Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based

David Derler[1], Kai Samelin[2], Daniel Slamanig[3], and Christoph Striecks[3]

[1] DFINITY
david@dfinity.org
[2] TÜV Rheinland i-sec GmbH
kaispapers@gmail.com
[3] AIT Austrian Institute of Technology
{daniel.slamanig,christoph.striecks}@ait.ac.at

Abstract. Blockchain technologies recently received a considerable amount of attention. While the initial focus was mainly on the use of blockchains in the context of cryptocurrencies such as Bitcoin, application scenarios now go far beyond this. Most blockchains have the property that once some object, e.g., a block or a transaction, has been registered to be included into the blockchain, it is persisted and there are no means to modify it again. While this is an essential feature of most blockchain scenarios, it is still often desirable—at times it may be even legally required—to allow for breaking this immutability in a *controlled way.*

Only recently, Ateniese et al. (EuroS&P 2017) proposed an elegant solution to this problem *on the block level*. Thereby, the authors replace standard hash functions with so-called chameleon-hashes (Krawczyk and Rabin, NDSS 2000). While their work seems to offer a suitable solution to the problem of controlled re-writing of blockchains, their approach is too coarse-grained in that it *only* offers an all-or-nothing solution. We revisit this idea and introduce the novel concept of policy-based chameleon-hashes (PCH). PCHs generalize the notion of chameleon-hashes by giving the party computing a hash the ability to associate access policies to the generated hashes. Anyone who possesses enough privileges to satisfy the policy can then find arbitrary collisions for a given hash. We then apply this concept to *transaction-level* rewriting within blockchains, and thus support *fine-grained and controlled* modifiability of blockchain objects. Besides modeling PCHs, we present a generic construction of PCHs (using a strengthened version of chameleon-hashes with ephemeral trapdoors which we also introduce), rigorously prove its security, and instantiate it with efficient building blocks. We report first implementation results.

Author list in alphabetical order. See https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This is the full version of a paper which appears in NDSS 2019, 26th Network and Distributed System Security Symposium, San Diego, USA, CA, Feb 24 - Feb 27, 2019. The Internet Society 2019. The proceedings version is available online at https://dx.doi.org/10.14722/ndss.2019.23066

1 Introduction

Blockchains technologies have attracted a tremendous amount of attention. This increase in interest was mainly triggered by the first large-scale application of blockchains, i.e., the decentralized cryptocurrency Bitcoin. Meanwhile applications go far beyond their use in cryptocurrencies. Examples include application domains such as supply chains, digital twins, insurance, healthcare, or energy.[4] In a nutshell, a blockchain is a decentralized, distributed, potentially public, and immutable log of objects such as transactions. It is created by establishing consensus between the chain's participants and can be thought of as a hash-chain which links blocks together. That is, each block includes the hash of the previous block as a reference to link them. Each block typically also includes some other information and a set of valid transactions, which, in turn, are usually accumulated into a single hash value by means of a Merkle tree [Mer89]. A transaction can be a monetary transaction (as in cryptocurrencies) or include any other object of interest which needs to be recorded, e.g., data related to smart contracts.

Blockchains can be of different types. They can be *public* as for example used within Bitcoin or Ethereum, where the consensus protocol is executed between many pseudonymous participants. Here, the blockchain can be read and written by everyone. In such blockchains, the consensus finding is typically either implemented via proofs of work (PoW), or proofs of stake (PoS) combined with some alternative, less resource intensive consensus finding algorithm (e.g., byzantine fault tolerance algorithms [LSP82]). Such public blockchains can also be viewed as *permissionless*, because everyone can join the system, can participate in the consensus protocol, and can also establish smart contracts. Blockchains, however, can also be *private* (also called enterprise or permissioned blockchains) like Hyperlegder, Ethereum Enterprise, Ripple, or Quorum. Here, all the participants and their (digital) identities are known to one or more trusted organizations. Actors have (policy-based) write and read permissions, and reading and writing usually requires consensus of several participants. Such private blockchains can thus be viewed as *permissioned*, because they restrict the actors who can contribute to the consensus on the system state to validate the block transactions. Hyperledger [ABB+18] for instance uses so-called endorsement policies in the form of monotone Boolean formulas, e.g., $(A \ \mathsf{AND}\ D)\ \mathsf{OR}\ C$ to determine which peers are

[4] http://www.businessinsider.de/blockchain-technology-applications-use-cases-2017-9?r=US&IR=T

required to endorse a transaction. Moreover, among others, they allow also to restrict access to approved actors who can create smart contracts.

Problem and Motivation. One particular issue that is of interest in this work is that once some object has been registered in the blockchain (be it private or public), the object is persisted as is and there is no means to alter it ever again. While this is one of the crucial properties of blockchains, it is often desirable—at times even legally required—to introduce features which allow to "break" the immutability of objects in the blockchain, preferably *in a fine-grained and controlled way*. While this might sound dangerous at first glance, there are surprisingly many scenarios which do benefit or even crucially require such a functionality.

With the increasing number of application of blockchains already listed, which may include sensitive information into blockchains, features to redact or correct objects in blockchains might evolve to an important requirement and may often be even legally obliged. For example, the upcoming general data protection regulation (GDPR)[5] of the European Union imposes the Right to be Forgotten as a key Data Subject Right. In light of this regulation, it is no longer legally possible to use immutable blockchains in processes where personal data are recorded within blockchains. Also various other legal regulations such as the United States Fair Credit Reporting Act, the Gramm-Leach-Bliley Act, and the Securities and Exchange Commission's Regulation S-P are relevant here.[6]

Mitigation Strategies. To mitigate this problem, there are different strategies that can be used. Central to the problem are thereby the questions of (1) who is allowed to perform these modifications and (2) what data can be modified. Thereby it seems desirable that (1) the person who introduces an object into the blockchain should be able to determine who will be able to modify the object if required and (2) only the object, i.e., the transaction, can be modified, while the blockchain (i.e., the chaining of the blocks) does not need to be touched. As we will discuss below, some straight-forward solutions do not satisfactorily address these issues.

A first strategy is to simply *create new objects*, i.e., a new version of a transaction or smart contract, to be integrated into the next block of the blockchain. This new object points to the old one and invalidates it. However, this keeps a history of all modification which is not always desired. Moreover, this may also infringe laws, e.g., EU privacy laws, and in particular the Right to be Forgotten, when the objects include sensitive

[5] https://www.eugdpr.org

[6] https://www.nytimes.com/2016/09/10/business/dealbook/downside-of-virtual-currencies-a-ledger-that-cant-be-corrected.html

and/or person related data, as in this case, the content does effectively not disappear from the blockchain.

Another strategy could be to simply perform a *hard-fork* whenever transactions in some block require to be edited or fixed, and to develop the new blockchain from there. Apart from being not oblivious to the users, i.e., it requires every user to download new client software which accepts the new chain, this is a significant intervention in the blockchain ecosystem on every correction. In particular, one needs to invalidate all confirmed later blocks including the modified one. The impractically of the method, for example, becomes apparent when thinking of a block from years ago, which needs to be removed due to data protection reasons.

Another solution is to *rewind and replay* the blockchain to the point where the modification needs to take place and to recompute everything from this point including a new consensus finding for all already computed blocks. For the same reasons as discussed above, this is highly inefficient, does not scale, and—likewise to the hard-fork strategy—is not oblivious to the users.

Arguably all the above strategies, besides their inefficiency, neither allow to control *who* will be able to modify nor *what* can be modified. A more desirable strategy is one which is controlled by the users in a fine-grained way, oblivious to the other users, highly efficient, and only requires changes that are local to the point where a transaction needs to be edited. A solution providing those properties may seem too good to be true, especially because the hash function involved in the block computation (or transaction aggregation) prevents any modification, i.e., any alteration of a transaction will change the hash value and break the link to the following block.

Existing Solution. Recently, Ateniese et al. [AMVA17] came up with a clever idea and showed that the problem of *rewriting entire blocks in a blockchain* can be efficiently solved by means of chameleon-hash functions [KR00]. A chameleon-hash (CH) is a hash function, where hashing is parametrized by a public key pk. It behaves like a collision resistant hash function as long as the trapdoor (the secret key sk corresponding to pk) is not known. Conversely, if the trapdoor sk is known, arbitrary collisions can be found. Using such hash functions as a replacement for collision resistant ones in blockchains allows to introduce some entity that possesses the trapdoor. By computing collisions in the hash function, this entity can efficiently edit the blockchain. This solution has recently seen practical adoption by Accenture.[7]

[7] https://www.accenture.com/us-en/service-blockchain-financial-services

Although very elegant, the approach by Ateniese et al. is rather limited. Firstly, it considers rewriting of a blockchain on the *block level*, i.e., to replace the hash of an entire block, which seems to be far too coarse-grained and powerful and rewriting on a transaction level seems more reasonable. Secondly, it can only be decided in a coarse-grained way who can compute collisions. This is because one always hashes with respect to a single fixed public key. Consequently, a single fixed secret key is useful to find collisions. Furthermore, the party who computes the hash is totally oblivious about who is later able to compute collisions in the chameleon-hash. This means that the party who computes the hash does not know who is allowed to rewrite the blockchain (apart from the entity behind pk). However, when an object should be included into the blockchain, the party performing this operation should be able to specify who is able to perform editing on this object in a fine-grained way. For example, for every transaction, one should be able to separately specify the identities of the user or roles of users within an organization (e.g., a data protection officer or a member of the board), which is required to later update/correct the respective object.

Our Envisioned Improved Solution. Our starting point is attribute-based access control (ABAC) [HFK+14], where users are tagged by (ad-hoc) attributes and there are policies that express (potentially complex) Boolean formulas over attributes. On a very high level, access decisions are made by evaluating the respective access policies on the set of attributes associated to a user. For instance, assume that a user has associated attributes $\{A, B, C\}$ out of the attribute set $\{A, B, C, D\}$ and access to a resource, protected by a policy $(A \mathsf{\ AND\ } D) \mathsf{\ OR\ } C$, then access for this user would be granted. Note that attributes can also directly describe users' identities $\{\text{user}_1, \text{user}_2, \text{user}_3, \dots\}$ and restricting the Boolean formulas to OR's allows for specifying the set of authorized users, e.g., $\text{user}_1 \mathsf{\ OR\ } \text{user}_2 \mathsf{\ OR\ } \text{user}_3$.

In addition, we consider a decentralized setting, where, in general, every entity can play the role of an attribute authority and tag other users with attributes (in our realization this will amount to issuing keys for corresponding attributes to those users). Then, when some user decides that some object should be integrated into the blockchain, the user can "tag" the object with an access policy corresponding to an attribute authority (managed by some other user) of it's choice (in our realization this amounts to hashing the object with a novel type of chameleon-hash). If at some later point in time the objects needs to be modified, every user that satisfies the associated policy can perform the update (in our realization this amounts to computing a collision in the novel type of chameleon-hash). An important property that we want to achieve thereby is that original and modified objects cannot be told apart (are indistinguishable) and that even if user keys associated to attributes leak, no information about the history of an object can be reconstructed (e.g., its previous state).

1.1 Contribution

In this work, we introduce a cryptographic solution to the scenario outlined above, i.e., the scenario of rewriting objects in blockchains in a flexible, controlled, and fine-grained way. To achieve this goal, we introduce the notion of policy-based chameleon-hashes (PCHs), which generalizes chameleon-hashes in the sense that hashing additionally takes an access policy as input and collision finding is much more fine-grained than in existing chameleon-hashing, i.e., a collision can only be found by users satisfying the policy specified during hashing. In particular, when computing a hash, an access policy can be included so that only entities that possess secret keys corresponding to attributes satisfying the access policy can find collisions. We rigorously model the security one would expect from such a primitive.

A cryptographic primitive that allows for elegantly modeling the access-control requirements in such a setting is ciphertext-policy attribute-based encryption (CP-ABE) which was first envisioned by Goyal, Pandey, Sahai, and Waters [GPSW06] and later efficiently instantiated by Bethencourt, Sahai, and Waters [BSW07]. Here, one specifies access policies over attributes upon computing ciphertexts, and secret keys are associated to attributes. Only someone who possesses a secret key whose corresponding attributes satisfy the access policy is able to decrypt. The important feature thereby is that the encrypting party does not even need to know the entities who will later be able to decrypt, but only needs to specify an access policy.

However, it turns out to be *non-trivial* to achieve a similar functionality in the context of chameleon-hashes, e.g., by extending the approach of Ateniese et al. [AMVA17]. The main technical hurdle, when going for a naive integration of the functionality of CP-ABE into chameleon-hashes, is that the party who computes a hash somehow needs to encrypt a trapdoor which will later be useful to compute collisions. Now, in conventional chameleon-hashes, the trapdoor, which enables computing collisions, is essentially the secret key corresponding to the public hashing key being fixed in the system parameters. This trapdoor remains the same for all hashes computed with respect to one public hashing key. Consequently, after computing one collision, one could compute a collision for any other chameleon-hash. Moreover, with such a naive solution, the hashing party (although it might not be authorized at all) could then compute arbitrary collisions. We, however, strive for a solution which allows us to have a separate trapdoor per hash, so that we are able to implement fine-grained access control. Conventional chameleon-hashes provide no security guarantees in such a setting.

To this end, we pursue a different path and carefully integrate the CP-ABE functionality with the recent concept of chameleon-hashes with ephemeral trapdoors (CHETs) [CDK+17] and present a generic construction of PCHs. For access policies representing the class of monotone Boolean formulas (which is well suited for access control), we can additionally take advantage of recent progress in very efficient CP-ABE schemes due to Agrawal and Chase [AC17]. Along the lines, we also introduce a novel CHET which is more efficient than the most practical known instantiation proposed by Camenisch et al. [CDK+17]. Putting all together, we obtain a very efficient concrete instantiation of a PCH. We support this claim with first implementation results of our primitive.

We discuss the application of PCHs for transaction-level rewriting of blockchains (cf. Section 5). Another application that comes to mind is the usage of PCHs instead of conventional chameleon-hashes in sanitizable signatures [ACdT05] to achieve more expressive delegations of editing rights. We leave a concrete and formal treatment open for future work. Moreover, we believe that PCHs will find many other applications.

1.2 Related Work

We already briefly discussed the work due to Ateniese et al. [AMVA17] which inspired our work. In another work Puddu et al. [PDC17] present mutable transactions for blockchains, where in their system all transactions are encrypted and mutation means that the respective decryption key is not provided anymore by validators. Mutations are subject to access control policies, but all the mechanisms are not cryptographic in nature.

Ferrara et al. [FFW13] discuss cryptographically enforced role-based access control (cRBAC) with the aim of introducing a precise syntax for a computational version of RBAC as well as rigorous definitions for cryptographic policy enforcement of a large class of RBAC security policies. They also show that an implementation of RBAC based on key-policy and ciphertext-policy attribute-based encryption (KP-ABE and CP-ABE, respectively) meets their security notions. Although their work has a totally different focus than ours, it shows that the use of attribute-based encryption is a good choice in realizing secure access control that meets real-world needs.

Damgård et al. [DHO16] introduced a primitive denoted as Access Control Encryption (ACE), which was later extended in [BMM17,KW17]. It allows a central party (called the sanitizer) to control for a set of parties which party is allowed to receive and send which message to other parties. This sanitizer processes all the messages and thereby enforces access-control policies. Although related (in that access-control policies need to be enforced), this primitive is not helpful in our setting.

Concurrent and Independent Work. In a concurrent and independent work Deuber et al. [DMT19] also propose a novel mechanism for editable (redactable) blockchains. Their goal is to avoid advanced cryptographic schemes and instead an edit operation in their setting can be proposed by any user and there is then a voting in the blockchain through a consensus, i.e., edits are only performed if approved by the blockchain policy (e.g., voted by the majority). If a proposed edit obtains enough votes, then the respective block is replaced by its new version in the blockchain (and the old state of the block needs to be kept available for validation - but the redacted data does not need to be kept for verification). Like Ateniese et al. [AMVA17], Deuber et al. [DMT19] exclusively focus on rewriting complete *blocks*, whereas our approach is more flexible as it also enables rewriting on the *transaction* level.

1.3 Preliminaries and Notation

We use κ to denote the security parameter and we use sans-serif letters, e.g., A, B, to denote algorithms. If not stated otherwise, all algorithms are required to run in probabilistic polynomial time (PPT), i.e., their running time can be bounded by a polynomial in their input length. Furthermore, all algorithms return a special symbol $\perp$ on error. By $y \leftarrow \mathsf{A}(1^\kappa, x)$, we denote that y is assigned the output of the potentially probabilistic algorithm A on input x and and fresh random coins. We assume 1^κ to be an implicit input to all algorithms. If $\forall c \exists \kappa_0 \forall \kappa \geq \kappa_0 : |f(\kappa)| \leq 1/\kappa^c$ for a function f, then we say f is negligible. For algorithms representing adversaries in the security experiments we use calligraphic letters, e.g., $\mathcal{A}$. Furthermore, we assume that all oracles defined within security experiments return $\perp$, as soon as any of the internally executed algorithms returns $\perp$. This allows for a more compact notation. Finally, similar to our notation in the context of algorithms, we use $y \leftarrow_r S$ to denote that an element is sampled uniformly at random from a finite set S and assigned to y.

2 Cryptographic Building Blocks

In this section, we provide some background including collision resistant hashing, ciphertext-policy attribute-based encryption (CP-ABE), introduce the notion of access structures that are associated to ciphertexts in CP-ABE formally and discuss how to encode such access structures. Then, we recall (and strengthen) chameleon-hashes with ephemeral trapdoors, which we require as an ingredient to our main construction.

Definition 1 (Access Structure). *Let $\mathbb{U}$ denote the universe of attributes. A collection $\mathbb{A} \in 2^{\mathbb{U}} \setminus \{\emptyset\}$ of non-empty sets is an access structure on $\mathbb{U}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets. A collection $\mathbb{A} \in 2^{\mathbb{U}} \setminus \{\emptyset\}$ is called monotone if $\forall\ B, C \in \mathbb{A}$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$.*

Attribute-Based Encryption. Let us recall the description of a CP-ABE scheme.

