Accountable Fine-Grained Blockchain Rewriting in the Permissionless Setting

Yangguang Tian, Bowen Liu, Yingjiu Li, Pawel Szalachowski, and Jianying Zhou

***Abstract*— Blockchain rewriting with fine-grained access control allows a user to create a transaction associated with a set of attributes, while a modifier who possesses sufficient rewriting privileges from a trusted authority satisfying the attribute set can anonymously rewrite the transaction. However, it lacks accountability and is not designed for open blockchains that require no centralized trust authority. In this work, we introduce accountable fine-grained blockchain rewriting in a permissionless setting. The property of accountability allows the modifier's identity and their rewriting privileges to be held accountable for the modified transactions in case of malicious rewriting. Our contributions are three-fold. First, we present a generic framework for secure blockchain rewriting in the permissionless setting. Second, we present an instantiation of our framework and show its practicality through evaluation analysis. Last, we demonstrate that our proof-of-concept implementation can be effectively integrated into open blockchains.**

***Index Terms*— Blockchain rewriting, accountability, open blockchains.**

I. Introduction

BLOCKCHAINS have received tremendous attention from research communities and industries in recent years. The concept was first introduced in the context of Bitcoin [39], where all payment transactions are appended in a public ledger, and each transaction is ordered and verified by network nodes in a peer-to-peer manner. Blockchain ledgers grow by one block at a time, where the new block in the chain is decided by a consensus mechanism (e.g., Proof-of-Work in Bitcoin [27]) executed by the network nodes. Usually, blockchains deploy hash-chains as an append-only structure, where the hash of a block is linked to the next block in the chain. Each block includes a set of valid transactions which are accumulated into a single hash value using the Merkle tree [38], and each transaction contains certain content which needs to be registered in the blockchain.

Blockchain was originally designed to be immutable, such that the registered content cannot be modified once they are appended. However, blockchain rewriting is required in practice, or even legally necessary in data regulation laws such as GDPR in Europe [1]. Since a blockchain platform in the permissionless setting is open, it is possible some users append transactions into a chain containing illicit content such as sensitive information, stolen private keys, and inappropriate videos [36], [37]. The existence of illicit content in the chain poses a significant challenge to law enforcement agencies like Interpol [49].

Blockchain rewriting can be realized by replacing a standard hash function, used for generating transaction hash in the blockchain, by a trapdoor-based chameleon hash [30]. The users who are given the trapdoor, which we call modifiers, can modify a mutable transaction. In other words, the same mutable transaction can be modified by multiple modifiers with the same privilege. Nonetheless, for most real-life blockchain applications, blockchain rewriting with fine-grained access control is more desired. In fine-grained access control, each mutable transaction is associated with a set of attributes, and each modifier is associated with a policy representing their rewriting privilege. A mutable transaction can be potentially modified by multiple modifiers if their rewriting privileges satisfy the set of attributes associated with the transaction.

A. Motivation

Blockchain rewriting with fine-grained access control has been studied in the permissioned setting [16], [48]; however, the proposed solution is not suitable for open blockchains in the permissionless setting for two reasons: 1) It requires a trusted authority to distribute rewriting privileges; however, such authority does not exist in the permissionless setting. 2) It lacks the accountability of identifying who are responsible for malicious modification of blockchain. For example, modifiers may add illicit or malicious content to mutable transactions, or delete legitimate or benign mutable transactions from blockchain. The main motivation of this work is to make fine-grained blockchain rewriting accountable in the permissionless setting.

In the permissionless setting, it is desired to achieve public accountability for fine-grained blockchain rewriting without relying on any trusted authority. Public accountability should

Manuscript received 19 March 2022; revised 25 September 2022, 26 June 2023, and 8 November 2023; accepted 3 December 2023. Date of publication 7 December 2023; date of current version 21 December 2023. This work was supported in part by the EU's Research and Innovation Program under Grant 101019645 (SECANT) and Grant 101095634 (ENTRUST) and in part by the U.K. Government Horizon Europe Guarantee and administered by UKRI. The work of Yangguang Tian was supported in part by the National Natural Science Foundation of China under Grant 61872264. The work of Yingjiu Li was supported in part by the Ripple University Blockchain Research Initiative. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Debdeep Mukhopadhyay. *(Corresponding author: Yangguang Tian.)*

Yangguang Tian is with the School of Computer Science and Electronic Engineering, University of Surrey, GU2 7XH Guildford, U.K. (e-mail: yangguang.tian@surrey.ac.uk).

Bowen Liu, Pawel Szalachowski, and Jianying Zhou are with the Information Systems Technology and Design Pillar, Singapore University of Technology and Design, Singapore 487372.

Yingjiu Li is with the Department of Computer and Information Science, University of Oregon, Eugene, OR 97403 USA.

This article has supplementary downloadable material available at https://doi.org/10.1109/TIFS.2023.3340917, provided by the authors.

Digital Object Identifier 10.1109/TIFS.2023.3340917

enable any user in the public to identify responsible modifiers who have made malicious modifications directly to mutable transactions. In the case of indirect modifications, authorized modifiers may generate an access device like a blackbox by packaging their rewriting privileges and distribute it to other users, who use the access device in making malicious modifications to mutable transactions. In this case, public accountability should enable the pubic to identify both responsible users who make malicious modifications, and the responsible rewriting privileges included in the access device.

B. Our Contributions

We introduce a new framework of accountable fine-grained blockchain rewriting in the permissionless setting. First, our framework relies on dynamic proactive secret sharing (DPSS) [35] to achieve strong security without relying on any trusted authority. We replace the trusted authority by a committee of multiple users for granting rewriting privileges, where each user holds a share of trust. We allow any user to join in and leave from a committee in any time epoch. We adapt the key-policy attribute-based encryption (KP-ABE) [44] to ensure fine-grained access control without a central authority. In particular, our adaption includes the following: 1) The master secret key in the framework is split into multiple key shares so that each user in a committee holds a single key share. 2) A certain number of shareholders in a committee can collaboratively recover the master secret key and distribute rewriting privileges to modifiers. 3) Any user can freely join/leave a committee, and the master secret key remains fixed across different committees. Our framework achieves strong security because its master secret key remains secure even if no more than a threshold number of shareholders are compromised in any committee.

Second, our framework achieves public accountability based on a novel combination of digital signature scheme, commitment scheme, and KP-ABE with public traceability (ABET for short). First, the digital signature helps the public to link a modified transaction to a modifier (or modifier's public key), as they sign the modified transaction using their signing keys, and the signed transaction is publicly verifiable. Second, the commitment scheme helps the public to link modifiers' public keys to responsible committees. Third, the ABET scheme helps the public to obtain a set of rewriting privileges from interacting with an access device in the case that an unauthorized user applies the access device in rewriting mutable transactions. Since there is no existing ABET to achieve this goal, we propose a new ABET scheme and apply it in open blockchains.

The major contributions of this work are summarized as follows.

- *Generic Framework.* We introduce a new generic framework of accountable fine-grained blockchain rewriting, which is based on an accountable policy and committee-based chameleon hash function (APC^2H for short). A unique feature of this framework is that it allows the fine-grained blockchain rewriting to be performed in the permissionless setting.
- *Public Accountability.* We introduce a new notion called public accountability. The modifiers' public keys and their rewriting privileges are publicly held accountable for the modified transactions.
- *New Primitive.* We present a new ABET scheme, which is of independent interest. The proposed ABET scheme is the first KP-ABE scheme with public traceability designed for decentralized systems.
- *Integration to Open Blockchains.* The proof-of-concept implementation shows that blockchain rewriting based on our approach incurs almost no overhead to chain validation when compared to the immutable blockchain.

II. Related Work

A. Blockchain Rewriting

Ateniese et al. [7] introduced the notion of blockchain rewriting. Their proposal is to replace the regular SHA256 hash function by a chameleon hash (CH) in blockchain generation [30]. The hashing of CH is parametrized by a public key `pk`, and CH behaves like a collision-resistant hash function if the chameleon secret key `sk` (or trapdoor) is unknown. A trapdoor holder (or modifier) can find collisions and output a new message-randomness pair without changing the hash value.

Camenisch et al. [14] introduced a new cryptographic primitive: chameleon hash with ephemeral trapdoor (CHET). CHET states that a modifier should have two trapdoors to find collisions: one trapdoor `sk` is associated with the public key `pk`; the other one is an ephemeral trapdoor etd chosen by the party who initially computed the hash value. CHET provides more control in rewriting in the sense that the party, who computed the hash value, can decide whether the holder of `sk` shall be able to rewrite the hash by providing or withholding the ephemeral trapdoor etd.

Derler et al. [16] proposed policy-based chameleon hash (PCH) to achieve fine-grained blockchain rewriting. The proposed PCH replaces the public key encryption scheme in CHET by a ciphertext-policy ABE scheme, such that a modifier must satisfy a policy to find collisions given a hash value. Later, Tian et al. proposed an accountable PCH (PCHBA) for for blockchain rewritings [48]. The proposed PCHBA enables the modifiers of transactions to be held accountable for the modified transactions. In particular, PCHBA allows a third party (e.g., key generation center) to resolve any dispute over modified transactions.

