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

David Derler
DFINITY
david@dfinity.org

Kai Samelin
TÜV Rheinland i-sec GmbH
kaispapers@gmail.com

Daniel Slamanig and Christoph Striecks
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.

I. 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.[1] 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 [36]. 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 [33]). 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 Hyperledger, 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 [3] for instance uses so-called endorsement policies in the form of monotone Boolean formulas, e.g., $(A \text{ AND } D) \text{ OR } C$ to determine which peers are 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

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

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 may be 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)[2] 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.[3]

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 modfiy 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 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, Magri, Venturi, and Andrade [6] 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 [31]. 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.[4]

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) [27], 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 $(A \text{ AND } D) \text{ OR } C$, 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., $\text{user}_1 \text{ OR } \text{user}_2 \text{ OR } \text{user}_3$.

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).

A. 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 [26] and later efficiently instantiated by Bethencourt, Sahai, and Waters [14]. 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. [6]. 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) [16] 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 [1]. Along the lines, we also introduce a novel CHET which is more efficient than the most practical known instantiation proposed by Camenisch et al. [16]. Putting all together, we obtain a very efficient concrete instantiation of a PCH. We support this claim with an implementation of our primitive.

We discuss the application of PCHs for transaction-level rewriting of blockchains (cf. Section V). Another application that comes to mind is the usage of PCHs instead of conventional chameleon-hashes in sanitizable signatures [4] 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.

B. Related Work

We already briefly discussed the work due to Ateniese et al. [6] which inspired our work. In another work Puddu et al. [37] 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. [22] 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. [21] introduced a primitive denoted as Access Control Encryption (ACE), which was later extended in [7], [29]. 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.

C. 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 $y \xleftarrow{\$} S$ to denote that an element is sampled uniformly at random from a finite set S and assigned to y.

II. 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:*

$\mathsf{Setup}_{\mathsf{ABE}}(1^\kappa)$: *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 is straightforward and given in the full version of this paper.

Security of CP-ABE. In the following, we recall IND-CCA2 security for CP-ABE (where we explicitly model key handles, cf. [30]).

$\mathbf{Exp}_{\mathcal{A},\mathsf{ABE}}^{\mathsf{IND\text{-}CCA2}}(\kappa)$:
  $(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{mpk}_{\mathsf{ABE}}) \leftarrow \mathsf{Setup}_{\mathsf{ABE}}(1^\kappa)$
  $b \leftarrow \{0, 1\}$
  $\mathcal{Q}, \mathcal{S} \leftarrow \emptyset, i \leftarrow 0$
  $(m_0, m_1, \mathbb{A}^*, \mathtt{state}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{mpk}_{\mathsf{ABE}})$
    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 $\perp$, if $(j, \mathsf{ssk}) \notin \mathcal{Q}$ for some 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 $c^* \leftarrow \perp$
  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 $\perp$, 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 $\perp$, 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

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

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

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

Monotone Span Programs. Monotone span programs (MSP) [28] (or, essentially linear secret-sharing schemes (LSSS) [11]) 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 [34, Appendix G] of [35], 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 [1] that we chose as a core building block for our instantiation in Section IV-D). 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 child 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. [16].

Definition 4 (Chameleon-Hashes). *A chameleon-hash CH with message space $\mathcal{M}$ consists of five algorithms* $(\mathsf{PPGen}_{\mathsf{CH}}, \mathsf{KGen}_{\mathsf{CH}}, \mathsf{Hash}_{\mathsf{CH}}, \mathsf{Verify}_{\mathsf{CH}}, \mathsf{Adapt}_{\mathsf{CH}})$, *such that:*

$\mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$**.** *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 $\bot$ otherwise.

Correctness. Correctness is straightforward and given in the full version of this paper. Note that the randomness is drawn by $\mathsf{Hash}_{\mathsf{CH}}$, and not outside. The intention is to capture "private-coin" constructions [6]. We provide the security notions in Appendix A1 and stress that we rely on the notions from [16] 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 [16]. 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 $\bot$ otherwise.

Correctness. Correctness is straightforward and given in the full version of this paper.

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 A2. 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.[5]

III. POLICY-BASED CHAMELEON-HASHING

In this section, we introduce and define a novel primitive which we term policy-based chameleon-hash. In Section III-A, we formally define policy-based chameleon-hashes. In Section III-B, 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 IV-D, after having discussed the selection of the underlying primitives, 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 then confirmed with a practical implementation in Section IV-E.