Definition 2 (CP-ABE). *A ciphertext-policy attribute-based encryption (CP-ABE) scheme is a tuple $(\mathsf{Setup}_{\mathsf{ABE}}, \mathsf{KGen}_{\mathsf{ABE}}, \mathsf{Enc}_{\mathsf{ABE}}, \mathsf{Dec}_{\mathsf{ABE}})$ of PPT algorithms defined as follows:*

$\mathsf{Setup}_{\mathsf{ABE}}(1^\kappa)$: *Takes as input a security parameter κ in unary and outputs a master secret and public key $(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{mpk}_{\mathsf{ABE}})$. We assume that all subsequent algorithms will implicitly receive the master public key $\mathsf{mpk}_{\mathsf{ABE}}$ as input which implicitly fixes a message and attribute space $\mathcal{M}$ and $\mathbb{U}$, respectively.*

$\mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$: *Takes as input the master secret key $\mathsf{msk}_{\mathsf{ABE}}$ and a set of attributes $\mathbb{S} \subseteq \mathbb{U}$ and outputs a secret key $\mathsf{sk}_{\mathbb{S}}$.*

$\mathsf{Enc}_{\mathsf{ABE}}(M, \mathbb{A})$: *Takes as input a message $M \in \mathcal{M}$ and an access structure $\mathbb{A}$ and outputs a ciphertext C.*

$\mathsf{Dec}_{\mathsf{ABE}}(\mathsf{sk}_{\mathbb{S}}, C)$: *Takes as input a secret key $\mathsf{sk}_{\mathbb{S}}$ and a ciphertext C and outputs a message M or $\perp$ in case decryption does not work.*

Correctness of CP-ABE requires that for all κ, for all access structures $\mathbb{A}$, all $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(1^\kappa)$, all $M \in \mathcal{M}$, all $\mathbb{S} \in \mathbb{U}$, all $\mathsf{sk}_{\mathbb{S}} \leftarrow \mathsf{KGen}(\mathsf{msk}, \mathbb{S})$ we have that $\mathsf{Pr}[\mathsf{Dec}(\mathsf{sk}_{\mathbb{S}}, \mathsf{Enc}(M, \mathbb{A})) = M] = 1$.

```
$\mathbf{Exp}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},\mathsf{ABE}}(\kappa)$:
  $(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{mpk}_{\mathsf{ABE}}) \leftarrow \mathsf{Setup}_{\mathsf{ABE}}(1^\kappa)$
  $b \leftarrow \{0,1\}$
  $\mathcal{Q}, \mathcal{S} \leftarrow \emptyset$, $i \leftarrow 0$
  $(m_0, m_1, \mathbb{A}^*, \mathtt{state}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{mpk}_{\mathsf{ABE}})$
    where $\mathcal{O} \leftarrow \{\mathsf{KGen}'_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \cdot), \mathsf{KGen}''_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \cdot),$
      $\mathsf{Dec}'_{\mathsf{ABE}}(\cdot,\cdot)\}$
    and $\mathsf{KGen}'_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \cdot)$ on input $\mathbb{S}$:
      return $\mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$ and set $\mathcal{S} \leftarrow \mathcal{S} \cup \mathbb{S}$
    and $\mathsf{KGen}''_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \cdot)$ on input $\mathbb{S}$:
      $\mathsf{ssk} \leftarrow \mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$ and set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i, \mathsf{ssk})\}$
      $i \leftarrow i + 1$
    and $\mathsf{Dec}'_{\mathsf{ABE}}(\cdot,\cdot)$ on input $j$, $c$:
      return $\bot$, if $(j, \mathsf{ssk}) \notin \mathcal{Q}$ for some $\mathsf{ssk}$
      return $\mathsf{Dec}_{\mathsf{ABE}}(\mathsf{ssk}, c)$
  if $m_0, m_1 \notin \mathcal{M} \;\vee\; |m_0| \neq |m_1| \;\vee \mathbb{A}^* \cap \mathcal{S} \neq \emptyset$, let $c^* \leftarrow \bot$
    else $c^* \leftarrow \mathsf{Enc}_{\mathsf{ABE}}(m_b, \mathbb{A}^*)$
  $b^* \leftarrow \mathcal{A}^{\mathsf{KGen}'''_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}},\cdot), \mathsf{KGen}''''_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}},\cdot), \mathsf{Dec}''_{\mathsf{ABE}}(\cdot,\cdot)}(c^*, \mathtt{state})$
    where $\mathsf{KGen}'''_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \cdot)$ on input $\mathbb{S}$:
      return $\bot$, if $\mathbb{S} \in \mathbb{A}^*$
      return $\mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$
    and $\mathsf{KGen}''''_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \cdot)$ on input $\mathbb{S}$:
      let $\mathsf{ssk} \leftarrow \mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$ and set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i, \mathsf{ssk})\}$
      $i \leftarrow i + 1$
    and $\mathsf{Dec}''_{\mathsf{ABE}}(\cdot,\cdot)$ on input $j$, $c$:
      return $\bot$, if $(j, \mathsf{ssk}) \notin \mathcal{Q}$ for some $\mathsf{ssk} \;\vee\; c = c^*$
      return $\mathsf{Dec}_{\mathsf{ABE}}(\mathsf{ssk}, c)$
  if $b^* = b$ return 1, else return 0
```

Fig. 1: ABE IND-CCA2 Security

Security of CP-ABE. In the following, we recall IND-CCA2 security for CP-ABE (where we explicitly model key handles, cf. [KW18]).

Definition 3 **(IND-CCA2-Security of CP-ABE).** *Let the advantage of an adversary $\mathcal{A}$ in the* IND-CCA2 *experiment* $\mathbf{Exp}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},\mathsf{ABE}}(\kappa)$ *be:*

$$\mathbf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},\mathsf{ABE}}(\kappa) := \left|\Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},\mathsf{ABE}}(\kappa) = 1\right] - 1/2\right|.$$

We call a CP-ABE scheme ABE *is* IND-CCA2 *secure if* $\mathbf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},\mathsf{ABE}}(\kappa)$ *is a negligible function in κ for all PPT adversaries $\mathcal{A}$.*

Monotone Span Programs. Monotone span programs (MSP) [KW93] (or, essentially linear secret-sharing schemes (LSSS) [Bei96]) consist of an integer matrix $\mathbf{M}$ which encodes monotone access structures. Monotone access structures are often represented as Boolean formulas over attributes with AND and OR operators and input attributes satisfy the formula if it evaluates to 1. Another way of representing such formulas is to think of access trees. In such a tree, the leafs form the input attributes while inner nodes are associated with the operators AND and OR. In the full version [LW10, Appendix G] of [LW11], Lewko and Waters describe an easy way to transform Boolean formulas with AND and OR operators into MSP (or, LSSS) matrices $\mathbf{M}$ (this transform is also used in the ABE scheme FAME by Agrawal and Chase [AC17] that we chose as a core building block for our instantiation in Section 4.4). Essentially, the encoding is as follows. The sharing vector of the LSSS matrix is $(1, 0, \ldots, 0)$ and the root of the tree is labeled with vector (1). Further, let l be a counter variable set to 0 in the beginning. We now go recursively down the tree levels. If the node is an AND operator, label one of the two children with $(v||1)$, where v is the label of the AND node padded with 0s to length l and $||$ denotes the concatenation. The other children is labeled with length-$(l+1)$ vector $(0, \ldots, 0, -1)$ and l is increased afterwards. If the node is an OR operator, label both children with v and do not increase l. Next, go on to the next node. Once labeling is completed, we now collect the labels of the leaf (i.e., attribute) nodes one-by-one to form the rows of the matrix $\mathbf{M}$. If there are empty matrix entries, fill them with 0s. The rows of $\mathbf{M}$ are associated with a function π that maps the row number to the corresponding attribute. The output of the encoding is $(\mathbf{M}, \pi)$. Note that we assume a canonical encoding of the access structure $\mathbb{A}$ and the tree s.t. computing $\mathbf{M}$ from a given $\mathbb{A}$ is deterministic. Decoding $(\mathbf{M}, \pi)$ works as follows. Let $\mathbb{S}$ be a set of attributes. For $\pi(i) \in \mathbb{S}$ there exist coefficients $\lambda_i \in \{0, 1, -1\}$ such that $\sum_{\pi(i)\in\mathbb{S}} \lambda_i(\mathbf{M})_i = (1, 0, \ldots, 0)$, where $(\mathbf{M})_i$ is the i-th row of $\mathbf{M}$. The output of decoding is the list $(\lambda_i)_{\pi(i)\in\mathbb{S}}$.

Chameleon-Hashes. Subsequently, we recall chameleon-hashes using the notion from Camenisch et al. [CDK+17].

Definition 4 (Chameleon-Hashes). *A chameleon-hash* CH *with message space* $\mathcal{M}$ *is a tuple* $(\mathsf{PPGen}_{\mathsf{CH}}, \mathsf{KGen}_{\mathsf{CH}}, \mathsf{Hash}_{\mathsf{CH}}, \mathsf{Verify}_{\mathsf{CH}}, \mathsf{Adapt}_{\mathsf{CH}})$ *of potentially probabilistic polynomial time algorithms, which are defined as follows:*

$\mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$**.** *The algorithm* $\mathsf{PPGen}_{\mathsf{CH}}$*, on input security parameter* κ *in unary, outputs public parameters* $\mathsf{pp}_{\mathsf{ch}}$*. For brevity, we assume that* $\mathsf{pp}_{\mathsf{ch}}$ *is an implicit input to all other algorithms.*

$\mathsf{KGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{ch}})$**.** *The algorithm* $\mathsf{KGen}_{\mathsf{CH}}$*, given the public parameters* $\mathsf{pp}_{\mathsf{ch}}$*, outputs the secret and public key* $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}})$*.*

$\mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m)$**.** *The algorithm* $\mathsf{Hash}_{\mathsf{CH}}$ *gets as input the public key* $\mathsf{pk}_{\mathsf{CH}}$ *and a message* $m \in \mathcal{M}$*, and outputs a hash* h *and randomness* r*.*

$\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, r, h)$**.** *The deterministic algorithm* $\mathsf{Verify}_{\mathsf{CH}}$ *gets as input the public key* $\mathsf{pk}_{\mathsf{CH}}$*, a message* m*, randomness* r*, and hash* h*. It outputs a decision* $d \in \{0, 1\}$ *indicating whether the hash* h *is valid.*

$\mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r, h)$**.** *The algorithm* $\mathsf{Adapt}_{\mathsf{CH}}$*, on input of the secret key* $\mathsf{sk}_{\mathsf{CH}}$*, message* m*, randomness* r*, hash* h*, and a additional message* m'*, outputs randomness* r'*.*

Note that we assume that the $\mathsf{Adapt}_{\mathsf{CH}}$ algorithm always verifies if the hash it is given is valid, and outputs $\perp$ otherwise.

Correctness. For a CH we require the correctness property to hold. In particular, we require that for all $\kappa \in \mathbb{N}$, for all $\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$, for all $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \leftarrow \mathsf{KGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{ch}})$, for all $m \in \mathcal{M}$, for all $(h, r) \leftarrow \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m)$, for all $m' \in \mathcal{M}$, we have for all for all $r' \leftarrow \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r, h)$, that $1 = \mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, r, h) = \mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m', r', h)$. This definition captures perfect correctness.

Note that the randomness is drawn by $\mathsf{Hash}_{\mathsf{CH}}$, and not outside. The intention is to capture "private-coin" constructions [AMVA17]. We provide the security notions in Appendix B.1 and stress that we rely on the notions from [CDK+17] with the exception that we provide a stronger form of indistinguishability, where the adversary is even allowed to know the secret key.

CHs with Ephemeral Trapdoors. We recall the notion of chameleon-hashes with ephemeral trapdoors (CHET) from [CDK+17]. This primitive is a variant of a chameleon-hash where, in addition to the long-term trapdoor, another ephemeral trapdoor (chosen during hashing) is required to compute collisions.

Definition 5 (Chameleon-Hashes with Ephemeral Trapdoors). *A chameleon-hash with ephemeral trapdoors* CHET *for message space* $\mathcal{M}$ *is a tuple of five algorithms* $(\mathsf{PPGen}_{\mathsf{CHET}}, \mathsf{KGen}_{\mathsf{CHET}}, \mathsf{Hash}_{\mathsf{CHET}}, \mathsf{Verify}_{\mathsf{CHET}}, \mathsf{Adapt}_{\mathsf{CHET}})$*, such that:*

$\mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)$: *On input security parameter* κ *in unary, this algorithm outputs the public parameters* pp*. We assume that they implicitly define the message space* $\mathcal{M}$*.*

$\mathsf{KGen}_{\mathsf{CHET}}(\mathsf{PP})$: *On input the public parameters* PP*, this algorithm outputs the long-term key pair* $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}})$.

$\mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m)$: *On input the public key* $\mathsf{pk}_{\mathsf{CHET}}$ *and a message* m*, this algorithm outputs a hash* h*, corresponding randomness* r*, as well as the ephemeral trapdoor* etd.

$\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, h, r)$: *On input the public key* $\mathsf{pk}_{\mathsf{CHET}}$*, a message* m*, a hash* h*, and randomness* r*, this algorithm outputs a bit* b.

$\mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{etd}, m, m', h, r)$: *On input secret key* $\mathsf{sk}_{\mathsf{CHET}}$*, ephemeral trapdoor* etd*, a message* m*, a message* m'*, hash* h*, randomness* r*, and trapdoor information* etd*, this algorithm outputs randomness* r'.

Note that we assume that the $\mathsf{Adapt}_{\mathsf{CHET}}$ algorithm always verifies if the hash it is given is valid, and outputs $\perp$ otherwise.

Correctness. For correctness, we require that for all $\kappa \in \mathbb{N}$, all $\mathsf{PP} \leftarrow \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$, all $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \leftarrow \mathsf{KGen}_{\mathsf{CH}}(\mathsf{PP})$, all $m, m' \in \mathcal{M}$, all $(h, r, \mathsf{etd}) \leftarrow \mathsf{Hash}(\mathsf{pk}_{\mathsf{CHET}}, m)$, all $r' \leftarrow \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{etd}, m, m', h, r)$, we have that $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, h, r) = 1 \;\wedge\; \mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m', h, r') = 1$.

For security, chameleon-hashes with ephemeral trapdoors are required to be indistinguishable, publicly collision resistant, and privately collision resistant. We postpone the formal definitions of these properties to Appendix B.3. In a nutshell, indistinguishability requires that an adversary cannot decide whether randomness was created through hashing or adaption. Public collision resistance requires that an outsider cannot find any collisions by itself, while private collision resistance enforces that even the holder of the long-term trapdoor cannot find collisions, if the ephemeral secret key is not known.[8]

3 Policy-Based Chameleon-Hashing

In this section, we introduce and define a novel primitive which we term policy-based chameleon-hash. In Section 3.1, we formally define policy-based chameleon-hashes. In Section 3.2, we show how to generically construct policy-based chameleon-hashes from a combination of the relatively recent concept of chameleon-hashes with ephemeral trapdoors (CHET) and ciphertext-policy attribute-based encryption (CP-ABE) schemes and rigorously prove the security of this generic construction. Later in Section 4.4, after having discussed the selection of the underlying primitives,

[8] Actually, we require some stronger definitions, which we also introduce.

we present a practically efficient instantiation of our generic construction for the class of policies represented by monotone access structures. The efficiency of this concrete instantiation is supported by first implementation results in Section 4.5.

3.1 Definitions

We define policy-based chameleon-hashes and their security.

Definition 6 (Policy-Based Chameleon-Hashes). *A policy-based chameleon-hash* PCH *with message space* $\mathcal{M}$ *consists of five algorithms* $(\mathsf{PPGen}_{\mathsf{PCH}}, \mathsf{KGen}_{\mathsf{PCH}}, \mathsf{Hash}_{\mathsf{PCH}}, \mathsf{Verify}_{\mathsf{PCH}}, \mathsf{Adapt}_{\mathsf{PCH}})$ *which are defined as follows.*

$\mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)$: *On input a security parameter* κ *in unary, this algorithm outputs the secret and public key* $(\mathsf{sk}_{\mathsf{PCH}}, \mathsf{pk}_{\mathsf{PCH}})$ *where* $\mathsf{pk}_{\mathsf{PCH}}$ *is implicitly available to all algorithms and determines* $\mathcal{M}$*.*

$\mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \mathbb{S})$: *On input a secret key* $\mathsf{sk}_{\mathsf{PCH}}$ *and a set of attributes* $\mathbb{S} \subseteq \mathbb{U}$*, key generation outputs a secret key* $\mathsf{sk}_{\mathbb{S}}$*.*

$\mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$**:** *On input a public key* $\mathsf{pk}_{\mathsf{PCH}}$*, access structure* $\mathbb{A} \subseteq 2^{\mathbb{U}}$*, and a message* $m \in \mathcal{M}$*, the hash algorithm outputs a hash* h *and randomness* r*.*

$\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}, m, h, r)$**:** *On input public key* $\mathsf{pk}_{\mathsf{PCH}}$*, message* m*, hash* h*, and randomness* r*, the verification outputs a bit* b*.*

$\mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{sk}_{\mathbb{S}}, m, m', h, r)$**:** *On input a secret key* $\mathsf{sk}_{\mathbb{S}}$*, messages* m *and* m'*, hash* h*, and randomness* r*, the adaptation algorithm outputs randomness* r'*.*

Note that we assume that the $\mathsf{KGen}_{\mathsf{PCH}}$ outputs $\perp$ if $\mathbb{S}$ is not contained in $\mathbb{U}$ and the $\mathsf{Adapt}_{\mathsf{PCH}}$ algorithm always verifies if the hash it is given is valid, and output $\perp$ otherwise.

Correctness. For correctness, we require that for all $\kappa \in \mathbb{N}$, for all $\mathbb{A} \subseteq 2^{\mathbb{U}}$, for all $\mathbb{S} \in \mathbb{A}$, for all $(\mathsf{sk}_{\mathsf{PCH}}, \mathsf{pk}_{\mathsf{PCH}}) \leftarrow \mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)$, for all $\mathsf{sk}_{\mathbb{S}} \leftarrow \mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \mathbb{S})$, for all $m \in \mathcal{M}$, for all $(h, r) \leftarrow \mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$, for all $m' \in \mathcal{M}$, for all $r' \leftarrow \mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{sk}_{\mathbb{S}}, m, m', h, r)$, we have that $1 = \mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, h, r) = \mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m', h, r')$.

