\Gamma'_{\text{privinfo}}$)
RFAME.Encrypt($\text{mpk}_{\text{ABE}}'$, $\Gamma'_{\text{privinfo}}$, ($\mathbf{M}_{\Upsilon_{\text{ABE}}}$,
    $\rho_{\Upsilon_{\text{ABE}}}$)) $\rightarrow X'$
$msg' = (m'$, digest, rand$'$, $\Gamma'_{\text{pubinfo}}$, $X'$, $\Upsilon_{\text{info}}$,
    $\Upsilon'_{\text{admin}}$, $\xi_{\Upsilon_{\text{info}}})$
 ReTRACe.Sign($\mathbb{GSK}$, $msg'$, $\Upsilon_{\text{GS}}$) $\rightarrow \xi_{msg'}$
 Call VerifyMiner(PubPar$'$, $msg'$, $\xi_{msg'}$, $\pi_{\omega'}$)
return ($msg'$, $\xi_{msg'}$)
```

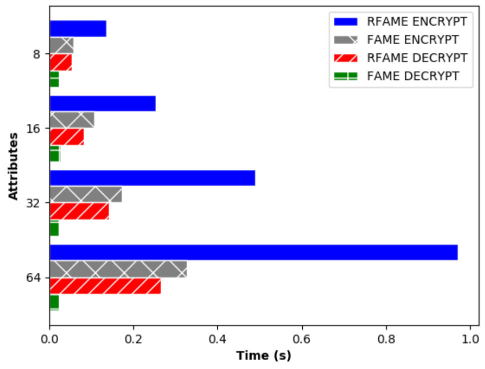
(b) ReTRACe**: Revoke Case 2: AIA revoking a single user**

Figure 6: ReTRACe **algorithms for revoking users**

et al. [27] for any policy-based chameleon hash scheme. We define revocation collision resistance, and strengthen the first three properties, by giving the adversary the ability to adapt messages and revoke messages. Indistinguishability requires that it should be computationally infeasible for an adversary to distinguish whether the randomness associated with a given message was generated as a result of a CreateMessage, AdaptMessage or RevokeUser. Public collision resistance requires that an adversary who knows neither the long-term nor the ephemeral trapdoor cannot produce valid collisions even after seeing past adaptations of messages and trapdoors, even with access to some attributes, but not the complete attribute set that can decrypt the ephemeral trapdoor.



(a) Key generation and Setup



(b) Encryption and Decryption

Figure 7: Timings for RFAME vs. FAME [1] (80 users per attribute)

Private collision resistance requires that an adversary that knows the long-term trapdoor, but not ephemeral trapdoor of the RCHET scheme, cannot produce valid collisions, even with knowledge of past message and trapdoor adaptations. This property should hold even if she has access to a subset of attributes, but not the complete set of attributes, needed to decrypt the current trapdoor. Revocation collision resistance requires that an adversary, who knows the long-term and ephemeral trapdoors, and has valid attributes to decrypt the ephemeral trapdoor, cannot produce valid collisions, if, either the RFAME policy changed to exclude her, or the AIA revoked some or all of her attributes necessary to decrypt the trapdoor. We have proven the IND-CPA security of RFAME; we apply the Fujisaki-Okamoto transform [30] to convert RFAME to an IND-CCA2 secure scheme to accomplish the proof. The formalization of the security properties and the proof of the pertinent theorem (next) are in Appendix 10.8.

Theorem 7.1. *If* RCHET *is secure,* RFAME *is fully IND-CCA2 secure, and* DGSS *is a secure dynamic group signature scheme then* ReTRACe *is secure.*

8 IMPLEMENTATION AND RESULTS

We implemented RFAME, RCHET, and ReTRACe in Python 3, and used Charm [19] for cryptographic modules. All the experiments were carried out on a machine with 64 GB RAM and an Intel(R) Core(TM) i7-6700K CPU clocked at 4.00 GHz. We implemented RCHET and RFAME to compare their performance against CHET and FAME, respectively, to quantify the price of adding revocation. We do not compare RFAME quantitatively with other revocable ABE schemes, since they do not provide the properties that RFAME provides (see Table 2 comparison). Using RCHET and RFAME we implement ReTRACe. Note that ReTRACe is the first system that provides transaction-level revocable blockchain rewrites, there is no equivalent state-of-the-art scheme to compare with.

RFAME Results: We set our ABE policies to contain a total of 8, 16, 32 and 64 attributes, and all our policies have two equisized conjunctive clauses separated by a single disjunction. In each run, 10, 20, 40, or 80 users signed up with the AIA for each attribute. The computation time increases linearly with the number of users, so for brevity, in Figure 7 we show results for RFAME and FAME for 80 users per attribute only. The setup times for RFAME are higher than for FAME because of the extra operations involved in computing the master public key (mpk_{ABE}) and master secret key (msk_{ABE}) during setup; and the growth of the public key size in RFAME is linear in the number of attributes (small-universe property).

In FAME, the size of one of the components of the ciphertext increases linearly in the number of attributes satisfying the given policy, whereas for RFAME there are two components whose size increases linearly, which accounts for the difference in the timings of the encryption and decryption operations. For decryption, the number of pairing operations is $6 + 2\times$(number of attributes satisfying a given policy) for RFAME, as compared to 6 pairing operations for FAME.

Table 3: Timing for the RFAME.Revoke **(time in secs)**

10 Users per attribute	0.115
20 Users per attribute	0.2
40 Users per attribute	0.364
80 Users per attribute	0.714

Table 3 shows the time taken when revoking one user from each attribute group with 10, 20, 40, and 80 users each, which results in the rekeying of the remaining users. The results are linear as expected because in each case 9, 19, 39, and 79 users got new keys, respectively. We expect this trend to continue as the number of users per attribute increases.

As mentioned before in Section 5, previous schemes do not provide efficient revocation. To carry out a user revocation under previous schemes, the entire system would have to be rekeyed using the Setup and Keygen functions, and all ciphertext re-encrypted regardless of whether the revoked user had access to the message or not. Thus, the cost in rekeying the users would be significantly lower in RFAME, especially if a user is in a single attribute group and is revoked.

RCHET Results: Table 4 compares the running times for CHET and RCHET. In RCHET, when compared to CHET, we have added one extra encryption and decryption, two NIZKPoK generation and verifications, and three modular exponentiations to all functions, except systemSetup and userKeySetup. Despite this, RCHET does not display a significant increase in latency, at the same time, providing the ability to adapt the trapdoor of a message digest. The time difference between RCHET and CHET algorithms is in the order of milliseconds and this is a minimal trade-off for the added functionality that RCHET provides.

ReTRACe Results: ReTRACe was implemented with the DGSS policies being the same as the ABE policies, and containing 20 users per attribute for 8 and 16 attributes. Except for the RFAME revocation component, whose running time is proportional to the number of users, the rest of the cryptographic primitives, i.e., DGSS, RCHET, and other RFAME algorithms, are independent of the number of users in the system. The running time for operations in ReTRACe would increase linearly with the number of users per attribute, as is evident from the RFAME results.

Table 4: Comparison of RCHET vs. CHET [18] (time in secs)

Algorithm	CHET	RCHET
Setup	0.537	0.5369
Chash	0.0216	0.0234
Verify	0.000697	0.000967
Adapt Message	0.0414	0.0415
Adapt Trapdoor	-	0.04305

Table 5 shows the timings of ReTRACe, with 20 users/attribute and messages with policies containing 8 and 16 attributes respectively. UserSetup and Keygen take significantly more time than the other functions as expected; both these functions involve all users in the system and are run only once at the beginning during system and users' setup. CreateMessage, Sign, Verify, and VerifyMiner would be run more frequently, and all have sub-second timings. For implementing Case 1 of ReTRACe.RevokeUser, we eliminate one attribute from Υ_{ABE}, and in Case 2 we revoke one user from the AIA that held an attribute in Υ_{ABE}. Case 2 takes longer because it includes the AIA's operations for revoking a user from a single attribute group and rekeying of the rest of the users in the same group, whereas Case 1 just changes the message policy and updates the message trapdoor.

Table 5: ReTRACe running time, 20 users/attribute (secs)

ReTRACe Algorithms	8 Attr	16 Attr
UserSetup and Keygen (for 20 users)	2.997	4.694
CreateMessage	0.473	0.963
Sign	0.0904	0.180
Verify	0.114	0.232
VerifyMiner	0.225	0.460
AdaptMessage	0.0928	0.152
RevokeUser (Case 1)	0.545	1.015
RevokeUser (Case 2) (for 19 users)	0.676	1.049

Implementation in Ethereum: ReTRACe can be plugged into an existing blockchain (e.g., Ethereum) by updating relevant cryptographic operations with equivalent ones in ReTRACe. For instance, in Ethereum the signature algorithm in the module "crypto/crypto.go" needs to be modified to use ReTRACe.Sign; "trie/trie.go" to use the digest of rewritable transactions at the leaves of the blocks' Merkle trees; ReTRACe.AdaptMessage and ReTRACe.RevokeUser need to be added to the "ethclient" module and ReTRACe.VerifyMiner to the "miner/miner.go" module. We are porting these modifications to Ethereum.

With ReTRACe-adapted Ethereum, an authorized user updates a transaction using the chameleon hash and then submits it to the transaction pool. In our design, the transactions will be updated with a binary flag ('0' $\leftarrow$ new; '1' $\leftarrow$ updated). A miner that picks up an updated transaction verifies the transaction using ReTRACe.Verify, updates the transaction in the block–the remaining transactions are untouched– and propagates the block for consensus. At each node storing the BC, the block with the updated transaction replaces the old block post transaction-verification.

The cost of ReTRACe operations in Ethereum (in gas) would be proportional to their computational cost shown in this section. We can calculate exact gas costs of our operations, but the exact cost is dynamic, varying based on many factors (number of pending transactions, minimum cost, etc.). At the base computation level, ReTRACe scales linearly with increasing number of attributes and users—highly desirable.

9 CONCLUSION

We present ReTRACe, a blockchain transaction rewriting framework building on a novel revocable chameleon hash with ephemeral trapdoor scheme and a novel revocable CP-ABE scheme. We discuss ReTRACe's contributions and functionalities that provide efficient and authorized transaction rewrites in blockchains, in addition to revocability and traceability of the users updating the transaction(s). We have performed rigorous security and experimental analyses to demonstrate ReTRACe's scalability.

ACKNOWLEDGMENTS

Research supported by NSF awards #1800088, #2028797, #1914635, Intel Labs, and the Federal Aviation Administration (FAA). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF, FAA, and Intel Inc.

REFERENCES

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

[2] Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton Andrade. 2017. Redactable blockchain–or–rewriting history in bitcoin and friends. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 111–126.

[3] Nuttapong Attrapadung and Hideki Imai. 2009. Attribute-Based Encryption Supporting Direct/Indirect Revocation Modes. In *12th IMA International Conference, Cryptography and Coding, Proceedings*. 278–300.

[4] Amos Beimel and Benny Chor. 1995. Secret Sharing with Public Reconstruction (Extended Abstract). In *Advances in Cryptology - CRYPTO, roceedings*. 353–366.

[5] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. 2016. NIZKs with an Untrusted CRS: Security in the Face of Parameter Subversion. In *Advances in Cryptology - ASIACRYPT, Proceedings, Part II*. 777–804.

[6] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. 2003. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In *Advances in Cryptology - EUROCRYPT, Proceedings*. 614–629.

[7] Mihir Bellare, Haixia Shi, and Chong Zhang. 2005. Foundations of Group Signatures: The Case of Dynamic Groups. In *Topics in Cryptology - CT-RSA, Proceedings*. 136–153.

[8] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP*. 459–474.

[9] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. 2019. Tesseract: Real-Time Cryptocurrency Exchange Using Trusted Hardware. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS*. 1521–1538.

[10] John Bethencourt, Amit Sahai, and Brent Waters. 2007. Ciphertext-Policy Attribute-Based Encryption. In *2007 IEEE Symposium on Security and Privacy (S&P 2007)*. 321–334.

[11] John Bethencourt, Amit Sahai, and Brent Waters. 2007. Ciphertext-Policy Attribute-Based Encryption. In *IEEE Symposium on Security and Privacy (S&P 2007)*. 321–334.

[12] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. 2008. Identity-based encryption with efficient revocation. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS*. 417–426.

[13] Dan Boneh. 1998. The Decision Diffie-Hellman Problem. In *Algorithmic Number Theory, Third International Symposium, ANTS, Proceedings*. 48–63.

[14] Dan Boneh, Xavier Boyen, and Hovav Shacham. 2004. Short Group Signatures. In *Advances in Cryptology - CRYPTO, Proceedings*, Matthew K. Franklin (Ed.). 41–55.

[15] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, and Jens Groth. 2016. Foundations of Fully Dynamic Group Signatures. *IACR Cryptol. ePrint Arch.* 2016 (2016), 368. http://eprint.iacr.org/2016/368

[16] business [n.d.]. The growing list of applications and use cases of blockchain technology in business and life. https://www.businessinsider.com/blockchain-technology-applications-use-cases.

[17] ca18 [n.d.]. California Consumer Privacy Act. https://oag.ca.gov/privacy/ccpa.

[18] Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. 2017. Chameleon-Hashes with Ephemeral Trapdoors - And Applications to Invisible Sanitizable Signatures. In *Public-Key Cryptography - PKC, Proceedings, Part II*. 152–182.

[19] charm [n.d.]. Charm: A tool for rapid cryptographic prototyping. http://charm-crypto.io.

[20] Melissa Chase. 2007. Multi-authority Attribute Based Encryption. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC, Proceedings*, Salil P. Vadhan (Ed.). 515–534.

[21] Jie Chen, Romain Gay, and Hoeteck Wee. 2015. Improved Dual System ABE in Prime-Order Groups via Predicate Encodings. In *Advances in Cryptology - EUROCRYPT, Proceedings, Part II*. 595–624.

[22] Sherman S. M. Chow. 2016. A Framework of Multi-Authority Attribute-Based Encryption with Outsourcing and Revocation. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies, SACMAT*. 215–226.

[23] Hui Cui, Robert H. Deng, Yingjiu Li, and Baodong Qin. 2016. Server-Aided Revocable Attribute-Based Encryption. In *Computer Security - ESORICS, Proceedings, Part II*. 570–587.

[24] Poulami Das, Lisa Eckey, Tommaso Frassetto, David Gens, Kristina Hostáková, Patrick Jauernig, Sebastian Faust, and Ahmad-Reza Sadeghi. 2019. Fastkitten: Practical smart contracts on bitcoin. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 801–818.

[25] Sourav Das, Vinay Joseph Ribeiro, and Abhijeet Anand. 2018. Yoda: Enabling computationally intensive contracts on blockchains with byzantine and selfish nodes. *arXiv preprint arXiv:1811.03265* (2018).

[26] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. 2015. Fully Secure Unbounded Revocable Attribute-Based Encryption in Prime Order Bilinear Groups via Subset Difference Method. *IACR Cryptology ePrint Archive* (2015). http://eprint.iacr.org/2015/293

[27] David Derler, Kai Samelin, Daniel Slamanig, and Christoph Striecks. 2019. Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based. In *26th Annual Network and Distributed System Security Symposium, NDSS*.

[28] Dominic Deuber, Bernardo Magri, and Sri Aravinda Krishnan Thyagarajan. 2019. Redactable Blockchain in the Permissionless Setting. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. 124–138.

[29] dhs [n.d.]. Department of Homeland Security: Blockchain and Suitability for Government Applciations. https://www.dhs.gov/sites/default/files/publications2018_AEP_Blockchain_and_Suitability_for_Government_Applications.pdf.

[30] Eiichiro Fujisaki and Tatsuaki Okamoto. 1999. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *Advances in Cryptology - CRYPTO, Proceedings*. 537–554.

[31] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. 2013. Attribute-Based Encryption for Circuits from Multilinear Maps. In *Advances in Cryptology - CRYPTO. Proceedings, Part II*. 479–499.

[32] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. 2013. Attribute-based encryption for circuits. In *Symposium on Theory of Computing Conference STOC*. 545–554.

[33] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. 2006. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS*. 89–98.