---

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

A. Definitions

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

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

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

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

Hash$_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$**:** *On input a public key* 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.*

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

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

Note that we assume that the KGen$_{\mathsf{PCH}}$ outputs $\bot$ if $\mathbb{S}$ is not contained in $\mathbb{U}$ and the Adapt$_{\mathsf{PCH}}$ algorithm always verifies if the hash it is given is valid, and output $\bot$ 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 (sk$_{\mathsf{PCH}}$, pk$_{\mathsf{PCH}}$) $\leftarrow$ PPGen$_{\mathsf{PCH}}(1^\kappa)$, for all sk$_{\mathbb{S}} \leftarrow$ KGen$_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \mathbb{S})$, for all $m \in \mathcal{M}$, for all $(h, r) \leftarrow$ Hash$_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$, for all $m' \in \mathcal{M}$, for all $r' \leftarrow$ Adapt$_{\mathsf{PCH}}(\mathsf{sk}_{\mathbb{S}}, m, m', h, r)$, we have that that $1 =$ Verify$_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, h, r) =$ 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. [6], 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 Hash$_{\mathsf{PCH}}$ or Adapt$_{\mathsf{PCH}}$. The messages are adaptively chosen by the adversary.

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

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

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

$\mathbf{Exp}_{\mathcal{A},\mathsf{PCH}}^{\mathsf{Ind}}(\kappa)$
  (sk$_{\mathsf{PCH}}$, pk$_{\mathsf{PCH}}$) $\leftarrow$ 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 HashOrAdapt$_{\mathsf{PCH}}(\mathsf{sk}, \cdot, \cdot, \cdot, \cdot, b)$ on input $m, m', \mathbb{S}, \mathbb{A}$:
      $(h_0, r_0) \leftarrow$ Hash$_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m', \mathbb{A})$
      $(h_1, r_1) \leftarrow$ Hash$_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$
      sk$_{\mathbb{S}} \leftarrow$ KGen$_{\mathsf{PCH}}(\mathsf{sk}, \mathbb{S})$
      $r_1 \leftarrow$ 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

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. [6]. That is, it grants the adversary $\mathcal{A}$ adaptive access to an Adapt$_{\mathsf{PCH}}$ oracle, and requires that it be intractable to find collisions for messages which were not queried to Adapt$_{\mathsf{PCH}}$. We note that this definition, analogous to [16], is even stronger than key-exposure freeness [5], [19].[6]

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

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

We say that a PCH *scheme provides outsider collision-resistance, if* $\mathbf{Adv}_{\mathcal{A},\mathsf{PCH}}^{\mathsf{SCRout}}(\kappa)$ *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

---

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

are not satisfied by their keys (oracle $\mathsf{KGen}'_{\mathsf{PCH}}$). 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 $\mathsf{Adapt}'_{\mathsf{PCH}}$).

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

Fig. 4: PCH Insider Collision-Resistance

$\underline{\mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)}:$ 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

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*

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

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 $\mathcal{A}$.

B. 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.

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 [17], [35]. 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.*

The theorem above can be proven by a straight forward reduction to strong indistinguishability of the underlying CHET. We provide the proof in the full version.

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.*

The theorem above can be proven by a straightforward reduction to public collision-resistance of the underlying CHET. We provide the proof in the full version.

Theorem 3. *If the* PCH *scheme in Scheme 1 is based on a strongly privately collision-resistant* CHET *and an* IND-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 $\Pr[S_i]$ 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 $\Pr[S_1] = \Pr[S_0] \cdot 1/q$.

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., $\Pr[S_2] = \Pr[S_1]$.

Game 3. As Game 2, but we change the simulation of the Hash algorithm within the i^*-th query to the 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 IND-CCA2 security of Π_{ABE}, i.e., $|\Pr[S_3] - \Pr[S_2]| \leq \mathbf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{B},\mathsf{ABE}}(\kappa)$. To prove the claim, we show that we can use an adaptive IND-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 IND-CCA challenger, set 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 KGen$_{\mathsf{CHET}}$ and all queries to the Hash'$_{\mathsf{CHET}}$ oracle up to the i^*-th query. In the i^*-th query to the 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 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 $\mathbf{Adv}^{\mathsf{SCRpriv}}_{\mathcal{C},\mathsf{CHET}}(\kappa) \geq \Pr[S_3]$.

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., $\Pr[S_3] \leq \mathbf{Adv}^{\mathsf{SCRpriv}}_{\mathcal{C},\mathsf{CHET}}(\kappa)$. 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. ∎

C. 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. If we speak of access policies in this work, we refer to monotone access policies. For access policies, we use monotone span programs (MSPs) as described in Section II 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 [26], 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. [25] based on lattices and by the work of Garg et al. [24] 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.