Furthermore, we require the following security properties.

Indistinguishability. Informally, indistinguishability requires that it be intractable to decide whether for a chameleon-hash its randomness is fresh or was created using the adaption algorithm even if the secret key is known. While such a property was not required in the work by Ateniese et al. [AMVA17], we believe that it could be useful in the blockchain context, because it helps to prevent outsiders (which later become insiders) from learning whether adaptations of certain objects, e.g., transactions, in the blockchain have taken place, when seeing the respective hashes and randomness. The security experiment grants the adversary access to the secret key and a left-or-right style HashOrAdapt oracle. It requires that the randomnesses r does not reveal whether it was obtained through $\mathsf{Hash}_{\mathsf{PCH}}$ or $\mathsf{Adapt}_{\mathsf{PCH}}$. The messages are adaptively chosen by the adversary.

```
$\mathbf{Exp}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$
    $(\mathsf{sk}_{\mathsf{PCH}}, \mathsf{pk}_{\mathsf{PCH}}) \leftarrow \mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)$
    $b \leftarrow \{0,1\}$
    $b^* \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt}_{\mathsf{PCH}}(\mathsf{sk},\cdot,\cdot,\cdot,\cdot,b)}(\mathsf{sk}_{\mathsf{PCH}}, \mathsf{pk}_{\mathsf{PCH}})$
        where $\mathsf{HashOrAdapt}_{\mathsf{PCH}}(\mathsf{sk},\cdot,\cdot,\cdot,\cdot,b)$ on input $m, m', \mathbb{S}, \mathbb{A}$:
            $(h_0, r_0) \leftarrow \mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m', \mathbb{A})$
            $(h_1, r_1) \leftarrow \mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$
            $\mathsf{sk}_{\mathbb{S}} \leftarrow \mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}, \mathbb{S})$
            $r_1 \leftarrow \mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{sk}_{\mathbb{S}}, m, m', h_1, r_1)$
            return $(h_b, r_b)$
    return 1, if $b = b^*$
    return 0
```

Fig. 2: PCH Indistinguishability

Definition 7 (Indistinguishability). *We define the advantage of an adversary $\mathcal{A}$ in the* Ind *experiment $\mathbf{Exp}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$ as*

$$\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{PCH}}(\kappa) := \left|\Pr\left[\mathbf{Exp}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{PCH}}(\kappa) = 1\right] - 1/2\right|.$$

We say a PCH *scheme is indistinguishable, if $\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$ is a negligible function in κ for all PPT adversaries $\mathcal{A}$.*

Outsider Collision Resistance. Outsider collision resistance essentially addresses, to some extent, the same requirements as covered by "enhanced collision resistance" in the work by Ateniese et al. [AMVA17]. That is, it grants the adversary $\mathcal{A}$ adaptive access to an $\mathsf{Adapt}_{\mathsf{PCH}}$ oracle, and requires that it be intractable to find collisions for messages which were not queried to $\mathsf{Adapt}_{\mathsf{PCH}}$. We note that this definition, analogous to [CDK+17], is even stronger than key-exposure freeness [AdM04,CZK04].[9]

```
$\mathbf{Exp}^{\mathsf{CROut}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$
  $(\mathsf{sk}_{\mathsf{PCH}}, \mathsf{pk}_{\mathsf{PCH}}) \leftarrow \mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)$
  $\mathcal{Q}, \mathcal{M} \leftarrow \emptyset$, $i \leftarrow 0$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{KGen}'_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}},\cdot),\mathsf{Adapt}'_{\mathsf{PCH}}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{PCH}})$
    where $\mathsf{KGen}'_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \cdot)$ on input $\mathbb{S}$:
      $\mathsf{sk}_{\mathbb{S}} \leftarrow \mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}, \mathbb{S})$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i, \mathsf{sk}_{\mathbb{S}})\}$
      $i \leftarrow i + 1$
    and $\mathsf{Adapt}'_{\mathsf{PCH}}(\cdot,\cdot,\cdot,\cdot,\cdot)$ on input $j, m, m', h, r$ :
      return $\perp$, if $\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, h, r) \neq 1 \ \vee$
      $(j, \mathsf{sk}_{\mathbb{S}}) \notin \mathcal{Q}$ for some $\mathsf{sk}_{\mathbb{S}}$
      $r' \leftarrow \mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, \mathsf{sk}_{\mathbb{S}}, m, m', h, r)$
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{m, m'\}$
      return $r'$
  return 1, if
    $\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m^*, h^*, r^*) = 1 \ \wedge$
    $\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m'^*, h^*, r'^*) = 1 \ \wedge$
    $m^* \notin \mathcal{M} \ \wedge \ m^* \neq m'^*$
  return 0
```

Fig. 3: PCH Outsider Collision Resistance

Definition 8 (Outsider Collision Resistance). *We define the advantage of an adversary $\mathcal{A}$ in the* CRout *experiment* $\mathbf{Exp}^{\mathsf{SCRout}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$ *as*

$$\mathbf{Adv}^{\mathsf{SCRout}}_{\mathcal{A},\mathsf{PCH}}(\kappa) := \Pr\left[\mathbf{Exp}^{\mathsf{SCRout}}_{\mathcal{A},\mathsf{PCH}}(\kappa) = 1\right].$$

We say that a PCH *scheme is outsider collision resistant, if* $\mathbf{Adv}^{\mathsf{SCRout}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$ *is a negligible function in κ for all PPT adversaries $\mathcal{A}$.*

Insider Collision Resistance. Insider collision resistance addresses the requirement that not even insiders who possess secret keys with respect to some attributes can find collisions for hashes which were computed with respect to policies which are not satisfied by their keys (oracle $\mathsf{KGen}'_{\mathsf{PCH}}$).

[9] Key-exposure only means that once a collision is made public, anyone can extract the secret key.

Intuitively, this notion enforces the attribute-based access-control policies, even if the adversary sees collisions for arbitrary attributes (oracles $\mathsf{KGen}''_{\mathsf{PCH}}$ and $\mathsf{Adapt}'_{\mathsf{PCH}}$).

```
$\mathbf{Exp}^{\mathsf{CRIns}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$
  $(\mathsf{sk}_{\mathsf{PCH}}, \mathsf{pk}_{\mathsf{PCH}}) \leftarrow \mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)$
  $\mathcal{S}, \mathcal{H}, \mathcal{Q} \leftarrow \emptyset$, $i \leftarrow 0$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk}_{\mathsf{PCH}})$
    where $\mathcal{O} \leftarrow \{\mathsf{KGen}'_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \cdot), \mathsf{KGen}''_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \cdot),$
      $\mathsf{Hash}'_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, \cdot, \cdot), \mathsf{Adapt}'_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, \cdot, \cdot, \cdot, \cdot)\}$
    and $\mathsf{KGen}'_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \cdot)$ on input $\mathbb{S}$:
      $\mathsf{sk}_{\mathbb{S}} \leftarrow \mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}, \mathbb{S})$
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbb{S}\}$
      return $\mathsf{sk}_{\mathbb{S}}$
    and $\mathsf{KGen}''_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \cdot)$ on input $\mathbb{S}$:
      $\mathsf{sk}_{\mathbb{S}} \leftarrow \mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}, \mathbb{S})$
      $\mathcal{Q} \cup \{(i, \mathsf{sk}_{\mathbb{S}})\}$
      $i \leftarrow i + 1$
    and $\mathsf{Hash}'_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, \cdot, \cdot)$ on input $m, \mathbb{A}$:
      $(h, r) \leftarrow \mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$
      $\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, \mathbb{A}, m)\}$
      return $(h, r)$
    and $\mathsf{Adapt}'_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, \cdot, \cdot, \cdot, \cdot, \cdot)$ on input $m, m', h, r, j$:
      return $\perp$, if $(j, \mathsf{sk}_{\mathbb{S}}) \notin \mathcal{Q}$ for some $\mathsf{sk}_{\mathbb{S}}$
      $r' \leftarrow \mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, \mathsf{sk}_{\mathbb{S}}, m, m', h, r)$
      if $(h, \mathbb{A}, m) \in \mathcal{H}$ for some $\mathbb{A}$, let $\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, \mathbb{A}, m')\}$
      return $r'$
  return 1, if
    $\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}, m^*, h^*, r^*) = \mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}, m'^*, h^*, r'^*) = 1 \wedge$
    $(h^*, \mathbb{A}, \cdot) \in \mathcal{H}$, for some $\mathbb{A}$ $\wedge$ $m^* \neq m'^*$ $\wedge$ $\mathbb{A} \cap \mathcal{S} = \emptyset$ $\wedge$
    $(h^*, \cdot, m^*) \notin \mathcal{H}$
  return 0
```

Fig. 4: PCH Insider Collision Resistance

Definition 9 (Insider Collision Resistance). *We define the advantage of an adversary $\mathcal{A}$ in the* SCRin *experiment* $\mathbf{Exp}^{\mathsf{SCRin}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$ *as*

$$\mathbf{Adv}^{\mathsf{SCRin}}_{\mathcal{A},\mathsf{PCH}}(\kappa) := \Pr\left[\mathbf{Exp}^{\mathsf{SCRin}}_{\mathcal{A},\mathsf{PCH}}(\kappa) = 1\right].$$

We say that a PCH *scheme is insider collision resistant, if the function* $\mathbf{Adv}^{\mathsf{SCRin}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$ *is a negligible function in* κ *for all PPT adversaries* $\mathcal{A}$*.*

3.2 Generic Construction

Our PCH construction is based on an IND-CCA2-secure CP-ABE scheme and a chameleon-hash with ephemeral trapdoors (CHET). We will sketch the overall idea first. The PCH setup runs the setup and the key generation of the CHET scheme as well as the key generation of the CP-ABE. Every participant obtains a secret key of the CHET and a secret key for the CP-ABE associated to a set of attributes. Hashing a message m to an access structure $\mathbb{A}$ means computing a CHET to the message m and encrypting the ephemeral trapdoor under $\mathbb{A}$ using the encryption algorithm of the CP-ABE. Collision-finding is possible if the $\mathsf{Adapt}_{\mathsf{PCH}}$ algorithm has access to the secret key of the CP-ABE for attributes $\mathbb{S}$ such that $\mathbb{S} \in \mathbb{A}$ is satisfied. This allows reconstructing the ephemeral trapdoor which in turn allows computing a collision in the CHET. The construction is depicted in Scheme 1.

$\underline{\mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)}$: Return $\mathsf{sk}_{\mathsf{PCH}} \leftarrow (\mathsf{msk}_{\mathsf{ABE}}, \mathsf{sk}_{\mathsf{chet}})$ and $\mathsf{pk}_{\mathsf{PCH}} \leftarrow (\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{pk}_{\mathsf{chet}})$, where

$$\begin{aligned}&\mathsf{pp}_{\mathsf{chet}} \leftarrow \mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa),\\&(\mathsf{sk}_{\mathsf{chet}}, \mathsf{pk}_{\mathsf{chet}}) \leftarrow \mathsf{KGen}_{\mathsf{CHET}}(\mathsf{pp}_{\mathsf{chet}}), \text{ and}\\&(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{mpk}_{\mathsf{ABE}}) \leftarrow \mathsf{Setup}_{\mathsf{ABE}}(1^\kappa).\end{aligned}$$

$\underline{\mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \mathbb{S})}$: Parse $\mathsf{sk}_{\mathsf{PCH}}$ as $(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{sk}_{\mathsf{chet}})$ and return $\mathsf{sk}_{\mathbb{S}} \leftarrow (\mathsf{sk}_{\mathsf{chet}}, \mathsf{ssk}')$, where

$$\mathsf{ssk}' \leftarrow \mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S}).$$

$\underline{\mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})}$: Parse $\mathsf{pk}_{\mathsf{PCH}}$ as $(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{pk}_{\mathsf{chet}})$ and return $(h, r) \leftarrow ((h_{\mathsf{chet}}, C), r_{\mathsf{chet}})$, where

$$\begin{aligned}&(h_{\mathsf{chet}}, r_{\mathsf{chet}}, \mathsf{etd}) \leftarrow \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{chet}}, m), \text{ and}\\&\text{and } C \leftarrow \mathsf{Enc}_{\mathsf{ABE}}(\mathsf{etd}, \mathbb{A}).\end{aligned}$$

$\underline{\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, h, r)}$: Parse $\mathsf{pk}_{\mathsf{PCH}}$ as $(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{pk}_{\mathsf{chet}})$, h as (h_{chet}, C), and r as r_{chet}. Return 1 if the following check holds and 0 otherwise:

$$\mathsf{CHET.Verify}(\mathsf{pk}_{\mathsf{chet}}, m, h_{\mathsf{chet}}, r_{\mathsf{chet}}) = 1.$$

$\underline{\mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{sk}_{\mathbb{S}}, m, m', h, r)}$: Parse $\mathsf{sk}_{\mathbb{S}}$ as $(\mathsf{sk}_{\mathsf{chet}}, \mathsf{ssk}')$ and h as (h_{chet}, C), and r as r_{chet}. Check whether $\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}, m, h, r) = 1$ and return $\perp$ otherwise. Compute $\mathsf{etd} \leftarrow \mathsf{Dec}_{\mathsf{ABE}}(\mathsf{ssk}', C)$ and return $\perp$ if $\mathsf{etd} = \perp$. Let $r' \leftarrow r'_{\mathsf{chet}}$, where

$$r'_{\mathsf{chet}} \leftarrow \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{chet}}, \mathsf{etd}, m, m', h, r_{\mathsf{chet}}).$$

Return $\perp$, if $\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m', h, r') = 0$ and r' otherwise.

Scheme 1: Black-box construction of a PCH scheme

Remark 1. In Scheme 1, a hash and also its verification does not allow to decide whether decrypting the ABE ciphertext will actually allow to compute a collision. We believe that many application scenarios do not require this. For instance, in the (permissioned) blockchain setting, when a party inserts a transaction, it is in the parties' best interest that this happens correctly and that its transaction could be rewritten if required. While one could clearly make the construction (as well as the model of PBCH) stronger, e.g., by requiring a non-interactive zero-knowledge (NIZK) proof that the CP-ABE ciphertext encrypts a valid CHET trapdoor, this would add a significant performance penalty. Obtaining an efficient construction in such a strong model, is a valuable avenue for future work.

Remark 2. We have based our construction on conventional CP-ABE, but to support multiple attribute authorities per policy, one could instead use a multi-authority CP-ABE scheme [Cha07,LW11]. We leave this for future work.

Now, we investigate the security of the PCH in Scheme 1.

Theorem 1. *If the* PCH *scheme in Scheme 1 is based on a strongly indistinguishable* CHET*, then the* PCH *scheme is strongly indistinguishable.*

Proof. We prove the theorem by constructing an efficient adversary $\mathcal{B}$ against $\mathbf{Exp}^{\mathsf{SInd}}_{\mathcal{B},\mathsf{CHET}}(\kappa)$, who uses an adversary $\mathcal{A}$ against $\mathbf{Exp}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$, with

$$\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{B},\mathsf{CHET}}(\kappa) \geq \mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{PCH}}(\kappa).$$

We let $\mathcal{B}$ proceed as follows:

- $\mathcal{B}$ obtains $(\mathsf{pk}_{\mathsf{chet}}, \mathsf{sk}_{\mathsf{chet}})$ from the indistinguishability challenger and completes Setup by running $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \Pi_{\mathsf{ABE}}.\mathsf{Setup}(1^\kappa)$ and setting $\mathsf{sk} \leftarrow (\mathsf{msk}, \mathsf{sk}_{\mathsf{chet}})$, and $\mathsf{pk} \leftarrow (\mathsf{mpk}, \mathsf{pk}_{\mathsf{chet}})$. Finally, it starts $\mathcal{A}$ on pk.
- For the $\mathsf{HashOrAdapt}$ oracle for $\mathcal{A}$, $\mathcal{B}$ internally uses the $\mathsf{HashOrAdapt}$ oracle provided by the challenger to obtain $(h_{\mathsf{chet}}, r_{\mathsf{chet}}, \mathsf{etd})$. Then it computes $C \leftarrow \Pi_{\mathsf{ABE}}.\mathsf{Enc}(\mathsf{etd}, \mathbb{A})$ and returns $((h_{\mathsf{chet}}, C), r_{\mathsf{chet}})$.
- As soon as $\mathcal{A}$ outputs its guess b, $\mathcal{B}$ forwards b to the challenger.

Now we observe that all oracles are simulated perfectly, and $\mathcal{B}$ wins with the same probability as $\mathcal{A}$ wins.

Theorem 2. *If the* PCH *scheme in Scheme 1 is based on a publicly collision resistant* CHET*, then the* PCH *scheme is outsider collision resistant.*

Proof. We prove the theorem by constructing an efficient adversary $\mathcal{B}$ against $\mathbf{Exp}^{\mathsf{CRpub}}_{\mathcal{B},\mathsf{CHET}}(\kappa)$, who uses an adversary $\mathcal{A}$ against $\mathbf{Exp}^{\mathsf{SCRout}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$, with

$$\mathbf{Adv}^{\mathsf{CRpub}}_{\mathcal{B},\mathsf{CHET}}(\kappa) \geq \mathbf{Adv}^{\mathsf{SCRout}}_{\mathcal{A},\mathsf{PCH}}(\kappa).$$

In particular, $\mathcal{B}$ proceeds as follows:

– $\mathcal{B}$ obtains $\mathsf{pk}_{\mathsf{chet}}$ from the indistinguishability challenger and completes Setup by running $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \Pi_{\mathsf{ABE}}.\mathsf{Setup}(1^\kappa)$ and setting $\mathsf{sk} \leftarrow (\mathsf{msk}, \perp)$, and $\mathsf{pk} \leftarrow (\mathsf{mpk}, \mathsf{pk}_{\mathsf{chet}})$. Finally, it starts $\mathcal{A}$ on pk.
– To simulate the Adapt' oracle for $\mathcal{A}$, $\mathcal{B}$ internally decrypts etd and then uses the Adapt oracle provided by the challenger.
– Whenever $\mathcal{A}$ outputs a collision being of the form $(m^*, r^*_{\mathsf{chet}}, m'^*, r'^*_{\mathsf{chet}}, (h^*_{\mathsf{chet}}, C^*))$, then $\mathcal{B}$ outputs $(m^*, r^*_{\mathsf{chet}}, m'^*, r'^*_{\mathsf{chet}}, h^*_{\mathsf{chet}})$ as a public collision for the CHET.