[34] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. 2018. Updatable and Universal Common Reference Strings with Applications to zk-SNARKs. In *Advances in Cryptology - CRYPTO Proceedings, Part III*. 698–728.

[35] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. 2018. ZEUS: Analyzing Safety of Smart Contracts.. In *Ndss*. 1–12.

[36] Mauricio Karchmer and Avi Wigderson. 1993. On Span Programs. In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference*. 102–111.

[37] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *IEEE Symposium on Security and Privacy, SP*. 839–858.

[38] Hugo Krawczyk and Tal Rabin. 2000. Chameleon Signatures. In *Proceedings of the Network and Distributed System Security Symposium, NDSS*.

[39] Allison B. Lewko and Brent Waters. 2011. Decentralizing Attribute-Based Encryption. In *Advances in Cryptology - EUROCRYPT, Proceedings*, Kenneth G. Paterson (Ed.). 568–588.

[40] Benoît Libert, Thomas Peters, and Moti Yung. 2012. Scalable Group Signatures with Revocation. In *Advances in Cryptology - EUROCRYPT, Proceedings*. 609–627.

[41] Medium. [n.d.]. Hundreds of Millions of Dollars Locked at Ethereum 0x0 Address and Smart ContractsâĂŹ Addresses. https://medium.com/@maltabba/hundreds-of-millions-of-dollars-locked-at-ethereum-0x0-address-and-smart-contracts-addresses-how-4144dbe3458a.

[42] Rafail Ostrovsky, Amit Sahai, and Brent Waters. 2007. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS*. 195–203.

[43] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *26th USENIX Security Symposium, USENIX Security*. 1199–1216.

[44] r3 [n.d.]. Banks complete 25 million euros securities transaction on blockchain platform. https://uk.reuters.com/article/uk-blockchain-securities/banks-complete-25-million-euros-securities-\transaction-on-blockchain-platform-\idUKKCN1GD4DW.

[45] Amit Sahai, Hakan Seyalioglu, and Brent Waters. 2012. Dynamic Credentials and Ciphertext Delegation for Attribute-Based Encryption. In *Advances in Cryptology - CRYPTO. Proceedings*. 199–217.

[46] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Bernardo Magri, Daniel Tschudi, and Aniket Kate. 2020. Reparo: Publicly Verifiable Layer to Repair Blockchains. *CoRR* abs/2001.00486 (2020). http://arxiv.org/abs/2001.00486

[47] Junichi Tomida, Yuto Kawahara, and Ryo Nishimaki. 2020. Fast, compact, and expressive attribute-based encryption. In *IACR International Conference on Public-Key Cryptography*. Springer, 3–33.

[48] Zhedong Wang, Xiong Fan, and Feng-Hao Liu. 2019. FE for Inner Products and Its Application to Decentralized ABE. In *Public-Key Cryptography - PKC, Proceedings, Part II*, Dongdai Lin and Kazue Sako (Eds.). 97–127.

[49] Brent Waters. 2011. Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization. In *Public Key Cryptography - PKC, Proceedings*. 53–70.

[50] Business Wire. 2016. Accenture Editable Blockchain. https://www.businesswire.com/news/home/20160920005551/en/Accenture-Debuts-Prototype-of-%E2%80%98Editable%E2%80%99-Blockchain-for-Enterprise-and-Permissioned-Systems.

[51] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. 2010. Attribute based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS*. 261–270.

10 APPENDIX

10.1 Computational Assumptions

Definition 10.1. *(Decision Linear Assumption (DLIN) [14]) Let* GroupGen$(1^\lambda) \longrightarrow (\mathbb{G}, \mathbb{H}, \mathbb{G}_T)$ *be a group generating algorithm, where* $|\mathbb{G}| = |\mathbb{H}| = |\mathbb{G}_T| = p$. *Let* $g \in \mathbb{G}$, $h \in \mathbb{H}$, *and* $e : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$. *Then* GroupGen *satisfies the decisional linear assumption (DLIN) if, for all probabilistic polynomial time A,*

$$Adv^A_{DLIN}(\lambda) = |Pr[A(1^\lambda, pub, D, T_0) = 1] - Pr[A(1^\lambda, par, D, T_1) = 1]|$$

is negligible in λ, *where* $pub = (\mathbb{G}, \mathbb{H}, \mathbb{G}_T, e, g, h)$, $a_1, a_2 \overset{R}{\leftarrow} \mathbb{Z}_p^*$, $s_1, s_2, \overset{R}{\leftarrow} \mathbb{Z}$, $D = (g^{a_1}, g^{a_2}, h^{a_1}, h^{a_2}, g^{a_1 s_1}, g^{a_2 s_2}, h^{a_1 s_1}, h^{a_2 s_2})$, $T_0 = (g^{s_1+s_2}, h^{s_1+s_2})$, $T_1 = (g^s, h^s)$.

Definition 10.2. *(DDH Problem [13]) We say that the DDH problem is hard relative to* $\mathcal{G}$ *if for all PPT algorithms A, there is a negligible function* negl *such that*

$$|Pr[A(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - Pr[A(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]| \leq \text{negl}(\lambda)$$

where in each case the probabilities are taken over the experiment in which $\mathcal{G}(1^\lambda)$ *outputs* $(\mathbb{G}, q, g)$, *and then uniform* $x, y, z \in \mathbb{Z}_q$ *are chosen.*

Definition 10.3. *(Monotone Access Structure [1]) If U denotes the universe of attributes, then an access structure* $\mathbb{A}$ *is a collection of non-empty subsets of U, i.e.,* $\mathbb{A} \subseteq 2^U \setminus 0$. *It is called monotone if for every* $B, C \subseteq U$ *such that* $B \subseteq C$, $B \in \mathbb{A} \implies C \in \mathbb{A}$.

Monotone access structures are encoded by monotone span programs [36], or linear secret sharing schemes [4], which we define below.

Definition 10.4. *(Monotone Span Program [36]) Let* $\Upsilon : \{0,1\}^n \rightarrow \{0,1\}$ *be a monotone Boolean function. A* monotone span program *for* Υ *over a field* $\mathbb{F}$ *is an* $l \times t$ *matrix* $\mathbf{M}$ *with entries in* $\mathbb{F}$, *along with a labeling function* $a : [l] \rightarrow [n]$ *that associates each row of* $\mathbf{M}$ *with an input variable of* Υ, *that, for every* $(x_1, \ldots, x_n) \in \{0,1\}^n$ *satisfies the following:*

$$\Upsilon(x_1, \ldots, x_n) = 1 \iff \exists \vec{v} \in \mathbb{F}^{1 \times l} : \vec{v}\mathbf{M} = [1, 0, 0, \ldots, 0] \; and (\forall\, i\; x_{a(i)} = 0 \rightarrow v_i = 0)$$

The Boolean function Υ *evaluates to 1, i.e.,* $\Upsilon(x_1, \ldots, x_n) = 1$ *if and only if the rows of* $\mathbf{M}$ *indexed by* $\{i | x_{a(i)} = 1\}$ *span the vector* $[1, 0, 0, \ldots, 0]$.

Alternatively, if S is a set of attributes and $I = \{i | i \in \{1, \ldots, n_1\}, \pi(i) \in S\}$ be a set of rows in $\mathbf{M}$ that belong to S. We say that $(\mathbf{M}, \pi)$ accepts S if there exist coefficients $\{\gamma_i\} i \in I$ such that $\sum_{i \in I} \gamma_i (\mathbf{M})_i = (1, 0, 0, \ldots, 0)$, where $(\mathbf{M})_i$ is the i^{th} row of $\mathbf{M}$.

A set S of attributes *satisfies* a Boolean formula if it causes the formula to evaluate to true on setting all inputs that map to some attributes in S to be true. The vast majority of practical ABE schemes in the cryptographic literature support access policies expressed as monotone span programs with AND and OR gates. There are a few inefficient schemes for supporting other kinds of gates [11, 33, 42], and ones based on lattices [31, 32] but we do not use them here.

10.2 Practical Applicability of ReTRACe

In this section we discuss two practical use cases of ReTRACe.

10.2.1 Smart Contracts. Smart contracts are high-level computer programs submitted to a BC by users in the system. Once written to the BC, a smart contract can be interacted with by users in the system who submit input transactions calling functions defined within the smart contract. Miners of the system execute the code of smart contracts on receiving input transactions. Smart contracts enforce agreements between two or more parties, and can be used in situations where otherwise a trusted third party or arbiter might be required, e.g., to enforce fair exchange of goods and services. There has been an extensive amount of research on smart contracts that address various interesting facets of smart contract deployment such as off-chain execution of smart contracts [24, 25], addressing and mitigating software bugs [35], and more. One of the problems of smart contracts is the amount of money locked up in them, e.g., due to errors in the smart contract code or due to user error while submitting a transaction. There are roughly more than $174 million dollars locked up at the $0x0$ address due to user error in submitting the transactions, about $1.2 million are locked up in the *ENG Smart Contract*, just to name a few. One could potentially use ReTRACe to solve this problem and reclaim the lost tokens and cryptocurrencies to the original senders [41].

The smart contract can be posted with as a ReTRACe message, allowing it to be modified or updated. The miners in the system are the only ones allowed to decrypt the trapdoor and update the smart contract through a specific process which validates the need for the update.

10.2.2 Financial Services Consortium. R3 [44] is a company that leads a global consortium of over two hundred members consisting of banks, trade associations and fintech companies, including U.S.-based members such as Bank of America, Goldman Sachs, Morgan Stanley, Wells Fargo, and international members such as Credit Suisse, Nomura, Deutsche Bank, Danske Bank, and several more. R3 has developed *Corda*, an open-source permissioned distributed ledger platform, designed to operate and execute financial transactions, while restricting access to transaction data. Corda has already been deployed to manage financial agreements, securities trading, inter-bank transactions, etc. between the members of the consortium.

Contracts between multiple banks involving trade instruments and securities, such as debts, bonds, money market instruments, equity warrants, convertibles, etc. are posted on the BC (i.e., Corda). This could be for reasons such as rogue employee posting sensitive information and/or offensive content and/or leaked encryption keys, error in the contract that needs to be expunged, etc. We envision ReTRACe could be very useful for such purposes, and fits the existing Corda ledger perfectly.

In this scenario, R3, in its capacity as leader of the consortium, can act as the AIA, and issue attributes to the members of the consortium. Although the members of the consortium include varied entities, let us consider banks as an example. A bank may have various internal departments, e.g., retail banking, credit operations, loan operations, private (high net-worth individuals) banking, etc., which are further divided into sub-departments, e.g., the loan operations departments could be have auto-loan, mortgage, student loan sub-departments. Each department or sub-department could be visualized as a group being headed by a group manager, GM, who issues signing keys to the group's members.

In this situation, a bank can post a contract to the BC, and create a trapdoor τ that enables specific banks' departments (after mutual consultations) to update the contract.[2] The trapdoor could be encrypted using the R3 public key, $\mathsf{mpk}_{\mathsf{ABE}}$, under an ABE policy, e.g., Υ_{ABE} = "Corporate Bonds Dept., Wells Fargo" $\vee$ "Equity Warrants Dept., Credit Suisse" $\vee$ "Money Market Instruments Dept., Deutsche Bank". The Υ_{GS} could be similarly constructed, e.g., Υ_{GS} = "Securities Dept., Wells Fargo" $\vee$ "Securities Dept., Credit Suisse" $\vee$ "Securities Dept., Deutsche Bank".

The advantage of keeping records on a BC is that any member of the consortium can, in the future, can verify a peer member's past records and thus creditworthiness, besides this platform is *already being deployed* by a multitude of banks and other fintech entities. Another advantage of using ReTRACe in this ledger is that any changes in contracts and parties can be seamlessly integrated, with minimal effort. Say, the bank that originally created a contract, τ and Υ_{ABE} now has sold some of its assets or transferred its liabilities and contracts to another member bank. All that needs to be done to enforce this change, is the banks need to update τ and Υ_{ABE}, and the old bank will no longer be able to update the contract. Finally, members of the general public should not be able to read the internal bank records, and certainly should not be able to write to them, which necessitates and justifies the use of a private permissioned blockchain such as Corda.

10.3 RCHET **Construction**

Our RCHET construction involves a non-interactive zero knowledge proof (NIZK), and hence requires a common reference string (crs), which is a random string produced by a trusted party, or group of parties, that is available to everyone in the system. The crs is generated by the two honest parties in our system: the AIA and the GM. We note that the crs is specific to our construction, and not integral to the idea of an RCHET scheme.

All NIZK proofs require a crs and the use of a crs has precedent: Camenisch *et al.* [18] used a crs in their CHET constructions and applications to *sanitizable signatures.* Hawk [37], a robust system for building privacy-preserving smart contracts uses a crs in pursuit of its goals. Succinct non-interactive arguments of knowledge which are used in various systems, the most prominent being ZCash [8], require and use a crs. Since, in our system, the AIA and GM are trusted, generating a crs should not be a problem; even if one relaxes these trust assumptions as outlined in Section 2 by using multi-AIA techniques, etc., the crs could be chosen using multi-party computations.

One could also use the methods of Groth *et al.* [34], where the crs is updatable and all parties contribute secret randomness to it. Another idea is to choose the crs using the methods of Bellare *et al.* [5] that guarantee security even when the crs is maliciously chosen.

10.4 RCHET **Security Properties and Proof**

Definition 10.5. *(Security of* RCHET*) An* RCHET *scheme is said to be secure if it possesses the following properties:*

1*) Correctness*: *We require that for all* $\lambda \in \mathbb{N}$*, for all* RCHET.systemSetup $(1^{\lambda}) \rightarrow$ (pubpar)*, for all* RCHET.userKeySetup (pubpar) $\rightarrow (sk_{ch}, pk_{ch})$*, for all* RCHET.cHash(sk_{ch}, pk_{ch}, m) $\rightarrow$ (digest, rand, $\Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}$)*, we have that* RCHET.verifyTransaction(pk_{ch}, m, digest, rand, Γ_{pubinfo}) $\rightarrow 1$. *We also require that for all* RCHET. adaptMessage(sk_{ch}, m, m', digest, rand, $\Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}$) $\rightarrow$ rand$'$*, we have that* RCHET.verifyTransaction(pk_{ch}, m, digest, rand$'$, Γ'_{pubinfo}) $\rightarrow 1$. *Furthermore, we require that for all* RCHET.adaptTrapdoor(sk_{ch}, m, m', digest, rand, Γ_{pubinfo}, Γ_{privinfo}) $\rightarrow$ (rand$'$, Γ'_{pubinfo}, $\Gamma'_{\text{privinfo}}$)*, we have that* RCHET.verifyTransaction(pk_{ch}, m, digest, rand$'$, Γ'_{pubinfo}) $\rightarrow 1$. *Here* $m, m' \in \mathcal{M}$, $\mathcal{M}$ *is a message space.*

2*) Indistinguishability*: *Let the advantage of an adversary,* $\mathcal{A}$*, in the indistinguishability game given in Figure 8 be defined as:*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{RCHET.Indistinguishability}}(\lambda) = Pr\left[\mathsf{Indistinguishability}^{\mathcal{A}}_{\mathsf{RCHET}}(\lambda) = 1\right].$$

An RCHET *scheme provides indistinguishability, if* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{RCHET.Indistinguishability}}(\lambda)$ *is a negligible function in* λ *for all PPT adversaries,* $\mathcal{A}$.

3*) Public collision-resistance*: *Let the advantage of an adversary,* $\mathcal{A}$*, in the public collision resistance game given in Figure 9 be defined as:* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{RCHET.PublicCollRes}}(\lambda) = Pr\left[\mathsf{PublicCollRes}^{\mathcal{A}}_{\mathsf{RCHET}}(\lambda) = 1\right]$. *An* RCHET *scheme provides public collision resistance, if* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{RCHET.PublicCollRes}}(\lambda)$ *is a negligible function in* λ *for all PPT adversaries,* $\mathcal{A}$.

4*) Private collision-resistance*: *Let the advantage of an adversary,* $\mathcal{A}$*, in the private collision resistance game given in Figure 10 be defined as:* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{RCHET.PrivateCollRes}}(\lambda) = Pr\left[\mathsf{PrivateCollRes}^{\mathcal{A}}_{\mathsf{RCHET}}(\lambda) = 1\right]$. *An* RCHET *scheme provides private collision resistance, if* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{RCHET.PrivateCollRes}}(\lambda)$ *is a negligible function in* λ *for all PPT adversaries,* $\mathcal{A}$.