IV. 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 an implementation of our concrete PCH.

A. 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 [1]. 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 [39] 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 [1].

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 [18]. Their scheme is fairly efficient (see [1] 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 [1]. Since FAME only provides IND-CPA security, but we require IND-CCA2 security, we apply a variant of the well known Fujisaki-Okamoto transform [23] (cf. [30]) 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.

B. Modified CHET

Camenisch et al. in [16] 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. [16], 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. [32], 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 A while our generic construction is given in Scheme 2.

$\mathsf{PPGen_{CHET}}(1^\kappa)$: On input a security parameter κ, let $\mathsf{PP_{CH}} \leftarrow \mathsf{PPGen_{CH}}(1^\kappa)$. Return $\mathsf{PP_{CHET}} \leftarrow \mathsf{PP_{CH}}$.

$\mathsf{KGen_{CHET}}(\mathsf{PP_{CHET}})$: On input $\mathsf{PP_{CHET}} = \mathsf{PP_{CH}}$, return $(\mathsf{sk_{CH}^1}, \mathsf{pk_{CH}^1}) \leftarrow \mathsf{KGen_{CH}}(\mathsf{PP_{CH}})$.

$\mathsf{Hash_{CHET}}(\mathsf{pk_{CHET}}, m)$: On input of $\mathsf{pk_{CHET}} = \mathsf{pk_{CH}^1}$ and m, let $(\mathsf{etd}, \mathsf{pk_{CH}^2}) \leftarrow \mathsf{KGen_{CH}}(\mathsf{PP_{CH}})$. Let $(h^1, r^1) \leftarrow \mathsf{Hash_{CH}}(\mathsf{pk_{CH}^1}, (m, \mathsf{pk_{CH}^1}, \mathsf{pk_{CH}^2}))$ and $(h^2, r^2) \leftarrow \mathsf{Hash_{CH}}(\mathsf{pk_{CH}^2}, (m, \mathsf{pk_{CH}^1}, \mathsf{pk_{CH}^2}))$. Return $((h^1, h^2, \mathsf{pk_{CH}^1}, \mathsf{pk_{CH}^2}), (r^1, r^2))$.

$\mathsf{Verify_{CHET}}(\mathsf{pk_{CHET}}, m, h, r)$: On input of $\mathsf{pk_{CHET}} = \mathsf{pk_{CH}^1}$, m, $h = (h^1, h^2, \mathsf{pk_{CH}^1}, \mathsf{pk_{CH}^2})$ and $r = (r^1, r^2)$, return 1, if $\mathsf{Verify_{CH}}(\mathsf{pk_{CH}^1}, (m, \mathsf{pk_{CH}^1}, \mathsf{pk_{CH}^2}), h^1, r^1) = 1$ and $\mathsf{Verify_{CH}}(\mathsf{pk_{CH}^2}, (m, \mathsf{pk_{CH}^1}, \mathsf{pk_{CH}^2}), h^2, r^2) = 1$. Otherwise, return 0.

$\mathsf{Adapt_{CHET}}(\mathsf{sk_{CHET}}, \mathsf{etd}, m, m', h, r)$: On input a secret key $\mathsf{sk_{CHET}} = \mathsf{sk_{CH}^1}$, etd, messages m and m', a hash $h = (h^1, h^2, \mathsf{pk_{CH}^1}, \mathsf{pk_{CH}^2})$ and $r = (r^1, r^1)$, first check that $\mathsf{Verify_{CHET}}(\mathsf{pk_{CHET}}, m, h, r) = 1$. Otherwise, return $\perp$. Let $r'^1 \leftarrow \mathsf{Adapt_{CH}}(\mathsf{sk_{CHET}^1}, (m, \mathsf{pk_{CH}^1}, \mathsf{pk_{CH}^2}), (m', \mathsf{pk_{CH}^1}, \mathsf{pk_{CH}^2}), r^1, h^1)$ and $r'^2 \leftarrow \mathsf{Adapt_{CH}}(\mathsf{etd}, (m, \mathsf{pk_{CH}^1}, \mathsf{pk_{CH}^2}), (m', \mathsf{pk_{CH}^1}, \mathsf{pk_{CH}^2}), r^2, h^2)$. Let $r' \leftarrow (r'^1, r'^2)$. If $\mathsf{Verify_{CHET}}(\mathsf{pk_{CHET}}, m', h, r') = 0$, return $\perp$. Return r'.

Scheme 2: Construction of a CHET

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 C.

Concrete Instantiation. For our concrete instantiation of the CHET, we use the RSA-based CH from [16] which builds upon the one presented by Brzuska et al. [15]. We recall this construction in Appendix B1 and show its security in our stronger model.