In another work, Puddu et al. [42] proposed μchain: a mutable blockchain. A transaction owner introduces a set of transactions, including an active transaction and multiple inactive transactions, where the inactive transactions are possible versions of the transaction data (namely, mutations) encrypted by the transaction owner, and the decryption keys are distributed among miners using Shamir's SSS [46]. The transaction owner enforces access control policies to define who is allowed to trigger mutations in which context. Upon receiving a mutation-trigger request, a set of miners runs a Multi Party Computation (MPC) protocol to recover the decryption key, decrypt the appropriate version of the transaction and publish it as an active transaction. μchain incurs

TABLE I

THE COMPARISON BETWEEN VARIOUS BLOCKCHAIN REWRITING SOLUTIONS. CH-BASED MEANS BLOCKCHAIN REWRITING IS REALIZED VIA THE CHAMELEON HASH FUNCTION. FINE-GRAINED (ACCESS CONTROL) MEANS THAT A TRANSACTION IS ASSOCIATED WITH AN ATTRIBUTE SET (OR AN ACCESS POLICY), AND THE TRANSACTION CAN BE MODIFIED BY ANYONE WHOSE REWRITING PRIVILEGE SATISFIES THE ATTRIBUTE SET (OR THE ACCESS POLICY)

	CH [7]	μchain [42]	PCH [16]	V-CH [19]	PCHBA [48]	Ours
CH-based	✓	×	✓	×	✓	✓
Permissionless	✓	✓	×	✓	×	✓
Fine-grained	×	×	✓	×	✓	✓
Accountability	✓	✓	×	✓	✓	✓

considerable overhead due to the use of MPC protocols across multiple miners. It works at both permissioned and permissionless blockchains.

Deuber et al. [19] introduced an efficient redactable blockchain in the permissionless setting. The proposed protocol relies on a consensus-based e-voting system [29], such that the modification is executed in the chain if a modification request from any public user gathers enough votes from miners (we call it V-CH for convenience). In a follow-up work, Thyagarajan et al. [47] introduced a protocol called Reparo to repair blockchains, which acts as a publicly verifiable layer on top of any permissionless blockchain. The unique feature of Reparo is that it is immediately integrable into open blockchains in a backward compatible fashion (i.e., any existing blockchains already containing illicit contents can be redacted).

There are mainly two types of blockchain rewritings in the literature: CH-based [7], [14], [15], [16], [48], and non CH-based [19], [42], [47]. CH-based blockchain rewritings allow one or more trusted modifiers to rewrite blockchain. The non-CH-based solution requires a threshold number of parties (or miners) to rewrite the blockchain. We stress that both aim to secure blockchain rewritings and one can apply both of them to redactable blockchains.

Table I shows a comparison between blockchain rewriting related solutions. In this work, we use chameleon hash cryptographic primitive to secure the blockchain rewriting. Our proposed solution supports fine-grained and accountable rewriting for open blockchains in the permissionless setting. It holds both the modifiers' public keys and their rewriting privileges accountable for the modified transactions. We stress that it is necessary to hold rewriting privileges accountable to the modified transactions; for example, a modifier may obtain various rewriting privileges from different committees. Overall, this work takes a significant step forward by allowing fine-grained blockchain rewriting to be performed in the permissionless setting compared to [16] and [48].

III. PRELIMINARY

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

A. Attribute-Based Encryption With Public Traceability

An attribute-based encryption with public traceability is shown as follows.

- $\mathsf{Setup}(1^\lambda)$: It takes a security parameter λ as input, outputs a master key pair $(\mathtt{msk}, \mathtt{mpk})$.
- $\mathsf{KeyGen}(\mathtt{msk}, \Lambda)$: It takes the master secret key $\mathtt{msk}$, an access policy Λ as input, outputs a decryption key $\mathtt{sk}_{\Lambda_i}$, which is associated with a unique index i. We assume an index space $\{1, \cdots, k\}$, where k denotes the maximal number of the index.
- $\mathsf{Enc}(\mathtt{mpk}, m, \delta, j)$: It takes the master public key $\mathtt{mpk}$, a message m, a set of attributes $\delta \subseteq \mathcal{U}$, and an index $j \in \{1, \cdots, k+1\}$ as input, outputs a ciphertext C. Note that C contains δ, not index j, and $\mathcal{U}$ is an attribute universe.
- $\mathsf{Dec}(\mathtt{mpk}, C, \mathtt{sk}_{\Lambda_i})$: It takes the master public key $\mathtt{mpk}$, a ciphertext C, and the decryption key $\mathtt{sk}_{\Lambda_i}$ as input, outputs the message m if $1 = \Lambda_i(\delta) \wedge j \leq i$.
- $\mathsf{Trace}(\mathtt{mpk}, \mathcal{O}, \epsilon)$: It takes master public key $\mathtt{mpk}$, a policy-specific decryption device $\mathcal{O}$, and a parameter $\epsilon > 0$ as input, outputs a set of indexes $\{1, \cdots, k\}$, where $\{1, \cdots, k\}$ denotes the index set of the accused decryption keys, and ϵ denotes the lower-bound of $\mathcal{O}$'s decryption ability.

Correctness and Security. An ABET scheme requires message-hiding, index-hiding and traceability. The message-hiding security is similar to the semantic security of conventional ABE scheme, except that every key query should include a unique index. We call it semantic security, and we require a CCA security (i.e., secure against chosen ciphertext attacks) due to the simulation of adapt queries in the adaptive collision-resistance security proof. Index-hiding is called ciphertext anonymity in this work. Specifically, the encryptor must generate a ciphertext on a message associated with a set of attributes and a *hidden* index $j \in \{1, \cdots, k+1\}$. In other words, the generated ciphertext reveals no information about index j to any third parties. Later, the Trace algorithm will use $j \in \{1, \cdots, k+1\}$ in generating ciphertext for tracing.

Traceability means that, given a policy-specific decryption device that includes a set of decryption keys, the tracing algorithm, which treats the decryption device as an oracle, can identify the accused decryption keys that have been used in constructing the decryption device. This decryption device represents a pirate decryption process: it may not be a physical box and may simply be some code on a computer. We denote policy-specific decryption device as access device because it accumulates various rewriting privileges for blockchain rewriting. We stress that the tracing algorithm works if an ABET scheme achieves semantic security and ciphertext anonymity.

The formal definitions of semantic security and ciphertext anonymity are referred to the supplemental material (Section II) [2], and the formal definition of traceability refers to [12], [13], and [32]. We provide a brief description of the traceability against policy-specific decryption device here: 1) the algorithm generates a ciphertext on a message under a set of attributes δ that satisfies Λ (i.e., $1 = \Lambda(\delta)$), and a hidden index from $\{1, \cdots, k+1\}$. 2) the algorithm sends the ciphertext to the decryption device and checks whether the decryption is successful. If decryption succeeds, the user outputs the accused index; otherwise, the user generates a new ciphertext under the same attribute set δ and a new index. 3) the algorithm continues this process until finding a set of accused indexes in $\{1, \cdots, k\}$.

B. Dynamic Proactive Secret Sharing

A dynamic proactive secret sharing DPSS consists of Share, Redistribute, and Open [8] protocols. It allows a dealer to share a secret s among a group of n_0 users such that the secret is secure against a *mobile* adversary, and allow any group of n_0-t users to recover the secret, where t denotes a threshold. The proactive security means that the execution of the protocol is divided into epochs, and a mobile adversary is allowed to corrupt users across all epochs, under the condition that no more than a threshold number of users are corrupted in any given epoch [41]. The Share and Open protocols can be realized via a secret sharing scheme (SSS) (e.g., Shamir's [46]). The Redistribute protocol prevents the mobile adversary from disclosing or destroying the secret and allows the set of the users and the threshold to change. The related works are referred to the supplemental material (Section III) [2]. Assuming that for each epoch i, no more than t users are corrupted, the following three properties hold:

- *Termination:* All honest users engaged in the protocol complete each execution of Share, Redistribute, and Open.
- *Correctness:* All honest users output a secret s' upon completing of Open, such that $s' = s$ if the dealer was honest during the execution of Share.
- *Secrecy:* If the dealer is honest, then s leaks no information to the adversary.

The definition described in [8] is for information-theoretically (or perfectly) secure protocols. We merely require DPSS scheme to be computationally secure in this work. Dynamic allows the set of users in a group (or committee) to be dynamically changed, which is useful in the permissionless blockchains. The Redistribute protocol has two processes: resharing the key shares to change the committee membership and threshold, updating the key shares across epochs to tackle mobile adversary.

- *Resharing the Key Shares [18].* We rely on a bivariate polynomial to share a secret s: $f(x, y) = \underline{s} + a_{0,1}x + a_{1,0}y + a_{1,1}xy + \cdots + a_{t_x,t_y}x^{t_x}y^{t_y}$, where t_x, t_y denote different thresholds. So there are two ways to share the secret s:
  1) If we fix $y = 0$, then the key shares include $\{f(i_0, 0), f(i_1, 0), \cdots, f(i_{t_x}, 0)\}$;
  2) If we fix $x = 0$, then the key shares include $\{f(0, j_0), f(0, j_1), \cdots, f(0, j_{t_y})\}$.

  We show how to transfer the ownership of the shareholders from committee A to committee B. First, we distribute key shares $\{f(i, y)\}$ to all users in committee A. Second, each user in committee A generates a set of temporary shares by running Shamir's SSS [46] on his own key share. In other words, his key share is the secret for SSS. Third, users in committee A send those temporary shares to users in committee B. Now, users in the committee B accumulate the received temporary shares and obtain another form of key shares $\{f(x, j)\}$ via interpolation of t_y temporary shares. To this end, the transfer between the two committees is successful. Note that either key shares $\{f(i, y)\}$ or $\{f(x, j)\}$ can be used to recover the secret s.
- *Updating the Key Shares [26].* Suppose that a bivariate polynomial is used to share the secret s: $f(x, y) = \underline{s} + a_{0,1}x + a_{1,0}y + a_{1,1}xy + a_{0,2}x^2 + a_{2,0}y^2 + a_{2,2}x^2y^2 + \cdots + a_{t_x,t_y}x^{t_x}y^{t_y}$. To update $f(x, y)$, we need another bivariate polynomial: $f'(x, y) = \underline{0} + a'_{0,1}x + a'_{1,0}y + a'_{1,1}xy + \cdots + a'_{t_x,t_y}x^{t_x}y^{t_y}$, which takes 0 as the secret. The reason is that the secret s in $f(x, y)$ will not be changed after updating by $f'(x, y)$. A crucial point is, we allow users in a new committee to collaboratively generate a polynomial $f'(x, y)$, thus the shareholders between old and new committees become independent. Note that t_x may not equal to t_y because the threshold between committees can be different, and we call it asymmetric bivariate polynomial.