5*) Revocation collision resistance*: *Let the advantage of an adversary,* $\mathcal{A}$*, in the revocation collision resistance game given in Figure 11 be defined as:*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{RCHET.RevocationCollRes}}(\lambda) = Pr\left[\mathsf{RevocationCollRes}^{\mathcal{A}}_{\mathsf{RCHET}}(\lambda) = 1\right].$$

[2]If a contract is unilaterally changed, the party that made the change can be traced and made to face consequences.

An RCHET *scheme provides revocation collision resistance, if* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{RCHET.RevocationCollRes}}(\lambda)$ *is a negligible function in* λ *for all PPT adversaries,* $\mathcal{A}$*.*

10.4.1 High-level Description of Games. Indistinguishability: We recall that the indistinguishability property requires that given an output, the adversary, $\mathcal{A}$, cannot tell if the output was a result of a cHash or adaptMessage or adaptTrapdoor. In our indistinguishability game in Figure 8, the adversary is given access to a HashOrAdapt oracle which takes as input a message m from $\mathcal{A}$, picks an integer $i \in \{0, 1, 2\}$, and returns the output of cHash or adaptMessage or adaptTrapdoor, respectively, depending on the value of i. $\mathcal{A}$ wins the game if it can guess i with probability greater than random guessing.

Public collision resistance: The public collision resistance property requires that an adversary who has neither the long-term trapdoor nor ephemeral trapdoor cannot successfully create collisions. In Figure 9, we need to give the adversary oracle access to both, the adaptMessage and adaptTrapdoor functionalities. We provide $\mathcal{A}$ access to a MsgOrTrap oracle, which takes an input bit b chosen by $\mathcal{A}$, and passes them on to an Adapt oracle, which either does an adaptMessage or adaptTrapdoor, depending on the value of b. The output of cHash and adaptMessage or adaptTrapdoor is returned to $\mathcal{A}$. $\mathcal{A}$ wins if it can successfully either adapt a message or adapt a trapdoor, where "successful" means that the output passes verification.

Private collision resistance: The private collision resistance property requires that even the holder of the long term trapdoor cannot find collisions, as long as the ephemeral trapdoor is unknown to them. In Figure 10, $\mathcal{A}$ picks the long term trapdoor and public key, the Adapt oracle creates digests, and either adapts a message or adapts the ephemeral trapdoor, per $\mathcal{A}$'s choice (governed by bit b). $\mathcal{A}$ is deemed to have won if $\mathcal{A}$ can successfully either adapt a message or adapt the trapdoor, and in both cases, produce a pre-image that maps on to one of the digests returned by the oracle.

Revocation collision resistance: Revocation collision resistance requires that someone who knows *both*, the long term and ephemeral trapdoors cannot successfully create collisions, after the ephemeral trapdoor has been updated, as long as the new trapdoor is unknown to them. In Figure 11, $\mathcal{A}$ creates a digest of a message, can adapt it and can also adapt the ephemeral trapdoor. The oracle Adapt is then invoked for updating the ephemeral trapdoor. After the update, $\mathcal{A}$ is tasked with either correctly updating the message or trapdoor. $\mathcal{A}$ wins if can successfully either adapt a message or adapt the trapdoor, and in both cases, produce a pre-image that maps on to one of the digests returned by the oracle.

10.4.2 RCHET *Proof.*

Proof. We need to prove our RCHET scheme provides indistinguishability, public collision resistance, private collision resistance and revocation collision resistance. We prove each property separately, and assume all communication between parties takes place via secure and authenticated channels. We need two additional games from [18], the zero knowledge game and simulation-sound extractability game, given in Figure 12a, and Figure 12b, respectively.

Indistinguishability: Trivial as an adversary will either see a hash or its adapted version (either as a result of an adaptTrapdoor or an adaptMessage), but will never see both, a hash and its adapted version at the same time.

Public Collision Resistance: Let us consider a sequence of games:

Game 0 (G0): The original public collision-resistance game.

Game 1 (G1): Same as **G0** but upon setup we obtain $(crs, \tau) \leftarrow S_1(1^\lambda)$, store τ and henceforth simulate all proofs using $S_2(crs, \tau, \cdot)$.

Transition - Game 0 $\longrightarrow$ *Game 1*: A distinguisher between **G0** and **G1** zero-knowledge distinguisher, i.e., $|\ Pr[\mathbf{G0}] - Pr[\mathbf{G1}]\ | \leq \nu_{zk}(\lambda)$.

Game 2 (G2): As **G1**, but upon setup we obtain $(crs, \tau, \chi) \leftarrow S_1(\lambda)$, and additionally store χ.

Transition - Game 1 $\longrightarrow$ *Game 2*: Under simulation sound extractability, this change is conceptual, i.e., $Pr[\mathbf{G1}] = Pr[\mathbf{G2}]$.

Game 3 (G3): As **G2**, but we simulate the Adapt oracle thus: to find a collision w.r.t. m, m', digest (b, h', π_t, C, C'), randomness (β, p, π_p), and trapdoor information $\Gamma_{\text{pubinfo}} = (\Delta, \pi_\Delta, D, \pi_D)$, $\Gamma_{\text{privinfo}} = (\delta, d, \text{etd})$, and $sk_{ch} = (SK, x)$, if rand $= (\beta, p, \cdot)$ corresponds to a previous Adapt query, set $AD = \top$ and $AD = \perp$ otherwise. If only p corresponds to a previous query, return $\perp$.

Transition - Game 2 $\longrightarrow$ *Game 3*: This change is conceptual, i.e., $Pr[\mathbf{G2}] = Pr[\mathbf{G3}]$ (observe that p is unconditionally binding, and, thus, modifying β implies that the check $b \overset{?}{=} \frac{h^\beta \cdot h'^a}{D \cdot \Delta}$ which is performed within Adapt fails, and the oracle would abort anyway).

Game 4 (G4): As **G3** but we further change the Adapt oracle thus: to find a collision w.r.t. m, m', digest (b, h', π_t, C, C'), randomness (β, p, π_p), and trapdoor information $\Gamma_{\text{pubinfo}} = (\Delta, \pi_\Delta, D, \pi_D)$, $\Gamma_{\text{privinfo}} = (\delta, d, \text{etd})$, and $sk_{ch} = (SK, x)$ do:

(1) If rand $= (\beta, p, \cdot)$ corresponds to a previous Adapt query, set $AD = \top$ and $AD = \perp$ otherwise. If only p corresponds to a previous query, return $\perp$.

(2) Decrypt $a \leftarrow \Pi.\text{Decrypt}(SK, C')$, check if $a \overset{?}{\leftarrow} H_k(m)$. $\boxed{\text{If } AD = \perp,}$ check if $b \overset{?}{=} \frac{h^\beta \cdot h'^a}{D \cdot \Delta}$. If checks fail, return $\perp$.

...

If $\mathcal{A}$ chose $b = 0$:

(4) $\boxed{\text{If } AD = \perp,}$ Decrypt $r \leftarrow \Pi.\text{Decrypt}(SK, C)$ and if $r = \perp$, return $\perp$.

else

(4) $\boxed{\text{If } AD = \perp,}$ Decrypt $r \leftarrow \Pi.\text{Decrypt}(SK, C)$ and if $r = \perp$, return $\perp$. Compute $a' \leftarrow H_k(m')$.

Transition - Game 3 $\longrightarrow$ *Game 4*: This change is conceptual, i.e., $Pr[\mathbf{G3}] = Pr[\mathbf{G4}]$ (the checks are only omitted if we know that they would not yield to an abort).

Game 5 (G5): As **G4**, but we further change the Adapt oracle as follows:

(1) If rand $= (\beta, p, \cdot)$ corresponds to a previous Adapt query, set $AD = \top$ and $AD = \perp$ otherwise. If only p corresponds to a previous query, return $\perp$.

(2) Decrypt $a \leftarrow \Pi.\text{Decrypt}(SK, C')$, check if $a \overset{?}{\leftarrow} H_k(m)$.

$\boxed{\text{If } AD = \perp,}$ check if $b \overset{?}{=} \frac{h^\beta \cdot h'^a}{D \cdot \Delta}$. If checks fail, return $\perp$.

...

If $\mathcal{A}$ chose $b = 0$:

(4) $\boxed{\text{If } AD = \perp,}$ Decrypt $r \leftarrow \Pi.\text{Decrypt}(SK, C)$ and if $r = \perp$, return $\perp$.

(5) Compute $a' \leftarrow H_k(m')$. $\boxed{\text{Compute } r' = garbage}$.

```
Game Indistinguishability$^{\mathcal{A}}_{\text{RCHET}}(\lambda)$
    1. systemSetup$(1^{\lambda}) \rightarrow$ (pubpar)
    2. userKeySetup(pubpar) $\rightarrow (sk_{ch}, pk_{ch})$
    3. $i \leftarrow \{0, 1, 2\}$
    4. $\mathcal{A}^{\text{HashOrAdapt}(sk_{ch},\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)}(pk_{ch}) \rightarrow i'$
        where oracle HashOrAdapt on input $(sk_{ch}, m, m')$ does:
        4.1. Do cHash$(sk_{ch}, pk_{ch}, m) \rightarrow$ (digest, rand$_0$, $\Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}$)
        4.2. If $i = 0$, Set $t_0$ = (digest, rand$_0$, $\Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}$)
        4.3. If $i = 1$, do adaptMessage$(sk_{ch}, m, m'$, digest, rand$_0$, $\Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}) \rightarrow$ rand$_1$
            Set $t_1$ = (digest, rand$_1$, $\Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}$)
        4.4. If $i = 2$, do adaptTrapdoor$(sk_{ch}, m, m'$, digest, rand$_0$, $\Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}) \rightarrow$ (rand$_2$, $\Gamma'_{\text{pubinfo}}, \Gamma'_{\text{privinfo}}$)
            Set $t_2$ = (digest, rand$_2$, $\Gamma'_{\text{pubinfo}}, \Gamma'_{\text{privinfo}}$)
        4.5. Verify output values, if any are $\perp$, return $\perp$
        4.6. Return $t_i$
    5. Return 1 if $(i' = i)$, else return 0.
```

Figure 8: RCHET **indistinguishability game**

```
Game PublicCollRes$^{\mathcal{A}}_{\text{RCHET}}(\lambda)$
    1. systemSetup$(1^{\lambda}) \rightarrow$ (pubpar)
    2. userKeySetup(pubpar) $\rightarrow (sk_{ch}, pk_{ch})$
    3. $Q \leftarrow \emptyset$, $\mathcal{A}$ picks $b \leftarrow \{0, 1\}$
    4. $\mathcal{A}^{\text{cHash}(sk_{ch},\cdot,\cdot)}(pk_{ch})$
        where oracle cHash on input $(sk_{ch}, pk_{ch}, m)$ does
        4.1. cHash$(sk_{ch}, pk_{ch}, m) \rightarrow$ (digest, rand, $\Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}$)
        4.2. return (digest, rand, $\Gamma_{\text{pubinfo}}$)
    5. $\mathcal{A}^{\text{Adapt}(sk_{ch},\cdot,\cdot,\cdot,\cdot,\cdot,\cdot),\text{MsgOrTrap}(\cdot)}(pk_{ch}) \rightarrow (m^*$, rand$^*$, $\Gamma^*_{\text{pubinfo}}, m^{*\prime}$, rand$^{*\prime}$, $\Gamma^{*\prime}_{\text{pubinfo}}$, digest$^*$)
        where oracle Adapt on input $(sk_{ch}, m, m')$, and oracle MsgOrTrap on input $b$ do:
        5.1. If $b = 0$, do adaptMessage$(sk_{ch}, m, m'$, digest, rand, $\Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}) \rightarrow$ rand$_0$
            Set $t_0$ = (rand$_0$, $\Gamma_{\text{pubinfo}}$)
        5.2. If $b = 1$, do adaptTrapdoor$(sk_{ch}, m, m'$, digest, rand, $\Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}}) \rightarrow$ (rand$_1$, $\Gamma'_{\text{pubinfo}}, \Gamma'_{\text{privinfo}}$)
            Set $t_1$ = (rand$_1$, $\Gamma'_{\text{pubinfo}}$)
        5.3. Verify output values, if any are $\perp$, return $\perp$
        5.4. $Q \leftarrow Q \cup \{m, m'\}$
        5.5. Return rand, $t_b$
    6. Return 1 if ((verifyTransaction$(pk_{ch}, m^*$, digest$^*$, rand$^*$, $\Gamma^*_{\text{pubinfo}}) \rightarrow 1$)
        $\wedge$(verifyTransaction$(pk_{ch}, m^{*\prime}$, digest$^*$, rand$^{*\prime}$, $\Gamma^*_{\text{pubinfo}}) \rightarrow 1$)$\vee$
        (verifyTransaction$(pk_{ch}, m^*$, digest$^*$, rand$^*$, $\Gamma^*_{\text{pubinfo}}) \rightarrow 1$)$\wedge$
        (verifyTransaction$(pk_{ch}, m^{*\prime}$, digest$^*$, rand$^{*\prime}$, $\Gamma^{*\prime}_{\text{pubinfo}}) \rightarrow 1$))$\wedge$
        $(m^{*\prime} \notin Q) \wedge (m^* \neq m^{*\prime}) \wedge (\Gamma^*_{\text{pubinfo}} \neq \Gamma^{*\prime}_{\text{pubinfo}})$. Else return 0.
```

Figure 9: RCHET **public collision resistance game**

(6) $\boxed{\text{Set } p' = h^{r'} \text{ and } \pi_{p'} \leftarrow \text{NIZKPoK}\{r' : p' = h^{r'}\}.}$
$\boxed{\text{Compute } \beta' \leftarrow (r + \frac{a.etd}{x} - \frac{a'.etd}{x} + \frac{\delta}{x} + \frac{d}{x}).}$
else
(4) $\boxed{\text{If } AD = \perp,}$ Decrypt $r \leftarrow \Pi$.Decrypt(SK, C) and if $r = \perp$, return $\perp$. Compute $a' \leftarrow H_k(m')$.
(6) $\boxed{\text{Set } r' = \perp, p' \leftarrow h^{r'}, \beta' \leftarrow (r + \frac{a.etd}{x} - \frac{a'.etd}{x} + \frac{\delta}{x} +}$
$\boxed{\frac{d'}{x}), \pi_{p'} \leftarrow \text{NIZKPoK}\{r' : p' = h^{r'}\}}$
Transition - Game 4 → Game 5:A distinguisher between **G4** and **G5** is an IND-CCA2 distinguisher for Π, i.e., $|\ Pr[\mathbf{G4}] - Pr[\mathbf{G5}]\ | \leq \nu_c(\lambda)$.
Game 6 (G6): As **G5** but for every query to Adapt, we store (β, p, π_p) if π_p was not previously simulated within Adapt in $R[(b, h', \pi_t, C, C')] \leftarrow (\beta, p, \pi_p)$. Now, for every forgery either both rand* or rand$^{*\prime}$ are fresh, or one of them contains a proof π_p (resp. π'_p) which we previously simulated in the Adapt oracle. If one of them contains such a proof, we replace the respective randomness tuple (β, p, π_p) by R[digest*].