C. 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. We assume that the reader is familiar with bilinear maps. In a nutshell, let BilGen be an algorithm outputting the parameters $(p, \hat{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$ for a Type-III pairing. For our instantiation, we choose the BN curve

family [9] and in particular the BN254 curve, which gives us around 100 bit of security. If one wants to go for a larger security level with comparable performance, one can follow the same lines as Zcash[7] and choose the BLS curve family [8] and in particular the curve BLS12-381 giving roughly 120 bits of security.[8]

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.

D. 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 IV-B. 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, 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 IV-A. 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 [20]). 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) [13][9] and let enc : $\{0,1\}^{\ell+\kappa} \rightarrow \mathbb{G}_{\mathsf{T}}$ 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 [16], we obtain the following.

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

E. Performance Evaluation

To evaluate the practicality of our scheme, we implemented our construction from Scheme 3 in Python 3.5.3 and base our implementation on the Charm [2] framework version 0.50[10], the implementation of FAME from the authors of [1][11], whereas we use our own implementation of CH. We performed the measurements on a laptop with an Intel Core i7-7600U CPU @ 2.80GHz with 16GB RAM running Ubuntu 18.04. All frequent operations, i.e., KGen, Hash, Verify and Adapt

for a policy of 64 attributes (consist of two OR clauses with 32 attributes connected via AND) are significantly below a second. Due to space constraints, we refer the reader to the full version for the detailed performance analysis.

V. 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. [6] 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. 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. [6], 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.

VI. 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

---

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

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

[9]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.

[10]https://github.com/JHUISI/charm

[11]https://github.com/sagrawal87/ABE

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

1) Choose prime e_1 s.t. $e_1 > N'$ with $N' = \max_r\{(N, \cdot, \cdot, \cdot) \leftarrow \mathsf{RSAKGen}(1^\kappa; r)\}$.
2) Run $(N_1, p_1, q_1, \cdot, \cdot) \leftarrow \mathsf{RSAKGen}(1^\kappa)$, choose a hash function $H_1 : \{0,1\} \to \mathbb{Z}^*_{N_1}$. 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 $a_1, a_2, b_1, b_2 \xleftarrow{\$} \mathbb{Z}^*_p$ and $w_1, w_2, w_3 \xleftarrow{\$} \mathbb{Z}_p$, hash functions $H_3 : \{0,1\}^* \to \mathbb{G}_1$ and $H_4 : \{0,1\}^* \to (\mathbb{Z}_p)^2$, an encryption scheme $(\mathsf{KGen}_{\mathsf{SE}}, \mathsf{Enc}_{\mathsf{SE}}, \mathsf{Dec}_{\mathsf{SE}})$, and set

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

$$\mathsf{msk}_{\mathsf{ABE}} \leftarrow (a_1, a_2, b_1, g_2, g_1^{w_1}, g_1^{w_2}, g_1^{w_3})$$

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 $(a_1, a_2, b_1, g_2, g_1^{w_1}, g_1^{w_2}, g_1^{w_3})$, pick $\rho_1, \rho_2 \xleftarrow{\$} \mathbb{Z}_p$ 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 $\sigma_s \xleftarrow{\$} \mathbb{Z}_p$ and set $\mathsf{sk}_s = (\mathsf{sk}_{s,1}, \mathsf{sk}_{s,2}, g_1^{-\sigma_s})$. 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 $\sigma' \xleftarrow{\$} \mathbb{Z}_p$ 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 $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{sk}_{\mathbb{S}, \mathsf{ABE}})$.

$\underline{\mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbf{M})}$: 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 $r_1 \xleftarrow{\$} \mathbb{Z}^*_{N_1}$, $r_2 \xleftarrow{\$} \mathbb{Z}^*_{N_2}$, 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 $r \xleftarrow{\$} \{0,1\}^\kappa$, $k \xleftarrow{\$} \mathsf{KGen}_{\mathsf{SE}}(1^\kappa)$ 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 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 $\mathsf{ct}_i \leftarrow (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2}, \mathsf{ct}_{i,3})$, $K \leftarrow \mathsf{encode}(k, r)$, compute $\hat{\mathsf{ct}} \leftarrow T_1^{u_1} \cdot T_2^{u_2} \cdot K$, $\tilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{Enc}_{\mathsf{SE}}(k, d_2)$, 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 $\mathsf{pk}_{\mathsf{CHET}} = (\kappa, N_1, e, H_1)$, verify whether $r_1 \in \mathbb{Z}^*_{N_1}$, $r_2 \in \mathbb{Z}^*_{N_2}$, and whether $h_1 = H_1((m, N_1, H_1, N_2, H_2))r_1^e \mod N_1$ 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.