Now we observe that all oracles are simulated perfectly, and $\mathcal{B}$ wins with the same probability as $\mathcal{A}$ wins.

Theorem 3. *If the* PCH *scheme in Scheme 1 is based on a strongly privately collision resistant* CHET *and an* $\mathsf{IND\text{-}CCA2}$*-secure* ABE*, then the* PCH *scheme is insider collision resistant.*

Proof. We prove the theorem above in a sequence of games, where we use $\mathsf{Pr}[S_i]$ to denote the success probability of the adversary in Game i. In addition we let the number of queries to the oracle Hash' be denoted by q.

Game 0. This is the original CRin security experiment from Figure 4 played with Scheme 1.

Game 1. As Game 0, but we guess the index i^* corresponding to the query to Hash' which returns the hash h^* which will be attacked by the adversary. We store the hash $h^* = (h^*_{\mathsf{CHET}}, C^*)$ as well as the corresponding randomness r^* and the ephemeral trapdoor etd^*. If we detect that our guess is wrong at some point during the simulation, we abort.

The winning probability in Game 1 is the same as in Game 0, unless an abort happens. Therefore we have that $\mathsf{Pr}[S_1] = \mathsf{Pr}[S_0] \cdot {}^1\!/_q$.

Game 2. As Game 1, but whenever we receive an adapt query for a hash $h = (h_{\mathsf{CHET}}, C)$, where $C = C^*$ we do not decrypt, but directly adapt using etd^*.

The winning probability in Game 2 is the same as in Game 1 under the perfect correctness of the encryption scheme, i.e., $\mathsf{Pr}[S_2] = \mathsf{Pr}[S_1]$.

Game 3. As Game 2, but we change the simulation of the Hash algorithm within the i^*-th query to the $\mathsf{Hash}'_{\mathsf{CHET}}$ oracle: instead of running $C \leftarrow \Pi_{\mathsf{ABE}}.\mathsf{Enc}(\mathsf{etd}, \mathbb{A})$, we run $C \leftarrow \Pi_{\mathsf{ABE}}.\mathsf{Enc}(0^{|\mathsf{etd}|}, \mathbb{A})$ and locally store etd.

We claim that Game 2 and Game 3 are indistinguishable under the $\mathsf{IND\text{-}CCA2}$ security of Π_{ABE}, i.e., $|\mathsf{Pr}[S_3] - \mathsf{Pr}[S_2]| \leq \mathbf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{B},\mathsf{ABE}}(\kappa)$. To prove the claim, we show that we can use an adaptive $\mathsf{IND\text{-}CCA}$ challenger to effectively interpolate between Game 2 and Game 3. In particular, consider the following hybrid game: Upon setup we obtain mpk from an $\mathsf{IND\text{-}CCA}$ challenger, set $\mathsf{msk} \leftarrow \perp$ and complete the remainder of the setup honestly. To simulate queries to the key generation oracles we use the respective oracles provided by the challenger. Decryption within the adapt oracle is done by using the decryption oracle provided by the challenger. Furthermore, upon the i^*-th query to Hash', we output $(\mathsf{etd}, 0^{|\mathsf{etd}|}, \mathbb{A}, \mathtt{state})$ to the challenger to obtain $(C^*, \mathtt{state})$ and set $C \leftarrow C^*$. For adapt queries with respect to the hash returned upon the i^*-th query to Hash', we directly adapt using etd without prior decryption. Now, observe that aborting as soon as we detect that our guess of index i^* is wrong ensures that we will never have to answer queries which involve queries to the challenger's oracle which would not be answered. This, in turn, means that if the bit b of the challenger is 0 we perfectly simulate Game 2, whereas we perfectly simulate Game 3 if $b = 1$. This proves the claim.

Reduction to Strong Private Collision Resistance. Now we are ready to describe the reduction to private collision resistance. In particular, we obtain $\mathsf{pp}_{\mathsf{CHET}}$ from a private collision resistance challenger $\mathcal{C}$ and honestly complete the setup. Then we simulate all oracles except the hash and the adapt oracle as in Game 3. In particular, we can internally simulate $\mathsf{KGen}_{\mathsf{CHET}}$ and all queries to the $\mathsf{Hash}'_{\mathsf{CHET}}$ oracle up to the i^*-th query. In the i^*-th query to the $\mathsf{Hash}'_{\mathsf{CHET}}$ oracle, we use the Hash' oracle provided by the private collision resistance challenger to obtain $(h_{\mathsf{chet}}, r_{\mathsf{chet}})$. As the ciphertext C already encrypts $0^{|\mathsf{etd}|}$ instead of etd we do not require to know etd. Likewise, for the adaption oracle, we only modify the simulation for queries with respect to the etd returned upon the i^*-th query to $\mathsf{Hash}'_{\mathsf{CHET}}$ in that we use the adaption oracle provided by the challenger to compute the adapted hashes. If the adversary eventually outputs a collision $(m^*, r^*_{\mathsf{chet}}, m'^*, r'^*_{\mathsf{chet}}, (h^*_{\mathsf{chet}}, C^*))$, we output $(m^*, r^*_{\mathsf{chet}}, m'^*, r'^*_{\mathsf{chet}}, h^*_{\mathsf{chet}})$ as a private collision for the CHET. Consequently, we have that $\mathbf{Adv}^{\mathsf{SCRpriv}}_{\mathcal{C},\mathsf{CHET}}(\kappa) \geq \mathsf{Pr}[S_3]$.

Overall Bound. As we have shown above, the advantage of any adversary in the final game is bounded by the advantage of any adversary in the private collision freeness game, i.e., $\mathsf{Pr}[S_3] \leq \mathbf{Adv}^{\mathsf{SCRpriv}}_{\mathcal{C},\mathsf{CHET}}(\kappa)$. This yields the following bound for the original game $\mathbf{Adv}^{\mathsf{sCRin}}_{\mathcal{A},\mathsf{PCH}}(\kappa) \leq q \cdot (\mathbf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{B},\mathsf{ABE}}(\kappa) + \mathbf{Adv}^{\mathsf{SCRpriv}}_{\mathcal{C},\mathsf{CHET}}(\kappa))$, which concludes the proof.

3.3 On the Choice of Access Policies

The policy expressiveness of our PCH construction is given by the policy expressiveness of the underlying CP-ABE scheme. In general, the most basic and reasonable access policies offer at least monotonic operators such as AND and OR. Basically, a monotone access policy for CP-ABE ensures that adding an attribute to the secret key does not lead to the case where the access policy cannot be satisfied anymore under that updated secret key (assuming that the secret key has satisfied the policy before of course). If we speak of access policies in this work, we refer to monotone access policies. Often, access policies can directly be represented in a tree-based fashion, i.e., see for example [GPSW06]. In many cases also threshold gates can be used. Usually, the underlying building block for encoding access policies are monotone span programs (MSP) [KW93]. Those techniques offer more generality than encoding the access policy as boolean formula and are the de-facto standard in the cryptographic literature for encoding policies in ABE schemes. For access policies, we use monotone span programs (MSPs) as described in Section 2 for encoding the access policy.

More expressive policies are known in the cryptographic literature, e.g., ones that allow for NOT-gates to be present within an access policy. However, the provided ABE schemes are often not as efficient compared to their monotonic counterparts in a practical sense. In [GPSW06], Goyal et al. describe an inefficient way of realizing more general access policies to allow the NOT operator. Furthermore, we can allow even for access policies that are represented as general circuits as described by the work of Gorbunov et al. [GVW13] based on lattices and by the work of Garg et al. [GGH$^+$13] based on multilinear maps. However, those scheme are not at all efficient yet. Hence, for practical considerations, schemes in the bilinear group setting supporting monotone access policies (i.e., monotone span programs) currently seem to be the optimal choice.

4 Instantiation and Evaluation

In this section, we start with discussing the choice of primitives to come up with an efficient instantiation of our generic approach to PCHs. We then present our construction and report on the evaluation of first implementation results of our concrete PCH.

4.1 Selecting a CP-ABE Scheme

In terms of practical CP-ABE instantiations, we consider the recent work by Agrawal and Chase on a very efficient CP-ABE scheme they dubbed FAME [AC17]. FAME supports unbounded ABE universes, has no restrictions on the monotone policies used, is based on efficient Type-III pairings, has constant-time decryption, and is adaptively secure under a standard assumption. Those features make FAME very versatile in practical environments as used in our context. The only restriction is that FAME does not support multiple attributes in a ciphertext policy (which is commonly referred to as one-use restriction). However, FAME can be adapted to allow up to a constant number of the repeating attributes within a policy by increasing the ciphertext and keys by a small factor.

The CP-ABE scheme by Waters [Wat11] yields a good candidate for our scenario as well; in particular, since his construction does not have the one-use restriction. However, Waters' scheme is only selectively secure and requires a q-type (i.e., non-standard and non-static) assumption. In a selective security model, the adversary has to output the target access policy before receiving the public parameters of the system which is clearly not a realistic scenario and too weak for the security required by PCHs. Furthermore, encryption and decryption is less efficient compared to FAME [AC17].

We also considered the state-of-the-art adaptively secure CP-ABE scheme from the literature, i.e., the work by Chen, Gay, and Wee [CGW15]. Their scheme is fairly efficient (see [AC17] for comparisons); however, not suitable for our scenario due to the restriction of a bounded universe of attributes.

Finally, we end up with selecting FAME [AC17]. Since FAME only provides IND-CPA security, but we require IND-CCA2 security, we apply a variant of the well known Fujisaki-Okamoto transform [FO99] (see [KW18]) to FAME. Basically, the encryption algorithm will encrypt as its message (m, r) with m the original message and r a sufficiently large randomly sampled bitstring (this requires to injectively encode (m, r) into the message space of ABE). The ABE encryption is derandomized and uses as the random coins $H(r, \mathbb{A})$ where H is a hash function modeled as a random oracle and $\mathbb{A}$ the used access policy to obtain the ciphertext C. The decryption algorithm applies the original decryption algorithm from IND-CPA-secure ABE to receive (m', r'). Then, it re-encrypts (m', r') using random coins $H(r, \mathbb{A})$ to obtain ciphertext C'. If it holds that $C = C'$, it outputs m' and otherwise it outputs $\perp$. Note that if we want to use the resulting scheme as an IND-CCA2-secure KEM, we can simply sample a random (say ℓ bit) key k and use (k, r) as the message to be encrypted with the ABE and k as the key (and also need to include k as input to H). Observe, that clearly for the transformation to work, $\mathbb{A}$ needs to be known to the decryption algorithm. We can safely assume that this can be inferred from a given ciphertext (of the IND-CPA-secure variant of ABE), i.e., by simply appending a canonical representation of $\mathbb{A}$ to the ciphertext.

4.2 Modified CHET

Camenisch et al. in [CDK+17] provide, among others, a generic construction of a CHET by combining two chameleon-hashes, both requiring collision resistance even in presence of a collision-finding oracle. The keys for the second chameleon-hash are drawn freshly for each new hash. Thus, the secret key for the second chameleon-hash is the ephemeral trapdoor. In this section, we provide a construction which is essentially the one given by Camenisch et al. [CDK+17], but we additionally check whether a hash h is valid after adaption and add the two public keys to the hash-computation, as already done by Krenn et al. [KPSS18], but in a slightly different context. This allows us to prove our stronger notion of private collision resistance required in this work. The formal definitions are given in Appendix B while our generic construction is given in Scheme 2.

Theorem 4 (Security of Scheme 2). *If* CH *is strongly indistinguishable, collision resistant, and correct, then the construction of a* CHET *given in Construction 2 is strongly indistinguishable, publicly collision resistant, strongly privately collision resistant, and correct.*

We provide a proof of this theorem in Appendix B.4.

Concrete Instantiation. For our concrete instantiation of the CHET, we use the RSA-based CH from [CDK+17] which builds upon the one presented by Brzuska et al. [BFF+09]. We recall this construction in Appendix B.2 and show its security in our stronger model.

$\underline{\mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)}$: On input a security parameter κ, let $\textsc{pp}_{\mathsf{CH}} \leftarrow \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$. Return $\textsc{pp}_{\mathsf{CHET}} \leftarrow \textsc{pp}_{\mathsf{CH}}$.

$\underline{\mathsf{KGen}_{\mathsf{CHET}}(\textsc{pp}_{\mathsf{CHET}})}$: On input $\textsc{pp}_{\mathsf{CHET}} = \textsc{pp}_{\mathsf{CH}}$, return $(\mathsf{sk}^1_{\mathsf{CH}}, \mathsf{pk}^1_{\mathsf{CH}}) \leftarrow \mathsf{KGen}_{\mathsf{CH}}(\textsc{pp}_{\mathsf{CH}})$.

$\underline{\mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m)}$: On input of $\mathsf{pk}_{\mathsf{CHET}} = \mathsf{pk}^1_{\mathsf{CH}}$ and m, return $((h^1, h^2, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), (r^1, r^2))$, where

$$\begin{aligned}
&(\mathsf{etd}, \mathsf{pk}^2_{\mathsf{CH}}) \leftarrow \mathsf{KGen}_{\mathsf{CH}}(\textsc{pp}_{\mathsf{CH}})\\
&(h^1, r^1) \leftarrow \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}^1_{\mathsf{CH}}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}))\\
&(h^2, r^2) \leftarrow \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}^2_{\mathsf{CH}}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}))
\end{aligned}$$

$\underline{\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, h, r)}$: On input of $\mathsf{pk}_{\mathsf{CHET}} = \mathsf{pk}^1_{\mathsf{CH}}$, m, $h = (h^1, h^2, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}})$ and $r = (r^1, r^2)$, return 1, if:

$$\begin{aligned}
&\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}^1_{\mathsf{CH}}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), h^1, r^1) = 1 \wedge\\
&\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}^2_{\mathsf{CH}}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), h^2, r^2) = 1
\end{aligned}$$

Otherwise, return 0.

$\underline{\mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{etd}, m, m', h, r)}$: On input a secret key $\mathsf{sk}_{\mathsf{CHET}} = \mathsf{sk}^1_{\mathsf{CH}}$, etd, messages m and m', a hash $h = (h^1, h^2, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}})$ and $r = (r^1, r^1)$, first check that $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, h, r) = 1$. Otherwise, return $\perp$. Return r', if $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m', h, r') = 1$ where

$$\begin{aligned}
&r'^1 \leftarrow \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}^1_{\mathsf{CHET}}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), (m', \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), r^1, h^1)\\
&r'^2 \leftarrow \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{etd}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), (m', \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), r^2, h^2)\\
&r' \leftarrow (r'^1, r'^2)
\end{aligned}$$

Otherwise, return $\perp$.

Scheme 2: Construction of a CHET

4.3 Selection of Suitable Parameters

Subsequently, we discuss the selection of parameters for the required cryptographic building blocks considering the cryptanalytic state-of-the-art.

Bilinear Groups. In the sequel, let BilGen be an algorithm that, on input a security parameter 1^κ, outputs $(p, \hat{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2) \leftarrow \mathsf{BilGen}(1^\kappa)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of prime order p with bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ and generators $g_i \in \mathbb{G}_i$ for $i \in \{1, 2\}$. We choose to use Type-III bilinear groups as they represent the state-of-the-art regarding efficiency and similarity of the security levels of the base and target groups. Thereby, we need to cope with recent advances for computing discrete logarithms in finite extension fields [KB16], which apply to the target group $\mathbb{G}_T$ of state-of-the-art pairing-friendly elliptic curve groups. This impact is assessed by Menezes et al. [MSS16] as well as Barbulescu and Duquesne [BD18], whereas former estimates are less conservative. Regarding a concrete choice of a curve family, we choose the popular BN curve family [BN06] and in particular the BN254 curve, which gives us around 100 bits of security. If one wants to go for a larger security level with comparable performance, one can follow the same lines as Zcash[10] and choose the BLS curve family [BLS03] and in particular the curve BLS12-381 giving roughly 120 bits of security.[11]

RSA. To have a security level of RSA parameters that is comparable to the one for our chosen bilinear groups, we selected moduli of 2048 bits in size.

4.4 A Concrete PBCH

We now present an efficient instantiation of a PCH. In particular, we instantiate our PCH using the concrete CHET introduced in Section 4.2. We stress that although we end up in a construction that instantiates one component based on the RSA setting and the other one in a prime order group, neither influences the other regarding choice of the security parameters and it is a compromise to obtain a good overall efficiency.

Our Construction. In Scheme 3, 4, and 5 we present our concrete instantiation. Likewise to the abstract algorithmic definition of PCHs, we assume that all algorithms implicitly have access to pk. Note that we use the CP-ABE scheme, which is made IND-CCA2 secure as discussed in Section 4.1. This scheme is used as a CCA secure KEM and combined with an IND-CCA2 secure symmetric encryption scheme to obtain a CCA2 secure hybrid encryption scheme (using the compiler formalized in [CS03]). Consequently, we encrypt the ephemeral trapdoor using the symmetric scheme (denoted by $(\mathsf{KGen}_{\mathsf{SE}}, \mathsf{Enc}_{\mathsf{SE}}, \mathsf{Dec}_{\mathsf{SE}})$ in our construction). It is easy to show that this modification preserves the adaptive IND-CCA2 security of the modified CP-ABE scheme. We note that the hash functions H_1, H_2, H_3, H_4 are modeled as a random oracles (ROs) [BR93][12] and let $\mathsf{enc} : \{0,1\}^{\ell+\kappa} \to \mathbb{G}_\mathsf{T}$ be in injective encoding function and ℓ be the maximum length of keys output by $\mathsf{KGen}_{\mathsf{SE}}(1^\kappa)$. Combining the results in Theorems 1-3, 4, and Theorem 1 from [CDK⁺17], we obtain the following.