Transition - Game 5 → Game 6: This change is conceptual, i.e.,

```
Game $\mathsf{PrivateCollRes}^{\mathcal{A}}_{\mathsf{RCHET}}(\lambda)$
    1. $\mathsf{systemSetup}(1^{\lambda}) \rightarrow (\mathsf{pubpar})$
    2. $\mathcal{A}(\mathsf{pubpar}) \rightarrow (sk^*_{ch}, pk^*_{ch})$
    3. $Q \leftarrow \emptyset$, $\mathcal{A}$ picks $b \leftarrow \{0, 1\}$
    4. $\mathcal{A}^{\mathsf{cHash}(\cdot,\cdot,\cdot)}(pk_{ch})$
        where oracle cHash on input $(sk^*_{ch}, pk^*_{ch}, m)$ does
        4.1. $\mathsf{cHash}(sk^*_{ch}, pk^*_{ch}, m) \rightarrow (\mathsf{digest}, \mathsf{rand}, \Gamma_{\mathsf{pubinfo}}, \Gamma_{\mathsf{privinfo}})$
        4.2. Return $(\mathsf{digest}, \mathsf{rand}, \Gamma_{\mathsf{pubinfo}})$
    5. $\mathcal{A}^{\mathsf{Adapt}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{MsgOrTrap}(\cdot)}(pk^*_{ch}) \rightarrow (m^*, \mathsf{rand}^*, \Gamma^*_{\mathsf{pubinfo}}, m^{*\prime}, \mathsf{rand}^{*\prime}, \Gamma^{*\prime}_{\mathsf{pubinfo}}, \mathsf{digest}^*)$
        where oracle Adapt on input $(sk^*_{ch}, m, m')$, and oracle MsgOrTrap on input $b$ do:
        5.1. If $b = 0$, do $\mathsf{adaptMessage}(sk^*_{ch}, m, m', \mathsf{digest}, \mathsf{rand}, \Gamma_{\mathsf{pubinfo}}, \Gamma_{\mathsf{privinfo}}) \rightarrow \mathsf{rand}_0$
            Set $t_0 = (\mathsf{rand}_0, \Gamma_{\mathsf{pubinfo}})$
        5.2. If $b = 1$, do $\mathsf{adaptTrapdoor}(sk^*_{ch}, m, m', \mathsf{digest}, \mathsf{rand}, \Gamma_{\mathsf{pubinfo}}, \Gamma_{\mathsf{privinfo}}) \rightarrow (\mathsf{rand}_1, \Gamma'_{\mathsf{pubinfo}}, \Gamma'_{\mathsf{privinfo}})$
            Set $t_1 = (\mathsf{rand}_1, \Gamma'_{\mathsf{pubinfo}})$.
        5.3. Verify output values, if any are $\perp$, return $\perp$
        5.4. $Q \leftarrow Q \cup \{\mathsf{digest}, m\}$
        5.5. Return $(\mathsf{digest}, \mathsf{rand}, t_b)$
    6. Return 1 if $((\mathsf{verifyTransaction}(pk^*_{ch}, m^*, \mathsf{digest}^*, \mathsf{rand}^*, \Gamma^*_{\mathsf{pubinfo}}) \rightarrow 1) \wedge$
        $(\mathsf{verifyTransaction}(pk^*_{ch}, m^{*\prime}, \mathsf{digest}^*, \mathsf{rand}^{*\prime}, \Gamma^*_{\mathsf{pubinfo}}) \rightarrow 1) \vee$
        $(\mathsf{verifyTransaction}(pk^*_{ch}, m^*, \mathsf{digest}^*, \mathsf{rand}^*, \Gamma^*_{\mathsf{pubinfo}}) \rightarrow 1) \wedge$
        $(\mathsf{verifyTransaction}(pk^*_{ch}, m^{*\prime}, \mathsf{digest}^*, \mathsf{rand}^{*\prime}, \Gamma^{*\prime}_{\mathsf{pubinfo}}) \rightarrow 1)) \wedge$
        $(\mathsf{digest}^*, m^{*\prime} \notin Q) \wedge (\mathsf{digest}^*, \cdot) \in Q \wedge (\Gamma^*_{\mathsf{pubinfo}} \neq \Gamma^{*\prime}_{\mathsf{pubinfo}})))$ else return 0.
```

Figure 10: RCHET **private collision resistance game**

```
Game $\mathsf{RevocationCollRes}^{\mathcal{A}}_{\mathsf{RCHET}}(\lambda)$
    1. $\mathsf{systemSetup}(1^{\lambda}) \rightarrow \mathsf{pubpar}$
    2. $\mathcal{A}(\mathsf{pubpar}) \rightarrow (sk^*_{ch}, pk^*_{ch})$
    3. $\mathcal{A}$ does $\mathsf{cHash}(sk^*_{ch}, pk^*_{ch}, m^*) \rightarrow (\mathsf{digest}^*, \mathsf{rand}^*, \Gamma^*_{\mathsf{pubinfo}}, \Gamma^*_{\mathsf{privinfo}})$
    4. $\mathcal{A}^{\mathsf{Adapt}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)}(pk^*_{ch}) \rightarrow (m^{*\prime\prime}, \mathsf{rand}^{*\prime\prime}, \{\Gamma^{*\prime\prime}_{\mathsf{pubinfo}}, \perp\}, \mathsf{digest}^*)$
        where oracle Adapt on input $(sk^*_{ch}, m^*, m^{*\prime}, \mathsf{digest}^*, \mathsf{rand}^*, \Gamma^*_{\mathsf{pubinfo}}, \Gamma^*_{\mathsf{privinfo}})$ does:
        4.1. If $\mathsf{verifyTransaction}(pk^*_{ch}, m^*, \mathsf{digest}^*, \mathsf{rand}^*, \Gamma^*_{\mathsf{pubinfo}}) \rightarrow 0$ return $\perp$
        4.2. Do $\mathsf{adaptTrapdoor}(sk^*_{ch}, m^*, m^{*\prime}, \mathsf{digest}^*, \mathsf{rand}^*, \Gamma^*_{\mathsf{pubinfo}}, \Gamma^*_{\mathsf{privinfo}}) \rightarrow (\mathsf{rand}^{*\prime}, \Gamma^{*\prime}_{\mathsf{pubinfo}}, \Gamma^{*\prime}_{\mathsf{privinfo}})$
        4.3. Verify output values, if any are $\perp$, return $\perp$
        4.4. Return $(m^{*\prime}, \mathsf{rand}^{*\prime}, \Gamma^{*\prime}_{\mathsf{pubinfo}})$
    5. Return 1 if $((\mathsf{verifyTransaction}(pk^*_{ch}, m^*, \mathsf{digest}^*, \mathsf{rand}^*, \Gamma^*_{\mathsf{pubinfo}}) \rightarrow 1) \wedge$
        $(\mathsf{verifyTransaction}(pk^*_{ch}, m^{*\prime\prime}, \mathsf{digest}^*, \mathsf{rand}^{*\prime\prime}, \Gamma^{*\prime}_{\mathsf{pubinfo}}) \rightarrow 1) \vee$
        $(\mathsf{verifyTransaction}(pk^*_{ch}, m^*, \mathsf{digest}^*, \mathsf{rand}^*, \Gamma^*_{\mathsf{pubinfo}}) \rightarrow 1) \wedge$
        $(\mathsf{verifyTransaction}(pk^*_{ch}, m^{*\prime\prime}, \mathsf{digest}^*, \mathsf{rand}^{*\prime\prime}, \Gamma^{*\prime\prime}_{\mathsf{pubinfo}}) \rightarrow 1))$. Else return 0.
```

Figure 11: RCHET **revocation collision resistance game**

$Pr[\mathbf{G5}] = Pr[\mathbf{G6}]$. Observe that the fact that a proof stems from a tuple returned by Adapt implies that a query with a tuple (β, p, π_p) where π_p was not simulated must once have happened. Further, the modified forgery is still a valid public collision freeness forgery.
Game 7 (G7): As **G6** but for the modified forgery we extract both rand and rand$'$ from π_p and π'_p contained in $\mathsf{rand}^* = (\beta, p, \pi_p)$ and $\mathsf{rand}^{*\prime} = (\beta', p', \pi'_p)$ If the extraction fails, we abort.
Transition - Game 6 → Game 7: Both games proceed identically, unless we have to abort, i.e., $|\ Pr[\mathbf{G6}] - Pr[\mathbf{G7}]\ | \leq 2\nu_e(\lambda)$.
Game 8 (G8): As G7 but for π_t contained in digest* we extract the etd and abort if the extraction fails.
Transition - Game 7 → Game 8: Both games proceed identically, unless we have to abort, i.e., $|\ Pr[\mathbf{G7}] - Pr[\mathbf{G8}]\ | \leq \nu_e(\lambda)$.
Game 9 (G9): As **G8** but we obtain a DL-challenge $(\mathbb{G}, g, q, g^x)$, perform the setup with respect to $(\mathbb{G}, g, q)$ and embed g^x into pk_{ch}.
Transition - Game 8 → Game 9: This change is conceptual, i.e.,

```
Game Zero − Knowledge$^{\mathsf{NIZKPoK}}_{\mathcal{A}}(\lambda)$
$b \leftarrow \{0, 1\}$
$(crs, \tau) \leftarrow S_1(1^\lambda)$
$a \leftarrow \mathcal{A}^{P_b(\cdot,\cdot)}(crs)$
   where oracle $P_0$ on input $(x, w)$ :
      return $\pi \leftarrow \{(w) : R(x, w) = 1\}$, if $(x, w) \in L$
      return $\perp$
   and oracle $P_1$ on input $(x, w)$ :
      return $\pi \leftarrow S_2(crs, \tau, x)$, if $(x, w) \in L$
      return $\perp$
return 1,  if $a = b$
return 0
```

(a) Zero knowledge game

```
Game SimSoundExt$^{\mathsf{NIZKPoK}}_{\mathcal{A},E}(\lambda)$
$(crs, \tau, \xi) \leftarrow S(1^\lambda)$
$(x, \pi) \leftarrow \mathcal{A}^{Sim(\cdot)}(crs)$
   where oracle $Sim$ on input $x$ :
      obtain $\pi \leftarrow S_2(crs, \tau, x)$
      set $Q^{Sim} \leftarrow Q^{Sim} \cup \{(x, \pi)\}$
$w \leftarrow \mathcal{E}(crs, \xi, x, \pi)$
return 1, if verify$(x, \pi) =$ true $\wedge$
      $(x, w) \notin R \wedge (x, \pi) \notin Q^{Sim}$
return 0
```

(b) Simulation sound extractability game

Figure 12: Additional games for RCHET **proof**

$| Pr[\mathbf{G8}] = Pr[\mathbf{G9}] |$.

Private Collision Resistance:
Game 0 (G0): The original private collision-resistance game.
Game 1 (G1): Same as **G0**, but upon setup we obtain $(crs, \tau) \leftarrow S_1(1^\lambda)$ upon setup, store τ and henceforth simulate all proofs using $S_2(crs, \tau, \cdot)$.
Transition - Game 0 ⟶ Game 1: A distinguisher between **G0** and **G1** is a zero-knowledge distinguisher, i.e., $| Pr[\mathbf{G0}] - Pr[\mathbf{G1}] | \leq \nu_{zk}(\lambda)$.
Game 2 (G2): As **G1** but upon setup we obtain $(crs, \tau, \chi) \leftarrow S_1(\lambda)$, and additionally store χ.
Transition - Game 1 ⟶ Game 2: Under simulation sound extractability, this change is conceptual, i.e., $Pr[\mathbf{G1}] = Pr[\mathbf{G2}]$.
Game 3 (G3): As **G2** but we modify the Adapt oracle so that it no longer draws etd uniformly at random but directly draws h' uniformly at random from G^*. To hash m w.r.t. $pk_{ch} = (PK, h, \pi_{pk})$ do $\boxed{h' \leftarrow G^*}$, $D \leftarrow g^d$, and $\cdots$
Game 4 (G4): As **G3** but for every π_p returned by cHash we record the value β so that $\beta = r$ in $R[\beta] \leftarrow r$:
Transition - Game 3 ⟶ Game 4: This change is conceptual, i.e., $Pr[\mathbf{G3}] = Pr[\mathbf{G4}]$
Game 5 (G5): As **G4** but pk^* output by the adversary. We extract x so that $g^x = h$. If the extraction fails, we abort.
Transition - Game 4 ⟶ Game 5: Both games proceed identically, unless we have to abort, i.e., $| Pr[\mathbf{G4}] - Pr[\mathbf{G5}] | \leq \nu_e(\lambda)$.
Game 6: As **G5**, but we obtain a DL instance (G, g, q, g^t), perform the setup with respect to (G, g, q) and further modify cHash thus: to hash m w.r.t. $pk_{ch} = (PK, h, \pi_{pk})$, do $\boxed{\text{Set } s \leftarrow \mathbb{Z}_q^*, h' \leftarrow (g^t)^s}$, $D \leftarrow g^d$, and $\cdots$. Furthermore, we record $S[h'] \leftarrow s$.
Transition - Game 5 ⟶ Game 6: This change is conceptual, i.e., $Pr[\mathbf{G5}] = Pr[\mathbf{G6}]$.
Game 7: As **G6**, but if π_p or π'_p contained in rand$* = (\beta, p, \pi_p)$ and rand$'* = (\beta', p', \pi'_p)$ do not correspond to a cHash answer we algebraically obtain r from $\beta \leftarrow (r + \frac{\delta}{x} + \frac{d}{x})$ and r' from $r' \leftarrow (\frac{rx + a \cdot \mathsf{etd} - a' \cdot \mathsf{etd} + \delta}{x})$, set $R[\beta] \leftarrow r$ or $R[\beta'] \leftarrow r'$.
Transition - Game 6 ⟶ Game 7: Both games proceed identically, i.e., $| Pr[\mathbf{G6}] = Pr[\mathbf{G7}]$.

Revocation Collision Resistance:
Game 0 (G0): The original revocation collision-resistance game.
Game 1 (G1): As Game 0, but after the Adapt oracle returns $(m^{*\prime}$, rand$^{*\prime}$, $\Gamma^{*\prime}_{\text{pubinfo}})$ and $\mathcal{A}$ runs RCHET.adaptTrapdoor and would fail in Step (6).
If the adversary is successful in Step (6) RCHET.adaptTrapdoor, it proved knowledge of $g^{\delta d}$, and it was able to extract δ and d values from Δ and D, respectively, thus breaking the DL assumption.