$\underline{\mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{sk}_{\mathbb{S}}, m, m', h, r)}$: 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 $r = (r_1, r_2)$ and assuming that $\mathbb{S}$ satisfies the access policy $\mathbb{A}$ used for encryption and associated to 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 $h_1 = H_1((m, N_1, H_1, N_2, H_2))r_1^e \mod N_1$ and $h_2 = H_2((m, N_1, H_1, N_2, H_2))r_2^e \mod N_2$ and return $\perp$, if any of the checks fails. Otherwise, let $x_1 \leftarrow H_1((m, N_1, H_1, N_2, H_2))$, $x'_1 \leftarrow H_1((m', N_1, H_1, N_2, H_2))$, $y_1 \leftarrow x_1 r_1^e \mod N_1$ as well as $x_2 \leftarrow H_2((m, N_1, H_1, N_2, H_2))$, $x'_2 \leftarrow H_2((m', N_1, H_1, N_2, H_2))$, $y_2 \leftarrow x_2 r_2^e \mod N_2$, compute $r'_1 \leftarrow (y_1(x_1^{'-1}))^{d_1} \mod N_1$, and finally $r'_2 \leftarrow (y_2(x_2^{'-1}))^{d_2} \mod N_2$.
4) Return $\perp$ if $h_1 \neq H_1((m', N_1, H_1, N_2, H_2))r_1^{'e} \mod N_1$ or $h_2 \neq H_2((m', N_1, H_1, N_2, H_2))r_2^{'e} \mod N_2$.
Finally, return $r' \leftarrow (r'_1, r'_2)$.

Scheme 3: Concrete construction of a PCH

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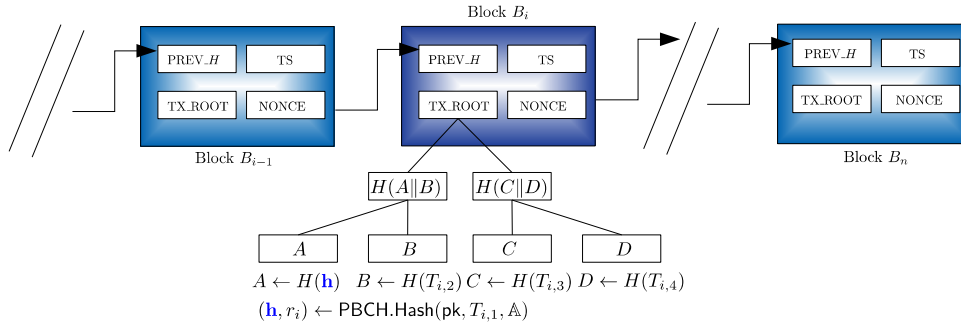
Fig. 5: Using a PCH for transaction-level rewrites

construction of this primitive from a CP-ABE scheme and a modified CHET, and provide first implementation results.

REFERENCES

[1] S. Agrawal and M. Chase, "FAME: Fast attribute-based message encryption," in *ACM CCS 17*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM Press, Oct. / Nov. 2017, pp. 665–682.

[2] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.

[3] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," *CoRR*, 2018.

[4] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik, "Sanitizable signatures," in *ESORICS 2005*, ser. LNCS, S. D. C. di Vimercati, P. F. Syverson, and D. Gollmann, Eds., vol. 3679. Springer, Heidelberg, Sep. 2005, pp. 159–177.

[5] G. Ateniese and B. de Medeiros, "On the key exposure problem in chameleon hashes," in *Security in Communication Networks, 4th International Conference, SCN 2004*, 2004.

[6] G. Ateniese, B. Magri, D. Venturi, and E. R. Andrade, "Redactable blockchain - or - rewriting history in bitcoin and friends," in *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017*, 2017.

[7] C. Badertscher, C. Matt, and U. Maurer, "Strengthening access control encryption," in *ASIACRYPT 2017, Part I*, ser. LNCS, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, Heidelberg, Dec. 2017, pp. 502–532.

[8] P. S. L. M. Barreto, B. Lynn, and M. Scott, "Constructing elliptic curves with prescribed embedding degrees," in *SCN 02*, ser. LNCS, S. Cimato, C. Galdi, and G. Persiano, Eds., vol. 2576. Springer, Heidelberg, Sep. 2003, pp. 257–267.

[9] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *SAC 2005*, ser. LNCS, B. Preneel and S. Tavares, Eds., vol. 3897. Springer, Heidelberg, Aug. 2006, pp. 319–331.

