Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Computing and Information Systems

School of Computing and Information Systems

6-2021

Non-equivocation in blockchain: Double-authentication-preventing signatures gone contractual

Yannan LI

Willy SUSILO

Guomin YANG
Singapore Management University, gmyang@smu.edu.sg

Yong YU

Tran Viet Xuan PHUONG

See next page for additional authors

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

Part of the Information Security Commons

Citation

LI, Yannan; SUSILO, Willy; YANG, Guomin; YU, Yong; PHUONG, Tran Viet Xuan; and LIU, Dongxi. Non-equivocation in blockchain: Double-authentication-preventing signatures gone contractual. (2021). *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, Virtual Conference, June 7-11*. 859-871.
Available at: https://ink.library.smu.edu.sg/sis_research/7404

This Conference Proceeding Article is brought to you for free and open access by the School of Computing and Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Computing and Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email cherylds@smu.edu.sg.

Author
Yannan LI, Willy SUSILO, Guomin YANG, Yong YU, Tran Viet Xuan PHUONG, and Dongxi LIU

This conference proceeding article is available at Institutional Knowledge at Singapore Management University:
https://ink.library.smu.edu.sg/sis_research/7404

Non-Equivocation in Blockchain: Double-Authentication-Preventing Signatures Gone Contractual

Yannan Li
Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Australia.
yl738@uowmail.edu.au

Willy Susilo
Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Australia.
wsusilo@uow.edu.au

Guomin Yang
Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Australia.
gyang@uow.edu.au

Yong Yu
School of Computer Science, Shaanxi Normal University, China.
yuyong@snnu.edu.cn

Tran Viet Xuan Phuong
Institute of Cybersecurity and Cryptology,School of Computing and Information Technology, University of Wollongong, Australia.
tran@uow.edu.au

Dongxi Liu
Data61, CSIRO, Australia.
Dongxi.Liu@data61.csiro.au

ABSTRACT

Equivocation is one of the most fundamental problems that need to be solved when designing distributed protocols. Traditional methods to defeat equivocation rely on trusted hardware or particular assumptions, which may hinder their adoption in practice. The advent of blockchain and decentralized cryptocurrencies provides an auspicious breakthrough paradigm to resolve the problem above. In this paper, we propose a blockchain-based solution to address *contractual* equivocation, which supports user-defined fine-grained policy-based equivocation. Specifically, users will be de-incentive if the statements they made breach the predefined access rules. The core of our solution is a newly introduced primitive named Policy-Authentication-Preventing Signature (PoAPS), which combined with a deposit mechanism allows a signer to make conflict statements corresponding to a policy to be penalized. We present a generic construction of PoAPS based on Policy-Based Verifiable Secret Sharing (PBVSS) and demonstrate its practicality via a concrete implementation in the blockchain. Compared with the existing solutions that only handle specific types of equivocation, our proposed approach is more generic and can be instantiated to deal with various kinds of equivocation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASIA CCS '21, June 7–11, 2021, Hong Kong, Hong Kong
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8287-8/21/06...$15.00
https://doi.org/10.1145/3433210.3437516

CCS CONCEPTS

• **Information systems** → *Distributed database transactions*; • **Security and privacy** → *Digital signatures*.

KEYWORDS

Equivocation; Blockchain; Digital signatures; Verifiable secret sharing

ACM Reference Format:
Yannan Li, Willy Susilo, Guomin Yang, Yong Yu, Tran Viet Xuan Phuong, and Dongxi Liu. 2021. Non-Equivocation in Blockchain: Double-Authentication-Preventing Signatures Gone Contractual. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21), June 7–11, 2021, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3433210.3437516