The advantage of the adversary is given by the following equation:

$$\begin{aligned}
&\mathsf{Adv}^{\mathcal{A}}_{\mathsf{RCHET}}(\lambda) \leq \Big[\mathsf{Adv}^{\mathcal{A}}_{\mathsf{NIZKPoK}}(\lambda) + \\
&\mathsf{Adv}^{\mathcal{A}}_{\mathsf{H}^{\mathsf{k}}_{\mathbb{Z}^*_{\mathsf{q}}}.\mathsf{CollisionResistance}}(\lambda) + \mathsf{Adv}^{\mathcal{A}}_{DL}(\lambda) + \\
&\mathsf{Adv}^{\mathcal{A}}_{DDH}(\lambda) + \mathsf{Adv}^{\mathcal{A}}_{\Pi.CCA}(\lambda)\Big]
\end{aligned}$$

□

10.5 RFAME Correctness

We need to verify that when (M, π) accepts S, decryption recovers the correct message with probability 1. For $l = 1, 2, 3$,

$$\begin{aligned}
\prod_{i \in I} ct_{i,l}^{\gamma_i} &= \prod_{i \in I} \Big(\mathcal{H}(\pi(i)l1)^{\gamma_i s_1} \cdot \mathcal{H}(\pi(i)l2)^{\gamma_i s_2} \cdot \\
&\qquad \prod_{j=1}^{n_2} \left[\mathcal{H}(0jl1)^{s_1} \cdot \mathcal{H}(0jl2)^{s_2} \right]^{\gamma_i (M)_{i,j}} \Big) \\
&= \Big(\prod_{j=1}^{n_2} \left[\mathcal{H}(0jl1)^{s_1} \cdot \mathcal{H}(0jl2)^{s_2} \right]^{\sum_{i \in I} \gamma_i (M)_{i,j}} \Big) \cdot \\
&\qquad \Big(\prod_{i \in I} \mathcal{H}(\pi(i)l1)^{\gamma_i s_1} \cdot \mathcal{H}(\pi(i)l2)^{\gamma_i s_2} \Big) \\
&= \mathcal{H}(01l1)^{s_1} \cdot \mathcal{H}(01l2)^{s_2} \cdot \\
&\qquad \prod_{i \in I} \mathcal{H}(\pi(i)l1)^{\gamma_i s_1} \cdot \mathcal{H}(\pi(i)l2)^{\gamma_i s_2}
\end{aligned}$$

Now, the product of all but the first term in the numerator, num, is given by:

$$\prod_{t\in\{1,2\}}\Big[e(\mathcal{H}(011t),h)^{b_1r_1s_t}\cdot e(\mathcal{H}(012t),h)^{b_2r_2s_t}$$
$$\cdot e(\mathcal{H}(013t),h)^{(r_1+r_2)s_t}\cdot\prod_{i\in I}\Big(e(\mathcal{H}(\pi(i)1t)^{\gamma_i},h)^{b_1r_1s_t}$$
$$\cdot e(\mathcal{H}(\pi(i)2t)^{\gamma_i},h)^{b_2r_2s_t}\cdot e(\mathcal{H}(\pi(i)3t)^{\gamma_i},h)^{(r_1+r_2)s_t}\Big)\Big]$$

The denominator, den, is given as:

$$\mathsf{den}=\prod_{i\in I}e(\mathsf{sk}^{\gamma_i}_{\rho(i),1},\mathsf{ct}_{\rho(i),1})\cdot\prod_{i\in I}e(\mathsf{sk}^{\gamma_i}_{\rho(i),2},\mathsf{ct}_{\rho(i),2})\cdot$$
$$\prod_{i\in I}e(\mathsf{sk}'_3\cdot\mathsf{sk}^{\gamma_i}_{\rho(i),3},\mathsf{ct}_{0,3})\cdot\prod_{t\in\{1,2\}}e(\mathsf{sk}'_t\cdot\mathsf{sk}''_t,\mathsf{ct}_{0,t})$$

We expand the first term of den as follows:

$$\prod_{i\in I}e(\mathsf{sk}^{\gamma_i}_{\rho(i),1},\mathsf{ct}_{\rho(i),1})=e(\mathcal{H}(\rho(i)11),h)^{\gamma_ib_1r_1s_1}\cdot$$
$$e(\mathcal{H}(\rho(i)21),h)^{\gamma_ib_2r_2s_1}\cdot$$
$$e(\mathcal{H}(\rho(i)31),h)^{\gamma_i(r_1+r_2)s_1}\cdot$$
$$e(q,h)^{\gamma_i\sigma_ys_1}\cdot e(q,h)^{\gamma_i\alpha_ys_1}$$

The expansion of the second and third terms of den are similar to the first term. We show the expansion of the fourth term below:

$$\prod_{t\in\{1,2\}}e(\mathsf{sk}'_t\cdot\mathsf{sk}''_t,\mathsf{ct}_{0,t})=e(\mathcal{H}(0111),h)^{b_1r_1s_1}\cdot$$
$$e(\mathcal{H}(0121),h)^{b_2r_2s_1}\cdot$$
$$e(\mathcal{H}(0131),h)^{(r_1+r_2)s_1}\cdot e(g,h)^{\sigma's_1}\cdot$$
$$e(\mathcal{H}(0112),h)^{b_1r_1s_2}\cdot$$
$$e(\mathcal{H}(0122),h)^{b_2r_2s_2}\cdot$$
$$e(\mathcal{H}(0132),h)^{(r_1+r_2)s_2}\cdot e(g,h)^{\sigma's_2}\cdot$$
$$e(g,h)^{d_1p_1a_1s_1}\cdot e(g,h)^{d_2p_2a_2s_2}$$

After all the terms of num and den are expanded, all terms cancel out, leaving only *msg*.

10.6 RFAME Security Analysis

We give the full IND-CPA security game for CPABE schemes in Definition 10.6.

Definition 10.6. *(CPABE Full IND-CPA Game)*

(1) Setup phase*: Challenger runs* Setup$(1^\lambda)\rightarrow(mpk,msk)$ *and obtains a master public and secret key pair. The mpk is given to $\mathcal{A}$.*
(2) Query phase*: $\mathcal{A}$ generates an attribute set $\mathbb{S}$ and sends $\mathbb{S}$ to challenger. Challenger generates secret keys for attributes in $\mathbb{S}$ by running* KeyGen$(msk,\mathbb{S})\rightarrow(sk_1,sk_2,\ldots,sk_{|\mathbb{S}|})$ *for all attributes in $\mathbb{S}$. The secret keys are given to $\mathcal{A}$. $\mathcal{A}$ can query the challenger a polynomially-bounded number of times.*
(3) Challenge phase*: $\mathcal{A}$ submits two messages, m_0,m_1, as well as a challenge access policy Υ to the challenger. $\mathcal{A}$ cannot submit a policy for which $\Upsilon(\mathbb{S})=1$. That is, no combination of $\mathcal{A}$'s attributes should satisfy Υ. Challenger does* Encrypt$(mpk,m_b,\Upsilon)\rightarrow C$, *where $b\in\{0,1\}$, and gives C to $\mathcal{A}$.*
(4) Query phase repeat*: $\mathcal{A}$ can again query for secret keys for attribute sets, except attribute sets that satisfy Υ.*
(5) Response phase*: $\mathcal{A}$ outputs b'.*

The advantage of an adversary in this game is defined as $Pr[b'-b]-\frac{1}{2}$.

Definition 10.7. *A ciphertext policy attribute-based encryption scheme is fully IND-CPA secure if all probabilistic polynomial time $\mathcal{A}$ have at most a negligible advantage in the game in Definition 10.6.*

We now give the full IND-CPA game for RFAME in Definition 10.8. We follow the representations of [1, 21] where the group elements are compactly represented as matrices, e.g., $[x]_1$ denotes g^x, $[y]_2$ denotes h^y and $[z]_T$ denotes $e(g,h)^z\in\mathbb{G}_T$, where $g\in\mathbb{G}$, $h\in\mathbb{H}$, $x,y\overset{R}{\leftarrow}\mathbb{Z}_q^*$, and $|\mathbb{G}|=|\mathbb{H}|=q$. Vector $(g^{v_1},g^{v_2},\ldots,g^{v_n})$ is denoted as $[v]_1$, and correspondingly for $[v]_2$. Similarly, $[M]_1$ denotes a matrix of elements fro group $\mathbb{G}$, and $[M]_2$ denotes a matrix of elements from group $\mathbb{H}$, while $e([A]_1,[B]_2)$ is defined as $[A^\top B]_T$.

Definition 10.8. *(Full IND-CPA game for* RFAME*)*
Setup phase*: The challenger runs the* Setup *algorithm to generate* $\mathsf{mpk_{ABE}},\mathsf{msk_{ABE}}$*. It generates the group parameters $(q,\mathbb{G},\mathbb{H},\mathbb{G}_T,e(g,h)$. It picks $(A,a^\perp)$, $(B,b^\perp)$, $(Z,\alpha^\perp)$, $(P,p^\perp)$, picks $d_1,d_2,d_3,p_1,p_2,p_3\overset{R}{\leftarrow}\mathbb{Z}_q$. It sets the vectors $d=(d_1,d_2,d_3)^\top$, $p=(p_1,p_2,p_3)^\top$. It sets* $\mathsf{mpk_{ABE}}=([A]_2,[Pd^\top A]_T,[Z]_2)$ *and* $\mathsf{msk_{ABE}}=(g,h,A,B,P,[d]_1,Z)$.
Query phase*: Adversary, $\mathcal{A}$, will create a set of attributes, $\mathbb{S}$ and will query the challenger for secret keys for all attributes in $\mathbb{S}$. Let us assume challenger maintains two lists, L,Q for simulating the two kinds of inputs/outputs of the random oracle $\mathcal{H}$. L has entries of the form (y,W_y) or (j,U_j) where y is a binary string, $j\in\mathbb{Z}^+$, and W_y,U_j are 3×3 matrices whose elements are drawn from $\mathbb{Z}_q$. List Q has entries of the form (f,r) where $r\in\mathbb{G}$, and f is either xlt or $0jlt$ for $l\in\{1,2,3\}$ and $t\in\{1,2\}$.*

$\mathcal{A}$ needs to request keys for attributes $y\in\mathbb{S}$. $\mathcal{A}$ will create attributes strings $y1t$, $y2t$, $y3t$ where $t\in\{1,2\}$. This is used by the challenger in the creation of the $\mathsf{sk}_{y,t}$. $\mathcal{A}$ will also construct bit strings of the form $01lt$ for $l\in\{1,2,3\}$ and $t\in\{1,2\}$. When $\mathcal{A}$ queries on an attribute $y\in\mathbb{S}$, the challenger retrieves matrices W_y and U_j from L. We recall that these matrices store attribute representations of the form ylt and $0jlt$. $\mathcal{A}$ can make one of three kinds of queries:
1) ylt: The challenger checks if $(ylt)\in Q$ for some r. If yes, it returns r, else it checks if $(y,W_y)\in L$ for some W_y. If yes, it computes $r=[(W_y^\top A)_{l,1}]_1$, add (ylt,r) to Q, and returns r. Else pick $W_y\overset{R}{\leftarrow}\mathbb{Z}_q$, add (y,W_y) to L, compute and return $r=[(W_y^\top A)_{l,t}]_1$, and add (ylt,r) to Q.
2) $0jlt$: The challenger checks if $(0jlt,r)\in Q$for some r. If yes, return r. Else check if $(j,U_j)\in L$ for some U_j. If yes, compute $r=[(U_j^\top A)_{l,t}]_1$, add $(0jlt,r)$ to Q an return r. Else pick $U_j\overset{R}{\leftarrow}\mathbb{Z}_q$, add (y,U_j) to L, compute $r=[(U_j^\top A)_{l,t}]_1$, add $(0jlt,r)$ to Q an return r.
3) Any other query, q: Challenger checks if $(q,r)\in Q$ for some r. If yes, return r. Else pick $r'\overset{R}{\leftarrow}\mathbb{G}$, add (q,r') to Q, and return r'.

Next, the challenger picks $r_1, r_2, \sigma_y, \sigma' \xleftarrow{R} \mathbb{Z}_q$. Let $r = (r_1, r_2)^\top$. The challenger needs to set up $\mathsf{sk}_0, \mathsf{sk}_y, \mathsf{sk}', \mathsf{sk}''$*, which it computes as follows:*

$$\mathsf{sk}_0 = [Br]_2 \quad \mathsf{sk}_y = [W_y Br + \sigma_y a^\perp + \sigma_y \alpha^\perp + \alpha_y]_1$$
$$\mathsf{sk}' = [d + U_1 Br + \sigma' a^\perp]_1 \quad \mathsf{sk}'' = [Pd]_1$$

The challenger then returns $(\mathsf{sk}_0, \{\mathsf{sk}_y\}_{y \in \mathbb{S}}, \mathsf{sk}', \mathsf{sk}'')$ *as the signing key for attributes in* $y \in \mathbb{S}$.

Challenge phase*: In the challenge phase, $\mathcal{A}$ will ask for an encryption of one of (m_0, m_1), and will send a policy (M, π), where M is a monotone span program representation of a policy, and π is a mapping of row numbers to attributes. The usual restriction holds that any combination of attributes in $\mathbb{S}$ that $\mathcal{A}$ has queried in the query phase should not satisfy (M, π), else challenger returns $\perp$. Let us assume M has n_1 rows and n_2 columns. The challenger picks $b \in \{0, 1\}$, retrieves $[(W_{\pi(i)}^\top A)_{l,t}]_1$, and $[(U_j^\top A)_{l,t}]_1$ for all $i \in [1..n_1]$ and all $j \in [1..n_2]$, $l, t \in Q$. The challenger picks $s_1, s_2 \xleftarrow{R} \mathbb{Z}_q$. It computes:*

$$\mathsf{ct}_0 = [As \cdot \alpha_{\rho(i)}]_2 \quad \mathsf{ct}_i = \left[W_{\pi(i)}^\top As + \sum_{j=1}^{n_2} (M)_{i,j} U_j^\top As \right]_1$$
$$\mathsf{ct}' = [d^\top PAs]_T \cdot m_b$$

.

The challenger then returns $(\mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_{n_1}, \mathsf{ct}')$ *for all i rows of M.*

Repeat query/output phase*: $\mathcal{A}$ can query the challenger on more attributes sets, $\mathbb{S}' \subset U$.*

Response phase*: $\mathcal{A}$ will output its guess b'.*

We now give the proof of Theorem 5.1

Proof. The proof is formulated via a series of hybrids, and is based on the proof structure of [1]. A key can be one of the following forms, with variables in the key being progressively replaced from the previous form: 1) Normal, as generated in the RFAME CPA game. 2) P-normal, where Br is replaced with $Br + \hat{r}a^\perp$ where $\hat{r} \xleftarrow{R} \mathbb{Z}_q$. 3) P-normal*, where we remove $\sigma_y a^\top + \sigma_y \alpha^\top \,\forall\, y \in \mathbb{S}$, and remove $\sigma' a^\top$ from a P-normal key. 4) Semi-normal*, where α_y is removed from a P-normal* key. 5) Normal*, where $Br + \hat{r}a^\top$ is replaced by Br in a P-normal* key. 6) P-SF*, where $\chi a^\top$ is added to the last component of a P-normal* key, where $\chi \xleftarrow{R} \mathbb{Z}_q$. 7) SF* where $Br + \hat{r}a^\top$ is replaced by Br in a P-SF* key.

The ciphertext is of the forms: 1) Normal*, which are the ciphertexts generated in the RFAME CPA game. 2) SF*, where the As in an Normal* ciphertext is replaced by $As + \hat{s}b^\top$, $\hat{s} \xleftarrow{R} \mathbb{Z}_q$. 3) Rnd*, where the entire message msg_b is replaced by $msg^* \xleftarrow{R} \mathbb{G}$.

The first step of the proof is to replace all keys that $\mathcal{A}$ queries in the RFAME CPA game progressively from Normal to Normal*, and to gradually convert the challenge ciphertext from Normal* to Rnd*. So, for all of $\mathcal{A}$'s queried keys in the query phase, first change the key from Normal to P-normal, then to P-normal*, to Normal*. We then take the challenge ciphertext, of the form Normal*, then convert it to SF*, and finally to Rnd*, and convert the keys from Normal* to P-SF*, and finally to SF*. The idea is, after these conversions have been applied, $\mathcal{A}$ will be handed SF* keys and Rnd* ciphertext. We describe the sequence of hybrids:

(1) Hyb_1 is the original RFAME CPA game.
(2) Hyb_2 is same as Hyb_1, additionally, the first $i-1$ keys queried by $\mathcal{A}$ are Normal*, i^{th} key is Normal*, and remaining keys are Normal.
(3) Hyb_3 is same as Hyb_2, additionally, the challenge ciphertext is SF*.
(4) Hyb_4 is same as Hyb_3, additionally, the first $i-1$ keys are SF*, i^{th} key is SF*, and remaining keys are Normal*.
(5) Hyb_5 is same as Hyb_4, additionally, the ciphertext is converted to Rnd*.

We now describe how the our representation models the ciphertext in our RFAME construction. For $i \in [1..n_1]$, and $l \in \{1, 2, 3\}$, and $s = (s_1, s_2)^\top$.

$$\mathsf{ct}_0 = [As \cdot \alpha_{\rho(i)}]_2 +, \mathsf{ct}' = [d^\top PAs]_T \cdot msg_b$$

$$\mathsf{ct}_{i,l} = \Big[(W_{\pi(i)}^\top A)_{l,1} s_1 + (W_{\pi(i)}^\top A)_{l,2} s_2 + \sum_j \{(U_j^\top A)_{l,1} s_1 + (U_j^\top A)_{l,2} s_2\} (M)_{i,j} \Big]_1$$

The key $\mathsf{sk}_{y,t}$ is defined as:

$$\mathsf{sk}_{y,t} = \Big[(W_y^\top A)_{1,t} \cdot \frac{b_1 r_1}{a_t + \alpha_y} + (W_y^\top A)_{2,t} \cdot \frac{b_2 r_2}{a_t + \alpha_y} + (W_y^\top A)_{3,t} \cdot \frac{r_1 + r_2}{a_t + \alpha_y} + \frac{\sigma_y}{a_t + \alpha_y} + \frac{\alpha_y}{a_t + \alpha_y} \Big]_1$$

which can be rewritten as $(W_y Br)_t + a_t^{-1} \alpha^{-1} [(W_y Br)_3 + \sigma_y + \alpha_y]$, where $r = (r_1 r_2)^\top$. The key $\mathsf{sk}_{y,3}$ is identically distributed to $[W_y Br + \sigma_y \alpha_y] a^\perp]_1$. The next part of the key, sk' is identically distributed to $[d + U_1 Br + \sigma' a^\top]_1$, and finally $\mathsf{sk}_0 = [Br]_2$.