[10] M. T. Beck, J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig, "Practical strongly invisible and strongly accountable sanitizable signatures," in *ACISP 17, Part I*, ser. LNCS, J. Pieprzyk and S. Suriadi, Eds., vol. 10342. Springer, Heidelberg, Jul. 2017, pp. 437–452.

[11] A. Beimel, "Secure schemes for secret sharing and key distribution," in *PhD thesis*, 1996.

[12] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, "The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme," *Journal of Cryptology*, vol. 16, no. 3, pp. 185–215, Jun. 2003.

[13] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *ACM CCS 93*, V. Ashby, Ed. ACM Press, Nov. 1993, pp. 62–73.

[14] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2007, pp. 321–334.

[15] C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk, "Security of sanitizable signatures revisited," in *PKC 2009*, ser. LNCS, S. Jarecki and G. Tsudik, Eds., vol. 5443. Springer, Heidelberg, Mar. 2009, pp. 317–336.

[16] J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig, "Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures," in *Public-Key Cryptography - PKC 2017.*, 2017.

[17] M. Chase, "Multi-authority attribute based encryption," in *TCC 2007*, ser. LNCS, S. P. Vadhan, Ed., vol. 4392. Springer, Heidelberg, Feb. 2007, pp. 515–534.

[18] J. Chen, R. Gay, and H. Wee, "Improved dual system ABE in prime-order groups via predicate encodings," in *EUROCRYPT 2015, Part II*, ser. LNCS, E. Oswald and M. Fischlin, Eds., vol. 9057. Springer, Heidelberg, Apr. 2015, pp. 595–624.

[19] X. Chen, F. Zhang, and K. Kim, "Chameleon hashing without key exposure," in *ISC*, 2004.

[20] R. Cramer and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack," *SIAM Journal on Computing*, vol. 33, no. 1, pp. 167–226, 2003.

[21] I. Damgård, H. Haagh, and C. Orlandi, "Access control encryption: Enforcing information flow with cryptography," in *TCC 2016-B, Part II*, ser. LNCS, M. Hirt and A. D. Smith, Eds., vol. 9986. Springer, Heidelberg, Oct. / Nov. 2016, pp. 547–576.

[22] A. L. Ferrara, G. Fuchsbauer, and B. Warinschi, "Cryptographically enforced RBAC," in *2013 IEEE 26th Computer Security Foundations Symposium*, 2013, pp. 115–129.

[23] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *CRYPTO'99*, ser. LNCS, M. J. Wiener, Ed., vol. 1666. Springer, Heidelberg, Aug. 1999, pp. 537–554.

[24] S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters, "Attribute-based encryption for circuits from multilinear maps," in *CRYPTO 2013, Part II*, ser. LNCS, R. Canetti and J. A. Garay, Eds., vol. 8043. Springer, Heidelberg, Aug. 2013, pp. 479–499.

[25] S. Gorbunov, V. Vaikuntanathan, and H. Wee, "Attribute-based encryption for circuits," in *45th ACM STOC*, D. Boneh, T. Roughgarden, and J. Feigenbaum, Eds. ACM Press, Jun. 2013, pp. 545–554.

[26] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *ACM CCS 06*, A. Juels, R. N. Wright, and S. Vimercati, Eds. ACM Press, Oct. / Nov. 2006, pp. 89–98, available as Cryptology ePrint Archive Report 2006/309.

[27] V. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to attribute based access control (abac) definition and considerations," NIST Special Publication 800-162, Tech. Rep., 2014.

[28] M. Karchmer and A. Wigderson, "On span programs," in *Proceedings of Structures in Complexity Theory*, 1993, pp. 102–111.

[29] S. Kim and D. J. Wu, "Access control encryption for general policies from standard assumptions," in *ASIACRYPT 2017, Part I*, ser. LNCS, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, Heidelberg, Dec. 2017, pp. 471–501.

[30] V. Koppula and B. 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.

[31] H. Krawczyk and T. Rabin, "Chameleon signatures," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA*, 2000.

[32] S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig, "Chameleon-hashes with dual long-term trapdoors and their applications," in *AfricaCrypt*, 2018.

[33] L. Lamport, R. E. Shostak, and M. C. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.

[34] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," Cryptology ePrint Archive, Report 2010/351, 2010, http://eprint.iacr.org/2010/351.

[35] A. B. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *EUROCRYPT 2011*, ser. LNCS, K. G. Paterson, Ed., vol. 6632. Springer, Heidelberg, May 2011, pp. 568–588.

[36] R. C. Merkle, "A certified digital signature," in *Advances in Cryptology - CRYPTO '89*, 1989.