C. Polynomial Commitments

We adapt a polynomial commitment scheme [28] to ensure it works in the asymmetric pairings.

- Setup$(1^\lambda, t)$: It takes a security parameter λ and t as input, outputs a key pair $(\mathtt{msk}, \mathtt{mpk})$, where $\mathtt{msk} = \alpha$, $\mathtt{mpk} = (g, g^\alpha, \cdots, g^{\alpha^t}, h, h^\alpha, \hat{\mathsf{e}})$.
- Commit$(\mathtt{mpk}, f(x))$: It takes the public key $\mathtt{mpk}$, and a polynomial $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_tx^t$ as input, outputs $C = \prod_{j=0}^{t}(g^{\alpha^j})^{a_j}$ as the commitment to $f(x)$.
- CreateWitness$(\mathtt{mpk}, C, f(x))$: It takes the public key $\mathtt{mpk}$, and the polynomial $f(x)$ as input, outputs a tuple $(i, f(i), w_i)$. Specifically, it computes a polynomial $\frac{f(x)-f(i)}{x-i}$ (note that the coefficients of the resulting quotient polynomial are $(\widehat{a_0}, \widehat{a_1}, \cdots, \widehat{a_t})$), and a witness $w_i = \prod_{j=0}^{t}(g^{\alpha^j})^{\widehat{a_j}}$.
- VerifyEval$(\mathtt{mpk}, C, i, f(i), w_i)$: It takes the public key $\mathtt{mpk}$, a commitment C, and the tuple $(i, f(i), w_i)$ as input, outputs 1 if $\hat{\mathsf{e}}(C/g^{f(i)}, h) = \hat{\mathsf{e}}(w_i, h^\alpha/h^i)$.

The witness w_i proves that $f(i)$ is a correct evaluation at $i \in \mathbb{Z}_q$, without revealing the polynomial $f(x)$. If the KZG commitment scheme is used in DPSS, the committee members can be held accountable in a committee. In particular, the KZG commitment scheme is publicly verifiable if we append commitments and witnesses to the blockchain. The appended commitments and witnesses can be confirmed in the blockchain using Proof-of-Work (PoW) consensus (it is not difficult to extend this assumption to other consensus like Proof of Stake [3], Proof of Space [22]).

IV. The Proposed Construction

In this section, we present the system model for accountable fine-grained blockchain rewriting in the permissionless setting. Next, we present the definition and the generic construction of APC^{2}H, respectively.

A. System Model

The system model involves three types of entities: user, modifier, and miner, in which the entities can intersect, such as a user can be a modifier and/or a miner. The communication model considers both on-chain and off-chain settings. The on-chain setting is the permissionless blockchain, where *read* is public, but *write* is granted to anyone who can show PoW. The off-chain setting assumes that every user has a point-to-point (P2P) channel with every other users. One may use Tor or transaction ghosting to establish a P2P channel [35]. Such

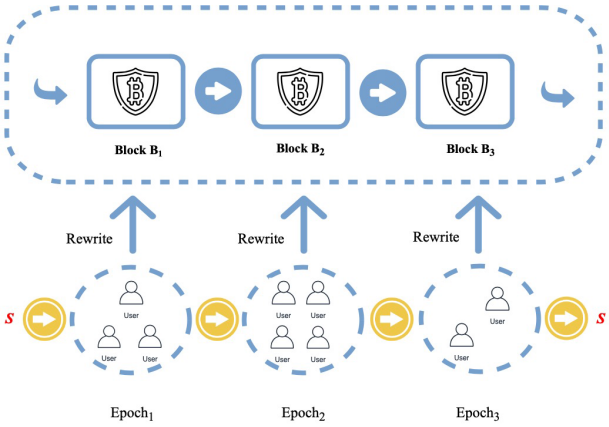

Fig. 1. Blockchain rewriting with dynamic committees. Users may join in or leave from a committee, and a designated modifier in a committee may rewrite the blockchain. The secret s remains fixed across different committees.

P2P channel works in a synchronous model, i.e., any message sent via this channel is received within a known bounded time-period. To setup the P2P channel, decentralized identifier (so called DID [34]) can be used to address centralization or identity information. DID provides an approach to achieve decentralized identity or self-sovereign identity. Specifically, the users can gather and manage their own credentials using self-created DIDs.

The system proceeds in fixed time periods called epochs. In the first epoch, a committee election protocol (e.g., Algorand's $BA*$ protocol [25], or other methods [33], [52]) is executed, so that a set of users can agree on an initial committee with Byzantine fault tolerance (e.g., up to 1/3 malicious members). The secret s in the initial committee can be generated by an honest user or in a distributed fashion [24]. Similarly, the setup of the commitment scheme can be performed by an honest user in the initial committee.

In Figure 1, a blockchain is generated by users who append their hashed contents to the blockchain. Later, modifiers with sufficient rewriting privileges are required to rewrite the hashed contents. We stress that the link of hash-chain remains intact after rewriting, and the secret remains fixed across different committees. We assume at most n users (i.e., protocol participants) exist in each epoch. We consider k dynamic committees, each of which has a varying number of committee members, and we denote $n_0 \leq n$ as a committee's size. The parameters n and k are independent. We also consider dynamic churn (i.e., join/leave) of the protocol participants. In particular, we do not assume that k committees exist in each different epoch (or we allow several committees to exist in the same epoch).

Remark: To prevent a malicious user from controlling a committee by launching Sybil attacks [20], we rely on the PoW-based identity generation mechanism [6], [33]. The mechanism allows all users to establish their identities in a committee, yet limiting the number of Sybil identities created by a malicious user. In Elastico [33], each user locally generates/establishes an identity consisting of a public key, an IP address, and a PoW solution. The user must solve a PoW puzzle which has publicly verifiable solutions to generate the final component of the identity. A PoW solution also allows other committee members to verify and accept the identity of a user. Because solving PoW requires computation, the number of identities that the malicious user can create is limited by a fraction of malicious computational power. One can refer to [33], [51], and [52] for the detailed discussion on Byzantine fault resiliency.

B. Definition

We assume that every user has a key pair $(\mathtt{sk}, \mathtt{pk})$ and that users' public keys are known to all system users. Each modifier possesses a set of attributes, and more than a threshold number of users in a committee can collectively grant a rewriting privilege to a modifier based on their attribute set. We assume a user with $\mathtt{pk}$ creates a transaction T, including a chameleon hash, a ciphertext that is labeled with a set of attributes, and a signature (i.e., signs T using his secret key $\mathtt{sk}$). A modifier with $\mathtt{pk}'$ who is granted the rewriting privilege from a committee, can rewrite the transaction T. During rewriting, the modifier signs the modified transaction using her secret key $\mathtt{sk}'$. We stress that our hash algorithm is a public process, anyone with a key pair can create a chameleon hash value. The adapt algorithm is a private process, such that a modifier with another key pair and sufficient rewriting privilege may adapt the chameleon hash value. We present the formal definition of APC^{2}H, which includes the following algorithms.

- $\mathsf{Setup}(1^\lambda)$: It takes a security parameter λ as input, outputs a master key pair $(\mathtt{msk}, \mathtt{mpk})$. Note that $\mathtt{msk}$ is shared among an initial committee C_0.
- $\mathsf{KeyGen}(\mathsf{C}_i, \Lambda)$: It takes a committee C_i, and a policy Λ as input, outputs a secret key $\mathtt{sk}_{\Lambda_i}$. The committee index $i \in \{1, \cdots, k\}$, where k denotes the total number of committees.
- $\mathsf{Hash}(\mathtt{mpk}, \mathtt{sk}, m, \delta, j)$: It takes the master public key $\mathtt{mpk}$, a secret key $\mathtt{sk}$, a message $m \in \mathcal{M}$, a set of attributes $\delta \subseteq \mathcal{U}$, and an index $j \in \{1, \cdots, k+1\}$ as input, outputs a chameleon hash h, a randomness r, and a signature σ. Note that $\mathcal{M} = \{0,1\}^*$ denotes a general message space.
- $\mathsf{Verify}(\mathtt{mpk}, \mathtt{pk}, h, m, r, \sigma)$: It takes the master public key $\mathtt{mpk}$, a public key $\mathtt{pk}$, chameleon hash h, message m, randomness r, signature σ as input, output a bit $b \in \{0,1\}$.
- $\mathsf{Adapt}(\mathtt{sk}_{\Lambda_i}, \mathtt{sk}', h, m, m', r, \sigma)$: It takes the secret key $\mathtt{sk}_{\Lambda_i}$, a secret key $\mathtt{sk}'$, chameleon hash h, messages m and m', randomness r, and signature σ as input, outputs r' and σ' if $1 = \Lambda(\delta)$ and $j \leq i$.
- $\mathsf{Judge}(\mathtt{mpk}, T, T', \mathcal{O})$: It takes the master public key $\mathtt{mpk}$, two transactions (T, T'), and an access device $\mathcal{O}$ that involved users' secret keys as input, outputs a transaction-committee pair (T', C_i), where $T' = (h, m', r', \sigma')$. It means a user with a secret key from committee C_i has modified transaction $T = (h, m, r, \sigma)$.