Now we define the hybrids as defined in [1], where $\mathcal{A}$'s key queries run from [1..N]. The Hyb_2 can be split up into four hybrids:

(1) $\mathsf{Hyb}_{2,1,n}$, which is the same as RFAME CPA game, except that the first $i-1$ keys are Normal*, i^{th} key is P-Normal, and remaining keys are Normal.
(2) $\mathsf{Hyb}_{2,2,n}$, which is same as $\mathsf{Hyb}_{2,1,n}$, except the i^{th} key becomes P-normal*.
(3) $\mathsf{Hyb}_{2,3,n}$, which is same as $\mathsf{Hyb}_{2,2,n}$, except the i^{th} key becomes Semi-normal*.
(4) $\mathsf{Hyb}_{2,4,n}$, which is the same as $\mathsf{Hyb}_{2,2,n}$, except that the i^{th} key becomes Normal*.

Similarly, Hyb_4 can be split up into:

(1) $\mathsf{Hyb}_{4,1,n}$, which is the same as Hyb_3 except that the first $i-1$ keys are SF*, i^{th} key is P-normal*, and remaining keys are Normal*.
(2) $\mathsf{Hyb}_{4,2,n}$ which is the same as $\mathsf{Hyb}_{4,1,n}$, except the i^{th} key becomes Semi-normal*.
(3) $\mathsf{Hyb}_{4,3,n}$, which is same as $\mathsf{Hyb}_{4,1,n}$, except the i^{th} key becomes P-SF*.

(4) $\mathsf{Hyb}_{4,4,n}$, which is the same as $\mathsf{Hyb}_{4,2,n}$, except that the i^{th} key becomes SF*.

We first need to show that Hyb_2 is indistinguishable from Hyb_1. Consider the four hybrids comprising Hyb_2. In $\mathsf{Hyb}_{2,1,n}$, the i^{th} key is P-normal, and remaining keys are Normal, whereas in Hyb_1, all keys were Normal. This proof is the same as the one in [1], and our extra α terms in the secret keys and ciphertext does not change anything. Similarly the arguments for the transition from $\mathsf{Hyb}_{2,1,n}$ to $\mathsf{Hyb}_{2,2,n}$ are the same as those in [1], and are given below.

We will need the following two lemmas from [1]:

Lemma 10.1. *For all $i \in [1..N]$ and PPT adversaries $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathcal{A}}_{(2,3,n-1),(2,3,n)}(\lambda) \leq \mathsf{Adv}^{\mathcal{B}}_{DLIN}(\lambda) + 1/q$$

Lemma 10.2. *For all $i \in [1..N]$ and PPT adversaries $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathcal{A}}_{(2,1,n),(2,2,n)}(\lambda) \leq 2/p$$

The transition from $\mathsf{Hyb}_{2,2,n}$ to $\mathsf{Hyb}_{2,3,n}$ is unique to RFAME, and involves the concept of Semi-normal* keys, which were not there in [1]. We need to prove that this transition is indistinguishable from $\mathcal{A}$'s standpoint.

Lemma 10.3. *For all PPT adversaries $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathcal{A}}_{(2,2,n),(2,3,n)}(\lambda) \leq \mathsf{Adv}^{\mathcal{B}}_{DLIN}(\lambda) + 1/p$$

Proof. We recollect that in the $\mathsf{Hyb}_{2,3,n}$ game, all keys are of the form Normal*, but the challenge ciphertext is Normal. In Hyb_3, all keys are Normal*, but the ciphertext is of the form SF*. $\mathcal{B}$ is the DLIN adversary, and simulates the challenger in the RFAME CPA game.

Let us rewrite the DLIN assumption as:

$$([A]_1, [A]_2, [As]_1, [As]_2) \approx ([A]_1, [A]_2, [s']_1, [s']_2) \qquad (1)$$

where

$$A = \begin{bmatrix} a_1 & 0 \\ 0 & a_2 \\ 1 & 1 \end{bmatrix}, \quad s = \begin{bmatrix} s_1 \\ s_2 \\ -1 \end{bmatrix} \quad s' = \begin{bmatrix} s_1 \\ s_2 \\ s \end{bmatrix}$$

Let $\mathcal{B}$ be a DLIN adversary that acts as a challenger for $\mathcal{A}$ in $\mathcal{A}$'s RFAME CPA game. $\mathcal{B}$ first gets its DLIN challenge as either $([B]_1, [B]_2, [Br^*]_1, [Br^*]_2)$ or $([B]_1, [B]_2, [r']_1, [r']_2)$ as its DLIN challenge. $\mathcal{B}$ then needs to setup the $\mathsf{mpk}_{\mathsf{ABE}}$, and provide $\mathcal{A}$ with keys in the query phase. $\mathcal{B}$ sets up the $\mathsf{mpk}_{\mathsf{ABE}}$ as $([A]_2, [Pd^\top, A]_T, [Z]_2)$ by sampling $d \leftarrow_\$ \mathbb{Z}_q,, (A, a^\perp), (Z, \alpha^\perp), (P, p^\perp) \leftarrow \mathsf{Samp}(q)$.

Now $\mathcal{B}$ needs to simulate keys for $\mathcal{A}$'s queries. Recollect that in $\mathsf{Hyb}_{2,2,n}$, the i^{th} key is of the form $\mathsf{sk}_0 = [Br + \hat{r}a^\perp]_2$, $\mathsf{sk}_{y,t} = [W_y(Br + \hat{r}a^\perp) + \alpha_y]_1$, $\mathsf{sk}' = [d + U_1 Br + \hat{r}a^\perp]_1$, and $\mathsf{sk}'' = [Pd]_1$. In $\mathsf{Hyb}_{2,3,n}$, all the keys, except the i^{th} key is an Semi-normal* key. This key needs to be generated by $\mathcal{B}$, and is of the form $\mathsf{sk}_0 = [Br + \hat{r}a^\perp]_2$, $\mathsf{sk}_{y,t} = [W_y Br + \hat{r}a^\perp]_1$, $\mathsf{sk}' = [d + U_1 Br + \hat{r}a^\perp]_1$, and $\mathsf{sk}'' = [Pd]_1$. The only term different from a P-normal* key and an Semi-normal* key is the α_y term. $\mathcal{B}$ can easily simulate this by doing $\alpha_y \leftarrow_\$ \mathbb{Z}_q$. If $\hat{r} = 0$, the view of $\mathcal{A}$ is identical to $\mathsf{Hyb}_{2,2,n}$, else it is similar to $\mathsf{Hyb}_{2,3,n}$. □

Lemma 10.4. *For all $i \in [1..N]$ and PPT adversaries $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathcal{A}}_{(2,3,3),(2,3,N)}(\lambda) \leq \mathsf{Adv}^{\mathcal{B}}_{DLIN}\lambda + 1/q$$

Proof. We want to prove that the view of an adversary in $\mathsf{Hyb}_{2,3,n}$ is identically distributed to its view in $\mathsf{Hyb}_{2,4,n}$. Recollect that the difference between $\mathsf{Hyb}_{2,3,n}$ and $\mathsf{Hyb}_{2,4,N}$ is that the i^{th} key is Normal* in the latter case. Let V be a matrix defined as $V^\top A = VB = 0$, and let $V = (b^\perp)^\top a^\perp$. Let β be defined ads the inner product of $a^\perp$ and $b^\perp$. Let $W_y^* = W_y - \sigma_x(\beta\hat{r})^{-1}V - \alpha_y(\beta\hat{r})^{-1}V - \alpha_y$, and $U_j^* = U_j - \sigma'(\beta\hat{r})^{-1}V$, where $\sigma_y, \sigma', \alpha_y, \hat{r} \leftarrow_\$ \mathbb{Z}_q$. Then the i^{th} key can be written as:

$$\begin{aligned}
&W_y^*(Br + \hat{r}a^\perp) + \sigma_y a^\perp + \alpha_y a^\perp \\
&\quad = (W_y - \sigma_y(\beta\hat{r})^{-1}V - \alpha_y(\beta\hat{r})^{-1}V - \alpha_y)(Br + \hat{r}a^\top) + \\
&\qquad \sigma_y a^\perp + \alpha_y a^\perp + \alpha_y \\
&\quad = W_y Br - \sigma_y(\beta\hat{r})^{-1}V\hat{r}a^\perp - \alpha_y(\beta\hat{r})^{-1}V\hat{r}a^\perp V\hat{r}a^\perp + \\
&\qquad W_y\hat{r}a^\perp + \sigma_y a^\perp + \alpha_y a^\perp + \alpha_y \\
&\quad = W_y(Br + \hat{r}a^\top) - \sigma_y\beta^{-1}\beta^{a^\top} - \alpha_y\beta^{-1}\beta^{a^\top}\sigma_y a^\perp + \alpha_y a^\perp \\
&\qquad = W_y(Br + \hat{r}a^\perp)
\end{aligned}$$

Similarly, $d + U_1^*(Br + \hat{r}a^\top) + \sigma' a^\top = d + U_1(Br + \hat{r}a^\perp)$. This is exactly how the i^{th} key of $\mathsf{Hyb}_{2,4,N}$ is distributed. □

At this point, the keys are all in the form Normal*. We observe that the next game transitions are $\mathsf{Hyb}_{2,4,N}$ to Hyb_3, and $\mathsf{Hyb}_{4,3,n-1}$ to $\mathsf{Hyb}_{4,3,N}$. These transition is similar to the one in [1], since, in game Hyb_3, all keys are Normal*, and only the challenge ciphertext is SF*. In the latter case, additionally, i^{th} key is P-normal*. We give their lemmas below:

Lemma 10.5. *For all PPT adversaries $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathcal{A}}_{(2,3,N),3}(\lambda) \leq \mathsf{Adv}^{\mathcal{B}}_{DLIN}\lambda + 1/q$$

Lemma 10.6. *For all $i \in [1..N]$ and PPT adversaries $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathcal{A}}_{(4,3,n-1),(4,1,n)}(\lambda) \leq \mathsf{Adv}^{\mathcal{B}}_{DLIN}\lambda + 1/q$$

We need to prove that $\mathcal{A}$'s view in $\mathsf{Hyb}_{4,3,n}$ is indistinguishable from the view in $\mathsf{Hyb}_{4,1,n}$.

Lemma 10.7. *For all $i \in [1..N]$ and PPT adversaries $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathcal{A}}_{(4,1,n),(4,3,n)}(\lambda) \leq \mathsf{Adv}^{\mathcal{B}}_{DLIN}\lambda + 1/q$$

Proof. In $\mathsf{Hyb}_{4,1,n}$ the i^{th} key is P-normal*, whereas in $\mathsf{Hyb}_{4,2,n}$, it is Semi-normal*. In both cases, the ciphertext is of the form SF*. Let us consider a matrix V such that $V = (b^\perp)^T a^\perp$. Let $\beta = \langle a^\perp, b^\perp \rangle$

Let V be a matrix defined as $V^\top A = VB = 0$, and let $\beta = (b^\perp)^\top a^\perp$. Let $W_y^* = W_y - \sigma_x(\beta\hat{r})^{-1}V - \alpha_y(\beta\hat{r})^{-1}V - \alpha_y$, and $U_j^* = U_j - \sigma'(\beta\hat{r})^{-1}V$, where $\sigma_y, \sigma', \alpha_y, \hat{r} \leftarrow_\$ \mathbb{Z}_q$. Let $w = (w_1, \ldots, w_{n_2})$ be the vector that satisfies (M, π), i.e., w is a [1,0,..,0] vector. Set $W_y =$

$W_y + \mu_y\beta^{-1}V$ and $U_j = U_j + \chi \cdot w_j\beta^{-1}V$, where $\mu_y, \chi \leftarrow_\$ \mathbb{Z}_q$. The i^{th} key for $\mathsf{sk_y}, \mathsf{sk}', \mathsf{sk}_0$, respectively, can be written as:

$$[W_y(Br + \hat{r}a^\perp) + \alpha_y + \mu_y\hat{r}a^\perp]$$

$$[d + U_1(Br + \hat{r}a^\perp) + \chi w_1\hat{r}a^\perp]$$

$$[Br + \hat{r}a^\perp]$$

The ciphertexts will correspondingly be:

$$\mathsf{ct}_i = W_{\pi(i)}^\top(As + \hat{s}b^\perp) + \sum_j (M)_{i,j}U_j^\top(As + \hat{s}b^\perp) + \left(\mu_{\pi(i)} + \chi\sum_j (M)_{i,j}w_j\right)\hat{s}b^\perp$$

$$\mathsf{ct}_0 = [As + \hat{s}b^\perp \cdot \alpha_{\pi(i)}]$$

$$\mathsf{ct}' = [d^\top PAs + \hat{s}b^\perp]_T \cdot m_b$$

We now replace $\mu_{\pi(i)} + \chi\sum_j (M)_{i,j}w_j$ by $\mu_{\pi(i)}$ (since $\sum_j (M)_{i,j}w_j = 0$, if y satisfies S), and replace $\chi w_1\hat{r}a^\top$ with $\chi a^\top$ in sk'. We then replace α_y from sk_y, since $[W_y(Br+\hat{r}a^\perp)+\alpha_y+\mu_y\hat{r}a^\perp]$ comes from an identical distribution. Finally, if we replace $W_y = W_y - \mu_y\beta^{-1}V$, the challenge ciphertext becomes SF*, and i^{th} key is in P-SF*, which is exactly $\mathcal{A}$'s view in $\mathsf{Hyb}_{4,3,n}$. □

At this point, the i^{th} key is P-SF*, and ciphertext is SF*. The last thing we need to cover is the transition from $\mathsf{Hyb}_{4,3,N}$ to Hyb_5. This is exactly the same as one given in [1], since our extra terms from the keys and ciphertext are already removed at this point.