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

[38] A. Sahai and B. R. Waters, "Fuzzy identity-based encryption," in *EUROCRYPT 2005*, ser. LNCS, R. Cramer, Ed., vol. 3494. Springer, Heidelberg, May 2005, pp. 457–473.

[39] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *PKC 2011*, ser. LNCS, D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, Eds., vol. 6571. Springer, Heidelberg, Mar. 2011, pp. 53–70.

Appendix

A. Security Properties of CHs and CHETs

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

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. [15] to a computational version, which is enough for most use-cases, including ours. However, compared to the existing definition in [10], [16], [32], the adversary is also allowed to know the secret key sk, but cannot generate it.

Note that we need to implicitly return $\bot$ 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 $\bot$. 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.

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

Fig. 6: CH Strong Indistinguishability

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

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

We say a CH scheme is indistinguishable, if $\mathbf{Adv}_{\mathcal{A},\mathsf{CH}}^{\mathsf{Ind}}(\kappa)$ 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 [5]: 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.

$\mathbf{Exp}_{\mathcal{A},\mathsf{CH}}^{\mathsf{CR}}(\kappa)$
$\quad \mathsf{PP}_{\mathsf{ch}} \leftarrow \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$
$\quad (\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \leftarrow \mathsf{KGen}_{\mathsf{CH}}(\mathsf{PP}_{\mathsf{ch}})$
$\quad \mathcal{Q} \leftarrow \emptyset$
$\quad (m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{Adapt}'_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, \cdot, \cdot, \cdot, \cdot)}(\mathsf{pk}_{\mathsf{CH}})$
$\quad\quad \text{where } \mathsf{Adapt}'_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, \cdot, \cdot, \cdot, \cdot) \text{ on input } m, m', r, h:$
$\quad\quad\quad r' \leftarrow \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r, h)$
$\quad\quad\quad \text{return } \bot, \text{ if } r' = \bot$
$\quad\quad\quad \mathcal{Q} \leftarrow \mathcal{Q} \cup \{m, m'\}$
$\quad\quad\quad \text{return } r'$
$\quad \text{return } 1, \text{ if } \mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m^*, h^*, r^*) = \mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m'^*,$
$\quad\quad r'^*, h^*) = 1 \ \wedge \ m'^* \notin \mathcal{Q} \ \wedge \ m^* \neq m'^*$
$\quad \text{return } 0$

Fig. 7: CH Collision-Resistance

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

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

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

Definition 12 (Secure Chameleon-Hashes). *We call a chameleon-hash CH secure, if it is correct, indistinguishable, and collision-resistant.*

Note that we do not require the uniqueness property [16] in the context of this paper which is why we omit the definition.

2) Security of CHETs: Subsequently we restate the security properties of CHET schemes from [16], 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 [16] 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 [16], [10], the adversary additionally receives the secret key sk, but cannot generate it.

$$
\begin{aligned}
&\mathbf{Exp}^{\mathsf{SInd}}_{\mathcal{A},\mathsf{CHET}}(\kappa) \\
&\quad \mathsf{PP}_{\mathsf{CHET}} \leftarrow \mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa) \\
&\quad (\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \leftarrow \mathsf{KGen}_{\mathsf{CHET}}(\mathsf{PP}_{\mathsf{CHET}}) \\
&\quad b \xleftarrow{\$} \{0,1\} \\
&\quad b^* \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}},\cdot,\cdot,b)}(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \\
&\qquad \text{where } \mathsf{HashOrAdapt}_{\mathsf{CHET}}(\mathsf{sk},\cdot,\cdot,b) \text{ on input } m, m': \\
&\qquad\quad \text{let } (h_0, r_0, \mathsf{etd}_0) \leftarrow \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m') \\
&\qquad\quad \text{let } (h_1, r_1, \mathsf{etd}_1) \leftarrow \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m) \\
&\qquad\quad \text{let } r_1 \leftarrow \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{etd}_1, m, m', h_1, r_1) \\
&\qquad\quad \text{return } (h_b, r_b, \mathsf{etd}_b) \\
&\quad \text{return } b = b^*
\end{aligned}
$$

Fig. 8: CHET Strong Indistinguishability

Definition 13 (Strong Indistinguishability). *We define the advantage of an adversary $\mathcal{A}$ in the strong indistinguishability experiment $\mathbf{Exp}^{\mathsf{SInd}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$ 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.*

Public Collision-Resistance. Public collision-resistance grants the adversary access to an $\mathsf{Adapt}_{\mathsf{CH}}$ oracle. It requires that it is intractable to produce collisions, other than the ones produced by the $\mathsf{Adapt}_{\mathsf{CH}}$ oracle. Due to lack of space and since the formal definition is not required for the proofs presented in this version we refer the reader to the full version or [16] for a formal definition of this property.