Corollary 1. *The construction in Scheme 3, 4, and 5 is an indistinguishable, outsider and insider collision resistant* PCH.

[10] https://z.cash/blog/new-snark-curve.html

[11] Estimates from a personal communication with Razvan Barbulescu.

[12] We note that any practical ABE scheme in literature that support an unlimited number of attributes from an unbounded domain are proven secure in the Random Oracle Model.

$\underline{\mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)}$: On input security parameter κ run

1. Choose prime e_1 s.t. $e_1 > N'$ with $N' = \max_r\{(N,\cdot,\cdot,\cdot,\cdot) \leftarrow \mathsf{RSAKGen}(1^\kappa; r)\}$.
2. Run $(N_1, p_1, q_1, \cdot, \cdot) \leftarrow \mathsf{RSAKGen}(1^\kappa)$, choose a hash function $H_1 : \{0,1\} \rightarrow \mathbb{Z}^*_{N_1}$. Compute d_1 s.t. $ed_1 \equiv 1 \mod (p_1 - 1)(q_1 - 1)$, set $\mathsf{sk}_{\mathsf{CHET}} \leftarrow (d_1)$ $\mathsf{pk}_{\mathsf{CHET}} \leftarrow (\kappa, N_1, e, H_1)$.
3. Run $(p, \hat{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2) \leftarrow \mathsf{BilGen}(1^\kappa)$. Pick $a_1, a_2, b_1, b_2 \leftarrow_r \mathbb{Z}_p^*$ and $w_1, w_2, w_3 \leftarrow_r \mathbb{Z}_p$, hash functions $H_3 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_4 : \{0,1\}^* \rightarrow (\mathbb{Z}_p)^2$, an encryption scheme $(\mathsf{KGen}_{\mathsf{SE}}, \mathsf{Enc}_{\mathsf{SE}}, \mathsf{Dec}_{\mathsf{SE}})$, and set

$$\mathsf{mpk}_{\mathsf{ABE}} \leftarrow (g_2, q_1 \leftarrow g_2^{a_1}, q_2 \leftarrow g_2^{a_2}, T_1 \leftarrow \hat{e}(g_1, g_2)^{w_1 a_1 + w_3}, T_2 \leftarrow \hat{e}(g_1, g_2)^{w_2 a_2 + w_3}, H_3, H_4, (\mathsf{KGen}_{\mathsf{SE}},, \mathsf{Enc}_{\mathsf{SE}}, \mathsf{Dec}_{\mathsf{SE}})), \text{and}$$
$$\mathsf{msk}_{\mathsf{ABE}} \leftarrow (a_1, a_2, b_1, b_2, g_2, g_1^{w_1}, g_1^{w_2}, g_1^{w_3})$$

and return $\mathsf{sk}_{\mathsf{PCH}} \leftarrow (\mathsf{msk}_{\mathsf{ABE}}, \mathsf{sk}_{\mathsf{CHET}})$ and $\mathsf{pk}_{\mathsf{PCH}} \leftarrow (\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{pk}_{\mathsf{CHET}})$.

$\underline{\mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \mathbb{S})}$: On input $\mathsf{sk}_{\mathsf{PCH}} = (\mathsf{msk}_{\mathsf{ABE}}, \mathsf{sk}_{\mathsf{CHET}})$ and attribute set $\mathbb{S}$, parse msk as $(a_1, a_2, b_1, b_2, g_2, g_1^{w_1}, g_1^{w_2}, g_1^{w_3})$, pick $\rho_1, \rho_2 \leftarrow_r \mathbb{Z}_p$ and compute

$$\mathsf{sk}_0 = (\mathsf{sk}_{0,1}, \mathsf{sk}_{0,2}, \mathsf{sk}_{0,3}) \leftarrow (g_2^{b_1\rho_1}, g_2^{b_2\rho_2}, g_2^{\rho_1+\rho_2}).$$

For all $s \in \mathbb{S}$ and $t = 1, 2$ compute

$$sk_{s,t} \leftarrow H_3(y\|1\|t)^{\frac{b_1\rho_1}{a_t}} \cdot H_3(y\|2\|t)^{\frac{b_2\rho_2}{a_t}} \cdot H_3(y\|3\|t)^{\frac{\rho_1+\rho_2}{a_t}} \cdot g_1^{\frac{\sigma_s}{a_t}},$$

where $\sigma_s \leftarrow_r \mathbb{Z}_p$ and set $\mathsf{sk}_s \leftarrow (\mathsf{sk}_{s,1}, \mathsf{sk}_{s,2}, g_1^{-\sigma_s})$. Moreover, for $t = 1, 2$ compute

$$\mathsf{sk}'_t \leftarrow g_1^{w_t} \cdot H_3(011\|t)^{\frac{b_1\rho_1}{a_t}} \cdot H_3(012\|t)^{\frac{b_2\rho_2}{a_t}} \cdot H_3(013\|t)^{\frac{\rho_1+\rho_2}{a_t}} \cdot g_1^{\frac{\sigma'}{a_t}},$$

where $\sigma' \leftarrow_r \mathbb{Z}_p$ and set $\mathsf{sk}' \leftarrow (\mathsf{sk}'_1, \mathsf{sk}'_2.g_1^{d_3} \cdot g_1^{-\sigma'})$. Set $\mathsf{sk}_{\mathbb{S},\mathsf{ABE}} \leftarrow (\mathsf{sk}_0, \{\mathsf{sk}_s\}_{s\in\mathbb{S}}, \mathsf{sk}')$ and return $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{sk}_{\mathbb{S},\mathsf{ABE}})$.

Scheme 3: Concrete PCH construction: parameter and key generation

4.5 Performance Evaluation

To demonstrate the practicality of our scheme, we implemented our construction in Python 3.5.3 and base our implementation on the Charm framework [AGM+13] version 0.50[13], the implementation of FAME from the authors of [AC17][14], whereas we use our own implementation of CH. We stress that we do not implement the IND-CCA2-secure variant of FAME, but use the IND-CPA-secure one. This is because our implementation is only intended to be used for evaluation purposes and we do not expect notable differences due to using the IND-CPA-variant. The measurements were performed on a laptop with an Intel Core i7-7600U CPU @ 2.80GHz with 16GB RAM running Ubuntu 18.04. For our benchmarks, we use access policies with $n \in \{8, 16, 32, 64\}$ attributes, where the poli-

[13] https://github.com/JHUISI/charm
[14] https://github.com/sagrawal87/ABE

```
$\underline{\mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbf{M})}$ : On input public key $\mathsf{pk} = (\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{pk}_{\mathsf{CHET}})$, a message $m$ and a
    matrix encoding $\mathbf{M}$ of access structure $\mathbb{A}$, parse $\mathsf{pk}_{\mathsf{CHET}} = (\kappa, N_1, e, H_1)$ and:
    1. Run $(N_2, p_2, q_2, \cdot, \cdot) \leftarrow \mathsf{RSAKGen}(1^\kappa)$, fix a hash function $H_2 : \{0,1\} \to \mathbb{Z}^*_{N_2}$. Compute $d_2$ s.t. $ed_2 \equiv 1 \mod (p_2 - 1)(q_2 - 1)$.
    2. Choose $r_1 \leftarrow_r \mathbb{Z}^*_{N_1}$, $r_2 \leftarrow_r \mathbb{Z}^*_{N_2}$, compute $h_1 \leftarrow H_1((m, N_1, H_1, N_2, H_2))r_1^e \mod N$ and $h_2 \leftarrow H_2((m, N_1, H_1, N_2, H_2))r_2^e$. Set $h' \leftarrow (h_1, h_2)$ and $r' \leftarrow (r_1, r_2)$.
    3. Choose $r \leftarrow_r \{0,1\}^\kappa$, $k \leftarrow_r \mathsf{KGen}_{\mathsf{SE}}(1^\kappa)$ compute $(u_1, u_2) \leftarrow H_4((r, \mathbb{A}))$ and $\mathsf{ct}_0 \leftarrow (q_1^{u_1}, q_2^{u_2}, g_2^{u_1+u_2})$. Assuming $\mathbf{M}$ has $\ell$ rows and $k$ columns, then for $i \in [\ell]$ and $z = 1, 2, 3$ compute

       $$\mathsf{ct}_{i,z} \leftarrow H_3(\pi(i)\|z\|1)^{u_1} \cdot H_3(\pi(i)\|z\|2)^{u_2} \cdot \prod_{j=1}^{k} \left[H_3(0\|j\|z\|1)^{u_1} \cdot H_3(0\|j\|z\|2)^{u_2}\right]^{(\mathbf{M})_{i,j}}.$$

       Set $\mathsf{ct}_i \leftarrow (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2}, \mathsf{ct}_{i,3})$, $K \leftarrow \mathsf{encode}(k, r)$, compute $\hat{\mathsf{ct}} \leftarrow T_1^{u_1} \cdot T_2^{u_2} \cdot K$, $\tilde{\mathsf{ct}} \leftarrow_r \mathsf{Enc}_{\mathsf{SE}}(k, d_2)$, and set $\mathsf{ct} \leftarrow (\mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell, \hat{\mathsf{ct}}, \tilde{\mathsf{ct}})$.
    Return $(h, r) \leftarrow ((h', N_2, H_2, \mathsf{ct}), r')$.
$\underline{\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, h, r)}$ : On input public parameters $\mathsf{pk} = (\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{pk}_{\mathsf{CHET}})$, message
    $m$, hash value $h = (h', N_2, H_2, \mathsf{ct})$, and randomness $r = (r_1, r_2)$, parse $\mathsf{pk}_{\mathsf{CHET}} = (\kappa, N_1, e, H_1)$, check if $r_1 \in \mathbb{Z}^*_{N_1}$, $r_2 \in \mathbb{Z}^*_{N_2}$, and if $h_1 = H_1((m, N_1, H_1, N_2, H_2))r_1^e \mod N_1$ and $h_2 = H_2((m, N_1, H_1, N_2, H_2))r_2^e \mod N_2$. If all checks hold, return 1 and 0 otherwise.
```

Scheme 4: Concrete PCH construction: hashing and verification

cies always consist of two OR clauses with $n/2$ attributes connected via AND. This is a rather pessimistic choice as for FAME, AND gates are more expensive than OR gates.

PBCH Algorithm	8 Att.	16 Att.	32 Att.	64 Att.
Setup	438	477	457	474
KGen	87	170	302	596
Hash	235	314	386	597
Verify	46	48	45	46
Adapt	286	403	379	385

Table 1: Performance evaluation: runtimes in [ms]

We examine that most runtimes of the algorithms are dominated by the operations related to the underlying primitives of our CHET. Algorithm Setup, which needs to generate a large prime $e > 2048$ bit, an 2048-bit RSA modulus N and a suitable prime-order group, is a one-time operation and independent of the number of attributes, and is therefore irrelevant from a practical perspective. The Hash and Adapt operations are also dominated by the modular exponentiations. Similar to Setup, Hash and Adapt are infrequent operations and in particular in the con-

$\underline{\mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{sk}_{\mathbb{S}}, m, m', h, r)}$: On input $\mathsf{sk}_{\mathbb{S}} = (\mathsf{sk}_{\mathsf{CHET}}, \mathsf{sk}_{\mathbb{S},\mathsf{ABE}})$ messages m and m', hash value $h = (h', N_2, H_2, \mathsf{ct})$ and randomness $r = (r_1, r_2)$, assuming that $\mathbb{S}$ satisfies access policy $\mathbb{A}$ used for the hash (otherwise return $\perp$), parse $\mathsf{sk}_{\mathsf{CHET}} = (d_1)$ and:

1. Compute coefficients $\{\lambda_i\}_{\pi(i)\in\mathbb{S}}$ for the MSP $(\mathbf{M}, \pi)$ associated to $\mathbb{A}$ as discussed in Section 2, compute
$$A \leftarrow \hat{\mathsf{ct}} \cdot \hat{e}(\prod_{i\in I} \mathsf{ct}_{i,1}^{\lambda_i}, \mathsf{sk}_{0,1}) \cdot \hat{e}(\prod_{i\in I} \mathsf{ct}_{i,2}^{\lambda_i}, \mathsf{sk}_{0,2}) \cdot \hat{e}(\prod_{i\in I} \mathsf{ct}_{i,3}^{\lambda_i}, \mathsf{sk}_{0,3}),$$
$$B \leftarrow \hat{e}(\mathsf{sk}'_1 \cdot \prod_{i\in I} \mathsf{sk}_{\pi(i),1}^{\lambda_i}, \mathsf{ct}_{0,1}) \cdot \hat{e}(\mathsf{sk}'_2 \cdot \prod_{i\in I} \mathsf{sk}_{\pi(i),2}^{\lambda_i}, \mathsf{ct}_{0,2}) \cdot \hat{e}(\mathsf{sk}'_3 \cdot \prod_{i\in I} \mathsf{sk}_{\pi(i),3}^{\lambda_i}, \mathsf{ct}_{0,3}),$$
$K' \leftarrow A \cdot B^{-1}$ and set $(k', r') \leftarrow \mathsf{encode}^{-1}(K')$.
2. Compute $(u'_1, u'_2) \leftarrow H_4((r', \mathbb{A}))$ and $\mathsf{ct}'_0 \leftarrow (q_1^{u'_1}, q_2^{u'_2}, g_2^{u'_1+u'_2})$. Assuming $\mathbf{M}$ has ℓ rows and k columns, then for $i \in [\ell]$ and $z = 1, 2, 3$ compute
$$\mathsf{ct}'_{i,z} \leftarrow H_3(\pi(i)\|z\|1)^{u'_1} \cdot H_3(\pi(i)\|z\|2)^{u'_2} \cdot \prod_{j=1}^{k} \left[H_3(0\|j\|z\|1)^{u'_1} \cdot H_3(0\|j\|z\|2)^{u'_2}\right]^{(\mathbf{M})_{i,j}}.$$
Set $\mathsf{ct}'_i \leftarrow (\mathsf{ct}'_{i,1}, \mathsf{ct}'_{i,2}, \mathsf{ct}'_{i,3})$, compute $\hat{\mathsf{ct}}' \leftarrow T_1^{u'_1} \cdot T_2^{u'_2} \cdot K'$, and set $\mathsf{ct}' \leftarrow (\mathsf{ct}'_0, \mathsf{ct}'_1, \ldots, \mathsf{ct}'_\ell, \hat{\mathsf{ct}}')$. If $\mathsf{ct} \neq \mathsf{ct}'$ return $\perp$. Otherwise compute $d'_2 \leftarrow \mathsf{Dec}_{\mathsf{SE}}(k', \tilde{\mathsf{ct}}')$ and return $\perp$ if $d'_2 = \perp$.
3. Check if $h_1 = H_1((m, N_1, H_1, N_2, H_2))r_1^e \bmod N_1$ and $h_2 = H_2((m, N_1, H_1, N_2, H_2))r_2^e \bmod N_2$ and return $\perp$, if any of the checks fails. Otherwise, let $x_1 \leftarrow H_1((m, N_1, H_1, N_2, H_2))$, $x'_1 \leftarrow H_1((m', N_1, H_1, N_2, H_2))$, $y_1 \leftarrow x_1 r_1^e \bmod N_1$ as well as $x_2 \leftarrow H_2((m, N_1, H_1, N_2, H_2))$, $x'_2 \leftarrow H_2((m', N_1, H_1, N_2, H_2))$, $y_2 \leftarrow x_2 r_2^e \bmod N_2$, compute $r'_1 \leftarrow (y_1(x_1'^{-1}))^{d_1} \bmod N_1$, and finally $r'_2 \leftarrow (y_2(x_2'^{-1}))^{d_2} \bmod N_2$.
4. Return $\perp$ if $h_1 \neq H_1((m', N_1, H_1, N_2, H_2))r_1'^e \bmod N_1$ or $h_2 \neq H_2((m', N_1, H_1, N_2, H_2))r_2'^e \bmod N_2$.

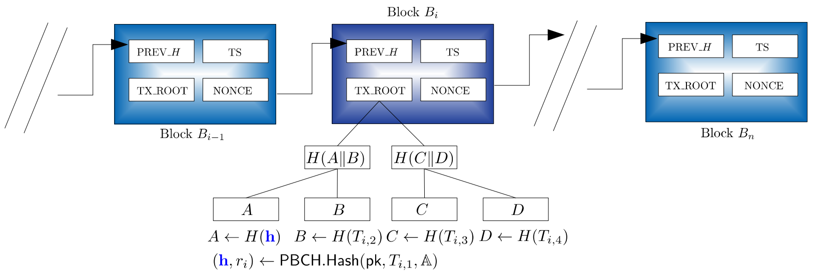
Finally, return $r' \leftarrow (r'_1, r'_2)$.

Scheme 5: Concrete PCH construction: adaptation

text of blockchains are totally acceptable. The operation that is most time critical, as it is required whenever verifying the transactions within a blockchain is the Verify algorithm, which is very efficient and also *constant*. A more detailed evaluation is presented in Appendix A.

5 Blockchain Transaction-Level Rewriting

In this section, we come back to the application of policy-based chameleon-hash functions (PCHs) to rewriting objects in blockchains, where we use the syntax of the Bitcoin blockchain for our discussion. We recall that while Ateniese et al. [AMVA17] target rewriting entire *blocks* within a blockchain, we propose *transaction-level rewriting*. Here, blocks in the blockchain remain intact but only specific transactions inside a block can be rewritten. We deem this application much more important than when focusing on blocks as it is much more fine-grained and keeps the overall blockchain intact.