Lemma 10.8. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathcal{A}}_{(4,3,N),5}(\lambda) \le 2/q$$

Putting everything together, from Lemma 10.1, Lemma 10.2, Lemma 10.3, Lemma 10.4, Lemma 10.5, Lemma 10.6, Lemma 10.7, and Lemma 10.8 we see that all the game transitions are indistinguishable from the point of view of $\mathcal{A}$, and Hyb_1 is indistinguishable from Hyb_5, thus concluding the proof. □

10.7 Dynamic Group Signature Scheme

Definition 10.9. *(Dynamic Group Signature Scheme, DGSS [15]) A DGSS scheme consists of the following algorithms:*

(1) $\mathsf{GSetup}(1^\lambda) \to PP$*: This algorithm is run to setup the public parameters of the system, PP.*
(2) $\mathsf{GKGen}(PP) \to (out_{GM}, st_{GM})$*: The group manager uses this algorithm to generate* $out_{GM} = (mpk, \mathrm{info}_0)$*, consisting of the manager's public key and the initial group information,* info_0*, and the resulting state,* st_{GM} *of the group manager. The group public key is* $gpk = (PP, mpk)$*.*
(3) Join*: To enroll a user as a member, the GM may run the interactive joining protocol with her. Their respective algorithms are:*
- $\mathsf{Join}_{User}^{\mathsf{WriteReg}(i,\cdot)}(M, st) \to (out, M_{GM}, st)$*: This algorithm specifies the user's execution of the interactive joining protocol with GM. Given an input message from GM and the user's internal state st, it returns a message for GM and a new state. In its first call, the algorithm is executed on initial input (init, gpk). In each instance of the protocol, the user is allowed a single call to the oracle* $\mathsf{WriteReg}(i,\cdot)$ *for writing into the registration table an entry* reg_i *corresponding to its identifier i. Joining session i terminates after at most* $k(\lambda)$ *rounds by a call returning* (gsk, M_{GM}, st)*, which includes the user's secret key* $gsk_i = gsk$*, an optional final message for the issuer including a termination message* done*, and the user final state. If it terminates with* $gsk = \perp$*, the user will consider it as a fail to join, and on failure it will always be the case that it ends with* $M_{GM} = (\mathrm{done}, \perp)$*. After termination, the user will ignore all future inputs to* Join_{User}*.*
- $\mathsf{Join}_{GM}^{\mathsf{ReadReg}(i)}(i, M_{GM}, st_{GM}) \to (out_{GM}, M, st_{GM})$*: This algorithm specifies GM's execution in the interactive joining protocol with a user. The GM keeps track of distinct instances of the protocol using unique identifiers i, which we, without loss of generality, assume are numbered 1, 2, 3, etc. The algorithm receives as input a session identifier i, a message* M_{GM} *received from the user, and the GM's internal state and it returns a message M for the user interacting in session i and updates the state* st_{GM}*. The algorithm has access to the oracle* $\mathsf{ReadReg}(i)$ *to read the entry* reg_i *in the registration table* Reg*. Each joining session will terminate after at most a polynomial number of rounds. We let* $k(\lambda)$ *be the maximal number of rounds before termination. Termination will be indicated in the local output* out_{GM} *of the manager GM and can be successful* $(\top)$ *or fail* $(\perp)$*, and if it fails the output message will be* $M_i = (\mathrm{done}, \perp)$*. After termination, GM ignores future calls with the same i.*
  For conciseness we will often refer to the user involved in the i^{th} *session of the* Join *protocol with the manager as user i. We note that the user may not be aware of her own session identifier i, since she may not be aware of how many other users are joining or have already joined the group.*

(4) $\mathsf{UpdateGroup}(\mathcal{R}, st_{GM}) \to (\mathrm{info}, st_{GM})$*: The group manager runs this algorithm to update the group information, where the set* $\mathcal{R}$ *consists of session identifiers associated with users to be revoked. The algorithm returns new public group information* info *and updates the state of GM. The group information* info *may or may not depend on the set of newly joined members of the group, which the group manager records in its internal state. The group information* info *is intended as group information pertaining to the group and we will in general assume anybody may have access to the sequence* $\mathrm{info}_0, \mathrm{info}_1$*, etc., the group manager creates during the lifetime of the group signature scheme.*
(5) $\mathsf{IsActive}(i, \tau, st_{GM}) \to \{0,1\}$*: The* Join *protocol and the* UpdateGroup *algorithm describe how an honest GM adds and revokes group members. The exact moment when a member is activated and able to sign is design specific. In some constructions, group members are implicitly activated after successfully terminating the* Join *protocols and may even be able to sign*

with respect to previous epochs; in others they are explicitly activated by GM when a new group information info_τ *is published. Consequently, different design choices lead to different time spans when members are allowed to sign. In order to take into account these differences in the security definitions without favoring a particular design paradigm, we use the* IsActive *procedure, which should be interpreted as the group manager's policy for when a member is considered active.*
The IsActive *algorithm takes as input a session identifier i, an epoch* τ *associated with group information* info_τ*, the group manager has published earlier, and the state of the group manager* st_{GM}*. We refer to a user as an active member of the group at epoch* τ *if and only if the algorithm returns 1. We place the following constraints on the policies an honest group manager can have for when a user is active:*

- *If* τ *is not associated with any* info *the group manager has published, the algorithm returns 0.*
- *If i is not associated with a joining session where the group manager has terminated successfully, the algorithm returns 0.*
- *If i was revoked when creating* info *for this epoch or earlier, the algorithm returns 0.*
- *If i is associated with a joining session where the group manager ended her part successfully before* info_τ *was created, and user i is not revoked at or before epoch* τ*, the algorithm returns 1.*

(6) $\text{Sign}(gsk, \text{info}, m) \rightarrow \Sigma$*: Given a user's group signing key gsk, group information* info*, and a message m, the signing algorithm outputs a group signature,* Σ*.*

(7) $\text{VerifySignature}(gpk, \text{info}, m, \Sigma) \rightarrow \{0, 1\}$*: The verification algorithm checks whether* Σ *is a valid group signature on m with respect to the group information* info *and outputs a bit: 1 for accept and 0 for reject.*

(8) $\text{UserTrace}(gpk, st_{GM}, \text{info}, m, \Sigma) \rightarrow (i, \pi)$*: The opening algorithm receives as input the group public key gpk, the state of the group manager, some public group information* info*, a message, and a signature. It returns a session identifier i together with a proof* π *attributing* Σ *to user i. If the algorithm is unable to attribute the signature to a particular group member, it returns* $(\perp, \pi)$ *to indicate that it could not attribute the signature.*

(9) $\text{Judge}(gpk, \text{info}, \text{reg}, m, \Sigma, \pi) \rightarrow \{0, 1\}$*: The judge algorithm checks the validity of a proof* π *attributing the signature* Σ *on m w.r.t. group information* info *to a user with registry record reg. It outputs 1 for accept and 0 for reject.*

10.8 ReTRACe Security Properties and Proof

We formalize the security properties in Definition 10.10. We recall that λ is the security parameter, U is the universe of attributes, and S is the set of attributes whose keys are requested by adversary $\mathcal{A}$.

Definition 10.10. *(Security of* ReTRACe*) A* ReTRACe *scheme is said to be secure if it possesses the following properties:*

1) Correctness*: We require that for all* $\lambda \in \mathbb{N}$*, for all* $S \in U$*, for all* $\text{ReTRACe.Keygen}(1^\lambda) \rightarrow (\text{PubPar}, \text{SecPar})$*, for all* ReTRACe. UserSetup $(\text{PubPar}, \text{SecPar}) \rightarrow (gsk, SK)$*, for all* ReTRACe.CreateMessage $(\text{key}, \text{PubPar}, m) \rightarrow (msg, \xi_{msg})$*, it holds true that* $\text{ReTRACe.Verify}(msg, \xi_{msg}, \text{PubPar}) \rightarrow 1$*, and* $\text{ReTRACe.VerifyMiner}(msg, \xi_{msg}, \text{PubPar}, \cdot) \rightarrow 1$*, We also require that for all* $\text{ReTRACe.AdaptMessage}(msg, m', \text{PubPar}) \rightarrow (msg', \xi_{msg'})$*, it holds that* $\text{ReTRACe.Verify}(msg', \xi_{msg'}, \text{PubPar}) \rightarrow 1$ *and correspondingly for* ReTRACe.VerifyMiner.

We also require that for all $\text{ReTRACe.RevokeUser}(msg, m'', \text{PubPar}) \rightarrow (msg'', \xi_{msg''})$*, it holds that* $\text{ReTRACe.Verify}(msg'', \xi_{msg''}) \rightarrow 1$*, and correspondingly for* ReTRACe.VerifyMiner*. Here* $m, m', m'' \in \mathcal{M}$ *where* $\mathcal{M}$ *is a message space.*

2) Indistinguishability*: Let the advantage of an adversary* $\mathcal{A}$ *in the indistinguishability game given in Figure 13 be defined as:*
$\text{Adv}^{\mathcal{A}}_{\text{ReTRACe.Indisguishability}}(\lambda) = Pr\,[\text{Indis}^{\mathcal{A}}_{\text{ReTRACe}}(\lambda) = 1]$.
A ReTRACe *scheme provides indistinguishability if*
$\text{Adv}^{\mathcal{A}}_{\text{ReTRACe.Indisguishability}}(\lambda)$ *is a negligible function in the security parameter* λ*.*

3) Public collision-resistance*: Let the advantage of an adversary* $\mathcal{A}$ *in the public collision resistance game given in Figure 14 be defined as:* $\text{Adv}^{\mathcal{A}}_{\text{ReTRACe.PublicCollRes}}(\lambda) = Pr\,[\text{PublicCollRes}^{\mathcal{A}}_{\text{ReTRACe}}(\lambda) = 1]$.
A ReTRACe *scheme provides public collision resistance if*
$\text{Adv}^{\mathcal{A}}_{\text{ReTRACe.PublicCollRes}}(\lambda)$ *is a negligible function in the security parameter* λ*.*

4) Private Collision Resistance*: Let the advantage of an adversary* $\mathcal{A}$ *in the private collision resistance game given in Figure 15 be defined as:* $\text{Adv}^{\mathcal{A}}_{\text{ReTRACe.PrivateCollRes}}(\lambda) = Pr[\text{PrivateCollRes}^{\mathcal{A}}_{\text{ReTRACe}}(\lambda) = 1]$. *A* ReTRACe *scheme provides private collision resistance if*
$\text{Adv}^{\mathcal{A}}_{\text{ReTRACe.PrivateCollRes}}(\lambda)$ *is a negligible function in the security parameter* λ*.*

5) Revocation collision resistance*: Let the advantage of an adversary* $\mathcal{A}$ *in the revocation collision resistance game given in Figure 16 be defined as:*

$$\text{Adv}^{\mathcal{A}}_{\text{ReTRACe.RevocationCollRes}}(\lambda) = Pr[\text{RevocationCollRes}^{\mathcal{A}}_{\text{ReTRACe}}(\lambda) = 1].$$

A ReTRACe *scheme provides public collision resistance if*
$\text{Adv}^{\mathcal{A}}_{\text{ReTRACe.PublicCollRes}}(\lambda)$ *is a negligible function in the security parameter* λ*.*

10.8.1 ReTRACe *Games and Proof.* In this section we give the proof of Theorem 7.1 w.r.t. the properties in Definition 10.10. We have proven the IND-CPA security of RFAME, but the proof requires an IND-CCA2 scheme. We apply the Fujisaki-Okamoto transform [30] to convert RFAME into an IND-CCA2-secure scheme, and then proceed with the proof.

Proof. **Indistinguishability**: Let $\mathcal{A}$ be a ReTRACe adversary, and let $\mathcal{B}$ be the adversary in the RCHET game. $\mathcal{B}$ obtains pk_{ch}, sk_{ch} from its RCHET challenger. We will focus on showing that $\mathcal{B}$ can successfully answer its RCHET challenge, if $\mathcal{A}$ can successfully win the ReTRACe indistinguishability game. $\mathcal{B}$ gets its $(sk_1, \cdots, sk_{|\mathcal{S}|})$ and $(gsk_1, \cdots, gsk_{|\mathcal{S}_{GS}|})$ from the ABE and GSS oracles respectively, and passes them on to $\mathcal{A}$. $\mathcal{B}$ simulates the CreateUpdateOrRevoke oracle by internally calling it own RCHET HashOrAdapt oracle, which returns $(\text{digest}, \text{rand}_0, \Gamma_{\text{pubinfo}}, \Gamma_{\text{privinfo}})$ to $\mathcal{A}$, Depending on $\mathcal{A}$'s choice of integer i, $\mathcal{B}$ sends i to its own RCHET challenger, and passes on the t_i that the RCHET challenger returns to $\mathcal{A}$. It is easy to see that if $\mathcal{A}$ wins the ReTRACe indistinguishability game, $\mathcal{B}$ will win its RCHET game.

Public Collision Resistance: $\mathcal{B}$ does the setup of keys, gives $\mathcal{A}$ PubPar. $\mathcal{B}$ gets its $(sk_1, \cdots, sk_{|\mathcal{S}|})$ and $(gsk_1, \cdots, gsk_{|\mathcal{S}_{GS}|})$ from the ABE and GSS oracles respectively, and passes them on to $\mathcal{A}$. $\mathcal{B}$ calls the AIARevoke oracle to revoke a user, and passes on the updated PubPar′ to $\mathcal{A}$. $\mathcal{B}$ then simulates the CreateUpdateOrRevoke oracle by internally invoking its own RCHET, RFAME and DGSS challengers. $\mathcal{B}$ creates two policy attributes, id_1, id_2 that satisfy $\mathcal{A}$'s Υ_{ABE} and Υ_{GS}. It then calls its own RCHET oracle to simulate AdaptMessage, and calls its DGSS oracle to produce valid signatures. Note that while simulating the CreateUpdateOrRevoke oracle, $\mathcal{B}$ will need to decrypt RFAME ciphertexts (contained within msg tuple) and needs access to a decryption oracle; this is where we need RFAME to be CCA2-secure. The rest of the simulation proceeds normally, with $\mathcal{B}$ returning either an adapted message or trapdoor to $\mathcal{A}$ depending on $\mathcal{A}$'s choice of i.

When $\mathcal{A}$ outputs a successful collision based on either a successful message adaptation or a successful trapdoor adaption, $\mathcal{B}$ will output this as a response to its RCHET challenger. $\mathcal{B}$ will output the signatures contained in ξ_{msg^*} as the response to its own DGSS challenger.

Private Collision Resistance: $\mathcal{B}$ will get the public parameters of ReTRACe, PubPar. It also gets the long-term trapdoor of the RCHET scheme, (SK, sk_{ch}). $\mathcal{B}$ passes along (SK, sk_{ch}) and PubPar to $\mathcal{A}$. $\mathcal{B}$ initializes Q, queries RFAME and GSS oracles for $\mathcal{S}_{\text{ABE}}$ and $\mathcal{S}_{\text{GS}}$, respectively, and passes on answers to $\mathcal{A}$. In the challenge phase, $\mathcal{B}$ will parse $\mathcal{A}$'s $\Upsilon_{\text{info}} = (\Upsilon_{\text{ABE}}, \Upsilon_{\text{GS}})$, do necessary checks, callsits RCHET and RFAME challenger for for simulating ReTRACe.CreateMessage, ReTRACe.AdaptMessage, and ReTRACe.RevokeUser. When $\mathcal{A}$ outputs its response, $\mathcal{B}$ will immediately break DGSS and RCHET similar to the public collision resistance game. For breaking RFAME, $\mathcal{B}$ needs to guess the bit b of its own challenger. Note that $\mathcal{A}$ can either request AdaptMessage or RevokeUser, but not both. When $\mathcal{B}$ simulates AdaptMessage, it uses m_0 of its own challenge as the Γ_{privinfo}, else it uses m_1. It $\mathcal{A}$ outputs a successful collision by doing an AdaptMessage, i.e., $(\text{verify}(\text{PubPar}, msg^*, \xi_{msg^*}) \longrightarrow 1 \wedge \text{verify}(\text{PubPar}, msg^{*\prime}, \xi_{msg^{*\prime}}) \longrightarrow 1)$, $\mathcal{B}$ outputs 0, else it outputs 1 as its guess for its RFAME challenger.