1 INTRODUCTION

Equivocation, which refers to dishonest parties conveying conflicting statements in a protocol, could cause various disasters, such as errors and attacks, to occur in a system. Typical examples of equivocation include double-spending in cryptocurrencies, issuing multiple certificates for an identity in public-key infrastructure, etc. Equivocation is one of the essential issues in designing protocols for distributed systems since it could cause Byzantine faults [9]. Traditional solutions involve a trusted party (or trusted hardware) [4], which can be effectively designed and implemented but unfortunately, is impractical. Other solutions rely on the assumptions of an honest majority or the synchrony of the system. Equivocation is also the main reason Byzantine agreement in general requires $N = 3f + 1$ nodes for at most f nodes failure [35]. Publicly verifiable append-only logs can adequately detect equivocation in a system. As a revolutionary technology, blockchain [8, 25, 26] provides a remedy to realize an immutable and decentralized public ledger that securely records all the transactions in the system, and hence paves a revolutionary mechanism for detecting and handling equivocation without relying on any trusted party. As an example, to prevent the typical multi-spending equivocation in a payment system, Bitcoin, the first blockchain-based cryptocurrency, allows users to sign multiple transactions, but only one of these transactions will be appended into the blockchain under the UTXO mechanism. Nevertheless, Bitcoin does not offer any solution to penalize or terminate equivocation and the panacea is not generally applicable in other application domains.

Equivocations in blockchain. As mentioned earlier, non-equivocation is a basic security requirement for computing systems, especially decentralized systems. Some early research works have indicated that it is impossible to deal with equivocation without trusted parties (trusted hardware) [24] or strong assumptions (high replication factors) [9]. The emergence of blockchain promises a breakthrough to address the issue based on a much weaker and more realistic assumption. Unfortunately, blockchain itself is a decentralized system, and hence, detecting and dealing with equivocation in the blockchain is also challenging. Recently, there have been several proposals that aim to identify equivocation effectively in the blockchain. Blockstack [1] uses a simple Kademlia-based Distributed Hash Table (DHT) as a discovery layer to ascertain the equivocation. Such Bitcoin witnessing approaches reduce the difficulty of equivocation to the hard problem of forking a chain. Still, they are inefficient as a user is required to download a large number of transactions to find out an equivocation. To efficiently verify the Bitcoin witnesses, Tomescu et al. [36] proposed Catena, a tamper-evident log built on Bitcoin blockchain. The new statement in their design is included via the OP-RETURN scripts. Aspegren [3] modified Catena and proposed a scalable non-equivocation based on b_verify for verifiable data management. Other solutions to blockchain-based non-equivocation contracts are based on the deposit mechanism with the help of the built-in time-lock script of Bitcoin [2, 6, 23]. It is also suggested that using Turing-complete smart contracts can also achieve non-equivocation contracts. Ruffing et al. [32] proposed a new primitive named *accountable assertions* to deal with the equivocation in distributed protocols such as Bitcoin [26], in which two conflict statements within one context are sufficient to extract a secret publicly. The existing solutions for handling equivocation mainly focused on specific issues (e.g., double spending/authentication), which significantly limits their application domains. Designing a contractual solution that can be instantiated to handle more general, e.g. user-defined policy-based equivocation is of both theoretical and practical interest. However, this remains an elusive research problem.

Contribution. In this paper, we propose a policy-based non-equivocation supporting general pre-defined policy as a solution to this proceeding problem. In summary, the contribution of this work is three-fold:

- We propose a Policy-Authentication-Preventing Signature (PoAPS) which allows the extraction of a signer's secret key when he/she signs messages violating a pre-defined policy. We formalize the definition and the security model of PoAPS and present a generic construction. The design is with the help of a new primitive named Policy-based Verifiable Secret Sharing (PBVSS), in which a secret can be shared based on general policies and each share supports public verifiability. We show an instantiation of the construction based on ECDSA together with a concrete PBVSS, and prove its security.
- We achieve policy-based non-equivocation in blockchain by integrating PoAPS with a blockchain-based deposit mechanism. Users who produce signatures for conflicting messages that violate a pre-defined policy will be penalized by the loss of the deposit. We propose a generic construction for this new primitive and then extend PoAPS from public extractability to supporting a designated beneficiary.
- We evaluate the proposed algorithms to test the time consumption off-chain. We also implement the system in a test net of blockchain to check the gas cost on-chain. Both implementation results demonstrate high efficiency and practicality.