Block B_i
PREV_H
TS
TX_ROOT
NONCE
Block B_{i-1}
Block B_n
$H(A\|B)$
$H(C\|D)$
A
B
C
D
$A \leftarrow H(\mathbf{h})$
$B \leftarrow H(T_{i,2})$
$C \leftarrow H(T_{i,3})$
$D \leftarrow H(T_{i,4})$
$(\mathbf{h}, r_i) \leftarrow \mathsf{PBCH.Hash}(\mathsf{pk}, T_{i,1}, \mathbb{A})$

Fig. 5: Using a PCH for transaction-level rewrites

We recall that each block in a blockchain stores a compact representation of a set of transactions, i.e., the root hash of a Merkle tree (denoted TX_ROOT) which accumulates all transactions associated to a block. Now, one way to integrate transaction-level rewriting capabilities into blockchains by means of PCHs is as follows. Every participant who engages in the role of an attribute authority includes pk using a transaction signed under the key corresponding to the public key of an address owner (we stress that there are various other ways of distributing the pk's in a way that they can be verified). The attribute authority can then issue PCH secret keys to other users. If a user wants to include a modifiable transaction the transaction needs to be hashed using the PCH. In Figure 5, we consider a toy example of a block (B_i) which accumulates four transactions $T_{i,1}, \ldots, T_{i,4}$. Let us assume that transaction $T_{i,1}$ should be rewritable by users that satisfy access policy $\mathbb{A}$. Then the last three transactions ($T_{i,2}$ to $T_{i,4}$) are processed as usual, i.e., input to the hash computation based on H, but the first transaction is preprocessed by means of the PCH and the hash value A is input in the Merkle tree. Observe that the randomness r_i is not included in the hash computation of the aggregation and is provided as non-hashed part of the transaction/block. When the transaction needs to be updated, everyone with a secret key satisfying $\mathbb{A}$ can compute a collision for hash value A and provide the new randomness r'_i. Note that in contrast to the scenario of Ateniese et al. [AMVA17], the hash function used to chain blocks remains to be a conventional collision resistant hash function and the PREV_H values are never updated.

Let us briefly recall how the security properties of the PCH come into play. Indistinguishability guarantees that it is not detectable whether a hash computed by means of the PCH has been adapted, i.e., whether a rewrite happened. We stress that this even holds if PCH secret keys that would allow to compute a collision are leaked. More importantly, the properties insider and outsider collision resistance guarantee that only someone in possession of a secret key (trapdoor) whose attributes satisfy the access policy used upon computing the hash is able to perform editing.

6 Conclusion

We tackle the problem of rewriting objects in blockchains in a way, flexible enough for real-world needs regarding the granularity of who can perform such an operation. With our challenging goal, to realize this functionality entirely by means of cryptography, in mind, we introduce the notion of policy-based chameleon-hashes (PCHs). This notion generalizes chameleon-hashes in the sense that hashing additionally takes a policy as input and collision finding is much more fine-grained than in existing chameleon-hashing, i.e., a collision can only be found by users satisfying the policy used during hashing. We rigorously model the security and present a generic construction of this primitive from a CP-ABE scheme and a modified CHET, and provide first implementation results.

Acknowledgments. We thank the anonymous reviewers for their helpful comments. Daniel Slamanig and Christoph Striecks have been supported by EU's Horizon 2020 ECSEL Joint Undertaking project SECREDAS under grant agreement No 783119.

References

ABB$^+$18. Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. *CoRR*, 2018.

AC17. Shashank Agrawal and Melissa Chase. FAME: Fast attribute-based message encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 665–682. ACM Press, October / November 2017.

ACdT05. Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS 2005*, volume 3679 of *LNCS*, pages 159–177. Springer, Heidelberg, September 2005.

AdM04. Giuseppe Ateniese and Breno de Medeiros. On the key exposure problem in chameleon hashes. In *Security in Communication Networks, 4th International Conference, SCN 2004*, 2004.

AGM$^+$13. Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.

AMVA17. Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton R. Andrade. Redactable blockchain - or - rewriting history in bitcoin and friends. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017*, 2017.

BCD$^+$17. Michael Till Beck, Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Practical strongly invisible and strongly accountable sanitizable signatures. In Josef Pieprzyk and Suriadi Suriadi, editors, *ACISP 17, Part I*, volume 10342 of *LNCS*, pages 437–452. Springer, Heidelberg, July 2017.

BD18. Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, 2018.

Bei96. Amos Beimel. Secure schemes for secret sharing and key distribution. In *PhD thesis*, 1996.

BFF$^+$09. Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 317–336. Springer, Heidelberg, March 2009.

BLS03. Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 257–267. Springer, Heidelberg, September 2003.

BMM17. Christian Badertscher, Christian Matt, and Ueli Maurer. Strengthening access control encryption. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 502–532. Springer, Heidelberg, December 2017.

BN06. Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, Heidelberg, August 2006.

BNPS03. Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

BSW07. John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society Press, May 2007.

CDK$^+$17. Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures. In *Public-Key Cryptography - PKC 2017.*, 2017.

CGW15. Jie Chen, Romain Gay, and Hoeteck Wee. Improved dual system ABE in prime-order groups via predicate encodings. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 595–624. Springer, Heidelberg, April 2015.

Cha07. Melissa Chase. Multi-authority attribute based encryption. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 515–534. Springer, Heidelberg, February 2007.

CS03. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

CZK04. Xiaofeng Chen, Fangguo Zhang, and Kwangjo Kim. Chameleon hashing without key exposure. In *ISC*, pages 87–98, 2004.

DHO16. Ivan Damgård, Helene Haagh, and Claudio Orlandi. Access control encryption: Enforcing information flow with cryptography. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 547–576. Springer, Heidelberg, October / November 2016.

DMT19. Dominic Deuber, Bernardo Magri, and Sri Aravinda Krishnan Thyagarajan. Redactable blockchain in the permissionless setting. In *IEEE Symposium on Security and Privacy (SP)*, pages 645–659, 2019.

FFW13. Anna Lisa Ferrara, Georg Fuchsbauer, and Bogdan Warinschi. Cryptographically enforced RBAC. In *2013 IEEE 26th Computer Security Foundations Symposium*, pages 115–129, 2013.

FO99. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, August 1999.

GGH$^+$13. Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 479–499. Springer, Heidelberg, August 2013.

GPSW06. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309.

GVW13. Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.

HFK$^+$14. Vincent C. Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (abac) definition and considerations. Technical report, NIST Special Publication 800-162, 2014.

KB16. Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 543–571. Springer, Heidelberg, August 2016.

KPSS18. Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Chameleon-hashes with dual long-term trapdoors and their applications. In *AfricaCrypt*, 2018.

KR00. Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA*, 2000.

KW93. Mauricio Karchmer and Avi Wigderson. On span programs. In *Proceedings of Structures in Complexity Theory*, pages 102–111, 1993.

KW17. Sam Kim and David J. Wu. Access control encryption for general policies from standard assumptions. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 471–501. Springer, Heidelberg, December 2017.

KW18. Venkata Koppula and Brent Waters. Realizing chosen ciphertext security generically in attribute-based encryption and predicate encryption. Cryptology ePrint Archive, Report 2018/847, 2018. https://eprint.iacr.org/2018/847.

LSP82. Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

LW10. Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. Cryptology ePrint Archive, Report 2010/351, 2010. http://eprint.iacr.org/2010/351.

LW11. Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 568–588. Springer, Heidelberg, May 2011.

Mer89. Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO '89*, 1989.

MSS16. Alfred Menezes, Palash Sarkar, and Shashank Singh. Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. In *Paradigms in Cryptology - Mycrypt 2016.*, 2016.

PDC17. Ivan Puddu, Alexandra Dmitrienko, and Srdjan Capkun. μchain: How to forget without hard forks. *IACR Cryptology ePrint Archive*, 2017:106, 2017.

Wat11. Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 53–70. Springer, Heidelberg, March 2011.

A Detailed Evaluation

We now present a more detailed evaluation of our implementation. In particular, Figure 6 provides a first overview (average), while Table 2 contains the bare numbers for our implementation. We omit setup in Figure 6, as this procedure is only executed once. The corresponding boxplots for each algorithm are given in Figure 7-Figure 9.

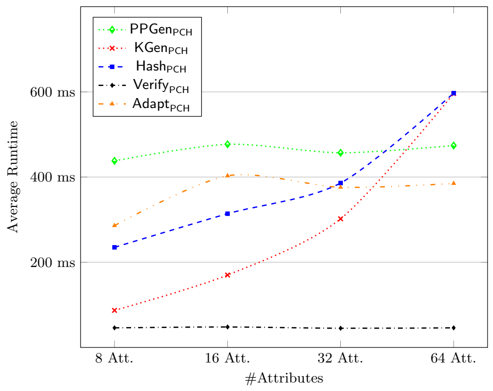
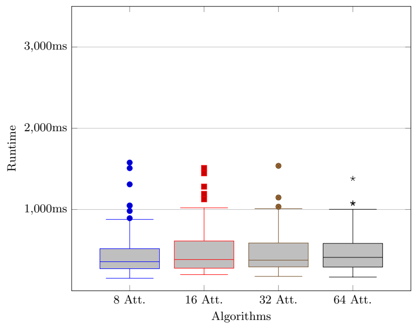
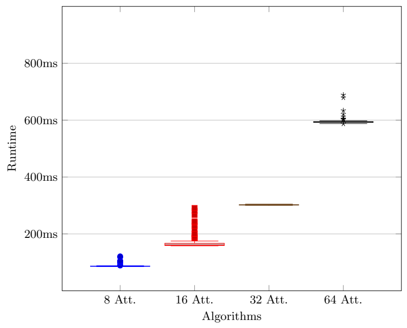


Fig. 6: Performance evaluation (runtimes in ms) with an IND-CPA-secure variant of FAME

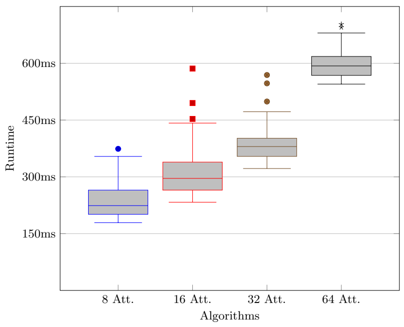


(a) Setup

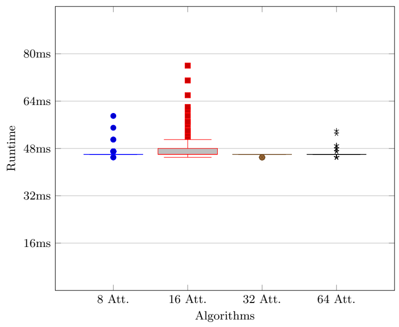


(b) Kgen

Fig. 7: Box-Plots of the measurements in ms for our protocol



(a) Hash



(b) Verify

Fig. 8: Box-Plots of the measurements in ms for our protocol

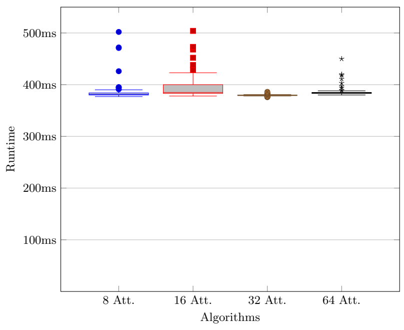


Fig. 9: Box-Plots of the Adapt measurements in ms for our protocol

This shows that setup is constant, regardless how many attributes are involved, which is also true for verification and adaption. However, things change for hashing and key-generation, which grow linearly with the amount of attributes. This, however, is easily explainable due to the used ABE scheme.

B Security Properties of CHs and CHETs

We now present the formal security definitions for CHs and CHETs.

B.1 Security of CHs

Below we present security notions of chameleon-hashes.

Strong Indistinguishability. Indistinguishability requires that the randomnesses r does not reveal if it was obtained through $\mathsf{Hash}_{\mathsf{CH}}$ or $\mathsf{Adapt}_{\mathsf{CH}}$. The messages are chosen by the adversary. We, however, relax the perfect indistinguishability definition of Brzuska et al. [BFF+09] to a computational version, which is enough for most use-cases, including ours. Compared to the existing definition in [BCD+17,CDK+17,KPSS18], the adversary is also allowed to know the secret key sk, but cannot generate it.

```
$\mathbf{Exp}^{\mathsf{SInd}}_{\mathcal{A},\mathsf{CH}}(\kappa)$
  $\mathsf{PP}_{\mathsf{ch}} \leftarrow \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$
  $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \leftarrow \mathsf{KGen}_{\mathsf{CH}}(\mathsf{PP}_{\mathsf{ch}})$
  $b \leftarrow \{0,1\}$
  $a \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}},\cdot,\cdot,b)}(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}})$
    where $\mathsf{HashOrAdapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}},\cdot,\cdot,b)$ on input $m, m'$:
      $(h, r) \leftarrow \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m')$
      $(h', r') \leftarrow \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m)$
      $r'' \leftarrow \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r', h')$
      if $b = 0$:
        return $(h, r)$
      if $b = 1$:
        return $(h', r'')$
  return 1, if $a = b$
  return 0
```

Fig. 10: CH Strong Indistinguishability

Note that we need to implicitly return $\perp$ in the $\mathsf{HashOrAdapt}$ oracle (in case of an error), as the adversary may try to enter a message $m \notin \mathcal{M}$, even if $\mathcal{M} = \{0,1\}^*$, which makes the algorithm output $\perp$. If we would not do this, the adversary could trivially decide which case it sees. For similar reasons these checks are also included in other definitions.

Definition 10 (Strong Indistinguishability). *We define the advantage of an adversary $\mathcal{A}$ in the* Ind *experiment* $\mathbf{Exp}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$ *as*

$$\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{CH}}(\kappa) := \left|\Pr\left[\mathbf{Exp}^{\mathsf{SInd}}_{\mathcal{A},\mathsf{CH}}(\kappa) = 1\right] - 1/2\right|.$$

We say a CH *scheme is indistinguishable, if* $\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{CH}}(\kappa)$ *is a negligible function in* κ *for all PPT adversaries* $\mathcal{A}$.

Collision Resistance. Collision resistance says, that even if an adversary has access to an adapt oracle, it cannot find any collisions for messages other than the ones queried to the adapt oracle. Note, this is an even stronger definition than key-exposure freeness [AdM04]: key-exposure freeness only requires that one cannot find a collision for some new "tag", i.e., for some auxiliary value for which the adversary has never seen a collision.

```
$\mathbf{Exp}^{\mathsf{CR}}_{\mathcal{A},\mathsf{CH}}(\kappa)$
  $\mathsf{pp_{ch}} \leftarrow \mathsf{PPGen_{CH}}(1^\kappa)$
  $(\mathsf{sk_{CH}}, \mathsf{pk_{CH}}) \leftarrow \mathsf{KGen_{CH}}(\mathsf{pp_{ch}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{Adapt}'_{\mathsf{CH}}(\mathsf{sk_{CH}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk_{CH}})$
    where $\mathsf{Adapt}'_{\mathsf{CH}}(\mathsf{sk_{CH}}, \cdot, \cdot, \cdot, \cdot)$ on input $m, m', r, h$:
      $r' \leftarrow \mathsf{Adapt_{CH}}(\mathsf{sk_{CH}}, m, m', r, h)$
      return $\bot$, if $r' = \bot$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m, m'\}$
      return $r'$
  return 1, if $\mathsf{Verify_{CH}}(\mathsf{pk_{CH}}, m^*, h^*, r^*) = \mathsf{Verify_{CH}}(\mathsf{pk_{CH}}, m'^*,$
    $r'^*, h^*) = 1 \ \wedge \ m'^* \notin \mathcal{Q} \ \wedge \ m^* \neq m'^*$
  return 0
```

Fig. 11: CH Collision Resistance

Definition 11 (Collision Resistance). *We define the advantage of an adversary $\mathcal{A}$ in the* CR *experiment* $\mathbf{Exp}^{\mathsf{CR}}_{\mathcal{A},\mathsf{CH}}(\kappa)$ *as*

$$\mathbf{Adv}^{\mathsf{CR}}_{\mathcal{A},\mathsf{CH}}(\kappa) := \Pr\left[\mathbf{Exp}^{\mathsf{CR}}_{\mathcal{A},\mathsf{CH}}(\kappa) = 1\right].$$

We say that a CH *scheme is collision resistant, if the function* $\mathbf{Adv}^{\mathsf{CR}}_{\mathcal{A},\mathsf{CH}}(\kappa)$ *is a negligible function in κ for all PPT adversaries $\mathcal{A}$.*

Weak Collision Resistance. Weak collision resistance is essentially the same as collision resistance, but the adversary does not get access to a collision-finding oracle. Thus, schemes which suffer from the key-exposure problem fall under this category.

```
$\mathbf{Exp}^{\mathsf{wCR}}_{\mathcal{A},\mathsf{CH}}(\kappa)$
    $\mathsf{PP_{ch}} \leftarrow \mathsf{PPGen_{CH}}(1^\kappa)$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen_{CH}}(\mathsf{PP_{ch}})$
    $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}(\mathsf{pk})$
    return 1, if $\mathsf{Verify}(\mathsf{pk}, m^*, h^*, r^*) = \mathsf{Verify}(\mathsf{pk}, m'^*, h^*, r'^*) = 1 \wedge$
        $m^* \neq m'^*$
    return 0
```

Fig. 12: CH Weak Collision resistance

Definition 12 (Weak Collision Resistance). *We define the advantage of an adversary $\mathcal{A}$ in the* wCR *experiment* $\mathbf{Exp}^{\mathsf{wCR}}_{\mathcal{A},\mathsf{CH}}(\kappa)$ *as*

$$\mathbf{Adv}^{\mathsf{wCR}}_{\mathcal{A},\mathsf{CH}}(\kappa) := \Pr\left[\mathbf{Exp}^{\mathsf{wCR}}_{\mathcal{A},\mathsf{CH}}(\kappa) = 1\right].$$

We say that a CH *scheme is weakly collision resistant, if the function* $\mathbf{Adv}^{\mathsf{wCR}}_{\mathcal{A},\mathsf{CH}}(\kappa)$ *is a negligible function in κ for all PPT adversaries $\mathcal{A}$.*

Uniqueness. Uniqueness requires that it be hard to come up with two different randomness values for the same message m^* such that the hashes are equal, for the same adversarially chosen pk^*.

```
$\mathbf{Exp}^{\mathsf{Uniq}}_{\mathcal{A},\mathsf{CH}}(\kappa)$
    $\mathsf{PP_{ch}} \leftarrow \mathsf{PPGen_{CH}}(1^\kappa)$
    $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*) \leftarrow \mathcal{A}(\mathsf{PP_{ch}})$
    return 1, if $\mathsf{Verify}(\mathsf{pk}^*, m^*, h^*, r^*) = \mathsf{Verify}(\mathsf{pk}^*, m^*, h^*, r'^*) = 1$
        $\wedge\ r^* \neq r'^*$
    return 0
```

Fig. 13: CH Uniqueness

Definition 13 (Uniqueness). *We define the advantage of an adversary $\mathcal{A}$ in the* Uniqu *experiment* $\mathbf{Exp}^{\mathsf{Uniq}}_{\mathcal{A},\mathsf{CH}}(\kappa)$ *as*

$$\mathbf{Adv}^{\mathsf{Uniq}}_{\mathcal{A},\mathsf{CH}}(\kappa) := \Pr\left[\mathbf{Exp}^{\mathsf{Uniq}}_{\mathcal{A},\mathsf{CH}}(\kappa) = 1\right].$$

We say that a CH *scheme is unique, if the function* $\mathbf{Adv}^{\mathsf{Uniq}}_{\mathcal{A},\mathsf{CH}}(\kappa)$ *is a negligible function in* κ *for all PPT adversaries* $\mathcal{A}$*.*

Definition 14 (Secure Chameleon-Hashes). *We call a chameleon-hash* CH *(weakly) secure, if it is correct, indistinguishable, and (weakly) collision resistant.*

Note that we do not require the uniqueness property [CDK+17] in the context of this paper.