Revocation Collision Resistance: $\mathcal{B}$ gets public parameters of the system, passes on PubPar to $\mathcal{A}$; in this case, $\mathcal{B}$ gets both, the longterm trapdoor and the ephemeral trapdoor of the RCHET scheme, $(SK, sk_{ch}, \Gamma_{\text{privinfo}} = (d, \delta, \text{etd}))$, all of which are passed along to $\mathcal{A}$. $\mathcal{B}$ queries its RFAME and GSS challengers, gets $\mathcal{S}_{\text{ABE}}, \mathcal{S}_{\text{GS}}$ and gives them to $\mathcal{A}$. $\mathcal{B}$ simulates CreateMessage, AdaptMessage and RevokeUser similar to the prior properties. When $\mathcal{A}$ outputs its response, $\mathcal{B}$ will immediately break the DGSS and RCHET schemes. For RFAME, we notice that we cannot use the same strategy as in the public and private collision resistance games, since in the revocation collision resistance game, $\mathcal{B}$ knows $(SK, sk_{ch}, \Gamma_{\text{privinfo}} = (d, \delta, \text{etd}))$. $\mathcal{B}$ in this case does not need to use its RFAME decryption oracle. $\mathcal{B}$ will construct its response to $\mathcal{A}$'s adaptTrapdoor query in a way that it sets $m_0 = \Gamma'_{\text{privinfo}}$, and $m_1 \leftarrow\!\!\$\, \mathbb{Z}_q^*$. When $\mathcal{A}$ returns a successful collision, $\mathcal{B}$ will take $\mathcal{A}$'s successful collision, $(\text{verify}(\text{PubPar}, msg^*, \xi_{msg^*}) \longrightarrow 1 \wedge \text{verify}(\text{PubPar}, msg^{*\prime\prime}, \xi_{msg^{*\prime\prime}}) \longrightarrow 1)$, and check if it can extract the $\Gamma'_{\text{privinfo}}$. If yes, $\mathcal{B}$ returns 0, else it returns 1. ◻

```
Game $\mathsf{Indis}^{\mathcal{A}}_{\mathsf{ReTRACe}}(\lambda)$
    1. $\mathsf{Keygen}(1^{\lambda}) \longrightarrow (\mathsf{SecPar}, \mathsf{PubPar}, sk_{ch})$
    2. $\mathcal{A}^{\mathsf{ABEKeyGen(SecPar)}}(\mathsf{PubPar}, \mathcal{S}_{\mathsf{ABE}}) \longrightarrow (sk_1, \ldots, sk_{|S|})$
    3. $\mathcal{A}^{\mathsf{GSKeyGen(SecPar)}}(\mathsf{PubPar}, \mathcal{S}_{\mathsf{GS}}) \longrightarrow (gsk_1, \ldots, gsk_{|\mathcal{S}_{\mathsf{GS}}|})$
    4. $i \longleftarrow \{0, 1, 2\}$
    5. $\mathcal{A}^{\mathsf{CreateUpdateOrRevoke(key,\cdot,\cdot,\cdot,\cdot)}}(\mathsf{PubPar}, \mathsf{info}_{\tau}, \Upsilon_{\mathsf{info}}, \Upsilon_{\mathsf{admin}}) \longrightarrow i'$
        where oracle HashOrAdapt on input $(\mathsf{key}, \mathsf{PubPar}, m, m')$ does:
        /* $\Upsilon_{\mathsf{info}}$ and $\Upsilon_{\mathsf{admin}}$ given by $A$ is used inside the CreateMessage */
        5.1 Parse $\Upsilon_{\mathsf{info}} = (\Upsilon_{\mathsf{ABE}}, \Upsilon_{\mathsf{GS}})$, check if $\Upsilon_{\mathsf{ABE}}(\mathcal{S}_{\mathsf{ABE}}) \neq 1$, and similarly for $\Upsilon_{\mathsf{admin}}$. If yes, return $\perp$
        5.2. Do $\mathsf{CreateMessage}(\mathsf{key}, \mathsf{PubPar}, m) \longrightarrow (msg, \xi_{msg})$
        5.3. If $i = 0$, Set $t_0 = (msg, \xi_{msg})$
        5.4. If $i = 1$, do $\mathsf{AdaptMessage}(\mathsf{key}, \mathsf{PubPar}, m', msg, \xi_{msg}) \longrightarrow (msg', \xi_{msg'})$
            Set $t_1 = (msg', \xi_{msg'})$
        5.5. If $i = 2$, do $\mathsf{RevokeUser}(\mathsf{key}, \mathsf{PubPar}, m', msg, \xi_{msg}) \longrightarrow (msg'', \xi_{msg''})$
            Set $t_2 = (msg'', \xi_{msg''})$
        5.6. Verify all returned values, if any are $\perp$, return $\perp$
        5.7. Return $t_i$
    6. Return 1 if $i' = i$, else return 0
```

Figure 13: ReTRACe **indistinguishability game**

```
Game $\mathsf{PublicCollRes}^{\mathcal{A}}_{\mathsf{ReTRACe}}(\lambda)$
    1. $\mathsf{Keygen}(1^{\lambda}) \longrightarrow (\mathsf{PubPar}, \mathsf{SecPar})$
    2. $Q \longleftarrow \emptyset$, picks $i \longleftarrow_{\$} \{0, 1, 2\}$
    3. $\mathcal{A}^{\mathsf{ABEKeyGen(SecPar)}}(\mathsf{PubPar}, \mathcal{S}_{\mathsf{ABE}}) \longrightarrow (sk_1, \ldots, sk_{|\mathcal{S}_{\mathsf{ABE}}|})$
    4. $\mathcal{A}^{\mathsf{GSKeyGen(SecPar)}}(\mathsf{PubPar}, \mathcal{S}_{\mathsf{GS}}) \longrightarrow (gsk_1, \ldots, gsk_{|\mathcal{S}_{\mathsf{GS}}|})$
    5. $\mathcal{A}^{\mathsf{AIARevoke}(\cdot, \mathsf{msk_{ABE}}, \nu)}(\mathsf{mpk_{ABE}}) \longrightarrow (\mathsf{mpk_{ABE}}')$ and $\mathcal{A}$ updates PubPar to PubPar$'$
    6. $\mathcal{A}^{\mathsf{CreateUpdateOrRevoke(key,\cdot,\cdot,\cdot,\cdot)}}(\mathsf{PubPar}, \Upsilon_{\mathsf{info}}, m, m', \Upsilon_{\mathsf{admin}}, \mathsf{PubPar}') \longrightarrow$
    $(msg^*, \xi_{msg^*}, msg^{*\prime}, \xi_{msg^{*\prime}})$
        where oracle CreateUpdateOrRevoke on input $(m, m', \mathsf{key}, \mathsf{PubPar}, \mathsf{PubPar}')$ does:
        /*$\Upsilon_{\mathsf{info}}$ and $\Upsilon_{\mathsf{admin}}$ given by $\mathcal{A}$ is used inside the CreateMessage */
        6.1. Parse $\Upsilon_{\mathsf{info}} = (\Upsilon_{\mathsf{ABE}}, \Upsilon_{\mathsf{GS}})$, if either $\Upsilon_{\mathsf{ABE}}(\mathcal{S}_{\mathsf{ABE}}) = 1$, $\Upsilon_{\mathsf{admin}}(\mathcal{S}_{\mathsf{ABE}}) = 1$ or $\Upsilon_{\mathsf{GS}}(\mathcal{S}_{\mathsf{GS}}) = 1$, return $\perp$
        6.2. Create two policy attributes, $id_1, id_2$ s.t., $\Upsilon_{\mathsf{ABE}}(id_1, id_2) = 1$ and $\Upsilon_{\mathsf{GS}}(id_1, id_2) = 1$
        6.3. Do $\mathsf{CreateMessage}(\mathsf{key}, \mathsf{PubPar}, m) \longrightarrow (msg, \xi_{msg})$
        6.4. If $i = 0$, do $\mathsf{AdaptMessage}(\mathsf{key}, \mathsf{PubPar}, m', msg, \xi_{msg}) \longrightarrow (msg', \xi_{msg'})$
            Set $t_0 = (\mathsf{PubPar}, m, msg, \xi_{msg}, m', msg', \xi_{msg'})$
        /* Next, we consider Case 1 of RevokeUser below, i.e., new $\Upsilon'_{\mathsf{info}}$ is created*/
        6.5. If $i = 1$, create $\Upsilon'_{\mathsf{info}} = (\Upsilon'_{\mathsf{ABE}}, \Upsilon_{\mathsf{GS}})$, s.t., $\Upsilon'_{\mathsf{ABE}}(id_1) = 1$
            Do $\mathsf{RevokeUser}(\mathsf{key}, \mathsf{PubPar}, m', msg, \xi_{msg}) \longrightarrow (msg', \xi_{msg'})$
            Set $t_1 = (\mathsf{PubPar}, m, msg, \xi_{msg}, m', msg', \xi_{msg'})$
        /* Now we consider Case 2 of RevokeUser below, i..e, when AIA revokes a user's $id_2$ attribute*/
        6.6. If $i = 2$, Use PubPar$'$
            Do $\mathsf{RevokeUser}(\mathsf{key}, \mathsf{PubPar}', m', msg, \xi_{msg}) \longrightarrow (msg', \xi'_{msg})$
            Set $t_2 = (\mathsf{PubPar}, m, msg, \xi_{msg}, m', msg', \xi_{msg'}, \mathsf{PubPar}')$
        5.7. $Q \longleftarrow Q \cup \{msg^*.m^*, msg^{*\prime}.m^*\}$
        5.8. return $t_i$
    6. If $(((\mathsf{verify}(\mathsf{PubPar}, msg^*, \xi_{msg^*}) \longrightarrow 1) \wedge (\mathsf{verify}(\mathsf{PubPar}, msg^{*\prime}, \xi_{msg^{*\prime}}) \longrightarrow 1) \wedge (msg^{*\prime} \notin Q) \wedge (msg^* \neq msg^{*\prime}))\vee$
        $((\mathsf{verify}(\mathsf{PubPar}, msg^*, \xi_{msg^*}) \longrightarrow 1) \wedge (\mathsf{verify}(\mathsf{PubPar}', msg^{*\prime}, \xi_{msg^{*\prime}})) \longrightarrow 1)\wedge$
        $(msg^{*\prime}.m^*, msg^*.m^* \notin Q) \wedge (msg^* \neq msg^{*\prime}))$ , return 1, else return 0.
```

Figure 14: ReTRACe **public collision resistance game**

```
**Game** $\mathsf{PrivateCollRes}^{\mathcal{A}}_{\mathsf{ReTRACe}}(\lambda)$
    1. $\mathsf{Keygen}(1^{\lambda}) \rightarrow (\mathsf{SecPar}, \mathsf{PubPar})$
    2. $\mathcal{A}(pp_{\mathsf{RCHET}}) \rightarrow (sk^*_{ch}, pk^*_{ch})$, $\mathcal{A}$ picks $i \leftarrow_{\$} \{0, 1, 2\}$
    3. $Q \leftarrow \emptyset$
    4. $\mathcal{A}^{\mathsf{ABEKeyGen}(\mathsf{SecPar})}(\mathsf{PubPar}, \mathcal{S}_{\mathsf{ABE}}) \rightarrow (sk_1, \ldots, sk_{|\mathcal{S}_{\mathsf{ABE}}|})$
    5. $\mathcal{A}^{\mathsf{GSKeyGen}(\mathsf{SecPar})}(\mathsf{PubPar}, \mathcal{S}_{\mathsf{GS}}) \rightarrow (gsk_1, \ldots, gsk_{|\mathcal{S}_{\mathsf{GS}}|})$
    6. $\mathcal{A}^{\mathsf{AIARevoke}(\cdot, \mathsf{msk}_{\mathsf{ABE}}, \nu)}(\mathsf{mpk}_{\mathsf{ABE}}) \rightarrow (\mathsf{mpk}_{\mathsf{ABE}}{}')$ and $\mathcal{A}$ updates PubPar to PubPar$'$
    7. $\mathcal{A}^{\mathsf{CreateUpdateOrRevoke}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{PubPar}, \Upsilon_{\mathsf{info}}, m, m', \Upsilon_{\mathsf{admin}}, \mathsf{PubPar}') \rightarrow$
    $(msg^*, \xi_{msg^*}, msg^{*\prime}, \xi_{msg^{*\prime}})$
        where oracle CreateUpdateOrRevoke on input $(m, m', \mathsf{key}, \mathsf{PubPar}, \mathsf{PubPar}')$ does:
        /*$\Upsilon_{\mathsf{info}}$ and $\Upsilon_{\mathsf{admin}}$ given by $\mathcal{A}$ is used inside the CreateMessage */
        7.1. Parse $\Upsilon_{\mathsf{info}} = (\Upsilon_{\mathsf{ABE}}, \Upsilon_{\mathsf{GS}})$, if either $\Upsilon_{\mathsf{ABE}}(\mathcal{S}_{\mathsf{ABE}}) = 1$, $\Upsilon_{\mathsf{admin}}(\mathcal{S}_{\mathsf{ABE}}) = 1$ or $\Upsilon_{\mathsf{GS}}(\mathcal{S}_{\mathsf{GS}}) = 1$, return $\perp$
        7.2. Create two policy attributes, $id_1, id_2$ s.t., $\Upsilon_{\mathsf{ABE}}(id_1, id_2) = 1$ and $\Upsilon_{\mathsf{GS}}(id_1, id_2) = 1$
        7.3. Do $\mathsf{CreateMessage}(\mathsf{key}, \mathsf{PubPar}, m) \rightarrow (msg, \xi_{msg})$
        7.4. If $i = 0$, do $\mathsf{AdaptMessage}(\mathsf{key}, \mathsf{PubPar}, m', msg, \xi_{msg}) \rightarrow (msg', \xi_{msg'})$
            Set $t_0 = (\mathsf{PubPar}, m, msg, \xi_{msg}, m', msg', \xi_{msg'})$
        /* Next, we consider Case 1 of RevokeUser below, i.e., new $\Upsilon'_{\mathsf{info}}$ is created*/
        7.5. If $i = 1$, create $\Upsilon'_{\mathsf{info}} = (\Upsilon'_{\mathsf{ABE}}, \Upsilon_{\mathsf{GS}})$, s.t., $\Upsilon'_{\mathsf{ABE}}(id_1) = 1$
            Do $\mathsf{RevokeUser}(\mathsf{key}, \mathsf{PubPar}, m', msg, \xi_{msg}) \rightarrow (msg', \xi_{msg'})$
            Set $t_1 = (\mathsf{PubPar}, m, msg, \xi_{msg}, m', msg', \xi_{msg'})$
        /* Now we consider Case 2 of RevokeUser below, i..e, when AIA revokes a user's $id_2$ attribute*/
        7.6. If $i = 2$, Use PubPar$'$
            Do $\mathsf{RevokeUser}(\mathsf{key}, \mathsf{PubPar}', m', msg, \xi_{msg}) \rightarrow (msg', \xi'_{msg})$
            Set $t_2 = (\mathsf{PubPar}, m, msg, \xi_{msg}, m', msg', \xi_{msg'}, \mathsf{PubPar}')$
        7.7. $Q \leftarrow Q \cup \{msg^*.m^*, msg^*.\mathsf{digest}^*\}$
        7.8. return $t_i$
    8. Return 1 if $(((\mathsf{verify}(\mathsf{PubPar}, msg^*, \xi_{msg^*}) \rightarrow 1) \wedge (\mathsf{verify}(\mathsf{PubPar}, msg^{*\prime}, \xi_{msg^{*\prime}}) \rightarrow 1) \wedge$
        $(msg^{*\prime} \notin Q) \wedge (msg^* \neq msg^{*\prime})) \vee ((\mathsf{verify}(\mathsf{PubPar}, msg^*, \xi_{msg^*}) \rightarrow 1) \wedge$
        $(\mathsf{verify}(\mathsf{PubPar}', msg^{*\prime}, \xi_{msg^{*\prime}})) \rightarrow 1) \wedge (msg^*.m^*, msg^*.\mathsf{digest}^*) \notin Q \wedge$
        $(\cdot, msg^*.\mathsf{digest}^*) \in Q$, else return 0.
```

Figure 15: ReTRACe **private collision resistance game**

```
**Game** $\mathsf{RevocationCollRes}^{\mathcal{A}}_{\mathsf{ReTRACe}}(\lambda)$
    1. $\mathsf{Keygen}(1^{\lambda}) \rightarrow (\mathsf{SecPar}, \mathsf{PubPar})$
    2. $\mathcal{A}(pp_{\mathsf{RCHET}}) \rightarrow (sk^*_{ch}, pk^*_{ch})$
    3. $\mathcal{A}$ does $\mathsf{CreateMessage}(\mathsf{key}, \mathsf{PubPar}, m) \rightarrow (msg^*, \xi_{msg^*})$
    4. $\mathcal{A}^{\mathsf{ABEKeyGen}(\mathsf{SecPar})}(\mathsf{PubPar}, \mathcal{S}_{\mathsf{ABE}}) \rightarrow (sk_1, \ldots, sk_{|\mathcal{S}_{\mathsf{ABE}}|})$
    5. $\mathcal{A}^{\mathsf{GSKeyGen}(\mathsf{SecPar})}(\mathsf{PubPar}, \mathcal{S}_{\mathsf{GS}}) \rightarrow (gsk_1, \ldots, gsk_{|\mathcal{S}_{\mathsf{GS}}|})$
    6. $\mathcal{A}^{\mathsf{Revoke}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{key}, \mathsf{PubPar}, m, msg, \xi_{msg}) \rightarrow$
    $(msg^*, \xi_{msg^*}, msg^{*\prime\prime}, \xi_{msg^{*\prime\prime}})$
        where oracle Revoke on input $(\mathsf{key}, \mathsf{PubPar}, m^{*\prime}, msg^*, \xi_{msg^*})$ does:
        6.1. If $\mathsf{verify}(\mathsf{PubPar}, msg^*, \xi_{msg^*}) \rightarrow 0$, return $\perp$
        6.2. Call $\mathsf{RevokeUser}(\mathsf{key}, \mathsf{PubPar}, m^{*\prime}, msg^*, \xi_{msg^*}) \rightarrow (msg^{*\prime}, \xi_{msg^{*\prime}})$
        6.3. return $(m^{*\prime}, msg^{*\prime}, \xi_{msg^{*\prime}})$
    7. Return 1 if $(\mathsf{verify}(\mathsf{PubPar}, msg^*, \xi_{msg^*}) \rightarrow 1 \wedge \mathsf{verify}(\mathsf{PubPar}, msg^{*\prime\prime}, \xi_{msg^{*\prime\prime}}) \rightarrow 1)$. Else return 0
```

Figure 16: ReTRACe **revocation collision resistance game**