Related works. One of the main contributions of this paper is PoAPS, the idea of which is inspired by a similar notion named double-authentication-preventing signatures (DAPS). DAPS was proposed by Poettering and Stebila [29, 30], in which the data to be signed is split into a subject and a message. If a signer signed two conflict messages for the same subject, the signing key of the signer would be revealed. Poettering et al. [29] presented a generic construction by involving the extractable two-to-one trapdoor function, which can be achieved by the group of quadratic residues modulo a Blum integer. Subsequently, Bellare et al. [5] proposed two highly efficient DAPS constructions based on identification trapdoor schemes. Subsequently, Derler et al. [13] generalized this notion into N-times-authentication-preventing signatures based on secret sharing schemes, where N signatures issued by the same signer within a subject are required to extract the signing key. Poettering et al. [28] put forward a shorter DAPS, which not only beats the signature size in [13], but also reduces the key size from linear to constant based on the identification scheme different from that of Fiat and Shamir [17]. Derler et al. modified the protocol in [13] and proposed a generic DAPS construction with constant keys and signatures size based on symmetric-key primitives [12]. Boneh et al. [7] proposed the notion of lattice-based DAPS and presented the first post-quantum DAPS system based on broadcast encryption [16]. Furthermore, they generalized the primitive by proposing predicate-authentication-preventing signatures without showing a concrete construction. The original source of the concept of DAPS actually dates back to leakage-deterring signatures from [22], which de-incentivizes the sharing of a signing key in whatever form, by embedding a secret to it. In this paper, we inherit this basic idea and design solutions to more generalized cases.

We should note that the new primitive we propose in this paper, namely PoAPS, is a generalization of the previous approaches. The extraction policy in our proposal is a tree-based multi-level structure. Thus the schemes mentioned above can be regarded as special cases of our proposal. To be more specific, the schemes with double-authentication/double-spending extractability correspond to a simple 2-ary structure in our scheme, and the N-times-authentication-preventing signatures are equivalent to our solution for a single-level tree with N leaf nodes. A typical application of the proposed primitive in this paper is to prevent contractual (policy-based) spending, which is a generalization of double-spending in e-cash and cryptocurrencies. More details and further potential applications are provided in Sect. 6.

Organization. The rest of the paper is organized as follows. We provide some preliminaries used through the paper in Sect. 2. The details of PoAPS are illustrated in Sect. 3. We present non-equivocation contracts in blockchain in Sect. 4. Implementation results and potential applications are shown in Sect. 5 and 6, respectively. We conclude the paper in Sect. 7.

2 PRELIMINARIES

In this section, we review some necessary tools used in policy-based non-equivocation in blockchain.

2.1 Digital signature

Digital signatures are cryptographic tools to authenticate an entity or check the integrity of a signed message. A digital signature Σ consists of the following 3 algorithms [21].

- $\mathsf{DKeyGen}(1^\lambda)$. On input a security parameter λ, this algorithm setups the system and returns a key pair (sk, pk).
- $\mathsf{DSign}(sk, m)$. This algorithm takes a message m and the secret key sk as input, it outputs a digital signature σ.
- $\mathsf{DVerify}(pk, m, \sigma)$. On input the message signature pair (m, σ), this algorithm outputs 0/1 to indicate the validity.

Elliptic curve digital signature algorithm (ECDSA) [21] (ECDSA = ($\mathsf{EKeyGen}$, ESign, $\mathsf{EVerify}$)) is a secure digital signature that widely used in many applications due to its short signature size, such as signing Bitcoin transactions.

A digital signature should be unforgeable under chosen-message attack, in which an attacker can learn the signatures on arbitrary messages of its choice. A formal definition is shown as follows.