Correctness. The *correctness* is held if: 1) For all λ, for all $\delta \in \mathcal{U}$, all keys $(\mathtt{msk}, \mathtt{mpk}) \leftarrow \mathsf{Setup}(1^\lambda)$, for all $\delta \in \Lambda$, for all $j \leq i$, for all $\mathtt{sk}_{\Lambda_i} \leftarrow \mathsf{KeyGen}(\mathsf{C}_i, \Lambda)$, for all $m \in \mathcal{M}$, for all $(h, r, \sigma) \leftarrow \mathsf{Hash}(\mathtt{mpk}, \mathtt{sk}, m, \delta, j)$, for all $m' \in \mathcal{M}$, for all $(r', \sigma') \leftarrow \mathsf{Adapt}(\mathtt{sk}_{\Lambda_i}, \mathtt{sk}', m, m', h, r, \sigma)$, we have $1 = \mathsf{Verify}(\mathtt{mpk}, \mathtt{pk}, h, m, r, \sigma) = \mathsf{Verify}(\mathtt{mpk}, \mathtt{pk}', h, m', r', \sigma')$. 2) The modified transaction is linked to a user and a com-

mittee $(T', \mathsf{C}_i) \leftarrow \mathsf{Judge}(\mathtt{mpk}, T, T', \mathcal{O})$. An APC^{2}H should satisfy indistinguishability, adaptive collision-resistance, and accountability. We defer the formal security definitions to the full version of our paper (Section IV) [2].

C. Generic Construction

The proposed APC^{2}H consists of the following building blocks.

- A chameleon hash scheme $\mathsf{CH} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Hash}, \mathsf{Verify}, \mathsf{Adapt})$.
- An attribute-based encryption scheme with public traceability $\mathsf{ABET} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$.
- A dynamic proactive secret sharing scheme $\mathsf{DPSS} = (\mathsf{Share}, \mathsf{Redistribute}, \mathsf{Open})$.
- A digital signature scheme $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$.

We assume the t-out-of-n_0 DPSS scheme to be executed over off-chain P2P channels and let all k committees have the same parameters (t, n_0). We define a function $F(a, b) = F(a) \bigodot F(b)$, where $\bigodot$ denotes a modular multiplication. For example, $F(\mathtt{sk}, \mathtt{sk}') = g^{\mathtt{sk}+\mathtt{sk}'} = g^{\mathtt{sk}} \cdot g^{\mathtt{sk}'}$.

We sketch the overall idea of APC^{2}H here. First, a user runs the setup of CH, ABET and Σ. In particular, the user runs the sharing protocol of DPSS so that each user in an initial committee holds a key share. Second, a threshold number of committee members run the open protocol of DPSS and the key generation of ABET to issue a secret key for a modifier based on an access policy. The modifier can be either one of the committee members or an external party. Also, any committee member may run the redistribute protocol of DPSS to redistribute key shares across committees. Third, any user may run the key generation and the hash processes of CH, the encryption process of ABET, and the signing process of Σ to obtain a hash value. Fourth, the modifier runs the verification processes of CH and Σ, the decryption process of ABET, the adapt process of CH, and the signing process of Σ to find a collision. Finally, any user may run the verification processes of CH and Σ, and the tracing process of ABET to identify a responsible modifier given a modified transaction. The proposed APC^{2}H is shown below.

- $\mathsf{Setup}(1^\lambda)$: A user takes a security parameter λ as input, outputs a public parameter $\mathsf{PP} = (\mathtt{mpk}_{\mathsf{ABET}}, \mathsf{PP}_\Sigma, \mathsf{PP}_{\mathsf{CH}})$, and a secret key $\mathtt{msk}_{\mathsf{ABET}}$, where $(\mathtt{msk}_{\mathsf{ABET}}, \mathtt{mpk}_{\mathsf{ABET}}) \leftarrow \mathsf{Setup}_{\mathsf{ABET}}(1^\lambda)$, $\mathsf{PP}_\Sigma \leftarrow \mathsf{Setup}_\Sigma(1^\lambda)$, $\mathsf{PP}_{\mathsf{CH}} \leftarrow \mathsf{Setup}_{\mathsf{CH}}(1^\lambda)$. The key shares $\{f(x, y)_0\} \leftarrow \mathsf{Share}_{\mathsf{DPSS}}(\mathtt{msk}_{\mathsf{ABET}})$ are distributed to users within committee C_0, where each user holds a key share. Besides, each user holds a key pair $(\mathtt{sk}, \mathtt{pk}) \leftarrow \mathsf{KeyGen}_\Sigma(\mathsf{PP}_\Sigma)$.
- $\mathsf{KeyGen}(\mathsf{C}_i, \Lambda)$: A group of t+1 users in committee C_i take their key shares $\{f(x, y)_i\}^{t+1}$ and a policy Λ as input, output a secret key $\mathtt{sk}_{\Lambda_i}$ for a modifier, where $\mathtt{sk}_{\Lambda_i} \leftarrow \mathsf{KeyGen}_{\mathsf{ABET}}(\mathtt{msk}_{\mathsf{ABET}}, \Lambda)$, $\mathtt{msk}_{\mathsf{ABET}} \leftarrow \mathsf{Open}_{\mathsf{DPSS}}(\{f(x, y)_i\}^{t+1})$, and key shares $\{f(x, y)_i\} \leftarrow \mathsf{Redistribute}_{\mathsf{DPSS}}(\{f(x, y)_{i-1}\})$.
- $\mathsf{Hash}(\mathsf{PP}, \mathtt{sk}, m, \delta, j)$: A user appends a message m, a set of attributes δ, and an index j to the blockchain, performs the following operations
  1) generate a chameleon hash $(h_{\mathsf{CH}}, \mathsf{r}) \leftarrow \mathsf{Hash}_{\mathsf{CH}}(\mathtt{pk}_{\mathsf{CH}}, m)$, where $(\mathtt{sk}_{\mathsf{CH}}, \mathtt{pk}_{\mathsf{CH}}) \leftarrow \mathsf{KeyGen}_{\mathsf{CH}}(\mathsf{PP}_{\mathsf{CH}})$.
  2) generate a ciphertext $C \leftarrow \mathsf{Enc}_{\mathsf{ABET}}(\mathtt{mpk}_{\mathsf{ABET}}, \mathtt{sk}_{\mathsf{CH}}, \delta, j)$, where $\mathtt{sk}_{\mathsf{CH}}$ denotes the encrypted message.
  3) generate a message-signature pair (c, σ_Σ), where $\sigma_\Sigma \leftarrow \mathsf{Sign}_\Sigma(\mathtt{sk}, c)$, and $c \leftarrow F(\mathtt{sk}, \mathtt{sk}_{\mathsf{CH}})$.
  4) output $(h, m, \mathsf{r}, \sigma)$, where $h \leftarrow (h_{\mathsf{CH}}, \mathtt{pk}_{\mathsf{CH}}, C)$, and $\sigma \leftarrow (c, \sigma_\Sigma)$.
- $\mathsf{Verify}(\mathsf{PP}, \mathtt{pk}, h, m, \mathsf{r}, \sigma)$: It outputs 1 if $1 \leftarrow \mathsf{Verify}_{\mathsf{CH}}(\mathtt{pk}_{\mathsf{CH}}, m, h_{\mathsf{CH}}, \mathsf{r})$ and $1 \leftarrow \mathsf{Verify}_\Sigma(\mathtt{pk}, c, \sigma_\Sigma)$, and 0 otherwise.
- $\mathsf{Adapt}(\mathtt{sk}_{\Lambda_i}, \mathtt{sk}', h, m, m', \mathsf{r}, \sigma)$: A modifier with a secret key $\mathtt{sk}_{\Lambda_i}$ and a new message m', performs the following operations
  1) check $1 \stackrel{?}{=} \mathsf{Verify}(\mathsf{PP}, \mathtt{pk}, h, m, \mathsf{r}, \sigma)$.
  2) compute $\mathtt{sk}_{\mathsf{CH}} \leftarrow \mathsf{Dec}_{\mathsf{ABET}}(\mathtt{mpk}_{\mathsf{ABET}}, C, \mathtt{sk}_{\Lambda_i})$.
  3) compute a new randomness $\mathsf{r}' \leftarrow \mathsf{Adapt}_{\mathsf{CH}}(\mathtt{sk}_{\mathsf{CH}}, m, m', h, \mathsf{r})$.
  4) generate a ciphertext $C' \leftarrow \mathsf{Enc}_{\mathsf{ABET}}(\mathtt{mpk}_{\mathsf{ABET}}, \mathtt{sk}_{\mathsf{CH}}, \delta, j)$.
  5) generate a message-signature pair (c', σ'_Σ), where $\sigma'_\Sigma \leftarrow \mathsf{Sign}_\Sigma(\mathtt{sk}', c')$, and $c' \leftarrow F(\mathtt{sk}', \mathtt{sk}_{\mathsf{CH}})$.
  6) output $(h, m', \mathsf{r}', \sigma')$, where $h \leftarrow (h_{\mathsf{CH}}, \mathtt{pk}_{\mathsf{CH}}, C')$, and $\sigma' \leftarrow (c', \sigma'_\Sigma)$.
- $\mathsf{Judge}(\mathsf{PP}, T, T', \mathcal{O})$: It takes the public parameter PP, two transactions (T, T'), and an access device $\mathcal{O}$ as input, outputs a transaction-committee pair (T', C_i) if the modified transaction T' links to a committee C_i, where $T' = (h, m', \mathsf{r}', \sigma')$.

Correctness of the Judge protocol. The Judge algorithm allows any public user to identify the responsible modifiers and their rewriting privileges given a modified transaction. The modified transaction can be easily linked to the modifier (or modifier's public key) because a digital signature scheme is used in the construction. Below, we explain the connection between the modified transaction and the responsible rewriting privileges (or committee indexes).

First, any public user verifies a connection between a transaction T and its modified version T'. The connection can be established, since both message-signature pair (c, σ_Σ) in T and message-signature pair (c', σ'_Σ) in T', are derived from the same chameleon trapdoor $\mathtt{sk}_{\mathsf{CH}}$. We require the underlying Σ scheme (e.g., Schnorr signature [45] and Waters signature [50]) to have homomorphic property regarding keys and signatures [17]. We rely on this homomorphic property to find connections between a transaction and its modified versions. The chameleon trapdoor $\mathtt{sk}_{\mathsf{CH}}$ is used in many modified versions of a mutable transaction because different modifiers may modify the same transaction. Here, we consider a single modified transaction T' for simplicity.