B.2 Instantiation of a Secure CH.

We recall a construction from [CDK+17] in Scheme 6 after recalling an RSA key generator.

RSA Key-Generator. Let $(N, p, q, e, d) \leftarrow \mathsf{RSAKGen}(1^\kappa)$ be an instance generator which returns an RSA modulus $N = pq$, where p and q are distinct primes, $e > 1$ an integer co-prime to $\varphi(n)$, and $de \equiv 1 \bmod \varphi(n)$. We require that $\mathsf{RSAKGen}$ always outputs moduli with the same bit-length, based on κ.

$\underline{\mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)}$: On input a security parameter κ it outputs the public parameters $\textsc{pp} \leftarrow (1^\kappa, e)$, where e is prime and $e > N'$ with $N' = \max_r\{N \in \mathbb{N} : (N, \cdot, \cdot, \cdot, \cdot) \leftarrow \mathsf{RSAKGen}(1^\kappa; r)\}$.

$\underline{\mathsf{KGen}_{\mathsf{CH}}(\textsc{pp})}$: On input $\textsc{pp} = (1^\kappa, e)$ run $(N, p, q, \cdot, \cdot) \leftarrow \mathsf{RSAKGen}(1^\kappa)$, choose a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_N^*$ (modeled as a random-oracle), compute d s.t. $ed \equiv 1 \bmod \varphi(N)$, set $\mathsf{sk} \leftarrow d$, $\mathsf{pk} \leftarrow (N, H)$, and return $(\mathsf{sk}, \mathsf{pk})$.

$\underline{\mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}, m)}$: On input a public key $\mathsf{pk} = (N, H)$ and a message m, choose $r \leftarrow_r \mathbb{Z}_N^*$, compute $h \leftarrow H(m)r^e \mod N$ and output (h, r).

$\underline{\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}, m, h, r)}$: On input public key $\mathsf{pk} = (N, H)$, a message m, a hash h, and a randomness $r \in \mathbb{Z}_N^*$, it computes $h' \leftarrow H(m)r^e \mod N$ and outputs 1 if $h' = h$ and 0 otherwise.

$\underline{\mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}, m, m', h, r)}$: On input a secret key $\mathsf{sk} = d$, messages m and m', a hash h, and randomness values r and r', the adaptation algorithm outputs $\perp$ if $\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}, m, h, r) \neq 1$. Otherwise, let $x \leftarrow H(m)$, $x' \leftarrow H(m')$, $y \leftarrow xr^e \mod N$ and return $r' \leftarrow (y(x'^{-1}))^d \mod N$.

Scheme 6: RSA-based Chameleon-Hash

In [CDK+17] Camenisch et al. show that the chameleon-hash in Scheme 6 is secure under the one-more RSA inversion assumption [BNPS03] in the random oracle model (ROM) [BR93].

Theorem 5 (cf. [CDK+17]). *If the one-more RSA-inversion assumption holds, then the chameleon-hash in Scheme 6 is correct, indistinguishable, collision resistant (and unique) in the random oracle model.*

We argue that the CH restated in Scheme 6 remains secure in our strengthened model. All properties, but strong indistinguishability, have already been proven by Camenisch et al. [CDK+17]. Thus, it remains to prove strong indistinguishability.

Proof. We prove strong indistinguishability by a sequence of games.

Game 0: The original strong indistinguishability game in the case $b = 0$.
Game 1: As Game 0, but we now make the transition to $b = 1$.
Transition - Game 0 → Game 1: As there is exactly one secret key (up to the group order, which can be ignored), which makes adaption work correctly, which we explicitly check, while r is always chosen randomly, the distributions are exactly equal and thus $|\Pr[S_0] - \Pr[S_1]| = 0$ follows.

In Game 0, we simulate the first distribution of the strong indistinguishability game, in Game 1 the second one. Both games are indistinguishable which concludes the proof.

B.3 Security of CHETs.

Subsequently we restate the security properties of CHET s from [CDK+17], where we adapt the notation to ours and also strengthen indistinguishability to what we call strong indistinguishability and private collision resistance to what we call strong private collision resistance. We note that [CDK+17] additionally define uniqueness of CHET schemes. However, in our case this notion is not required.

Strong Indistinguishability. Strong Indistinguishability requires that it be intractable for outsiders to distinguish whether a given randomness corresponds to an output of $\mathsf{Hash}_{\mathsf{CHET}}$ or $\mathsf{Adapt}_{\mathsf{CHET}}$. Note that, when compared to the definitions in [CDK+17,BCD+17], the adversary additionally receives the secret key sk, but cannot generate it.

Definition 15 (Strong Indistinguishability). *We define the advantage of an adversary $\mathcal{A}$ in the strong indistinguishability experiment as*

$$\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{CHET}}(\kappa) := \left|\Pr\left[\mathbf{Exp}^{\mathsf{SInd}}_{\mathcal{A},\mathsf{CHET}}(\kappa) = 1\right] - 1/2\right|.$$

We say that a CHET scheme is strongly indistinguishable, if $\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$ is a negligible function in κ for all PPT adversaries.

```
$\mathbf{Exp}^{\mathsf{SInd}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$
  $\mathsf{pp_{CHET}} \leftarrow \mathsf{PPGen_{CHET}}(1^\kappa)$
  $(\mathsf{sk_{CHET}}, \mathsf{pk_{CHET}}) \leftarrow \mathsf{KGen_{CHET}}(\mathsf{pp_{CHET}})$
  $b \leftarrow_r \{0,1\}$
  $b^* \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt_{CHET}}(\mathsf{sk_{CHET}},\cdot,\cdot,b)}(\mathsf{sk_{CHET}}, \mathsf{pk_{CHET}})$
    where $\mathsf{HashOrAdapt_{CHET}}(\mathsf{sk},\cdot,\cdot,b)$ on input $m, m'$:
      let $(h_0, r_0, \mathsf{etd}_0) \leftarrow \mathsf{Hash_{CHET}}(\mathsf{pk_{CHET}}, m')$
      let $(h_1, r_1, \mathsf{etd}_1) \leftarrow \mathsf{Hash_{CHET}}(\mathsf{pk_{CHET}}, m)$
      let $r_1 \leftarrow \mathsf{Adapt_{CHET}}(\mathsf{sk_{CHET}}, \mathsf{etd}_1, m, m', h_1, r_1)$
      return $(h_b, r_b, \mathsf{etd}_b)$
  return $b = b^*$
```

Fig. 14: CHET Strong Indistinguishability

Public Collision Resistance. Public collision resistance grants the adversary access to an $\mathsf{Adapt_{CH}}$ oracle. It requires that it is intractable to produce collisions, other than the ones produced by the $\mathsf{Adapt_{CH}}$ oracle.

Definition 16 (Public Collision Resistance). *We define the advantage of an adversary $\mathcal{A}$ in the public collision resistance experiment as*

$$\mathbf{Adv}^{\mathsf{CRpub}}_{\mathcal{A},\mathsf{CHET}}(\kappa) := \Pr\left[\mathbf{Exp}^{\mathsf{CRpub}}_{\mathcal{A},\mathsf{CHET}}(\kappa) = 1\right].$$

We say that a CHET *scheme is publicly collision resistant, if* $\mathbf{Adv}^{\mathsf{CRpub}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$ *is a negligible function in κ for all PPT adversaries.*

Private Collision Resistance. Private collision resistance requires that it is even intractable for the holder of the secret key sk to find collisions without knowledge of etd.

Definition 17 (Private Collision Resistance). *We define the advantage of an adversary $\mathcal{A}$ in the private collision resistance experiment as*

$$\mathbf{Adv}^{\mathsf{CRpriv}}_{\mathcal{A},\mathsf{CHET}}(\kappa) := \Pr\left[\mathbf{Exp}^{\mathsf{CRpriv}}_{\mathcal{A},\mathsf{CHET}}(\kappa) = 1\right].$$

We say that a CHET *scheme is privately collision resistant, if* $\mathbf{Adv}^{\mathsf{CRpriv}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$ *is a negligible function in κ for all PPT adversaries.*

Strong Private Collision Resistance. Strong private collision resistance requires that it is even intractable for the holder of the secret key

```
$\mathbf{Exp}^{\mathsf{CRpub}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$
  $\mathsf{pp}_{\mathsf{CHET}} \leftarrow \mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)$
  $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \leftarrow \mathsf{KGen}_{\mathsf{CHET}}(\mathsf{pp}_{\mathsf{CHET}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{Adapt}'_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}},\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk})$
    where $\mathsf{Adapt}'_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}},\cdot,\cdot,\cdot,\cdot,\cdot)$ on input $\mathsf{etd}, m, m', h, r$:
      $r' \leftarrow \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{etd}, m, m', h, r)$
      if $r' \neq \perp$, let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m, m'\}$
      return $r'$
  return 1, if $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m^*, h^*, r^*) = 1 \wedge$
    $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m'^*, h^*, r'^*) = 1 \wedge$
    $m'^* \notin \mathcal{Q} \ \wedge \ m^* \neq m'^*$
  return 0
```

Fig. 15: CHET Public Collision Resistance

sk to find collisions without knowledge of etd. Compared to the definition by Camenisch et al. [CDK+17], however, we also allow the adversary to request arbitrary collisions; the ephemeral trapdoor to use is indexed by the handle. This allows for a completely stateless primitive.

Definition 18 (Strong Private Collision Resistance). *We define the advantage of an adversary $\mathcal{A}$ in the strong private collision resistance experiment as*

$$\mathbf{Adv}^{\mathsf{SCRpriv}}_{\mathcal{A},\mathsf{CHET}}(\kappa) := \Pr\left[\mathbf{Exp}^{\mathsf{SCRpriv}}_{\mathcal{A},\mathsf{CHET}}(\kappa) = 1\right].$$

We say that a CHET scheme is strongly privately collision resistant, if $\mathbf{Adv}^{\mathsf{SCRpriv}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$ is a negligible function in κ for all PPT adversaries.

Uniqueness. Uniqueness requires that it be hard to come up with two different randomness values for the same message m^* such that the hashes are equal, for the same adversarially chosen pk^*.

Definition 19 (Uniqueness). *We define the advantage of an adversary $\mathcal{A}$ in the uniqueness experiment as*

$$\mathbf{Adv}^{\mathsf{Uniq}}_{\mathcal{A},\mathsf{CHET}}(\kappa) := \Pr\left[\mathbf{Exp}^{\mathsf{Uniq}}_{\mathcal{A},\mathsf{CHET}}(\kappa) = 1\right].$$

We say that a CHET scheme provides uniqueness, if $\mathbf{Adv}^{\mathsf{Uniq}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$ is a negligible function in κ for all PPT adversaries.

```
$\mathbf{Exp}^{\mathsf{CRpriv}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$
  $\mathsf{PP_{CHET}} \leftarrow \mathsf{PPGen_{CHET}}(1^\kappa)$
  $\mathcal{Q} \leftarrow \emptyset$
  $i \leftarrow 0$
  $(\mathsf{pk}^*, m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{Hash}'_{\mathsf{CHET}}(\cdot,\cdot)}(\mathsf{PP_{CHET}})$
    where $\mathsf{Hash}'_{\mathsf{CHET}}$ on input $\mathsf{pk}$, $m$:
      $(h, r, \mathsf{etd}) \leftarrow \mathsf{Hash_{CHET}}(\mathsf{pk}, m)$
      return $\perp$, if $r = \perp$
      $i \leftarrow i + 1$
      let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}, h, m, \mathsf{etd}, i)\}$
      return $(h, r)$
  return 1, if $\mathsf{Verify_{CHET}}(\mathsf{pk}^*, m^*, r^*, h^*) = 1 \wedge$
    $\mathsf{Verify_{CHET}}(\mathsf{pk}^*, m'^*, r'^*, h^*) = 1 \ \wedge\ m^* \neq m'^* \ \wedge$
    $(\mathsf{pk}^*, h^*, m^*, \cdot, \cdot) \notin \mathcal{Q} \ \wedge\ (\mathsf{pk}^*, h^*, \cdot, \cdot, \cdot) \in \mathcal{Q}$
  return 0
```

Fig. 16: CHET Private Collision Resistance

Definition 20 (Secure CHET). *We say that a* CHET *is secure if it is correct and provides strong indistinguishability, public collision resistance, and strong private collision resistance. We say that a* CHET *is weakly secure it is correct and provides strong indistinguishability, public collision resistance, and private collision resistance.*

B.4 Proofs for CHET

Theorem 6. *If* CH *is collision resistant, then the construction of a* CHET *given in Scheme 2 offers strong private collision resistance.*

Proof. Assume an adversary $\mathcal{A}$ who can break strong private collision resistance. We can then construct an adversary $\mathcal{B}$ which breaks the collision resistance of CH.

In particular, the reduction works as follows. $\mathcal{B}$ receives $\mathsf{pk_{ch}} = \mathsf{pk_{ch}}'$ as its own challenge. Note that $\mathsf{PP_{ch}}$ is implicit in $\mathsf{pk_{ch}}$ and let q be an upper bound on the number of queries to the $\mathsf{Hash}'_{\mathsf{CHET}}$-oracle. Our reduction draws a random index $i \leftarrow \{1, 2, \ldots, q\}$. It then initializes the adversary $\mathcal{A}$ by supplying $\mathsf{PP_{chet}} = \mathsf{PP_{ch}}$. For every adaption query $j \neq i$, $\mathcal{B}$ generates a new key pair for a CH and proceeds honestly, storing the corresponding secret and public keys. This does not change the view of the adversary so far. For the ith query, however, $\mathcal{B}$ embeds $\mathsf{pk_{ch}}'$ as the

```
$\mathbf{Exp}^{\mathsf{SCRpriv}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$
  $\mathsf{pp}_{\mathsf{CHET}} \leftarrow \mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)$
  $\mathcal{Q} \leftarrow \emptyset$
  $i \leftarrow 0$
  $(\mathsf{pk}^*, m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{Hash}'_{\mathsf{CHET}}(\cdot,\cdot),\mathsf{Adapt}'_{\mathsf{CHET}}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pp}_{\mathsf{CHET}})$
    where $\mathsf{Hash}'_{\mathsf{CHET}}$ on input $\mathsf{pk}$, $m$:
      $(h, r, \mathsf{etd}) \leftarrow \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}, m)$
      return $\perp$, if $r = \perp$
      $i \leftarrow i + 1$
      let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}, h, m, \mathsf{etd}, i)\}$
      return $(h, r)$
    and $\mathsf{Adapt}'_{\mathsf{CHET}}$ on input $\mathsf{sk}$, $h$, $r$, $m$, $m'$, $i$:
      return $\perp$, if $(\mathsf{pk}, h', m'', \mathsf{etd}, i) \notin \mathcal{Q}$ for some $h'$, $m''$, $\mathsf{etd}$, $\mathsf{pk}$
      $r' \leftarrow \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}, \mathsf{etd}, m, m', h, r)$
      if $r' \neq \perp$, let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}, h', m, \mathsf{etd}, i), (\mathsf{pk}, h', m', \mathsf{etd}, i)\}$
      return $r'$
  return 1, if $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, r^*, h^*) = 1 \wedge$
    $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}^*, m'^*, r'^*, h^*) = 1 \ \wedge\ m^* \neq m'^* \ \wedge$
    $(\mathsf{pk}^*, h^*, m^*, \cdot, \cdot) \notin \mathcal{Q} \ \wedge\ (\mathsf{pk}^*, h^*, \cdot, \cdot, \cdot) \in \mathcal{Q}$
  return 0
```

Fig. 17: CHET Strong Private Collision Resistance

public key of the second CH and for the respective query to the collision finding oracle $\mathsf{Adapt}'_{\mathsf{CHET}}$, $\mathcal{B}$ proceeds as follows if a collision is to be found for the embedded challenge: $\mathcal{B}$ uses its own oracle to find the collisions for h^2, while h^1 can be calculated honestly, as for that the adversary supplies the secret key. If this case happens can easily be identified due to the supplied handle. Note, we always check whether adaption was successful. Clearly, this simulation does not change the view of the adversary. Then, after the adversary $\mathcal{A}$ outputs $(\mathsf{pk}^*, m^*, r^*, m'^*, r'^*, h^*)$, where $r^* = (r_1^*, r_2^*)$, $r'^* = (r_1'^*, r_2'^*)$ and $h^* = (h_1^*, h_2^*, \mathsf{pk}^*, \mathsf{pk_{ch}}'')$, $\mathcal{B}$ can return $((m^*, \mathsf{pk}^*, \mathsf{pk_{ch}}'), r_1^*, (m'^*, \mathsf{pk}^*, \mathsf{pk_{ch}}'), r_1'^*, h_1^*)$ as its own forgery attempt, if $\mathsf{pk_{ch}}' = \mathsf{pk_{ch}}''$. In all other cases, $\mathcal{B}$ must abort. Assuming that $\mathsf{pk_{ch}}' = \mathsf{pk_{ch}}''$ holds, we already know that m^* or pk^* must be "fresh", and thus $(m^*, \mathsf{pk}^*, \mathsf{pk_{ch}}')$ is fresh as well. This concludes the proof.

```
$\mathbf{Exp}^{\mathsf{Uniq}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$
  $\textsc{pp} \leftarrow \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$
  $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*) \leftarrow \mathcal{A}(\textsc{pp})$
  return 1, if $\mathsf{Verify}(\mathsf{pk}^*, m^*, h^*, r^*) = \mathsf{Verify}(\mathsf{pk}^*, m^*, h^*, r'^*) = 1$
    $\wedge\ r^* \neq r'^*$
  return 0
```

Fig. 18: CHET Uniqueness

C A Weakly Secure CHET

Camenisch et al. in [CDK+17] provide, among others, a generic construction of a CHET by combining two chameleon-hashes, both requiring collision resistance even in presence of a collision-finding oracle. The keys for the second chameleon-hash are drawn freshly for each new hash. Thus, the secret key for the second chameleon-hash is the ephemeral trapdoor. In this section, we show a black-box construction along the same lines, but we prove that the second chameleon-hash is not required to be collision resistant w.r.t. to the collision-finding oracle. Thus, the second chameleon-hash can be constructed from simpler primitives and in particular leads to more efficient instantiations. In more detail, the second chameleon-hash can even suffer from the key-exposure problem [AdM04,CZK04], while the final construction remains secure. In particular, the first chameleon-hash (CH_s) needs to be fully secure while the second one, i.e., CH_w, is only required to be *weakly* secure.