Strong Private Collision-Resistance. Strong private collision-resistance requires that it is even intractable for the holder of the secret key sk to find collisions without knowledge of etd. Compared to the definition by Camenisch et al. [16], 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 14 (Strong Private Collision-Resistance). *We define the advantage of an adversary $\mathcal{A}$ in the private collision-resistance experiment $\mathbf{Exp}^{\mathsf{SCRpriv}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$ as*

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

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

Fig. 9: CHET Strong Private Collision-Resistance

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

B. Secure Chameleon-Hashes

1) Instantiation of a Secure CH*:* We recall a construction from [16] in Scheme 4 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 RSAKGen always outputs moduli with the same bit-length, based on κ.

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

$\mathsf{KGen}_{\mathsf{CH}}(\mathsf{PP})$: On input $\mathsf{PP} = (1^\kappa, e)$ run $(N, p, q, \cdot, \cdot) \leftarrow \mathsf{RSAKGen}(1^\kappa)$, choose a hash function $H : \{0,1\}^* \to \mathbb{Z}_N^*$ (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 $(\mathsf{sk}, \mathsf{pk})$.

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

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

$\mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}, m, m', h, r)$: 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 $\bot$ if $\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}, m, h, r) \neq 1$. Otherwise, let $x \leftarrow H(m)$, $x' \leftarrow H(m')$, $y \leftarrow xr^e \bmod N$ and return $r' \leftarrow (y(x'^{-1}))^d \bmod N$.

Scheme 4: RSA-based Chameleon-Hash

In [16] Camenisch et al. show that the chameleon-hash in Scheme 4 is secure under the one-more RSA inversion assumption [12] in the random oracle model (ROM) [13]. We argue that the CH restated in Scheme 4 remains secure in our strengthened model. All properties, but strong indistinguishability, have already been proven by Camenisch et al. [31]. 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. ∎

C. Secure CHETs

The generic construction is given in Construction 2. This construction is essentially the one given by Camenisch et al. [16], 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. [32]. This is crucial to achieve our strengthened security properties.

Correctness follows from inspection and from the correctness of the underlying chameleon hash. Public collision resistance of this construction was already proven in [16]. We prove the remaining properties below.

Theorem 5. *If* CH *is collision-resistant, then the construction of a* CHET *given in Scheme 2 is strongly privately collision-resistant.*

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 PP_{ch} is implicit in pk_{ch} and let q be an upper bound on the number of queries to the Hash'_{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 public key of the second CH and for the respective query to the collision finding oracle Adapt'_{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. ∎

Theorem 6. *If* CH *is strongly indistinguishable, then the construction of a* CHET *given in Scheme 2 is strongly indistinguishable.*

Proof: First, we prove strong indistinguishability by a sequence of games.

Game 0: The original strong indistinguishability game in the case $b = 1$.

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

Transition - Game 0 → Game 1: We claim that Game 0 and Game 1 are indistinguishable under the strong indistinguishability of CH. More formally, assume that the adversary $\mathcal{A}$ can distinguish this hop. We can then construct an adversary $\mathcal{B}$ which breaks the indistinguishability of CH. In particular, the reduction works as follows. $\mathcal{B}$ receives PP_{ch} as its own challenge, passing them through to $\mathcal{A}$ within PP_{chet} (generating the rest honestly), 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. Note, the HashOrAdapt always checks if the adaption was successful, and thus so does $\mathcal{B}$, making the distributions equal. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{CH\text{-}sInd}}(\kappa)$ follows.

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

Transition - Game 1 → Game 2: We claim that Game 1 and Game 2 are indistinguishable under the strong indistinguishability of CH. More formally, assume that the adversary $\mathcal{A}$ can distinguish this hop. We can then construct an adversary $\mathcal{B}$ which breaks the indistinguishability of CH. In particular, the reduction works as follows. $\mathcal{B}$ receives PP_{ch} as its own challenge, passing them through to $\mathcal{A}$ within PP_{chet} (generating the rest honestly), and proceeds as in the prior hop, with the exception that it uses the HashOrAdapt oracle to generate h^2. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. Clearly, the simulation is perfect from $\mathcal{A}$'s point of view. Note, the HashOrAdapt always checks if the adaption was successful, and thus so does $\mathcal{B}$, making the distributions equal. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{CH\text{-}sInd}}(\kappa)$ follows.

We are now in the case $b = 0$. However, as the adversary only sees negligible changes, strong indistinguishability is proven.

∎