Second, any public user obtains a set of accused committees from interacting with an access device $\mathcal{O}$, such that $\{1, \cdots, k\} \leftarrow \mathsf{Trace}_{\mathsf{ABET}}(\mathtt{mpk}_{\mathsf{ABET}}, \mathcal{O}, \epsilon)$. Specifically, the public sends an indexed ciphertext encrypting a message under a set of attributes (which satisfies the access policy involved in $\mathcal{O}$) to $\mathcal{O}$. The public outputs the committee index (we call it

accused committee) if decryption succeeds. The public repeats this tracing process until output a set of accused committees.

Third, if a user with $\mathtt{pk}'$ acts as a modifier in an accused committee, the public outputs (T', C_i). It means that a transaction T' is indeed modified by the user $\mathtt{pk}'$ whose rewriting privilege is granted from committee C_i. Because we allow the commitment scheme to be used in DPSS, the user $\mathtt{pk}'$ is held accountable in a committee. More specifically, user $\mathtt{pk}'$ joins in committee C_i by showing a commitment on her key share to other committee members, and further detail is given in the instantiation. If user $\mathtt{pk}'$ acts as modifiers for many accused committees, the public outputs $(T', \{\mathsf{C}_i\})$. However, if user $\mathtt{pk}'$ did not join in any accused committees, the public still outputs the indexes of the accused committees. This is the second case of blockchain rewriting: an unauthorized user has no granted rewriting privileges from any committee but uses an access device to rewrite mutable transactions.

To conclude, we achieve public accountability via three steps: 1) Verify a modified transaction; 2) Find an accused committee; 3) Link the modified transaction to the accused committee. We also consider that a committee may have multiple modifiers equipped with different rewriting privileges, but they should have the same committee index. In this case, the public still identifies the responsible modifiers in the same committee as the modifiers are required to sign the modified transactions.

Security analysis of APC^{2}H. The indistinguishability of APC^{2}H can be straightforwardly reduced to the indistinguishability of the underlying CH. Similarly, the accountability of APC^{2}H can be easily reduced to the correctness of the DPSS and the unforgeability of Σ. Adaptive collision-resistance can be proven by reductions to the CCA security of ABET, the secrecy of DPSS, and the collision-resistance of CH. One reduction involves CCA security, such that a simulator replaces the encrypted chameleon secret key with an empty value. During the simulation, the simulator answers key generation queries honestly using the key generation oracle provided by the corresponding challenger. For adapt queries, the simulator can decrypt the ciphertext from the adversary using the decryption oracle provided by the challenger and simulate collisions. To simulate the challenge hash query, the simulator submits two values $(\mathtt{sk}_{\mathsf{CH}}, \perp)$ to the challenger and obtains a challenge ciphertext C^*. For the challenge adapt query, the simulator simulates a collision using $\mathtt{sk}_{\mathsf{CH}}$ without decrypting C^*. A detailed security analysis of APC^{2}H is referred to the supplemental material (Section IV) [2].

V. Instantiation and Implementation

In this section, we explain the proposed ABET scheme first. Then, we show the proposed instantiation, give the implementation and evaluation analysis.

A. The Proposed ABET Scheme

For constructing a practical ABET, we require that the underlying KP-ABE scheme should have a minimal number of elements in master secret key, while the size of the ciphertext is constant (i.e., independent of the number of committees). Therefore, we rely on a KP-ABE scheme [44] and a hierarchical identity-based encryption (HIBE) scheme [11] to construct our ABET scheme, respectively. First, the KP-ABE [44] can be viewed as the stepping stone to construct ABET. Its master secret key has a single element, which requires a single execution of the DPSS. Its security is based on q-type assumption in the standard model, and it works in prime-order group. One may use more efficient ABE schemes such as [4] and [43]. However, the master secret key in the ABE scheme [4] includes several elements, which requires several executions of the DPSS. The master secret key in a recent ABE scheme [43] contains one element only, but it is proven secure in the generic group model. Constructing the practical ABET scheme based on the recent ABE scheme [43] would be a promising future work [43]. Second, the HIBE scheme [11] has constant-size ciphertext. Specifically, the ciphertext has just three group elements, and the decryption requires only two pairing operations. In particular, the HIBE's master secret key has one element, which can be shared with KP-ABE. We note that several ABET schemes have been proposed in [31], [32], [40] and [48], but they are ciphertext-policy ABE based. They cannot be applied to open blockchains with decentralized access control, such that a committee of multiple users share a common secret and manage access control. Our proposed ABET scheme makes it possible based on the KP-ABE scheme.

The ABET scheme described above lacks anonymity because its ciphertext reveals committee's index information to the public. As a result, the index-hiding required in the ABET scheme cannot be held (see Section III-A). We realize the ABET scheme with ciphertext anonymity using asymmetric pairings, i.e., $\hat{\mathsf{e}} : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$ (which is used in [21]). The basic idea is, the index-based elements in a modifier's decryption key belong to group $\mathbb{G}$. The index-based elements in a ciphertext belong to group $\mathbb{H}$ so that the ciphertext can hide the committee's index if the master secret key is unknown. The proposed ABET scheme can be found in the instantiation. The security analysis is referred to the supplemental material (Section II) [2].

B. Instantiation

First, we modify the discrete logarithm (DL)-based chameleon hash [30] and apply it to our instantiation. Specifically, we modify the hash process as $h = g^r \cdot \mathtt{pk}^m$, and the adapt process becomes $r' = r + (m - m') \cdot \mathtt{sk}$, where $\mathtt{pk} = g^{\mathtt{sk}}$. The adapt process will have a better performance compared to [30] because the cost of multiplication is lower than the division. Second, we use the proposed ABET scheme to construct our instantiation. Specifically, the Setup and KeyGen of ABET are directly used in the instantiation. The Enc, Dec and Trace of ABET are presented in the step (2) of Hash, Adapt and Judge, respectively. Third, we rely on a recent work [35] to initiate the DPSS scheme. We particularly show an instantiation of DPSS with a KZG commitment scheme [28], which allows users to be held accountable in a committee. Last, we use Schnorr signature [45] to instantiate digital signature scheme due to its inherent homomorphic properties.

We denote an index space as $\{I_1, \cdots, I_k\} \in (\mathbb{Z}_q)^k$, which is associated with k committees. We define a hierarchy as follows: index i is close to the root node k, and index j is close to the leaf node. We assume each committee has n_0 users, and the threshold is t, where $t < n_0/2$ according to [35]. Let $\mathtt{H} : \{0,1\}^* \to \mathbb{Z}_q$ be a hash function, and the size of hash output $\mathtt{H}$ is l. The concrete instantiation is shown below.