Theorem 7. *If* CH_s *is secure,* CH_w *is weakly secure, then Scheme 7 is a weakly secure* CHET.

Proof. We prove each property on its own.

Correctness. Follows by the correctness from the underlying chameleon-hashes.

Strong Indistinguishability. We prove strong indistinguishability by a sequence of games. This follows by a simple argument, as given below.

Game 0. The original strong indistinguishability game, where $b = 0$.

Game 1. As Game 0, but instead of calculating the hash h_s as in the game, directly hash.

$\underline{\mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)}$: On input a security parameter κ, let $\mathsf{PP}_{\mathsf{ch}s} \leftarrow \mathsf{CH}_s.\mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$ and $\mathsf{PP}_{\mathsf{ch}w} \leftarrow \mathsf{CH}_w.\mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$. Return $\mathsf{PP}_{\mathsf{ch}} \leftarrow (\mathsf{PP}_{\mathsf{ch}s}, \mathsf{PP}_{\mathsf{ch}w})$.

$\underline{\mathsf{KGen}_{\mathsf{CH}}(\mathsf{PP}_{\mathsf{ch}})}$: On input $\mathsf{PP}_{\mathsf{ch}} = (\mathsf{PP}_{\mathsf{ch}s}, \mathsf{PP}_{\mathsf{ch}w})$ run and return $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{CH}_s.\mathsf{KGen}_{\mathsf{CH}}(\mathsf{PP}_{\mathsf{ch}s})$.

$\underline{\mathsf{Hash}(\mathsf{pk}, m)}$: On input of pk and m, let $(\mathsf{etd}, \mathsf{pk}_w) \leftarrow \mathsf{CH}_w.\mathsf{KGen}_{\mathsf{CH}}(\mathsf{PP}_{\mathsf{ch}w})$. Let $(h_s, r_s) \leftarrow \mathsf{CH}_s.\mathsf{Hash}(\mathsf{pk}, m)$ and $(h_w, r_w) \leftarrow \mathsf{CH}_w.\mathsf{Hash}(\mathsf{pk}_w, m)$. Return $((h_s, h_w, \mathsf{pk}_w), (r_s, r_w))$.

$\underline{\mathsf{Verify}(\mathsf{pk}, m, h, r)}$: On input of pk, m, $h = (h_s, h_w, \mathsf{pk}_w)$ and $r = (r_s, r_w)$, return 1, if $\mathsf{CH}_s.\mathsf{Verify}(\mathsf{pk}, m, h_s, r_s) = 1$ and $\mathsf{CH}_w.\mathsf{Verify}(\mathsf{pk}_w, m, h_w, r_w) = 1$. Otherwise, return 0.

$\underline{\mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}, \mathsf{etd}, m, m', h, r)}$: On input a secret key sk, messages m and m', a hash $h = (h_s, h_w, \mathsf{pk}_w)$ and $r = (r_s, r_w)$, first check that $\mathsf{Verify}(\mathsf{pk}, m, h, r) = 1$. Otherwise, return $\perp$. Let $r'_s \leftarrow \mathsf{CH}_s.\mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}, m, m', r_s, h_s)$ and $r'_w \leftarrow \mathsf{CH}_s.\mathsf{Adapt}_{\mathsf{CH}}(\mathsf{etd}, m, m', r_w, h_w)$. Return (r'_s, r'_w).

Scheme 7: Construction of a Weakly Secure CHET

We claim that Game 0 and Game 1 are strongly indistinguishable under the strong indistinguishability of the chameleon-hashes. More formally, assume that the adversary can distinguish this hop. We can then construct an adversary $\mathcal{B}$ which breaks the indistinguishability of CH_s. In particular, the reduction works as follows. $\mathcal{B}$ receives pk_c and sk_c as its own challenge, passing them through to $\mathcal{A}$, and proceeds as in the prior hop, with the exception that it uses the HashOrAdapt oracle to generate h_1. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. Clearly, the simulation is perfect from $\mathcal{A}$'s point of view. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{CHs\text{-}ind}}(\kappa)$ follows.

Game 2. As Game 1, but instead of calculating the hash h_w as in the game, directly hash.

We claim that Game 1 and Game 2 are strongly indistinguishable under the strong indistinguishability of the chameleon-hashes. More formally, assume that the adversary can distinguish this hop. We can then construct an adversary $\mathcal{B}$ which breaks the indistinguishability of the chameleon-hashes. In particular, the reduction works exactly as before. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{CHw\text{-}ind}}(\kappa)$ follows.

We are now in the case $b = 1$. As each hop changes the view only negligibly, strong indistinguishability is proven. As each hop only changes the view of the adversary negligibly, this proves that our construction is strongly indistinguishable.

Public Collision Resistance. Assume, towards contradiction, that our scheme is not publicly collision resistant. We can then construct an adversary $\mathcal{B}$ which breaks the collision resistance of CH_s. In particular, the parameter received from $\mathcal{B}$'s own challenger are embedded in $\mathsf{PP}_{\mathsf{ch}s}$, while $\mathsf{PP}_{\mathsf{ch}w}$ is generated honestly. Likewise, $\mathcal{B}$ receives the pk of CH_s to forge. The pk for CH_w is generated honestly. All values are then given to $\mathcal{A}$. Then, for every query to the adaption oracle, $\mathcal{B}$ asks its own oracle to generate the collision for CH_s. The collision for CH_w is generated honestly. Then, $r' = (r'_s, r'_w)$ is returned to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs $(m^*, r^*, m'^*, r'^*, h^*)$. As, by assumption, $\mathcal{B}$ has never asks its own oracle to generate a collision for m^* and m'^*, $\mathcal{B}$ can return $(m^*, r^*_s, m'^*, r'^*_s, h^*)$ as its own forgery of CH_s. The reduction is tight, i.e., $\nu_{\mathsf{CHs\text{-}collres}}(\kappa)$.

Private Collision Resistance. Assume, towards contradiction, that our scheme is not privately collision resistant. We can then construct an adversary $\mathcal{B}$ which breaks the collision resistance of CH_w. In particular, the parameter received from $\mathcal{B}$'s own challenger are embedded in $\mathsf{PP}_{\mathsf{ch}w}$, while $\mathsf{PP}_{\mathsf{ch}s}$ are generated honestly. Likewise, the reduction $\mathcal{B}$ receives pk_w. All values, but pk_w, are given to $\mathcal{A}$. $\mathcal{A}$ will then output pk^*. For the next step, let q be an upper bound on the queries made to the hashing oracle. The reduction draws a random index $i \in \{1, 2, \ldots q\}$. At the ith query, $\mathcal{B}$ hashes with pk_w and returns all values to $\mathcal{A}$. For all other queries, the reduction draws pk_w honestly. So far, the simulation is perfect from $\mathcal{A}$'s point of view. Eventually, $\mathcal{A}$ outputs $(m^*, r^*, m'^*, r'^*, h^*)$. Assuming that the hash value h^* was returned from the hashing oracle, a collision for pk_w was found. Thus, $\mathcal{B}$ can return $(m^*, r^*_w, m'^*, r'^*_w, h^*_w)$ as its own forgery, where $r'^* = (r'^*_s, r'^*_w)$, $h^* = (h^*_s, h^*_w)$, and $r^* = (r^*_s, r^*_w)$. As, however, $\mathcal{B}$ needs to guess where the adversary finds the collision, we have a loss of $q\nu_{\mathsf{CHw\text{-}collres}}(\kappa)$.

Uniqueness. We prove uniqueness using the following sequence of games, where we let S_i denote the event that the adversary wins Game i.

Game 0. The original uniqueness game.

Game 1. As Game 0, but we abort if the adversary output $r^* = (r^*_s, r^*_w)$ and $r'^* = (r'^*_s, r'^*_w)$ such that $r'^*_s \neq r^*_s$ but the winning conditions are met.

We claim that Game 0 and Game 1 are indistinguishable under the uniqueness of CH_s. Both games proceed identically, unless the abort event $\mathsf{Pr}[E]$ happens, i.e., we have $|\mathsf{Pr}[S_0] - \mathsf{Pr}[S_1]| \leq \mathsf{Pr}[E]$. We bound $\mathsf{Pr}[E]$ by presenting a reduction $\mathcal{B}$, which works as follows. It receives $\mathsf{PP}_{\mathsf{ch}s}$ and generates $\mathsf{PP}_{\mathsf{ch}w}$ honestly. Both are given to the adversary. Once the adversary $\mathcal{A}$ outputs $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*)$, where $\mathsf{pk}^* = (\mathsf{pk}^*_s, \mathsf{pk}^*_w)$ and $h^* = (h^*_s, h^*_w)$, while r^* and r'^* are as before. $\mathcal{B}$ can simply return $(\mathsf{pk}^*_s, m^*, r^*_s, r'^*_s, h^*_s)$ as its own forgery. That is, $\mathsf{Pr}[E] \leq \nu_{\mathsf{CHs\text{-}uni}}(\kappa)$.

Game 2. As Game 1, but we abort if the adversary outputs $r^* = (r_s^*, r_w^*)$ and $r'^* = (r_s'^*, r_w'^*)$ such that $r_w'^* \neq r_w^*$ but the winning conditions are met.

We claim that Game 1 and Game 2 are indistinguishable under the uniqueness of CH_w. Both games proceed identically, unless the abort event $\mathsf{Pr}[E]$ happens, i.e., we have $|\mathsf{Pr}[S_1] - \mathsf{Pr}[S_2]| \leq \mathsf{Pr}[E]$. We bound $\mathsf{Pr}[E]$ by presenting a reduction $\mathcal{B}$, which works as follows. It receives $\mathsf{PP}_{\mathsf{ch}w}$ and generates $\mathsf{PP}_{\mathsf{ch}s}$ honestly. Both are given to the adversary. Once the adversary $\mathcal{A}$ outputs $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*)$, where $\mathsf{pk}^* = (\mathsf{pk}_s^*, \mathsf{pk}_w^*)$ and $h^* = (h_s^*, h_w^*)$, while r^* and r'^* are as before. $\mathcal{B}$ can simply return $(\mathsf{pk}_w^*, m^*, r_w^*, r_w'^*, h_w^*)$ as its own forgery. That is, $|\mathsf{Pr}[E]| \leq \nu_{\mathsf{CHw\text{-}uni}}(\kappa)$ follows.

A Note on Tightness. In Scheme 7 we use both chameleon-hashes as a black-box. This leads to a loss of a polynomial factor q in the proof of the private collision resistance, as the reduction needs to guess for which hash the adversary finds the collision. If, however, one is willing to use the chameleon-hash restated in Scheme 8 as CH_w, one can also get rid of that factor q. Namely, as the challenge of the DL-challenger consists of g^x, one can use random self-reducibility to achieve a tight bound, i.e., by embedding g^{xx_i} for a random x_i for each hash-query. If a collision is found by the adversary, the reduction knows each x_i and can thus still extract x. However, clearly this is now a non black-box reduction.

Remark 3. We stress that Scheme 7 does not provide strong private collision-resistance, i.e., is not useful in our PCH construction.

C.1 Instantiation of a (Weakly) Secure CH.

For completeness we include the initial CH construction by Krawczyk and Rabin [KR00] in Scheme 8, which is a suitable scheme to instantiate CH_w. The description was slightly adjusted to fit out framework and made a bit more general. In that scheme, $(\mathbb{G}, g, q) \leftarrow \mathsf{GGen}(1^\kappa)$ is an instance generator which returns a prime-order, and multiplicatively written, group $\mathbb{G}$ where the discrete logarithm problem is hard, along with a generator g such that $\langle g \rangle = \mathbb{G}$. We stress that this CH is also secure in our strengthened model. In particular, as the randomness r is drawn uniformly, the adapted randomness r' still remains uniform, which is independent from the question whether the adversary knows sk or not. Thus, the distributions are equal, and an adversary gains no advantage by knowing sk.

Collision Resistant Hash Function Families. In the scheme we also require collision resistant hashing, which we formally define below.

Definition 21 (Collision Resistant Hash Functions). *We say that a family* $\{H_{\mathcal{R}}^{k}\}_{k \in \mathcal{K}}$ *of hash functions* $\mathcal{H}_{\mathcal{R}}^{k} : \{0,1\}^* \to \mathcal{R}$ *indexed by key* $k \in \mathcal{K}$ *is collision resistant if*

$$\mathbf{Adv}_{\mathcal{A},H}^{\mathsf{CollRes}}(\kappa) := \Pr\left[\begin{matrix} k \leftarrow \mathcal{K}, \\ (v, v') \leftarrow \mathcal{A}(k) \end{matrix} : \begin{matrix} H_{\mathcal{R}}^{k}(v) = H_{\mathcal{R}}^{k}(v') \wedge \\ v \neq v' \end{matrix}\right]$$

is negligible for all PPT adversaries $\mathcal{A}$*.*

We sometimes write "pick a hash function H" and do not make the key k and domains explicit, if clear from the context.

$\underline{\mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)}$: On input a security parameter κ it outputs the public parameters $(\mathbb{G}, g, q, H)$, where $(\mathbb{G}, g, q) \leftarrow \mathsf{GGen}(1^\kappa)$ and H freshly chosen collision resistant hash function $H : \{0,1\}^* \to \mathbb{Z}_q^*$.

$\underline{\mathsf{KGen}_{\mathsf{CH}}(\mathsf{PP}_{\mathsf{ch}})}$: On input $\mathsf{PP}_{\mathsf{ch}} = (\mathbb{G}, g, q, H)$, draw $x \leftarrow \mathbb{Z}_q$. Let $\mathsf{pk} \leftarrow g^x$. Return $(\mathsf{sk}, \mathsf{pk}) = (x, g^x)$.

$\underline{\mathsf{Hash}(\mathsf{pk}, m)}$: On input a public key $\mathsf{pk} = g^x$ and a message m, choose $r \leftarrow_r \mathbb{Z}_q^*$. Let $h \leftarrow g^{H(m)}\mathsf{pk}^r$. Return (h, r).

$\underline{\mathsf{Verify}(\mathsf{pk}, m, h, r)}$: On input public key $\mathsf{pk} = g^x$, a message m, a hash h, and a randomness $r \in \mathbb{Z}_q^*$, compute $h' \leftarrow g^{H(m)}\mathsf{pk}^r$ and output 1, if $h' = h$ and 0 otherwise.

$\underline{\mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}, m, m', h, r)}$: On input a secret key $\mathsf{sk} = x$, messages m and m', a hash h, and randomness values r and r', the adaptation algorithm outputs $\perp$ if $\mathsf{Verify}(\mathsf{pk}, m, h, r) \neq 1$. Return $r' \leftarrow \frac{H(m)+xr-H(m')}{x}$.

Scheme 8: DL based chameleon-hash

In Scheme 8 we use the fact (shown by Krawczyk and Rabin [KR00]) that applying a collision-resistant hash-function prior to chameleon-hashing can be used to extend the message space of the chameleon-hash. It can also easily be seen that this construction is unique. Moreover, it is obvious that seeing a collision (if not stemming from the collision resistant hash function) allows to easily extract the secret key x by computing $x \leftarrow \frac{H(m)-H(m')}{r'-r}$.

Table 2: Percentiles in ms for our protocol

Alg.	Setup				KGen				Hash				Verf				Adapt			
Att.	8	16	32	64	8	16	32	64	8	16	32	64	8	16	32	64	8	16	32	64
Min.:	154	199	177	169	85	158	300	587	179	233	322	545	45	45	45	45	377	378	376	380
25%:	272	278	294	292	86	159	301	592	201	265	354	568	46	46	46	46	380	383	379	383
Med.:	361	384	377	411	86	161	302	593	224	296	380	594	46	46	46	46	381	385	379	384
75%:	518	613	588	583	87	167	303	595	265	339	402	618	46	48	46	46	384	400	380	385
90%:	723	780	755	865	88	186	303	598	294	393	438	646	47	54	46	47	389	429	381	388
95%:	982	1′021	923	947	99	228	304	605	302	439	469	660	47	59	46	48	395	473	382	397
Max.:	1′578	1′511	1′538	1′381	121	291	305	689	374	751	569	703	59	76	46	54	502	684	386	450
Avg.:	438	477	457	474	87	170	302	596	235	314	386	597	46	48	45	46	286	403	379	385
SD:	264.12	268.56	240.36	244.79	5.67	24.97	1.06	13.92	41.95	75.27	43.31	35.57	1.73	5.27	0.36	1.24	17.96	50.89	1.55	8.98