- $\mathsf{Setup}(1^\lambda)$: It takes a security parameter λ as input, outputs a master public key $\mathtt{mpk} = (g, u, v, w, h, \hat{\mathtt{e}}(g,h)^\alpha, \{g_1^\alpha, \cdots g_k^\alpha\}, \{h_1^\alpha, \cdots h_k^\alpha\}, g^\beta, h^{1/\alpha}, h^{\beta/\alpha}, \hat{\mathtt{e}}(g,h)^{\theta/\alpha})$, and a master secret key $\mathtt{msk} = (\alpha, \beta, \theta)$, where $(\alpha, \beta, \theta) \in \mathbb{Z}_q^*$ $\{z_1, \cdots, z_k\} \in \mathbb{Z}_q$, $(u, v, w) \in \mathbb{G}$, $\{g_1, \cdots, g_k\} = \{g^{z_1}, \cdots, g^{z_k}\}$, $\{h_1, \cdots, h_k\} = \{h^{z_1}, \cdots, h^{z_k}\}$. The master key generation is essentially the same as Setup of ABET.
  Regarding DPSS, the key shares of α and θ are distributed to users in committee C_0 by running the Share protocol of the DPSS scheme.
- $\mathsf{KeyGen}(\mathsf{C}_i, (\mathbf{M}, \pi))$: It inputs a committee C_i with index $(I_1, \cdots, I_i)$, and an access policy $(\mathbf{M}, \pi)$ ($\mathbf{M}$ has n_1 rows and n_2 columns, $\pi : \{1, \cdots, n_1\} \to \mathcal{U}$ denotes a mapping function), outputs a secret key $\mathtt{sk}_{\Lambda_i}$ for a modifier. Specifically, a group of $t+1$ members in committee C_i first recover secrets α and θ. Then, they pick $\{t_1, \cdots, t_{n_1}\} \in \mathbb{Z}_q$, for all $\tau \in [n_1]$, compute $\mathtt{sk}_{(\tau,1)} = g^{s_\tau} w^{t_\tau}$, $\mathtt{sk}_{(\tau,2)} = (u^{\pi(\tau)} v)^{-t_\tau}$, $\mathtt{sk}_{(\tau,3)} = h^{t_\tau}$, where s_τ is a key share from α. Eventually, they pick $\{r_1, \cdots, r_{n_1}\} \in \mathbb{Z}_q$, compute $\mathtt{sk}_0 = (g^{t^*/\alpha}, g^{r^*})$, $\mathtt{sk}_1 = g^\theta \cdot \widehat{i}^{t^*} \cdot g^{\beta \cdot r^*}$, $\mathtt{sk}_2 = \{g_{i-1}^{\alpha \cdot t^*}, \cdots g_1^{\alpha \cdot t^*}, g^{1/\alpha}\}$ (note that $g^{1/\alpha}$ is used only for delegation[1]), where $t^* = \sum_{\tau \in |n_1|}(t_\tau)$, $r^* = \sum_{\tau \in |n_1|}(r_\tau)$, and $\widehat{i} = g_k^{\alpha I_1} \cdots g_i^{\alpha I_i} \cdot g \in \mathbb{G}$ is associated with the committee C_i indexed $(I_1, \cdots, I_i)$. The secret key is $\mathtt{sk}_{\Lambda_i} = (\{\mathtt{sk}_\tau\}_{\tau \in [n_1]}, \mathtt{sk}_0, \mathtt{sk}_1, \mathtt{sk}_2)$. The secret key generation is essentially the same as KeyGen of ABET.
  Regarding DPSS, a group of $t+1$ members should recover the secrets α and θ by running the Open protocol of the DPSS scheme, prior to issuing a secret key to a modifier. Besides, the key shares of α and θ can be redistributed between committees by running the $\mathsf{Redistribute}$ protocol of the DPSS scheme (see the correctness of $\mathsf{Redistribute}$ protocol below).
- $\mathsf{Hash}(\mathtt{mpk}, \mathtt{sk}, m, \delta, j)$: To hash a message $m \in \mathbb{Z}_q$ under a set of attributes δ, and an index $(I_1, \cdots I_j)$, a user performs the following operations
  1) choose a randomness $\mathsf{r} \in \mathbb{Z}_q^*$, and a trapdoor R, compute a chameleon hash $b = g^{\mathsf{r}} \cdot p'^m$ where $p' = g^e$, $e = \mathtt{H}(\mathsf{R})$. Note that R denotes a short bit-string.
  2) generate a ciphertext on message $M = \mathsf{R}$ under a set of attributes $\delta = \{A_1, \cdots, A_{|\delta|}\}$ and index $(I_1, \cdots I_j)$. It first picks $s, r_1, r_2, \cdots, r_{|\delta|} \in \mathbb{Z}_q$, for $\tau \in |\delta|$ computes $ct_{(\tau,1)} = h^{r_\tau}$ and $ct_{(\tau,2)} = (u^{A_\tau} v)^{r_\tau} w^{-s}$. Then, it computes $ct = (\mathsf{R}||0^{l-|\mathsf{R}|}) \oplus \mathtt{H}(\hat{\mathtt{e}}(g,h)^{\alpha s}) \oplus \mathtt{H}(\hat{\mathtt{e}}(g,h)^{\theta s/\alpha})$, $ct_0 = (h^s, h^{s/\alpha}, h^{\beta \cdot s/\alpha})$, and $ct_1 = \widehat{j}^s$, where $\widehat{j} = h_k^{\alpha I_1} \cdots h_j^{\alpha I_j} \cdot h \in \mathbb{H}$. Eventually, it sets $C = (ct, \{ct_{(\tau,1)}, ct_{(\tau,2)}\}_{\tau \in [\delta]}, ct_0, ct_1)$.
  3) generate a signature $epk = g^{esk}, \sigma = esk + \mathtt{sk} \cdot \mathtt{H}(epk||c)$, where (esk, epk) denotes an ephemeral key pair, and $c = g^{\mathtt{sk} + (\mathsf{R}||0^{l-|\mathsf{R}|})}$ denotes a signed message.
  4) output $(m, p', b, \mathsf{r}, C, c, epk, \sigma)$.
- $\mathsf{Verify}(\mathtt{mpk}, \mathtt{pk}, m, p', b, \mathsf{r}, c, epk, \sigma)$: Anyone can verify whether a given hash (b, p') is valid, it outputs 1 if $b = g^{\mathsf{r}} \cdot p'^m$, and $g^\sigma = epk \cdot \mathtt{pk}^{\mathtt{H}(epk||c)}$.
- $\mathsf{Adapt}(\mathtt{sk}_{\Lambda_i}, \mathtt{sk}', m, m', p', b, \mathsf{r}, C, c, epk, \sigma)$: A modifier with a secret key $\mathtt{sk}_{\Lambda_i}$, and a new message $m' \in \mathbb{Z}_q$, performs the following operations
  1) check $1 \stackrel{?}{=} \mathsf{Verify}(\mathtt{mpk}, \mathtt{pk}, m, p', b, \mathsf{r}, c, epk, \sigma)$.
  2) run the following steps to obtain trapdoor R:
     a) generate a delegated key w.r.t an index $(I_1, \cdots I_{i+1})$. It picks $t' \in \mathbb{Z}_q$, computes $\mathtt{sk}_0 = (g^{(t^*+t')/\alpha}, g^{r^*})$, $\mathtt{sk}_1 = g^\theta \cdot \widehat{i}^{t^*} \cdot g^{\beta \cdot r^*} \cdot (g_{i-1}^{\alpha \cdot t^*})^{I_{i+1}} \cdot (g_k^{\alpha \cdot I_1} \cdots g_{i-1}^{\alpha \cdot I_{i+1}} \cdot g)^{t'}$, $\mathtt{sk}_2 = \{g_{i-2}^{\alpha \cdot t^*} \cdot g_{i-2}^{\alpha \cdot t'}, \cdots g_1^{\alpha \cdot t^*} \cdot g_1^{\alpha \cdot t'}, g^{1/\alpha}\}$. The delegated secret key is $\mathtt{sk}_{\Lambda_{i+1}} = (\{\mathtt{sk}_\tau\}_{\tau \in [n_1]}, \mathtt{sk}_0, \mathtt{sk}_1, \mathtt{sk}_2)$.
     b) if the attribute set δ involved in the ciphertext satisfies the policy $(\mathbf{M}, \pi)$, then there exists constants $\{\gamma_\mu\}_{\mu \in I}$ according to [9]. It computes B as follows.

$$
\begin{aligned}
B &= \prod_{\mu \in I} (\hat{\mathtt{e}}(\mathtt{sk}_{(\mu,1)}, ct_{(0,1)}) \hat{\mathtt{e}}(\mathtt{sk}_{(\mu,2)}, ct_{(\mu,1)}) \\
&\quad \hat{\mathtt{e}}(ct_{(\mu,2)}, \mathtt{sk}_{(\mu,3)}))^{\gamma_\mu} \\
&= \hat{\mathtt{e}}(g,h)^{s \sum_{\mu \in I} \gamma_\mu s_\mu} = \hat{\mathtt{e}}(g,h)^{\alpha s}, \\
&\quad where \sum_{\mu \in I} \gamma_\mu s_\mu = \alpha.
\end{aligned}
$$

     Since the key delegation process will not affect the attribute-based components $\{\mathtt{sk}_\tau\}_{\tau \in [n_1]}$, B is computed once. Note that $ct_{(0,1)}, ct_{(0,2)}$ denote the first and second element of ct_0, and the same rule applies to $\mathtt{sk}_0$.
     c) check $(\mathsf{R}||0^{l-|\mathsf{R}|}) \stackrel{?}{=} ct \oplus \mathtt{H}(B) \oplus \mathtt{H}(A)$, where $A = \frac{\hat{\mathtt{e}}(\mathtt{sk}_1, ct_{(0,2)})}{\hat{\mathtt{e}}(\mathtt{sk}_{(0,1)}, ct_1) \hat{\mathtt{e}}(\mathtt{sk}_{(0,2)}, ct_{(0,3)})}$. The format "$||0^{l-|\mathsf{R}|}$" is used to ensure that the encrypted value R can be decrypted successfully with certainty $1 - 2^{l-|\mathsf{R}|}$. Since this technique can decide when the delegation process terminates, the hidden index j of C is known to the modifier.
  3) compute a new randomness $\mathsf{r}' = \mathsf{r} + (m - m') \cdot e$, where $e = \mathtt{H}(\mathsf{R})$.
  4) generate a new ciphertext C' on the same message $M = \mathsf{R}$ using the attribute set δ and index $(I_1, \cdots, I_j)$.
  5) generate a signature $epk' = g^{esk'}, \sigma' = esk' + \mathtt{sk}' \cdot \mathtt{H}(epk'||c')$, where $c' = g^{\mathtt{sk}' + (\mathsf{R}||0^{l-|\mathsf{R}|})}$.
  6) output $(m', p', b, \mathsf{r}', C', c', epk', \sigma')$.
- $\mathsf{Judge}(\mathtt{mpk}, T, T', \mathcal{O})$: Given two transactions (T, T') and an access device $\mathcal{O}$, where $T = (m, b, p', C, c, epk, \sigma,)$, $T' = (m', b, p', C', c', epk', \sigma')$, any public user performs the following operations.
  1) verify the connection between T' and T' as follows
     – verify chameleon hash $b = g^{\mathsf{r}} \cdot p'^m = g^{\mathsf{r}'} \cdot p'^{m'}$.

[1]We discover that the underlying ABE scheme [44] cannot support key delegation. We add $g^{1/\alpha}$ to users' decryption keys, so the key holders can privately delegate their decryption keys (similar to the technique used in [10]).

- verify message-signature pair (c, σ) under $(epk, \mathtt{pk})$, and message-signature pair (c', σ') under $(epk', \mathtt{pk}')$.
- verify $\mathtt{pk}' = \mathtt{pk} \cdot \Delta(\mathtt{sk})$, where $\Delta(\mathtt{sk}) = c'/c = g^{\mathtt{sk}'-\mathtt{sk}}$. Note that $\Delta(\mathtt{sk})$ means the difference or shift between two keys, and (c, c') are derived from the same chameleon trapdoor R.

2) obtain a set of accused committees from interacting with an access device $\mathcal{O}$. This interaction process is referred to ABET's public traceability at Section III-A.
3) output a transaction-committee pair (T', C_i). We rely on the KZG commitment and PoW consensus to hold a modifier $\mathtt{pk}'$ accountable in an accused committee C_i (see the correctness of KZG commitment below).

Correctness of the Redistribute protocol. Two secrets need to be distributed: (α, θ). We assume the readers are familiar with Shamir's SSS; thus, we omit the correctness of Share and Open protocols in our instantiation. Now, we show users in committee C_{i-1} securely *handoff* their key shares of secret α to users in committee C_i. According to the DPSS scheme in [35], an asymmetric bivariate polynomial is used: $f(x, y) = \underline{\alpha} + a_{0,1}x + a_{1,0}y + a_{1,1}xy + a_{1,2}xy^2 + \cdots + a_{t,2t}x^t y^{2t}$. Each user in committee C_{i-1} holds a *full* key share after running Share protocol. For example, a user with $\mathtt{pk}$ holds a key share $f(i, y)$, which is a polynomial with dimension t. Overall, the handoff (i.e., Redistribute protocol) includes three phases: share reduction, proactivization, and full-share distribution.

- *Share Reduction.* It requires each user in committee C_{i-1} reshares its full key share. For example, user $\mathtt{pk}$ derives a set of *reduced* shares $\{f(i, j)\}_{j\in[1,n_0]}$ from its key share $f(i, y)$ using SSS. Then, each user distributes the reduced shares to users in committee C_i, which includes a user with $\mathtt{pk}'$. As a result, each user in C_i obtains a reduced share $f(x, j)$ by interpolating the received shares $\{f(i, j)\}_{i\in[1,t]}$. Note that the dimension of $f(x, j)$ is $2t$, and $2t$+1 of these reduced key shares $\{f(x, j)\}_{j\in[1,2t+1]}$ can recover α (see Section III-B). The goal of this dimension-switching (from t to $2t$) is to achieve optimal communication overhead, such that only $2t$+1 users in committee C_i are required to update $f(x, j)$.
- *Proactivization.* It requires $F(x, j) = f(x, j) + f'(x, j)$, where $f'(x, y)$ is a new asymmetric bivariate polynomial with dimension $(t, 2t)$ and $f'(0, 0) = 0$. We provide more details of $f'(x, y)$ later.
- *Full-share Distribution.* It requires each user in committee C_i to recover its full key share with dimension t. For example, a full key share $F(i, y)$ is recovered by interpolating the reduced shares $\{F(i, j)\}_{j\in[1,2t+1]}$ in committee C_i. This full key share $F(i, y)$ belongs to user $\mathtt{pk}'$, and t+1 of these full key shares can recover α.

Now we show the generation of an asymmetric bivariate polynomial $f'(x, y)$ with dimension $(t, 2t)$ such that $f'(0, 0) = 0$, which is used to update the reduced key shares $f(x, j)$ during proactivization. We denote a subset of C_i as $\mathcal{U}'$, which includes $2t$+1 users. The generation of $f'(x, y)$ requires two steps: univariate zero share, and bivariate zero share.

- *Univariate Zero Share.* It requires each user in $\mathcal{U}'$ to generate a key share $f'_j(y)$ from a common univariate polynomial with dimension $2t$. First, each user i generates a univariate polynomial $f'_i(y) = 0 + a'_1 y + a'_2 y^2 + \cdots + a'_{2t} y^{2t}$, and broadcasts it to all users in $\mathcal{U}'$. Second, each user in $\mathcal{U}'$ generates a common univariate polynomial $f'(y) = \sum_{i\in[1,2t+1]} f'_i(y)$ by combining all received polynomials, and obtains a key share $f'_j(y)$ from $f'(y)$.
- *Bivariate Zero Share.* It requires each user in committee C_i to generate a key share $f'(x, j)$ from a common bivariate polynomial with dimension $(t, 2t)$. First, each user in $\mathcal{U}'$ generates a set of reduced shares $\{f'(i, y)\}_{i\in[1,n_0]}$ with dimension t from its key share $f'_j(y)$ (i.e., resharing process), where $f'(i, y) = 0 + a'_{1,0}y + a'_{2,0}y^2 + \cdots + a'_{2t,0}y^{2t}$. Since the reduced shares are distributed to all users in committee C_i, a common bivariate polynomial with dimension $(t, 2t)$ is established: $f'(x, y) = 0 + a'_{0,1}x + a'_{1,0}y + a'_{1,1}xy + a'_{1,2}xy^2 + \cdots + a'_{t,2t}x^t y^{2t}$. Second, each user in committee C_i obtains a reduced key share $f'(x, j)$ by interpolating the received shares $\{f'(i, j)\}_{j\in[1,2t+1]}$. The key share $f'(x, j) = 0 + a'_{0,1}x + a'_{0,2}x^2 + \cdots + a'_{0,t}x^t$ is used to update $f(x, j)$ in the proactivization.

The asymmetric bivariate polynomial $f'(x, y)$ can be reused in another proactivization when sharing secret θ. In other words, multiple handoff protocols with respect to different secrets can be updated using the same bivariate polynomial, with the condition that these handoff protocols are executed within the same committee.

Correctness of KZG commitment. We show that the committee members can be held accountable in a committee.

- *Share Reduction.* We require user $\mathtt{pk}'$ in committee C_i to generate a commitment $C_{f(x,j)}$, which is a KZG commitment to the reduced key shares $\{f(i, j)\}_{j\in[1,2t+1]}$, and a set of witnesses $\{w_{f(i,j)}\}_{j\in[1,2t+1]}$. A witness $w_{f(i,j)}$ means the witness to evaluation of $f(x, j)$ at i. Note that $i \in [1, 2t+1]$ indicates the order of user $\mathtt{pk}'$'s public key in committee C_i (we order nodes lexicographically by users' public keys and choose the first $2t + 1$).
- *Full-share Distribution.* We require user $\mathtt{pk}'$ in committee C_i to generate a commitment $C_{F(x,j)}$, which is a KZG commitment to the reduced key shares $\{F(i, j)\}_{j\in[1,2t+1]}$, and a set of witnesses $w_{F(i,j)}$. A witness $w_{F(i,j)}$ means the witness to evaluation of $F(x, j)$ at i.
- *PoW Consensus.* We require user $\mathtt{pk}'$ to hash the KZG commitment and the set of witnesses, store them to an immutable transaction, and put them on-chain for PoW consensus.

The commitment and witness can also ensure the correctness of handoff described in the DPSS scheme above. Specifically, new committee members can verify the correctness of reduced shares from old committee members, thus the correctness of dimension-switching. The proof of correctness is publicly verifiable, such that any public user can verify that $f(i, j)$ (or $F(i, j)$) is the correct evaluation at i (i.e., user $\mathtt{pk}'$) of the polynomial committed by $C_{f(x,j)}$ (or $C_{F(x,j)}$) in committee C_i.

C. Implementation and Evaluation

We evaluate the performance of the proposed solution based on a proof-of-concept implementation in Python and Flask

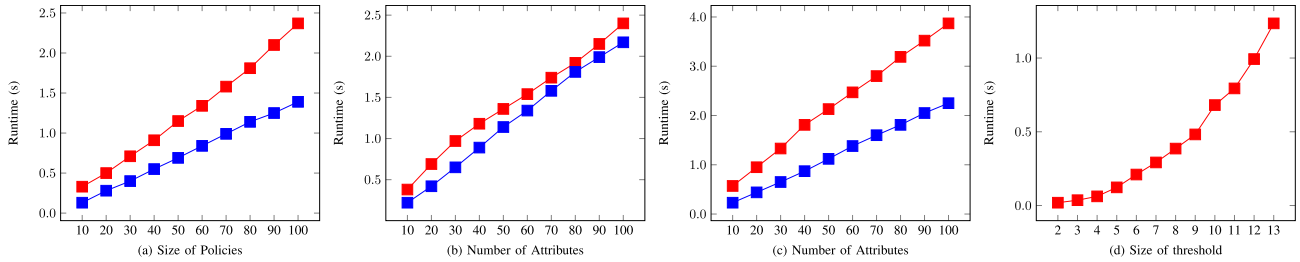

Fig. 2. Execution time of KeyGen, Hash, Adapt algorithms, and DPSS scheme [35]. Red line (with solid square) is for this work, while blue line (with solid dot) is for [48].

framework. We create a mutable open blockchain system with basic functionalities and a PoW consensus mechanism. The simulated open blockchain system is "healthy", satisfying the properties of persistence and liveness [23]. The system is specifically designed to include ten blocks, each block consists of 100 transactions. Please note, that our implementation can easily extend it to real-world applications such as a block containing 3500 transactions. We simulate ten nodes in a peer-to-peer network, each of them is implemented as a lightweight blockchain node. They can also be regarded as the users in a committee. A chain of blocks is established with PoW mechanism by consolidating transactions broadcast by the ten nodes. We show how to apply our proposed solution to open blockchains in the full version of our paper (Section V) [2].

First, if users append mutable transactions to blockchain, they use the proposed solution to hash the registered message m. Later, a miner uses the conventional hash function SHA-256 $\mathtt{H}$ to hash the chameleon hash output h and validates $\mathtt{H}(h)$ using a Merkle tree. Note that the non-hashed components such as randomness r, are parts of a mutable transaction $T = (\mathtt{pk}_{\mathsf{CH}}, m, h, \mathsf{r}, C, c, \sigma)$. As a consequence, a modifier can replace T by $T' = (\mathtt{pk}_{\mathsf{CH}}, m', h, \mathsf{r}', C', c', \sigma')$ without changing the hash output $\mathtt{H}(h)$.

Second, we mimic a dynamic committee that includes five users, we split the master secret key into five key shares so that each user in a committee holds a key share. We simulate the basic functionality of DPSS, including resharing and updating key shares. Any user can join in or leave from a committee by transmitting those key shares between committee members. In particular, we simulate three users in a committee can collaboratively recover the master secret key and grant rewriting privileges to the modifiers.

We implement our proposed solution using the Charm framework [5] and evaluate its performance on a PC with Intel Core i5 (2.7GHz×2) and 8GB RAM. We use Multiple Precision Arithmetic Library, Pairing-Based Cryptography (PBC) Library, and we choose MNT224 curve for pairing, which is the best Type-III paring in PBC. We instantiate the hash function and the pseudo-random generator with the corresponding standard interfaces provided by the Charm framework.

First, we made an overhead comparison against [16], [48], showing that our storage cost is slightly higher than [16] because our scheme includes digital signature and ABET, and it is lower than [48] if a mutable transaction involves less than 100 attributes. We also implemented the scheme in [48], showing that the execution times of our keygen, hash, and adapt algorithms are slightly higher than [48]. The execution time of KeyGen, Hash, and Adapt algorithms are measured and shown in Figure 2 (a-c).

Second, we evaluate the execution time of a t-out-of-n_0 DPSS protocol, where n_0 indicates the number of users in a committee and t is the threshold. Let $t < n_0/2$ be a safe threshold. The overhead includes the distribution cost between committee members, and the polynomial calculation cost. The evaluation in [35] provides storage overhead only, but we provide the execution time of DPSS in Figure 2 (d). Our evaluation shows that the maximal threshold is $t = 13$ if we run SSS on a standard computer. If $t = 14$, a secret share such as $f(23) = secret + (23)^1 + \cdots + (23)^{14}$ will overflow.

To conclude, the implementation performs the resharing twice and updating once regarding two shared secrets. Besides, the number of updating process is constant in a committee, independent of the number of shared secrets used in ABET. Since only two secrets are needed to be shared and recovered, we argue that the proposed ABET scheme is the most practical one if ABET includes DPSS. On the security-front, because every committee has at most $\frac{n_0}{3}$ malicious members [33] and $\frac{n_0}{2}+1$ committee members recover the shared secrets [35], the malicious committee members cannot dictate the committee and control the rewriting privileges. For the storage cost, each mutable transaction needs to store $T = (\mathtt{pk}_{\mathsf{CH}}, m, h, \mathsf{r}, C, c, \sigma)$. So, the storage cost of a mutable transaction includes: 1) $2\mathcal{L}_{\mathbb{Z}_q} + 3\mathcal{L}_{\mathbb{G}}$ regarding DL-based chameleon hash; 2) $\mathcal{L}_{\mathbb{Z}_q} + |\delta| \times \mathcal{L}_{\mathbb{G}} + (|\delta| + 4) \times \mathcal{L}_{\mathbb{H}}$ regarding ABET; 3) $\mathcal{L}_{\mathbb{Z}_q} + 2\mathcal{L}_{\mathbb{G}}$ regarding digital signature. The committee's on-chain storage cost regarding DPSS [35] is $2(t+1) \times [\mathcal{L}_{\mathbb{G}} + (2t+1)(\mathcal{L}_{\mathbb{Z}_q} + \mathcal{L}_{\mathbb{H}})]$.

VI. Conclusion

In this paper, we proposed a new framework of accountable fine-grained blockchain rewriting. The proposed framework is designed for open blockchains that require no trust assumptions. Besides, the proposed framework achieves public accountability, meaning that the modifiers' public keys and their rewriting privileges can be publicly held accountable for the modified transactions. We presented a practical instantiation, and showed that the proposed solution is suitable for open blockchain applications. In particular, the proof-of-concept implementation demonstrated that our proposed solution can be easily integrated into the existing open blockchains.

REFERENCES

[1] *General Data Protection Regulation*. [Online]. Available: https://eugdpr.org

[2] *Our Full Paper and Source Code*. [Online]. Available: https://github.com/Yangguang-Tian-Surrey/Accountable-Fine-Grained-Blockchain-Rewriting-in-the-Permissionless-Setting/tree/main

[3] *Proof of Stake*. [Online]. Available: https://en.wikipedia.org/wiki/Proof_of_stake

[4] S. Agrawal and M. Chase, "FAME: Fast attribute-based message encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 665–682.

[5] J. A. Akinyele et al., "Charm: A framework for rapidly prototyping cryptosystems," *J. Cryptograph. Eng.*, vol. 3, no. 2, pp. 111–128, Jun. 2013.

[6] M. Andrychowicz and S. Dziembowski, "Pow-based distributed cryptography with no trusted setup," in *Proc. CRYPTO*, 2015, pp. 379–399.

[7] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable blockchain-or-rewriting history in Bitcoin and friends," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Apr. 2017, pp. 111–126.

[8] J. Baron, K. El Defrawy, J. Lampkins, and R. Ostrovsky, "Communication-optimal proactive secret sharing for dynamic groups," in *Proc. ACNS*, 2015, pp. 23–41.

[9] A. Beimel, "Secure schemes for secret sharing and key distribution," M.S. thesis, Israel Inst. Technol., Technion, Haifa, Israel, 1996.

[10] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy*, May 2007, pp. 321–334.

[11] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *Proc. CRYPTO*, 2005, pp. 440–456.

[12] D. Boneh, A. Sahai, and B. Waters, "Fully collusion resistant traitor tracing with short ciphertexts and private keys," in *Proc. EUROCRYPT*, 2006, pp. 573–592.

[13] D. Boneh and B. Waters, "A fully collusion resistant broadcast, trace, and revoke system," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, Oct. 2006, pp. 211–220.

[14] J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig, "Chameleon-hashes with ephemeral trapdoors," in *Proc. PKC*, 2017, pp. 152–182.

[15] D. Derler, K. Samelin, and D. Slamanig, "Bringing order to chaos: The case of collision-resistant chameleon-hashes," in *Proc. PKC*, 2020, pp. 462–492.

[16] D. Derler, K. Samelin, D. Slamanig, and C. Striecks, "Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019.

[17] D. Derler and D. Slamanig, "Key-homomorphic signatures: Definitions and applications to multiparty signatures and non-interactive zero-knowledge," *Designs, Codes Cryptogr.*, vol. 87, no. 6, pp. 1373–1413, Jun. 2019.

[18] Y. Desmedt and S. Jajodia, "Redistributing secret shares to new access structures and its applications," Tech. Rep., 1997.

[19] D. Deuber, B. Magri, and S. A. K. Thyagarajan, "Redactable blockchain in the permissionless setting," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 124–138.

[20] J. R. Douceur, "The Sybil attack," in *Proc. Int. Workshop Peer-to-Peer Syst.*, 2002, pp. 251–260.

[21] L. Ducas, "Anonymity from asymmetry: New constructions for anonymous HIBE," in *Proc. CT-RSA*, 2010, pp. 148–164.

[22] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," in *Proc. CRYPTO*, 2015, pp. 585–605.

[23] J. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin backbone protocol: Analysis and applications," in *Proc. EUROCRYPT*, 2015, pp. 281–310.

[24] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *Proc. EUROCRYPT*, 1999, pp. 295–310.

[25] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 51–68.

[26] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Proc. CRYPTO*, 1995, pp. 339–352.

[27] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in *Secure Information Networks*, 1999, pp. 258–272.

[28] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Proc. ASIACRYPT*, 2010, pp. 177–194.

[29] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach, "Analysis of an electronic voting system," in *Proc. IEEE Symp. Secur. Privacy*, May 2004, pp. 27–40.

[30] H. Krawczyk and T. Rabin, "Chameleon signatures," in *Proc. NDSS*, 2000, pp. 1–12.

[31] J. Lai and Q. Tang, "Making any attribute-based encryption accountable, efficiently," in *Proc. ESORICS*, 2018, pp. 527–547.

[32] Z. Liu, Z. Cao, and D. S. Wong, "Blackbox traceable CP-ABE: How to catch people leaking their keys by selling decryption devices on eBay," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 475–486.

[33] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 17–30.

[34] D. Maram et al., "CanDID: Can-do decentralized identity with legacy compatibility, Sybil-resistance, and accountability," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 1348–1366.

[35] S. K. D. Maram et al., "CHURP: Dynamic-committee proactive secret sharing," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 2369–2386.

[36] R. Matzutt et al., "A quantitative analysis of the impact of arbitrary blockchain content on Bitcoin," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2018, pp. 420–438.

[37] R. Matzutt, O. Hohlfeld, M. Henze, R. Rawiel, J. H. Ziegeldorf, and K. Wehrle, "POSTER: I don't want that content! On the risks of exploiting Bitcoin's blockchain as a content store," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1769–1771.

[38] R. C. Merkle, "A certified digital signature," in *Proc. CRYPTO*, 1989, pp. 218–238.

[39] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.

[40] J. Ning, Z. Cao, X. Dong, J. Gong, and J. Chen, "Traceable CP-ABE with short ciphertexts: How to catch people selling decryption devices on eBay efficiently," in *Proc. ESORICS*, 2016, pp. 551–569.

[41] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks," in *Proc. ACM PODC*, 1991, pp. 51–59.

[42] I. Puddu, A. Dmitrienko, and S. Capkun, "μchain: How to forget without hard forks," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 106, Feb. 2017.

[43] D. Riepel and H. Wee, "FABEO: Fast attribute-based encryption with optimal security," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 2491–2504.

[44] Y. Rouselakis and B. Waters, "Practical constructions and new proof methods for large universe attribute-based encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 463–474.

[45] C. P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptol.*, vol. 4, no. 3, pp. 161–174, Jan. 1991.

[46] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[47] S. A. K. Thyagarajan, A. Bhat, B. Magri, D. Tschudi, and A. Kate, "Reparo: Publicly verifiable layer to repair blockchains," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2021, pp. 37–56.

[48] Y. Tian, N. Li, Y. Li, P. Szalachowski, and J. Zhou, "Policy-based chameleon hash for blockchain rewriting with black-box accountability," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2020, pp. 813–828.

[49] G. Tziakouris, "Cryptocurrencies—A forensic challenge or opportunity for law enforcement? An INTERPOL perspective," *IEEE Secur. Privacy*, vol. 16, no. 4, pp. 92–94, Jul. 2018.

[50] B. Waters, "Efficient identity-based encryption without random oracles," in *Proc. EUROCRYPT*, 2005, pp. 114–127.

[51] H. Yu, I. Nikolic, R. Hou, and P. Saxena, "OHIE: Blockchain scaling made simple," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 90–105.

[52] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling blockchain via full sharding," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 931–948.