Definition 2.1. A digital signature Σ is existentially unforgeable under chosen message attack (EUF-CMA), if for all PPT adversaries $\mathcal{A}$, we have,

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{A}} = \Pr[\mathsf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{A}}(\lambda) = 1] \le negl(\lambda),$$

where the experiment $\mathsf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{A}}(\lambda)$ is shown in Figure 1.

```
$\mathbf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{A}}(\lambda)$:
    $(sk_\sigma, pk_\sigma) \leftarrow \mathsf{DKeyGen}(1^\lambda)$.
    $\mathcal{Q} \leftarrow \emptyset$
    $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{DSign}'(sk,\cdot)}(pk)$
        where $\mathsf{DSign}'$ on input $m$:
            $\sigma \leftarrow \mathsf{DSign(sk,m)}$, $\mathcal{Q} \leftarrow \mathcal{Q} \cup m$
            return $\sigma$
    return 1, if $\mathsf{DVerify}(pk, m^*, \sigma^*) = 1 \wedge m^* \notin \mathcal{Q}$
    return 0
```

Figure 1: EUF-CMA security

2.2 Elliptic curve digital signature algorithm

Elliptic curve digital signature algorithm (ECDSA) [21] is widely used due to its short length of the generated signature. ECDSA is also the default signature algorithm to sign transactions in Bitcoin [26] system.

- $\mathsf{EKeyGen}(1^\lambda)$. Let G be an elliptic curve group, with a prime order p. g is the generator of G. Choose $x \in Z_p^*$ and compute $y = g^x$. Return the key pair as $(sk, pk) = (x, y)$.
- $\mathsf{ESign}(sk, m)$. On receiving a message m, execute the following steps to generate a signature.
  - choose a random $k \in Z_p^*$ and compute $R = g^k$, where $R = (R_x, R_y)$.
  - define $r = R_x \ mod \ p$. If $r = 0$, go back to step 1.
  - let $s = k^{-1}(H(m) + rx) mod p$. If $s = 0$, go to step 1.
  - return $\sigma = (r, s)$.
- $\mathsf{EVerify}(pk, m, \sigma)$. On receiving (m, σ), check if $r = 0$ or $s = 0$, return $\perp$. Otherwise, run the following steps.
  - compute $z = H(m)$.
  - let $u_1 = zs^{-1}$ and $u_2 = rs^{-1}$
  - compute $\tilde{R} = (\tilde{R_x}, \tilde{R_y}) = g^{u_1} y^{u_2}$.
  - if $\tilde{R_x} = r \mod p$, return 1. Otherwise, return $\perp$.

The ECDSA satisfies EUF-CMA if the elliptic curve discrete logarithm assumption [21] holds.

2.3 Verifiable secret sharing

Secret sharing schemes, as put forth by Shamir [34], are a secure mechanism to store sensitive information among multiple users. A (t, n) secret sharing scheme includes n players $P_1, \cdots, P_n$ and a dealer. The dealer generates n secret shares and distributes them to each player a unique share. Any subsets of the players containing t or more players can reconstruct the secret s, while less than t pieces of secret shares gather no information about the secret. Verifiable secret sharing schemes (VSSS) [10] can resist against a malicious dealer from distributing an invalid share, which allows the participants to verify the received secret shares without knowing the secret. A typical (t, n) verifiable secret sharing consists of the following phases.

- $\mathsf{SShareGen}$ $(s) \rightarrow (s_i, aux_i)$: On input a secret s, the dealer generates n shares and distributes the shares s_i

to each player together with an auxiliary information aux_i for verification.

- $\mathsf{SVerify}(s_i, aux_i) \rightarrow (0/1)$. On input a secret share and the auxiliary information, this algorithm outputs 1 to indicate this is a valid share, and otherwise, outputs 0.
- $\mathsf{SReconstruct}(\{(x_i, s_i)\}_{i=1}^{t}) \rightarrow (s)$: If a set of players $S \subset P$, where $|S| \geq t$, are collecting their shares s_i together as input, this algorithm outputs the original secret s.

A verifiable secret sharing is supposed to satisfy secrecy and verifiability [18, 27]. Secrecy depicts the confidentiality of the secret based on a share and verifiability allows players to check the consistency of the shares. Formal definitions are provided as follows.

Definition 2.2. **(Secrecy).** The secret shares reveal no information about the secret s. Less than t shares disclose no information about the secret s.

Definition 2.3. **(Verifiability).** If a share s_i generated by a dealer with a secret s can pass the $\mathsf{SVerify}$ algorithm with the help of the auxiliary information aux_i, this share must be valid. Specifically, if such t shares are collected, the secret s can be recovered.

3 POLICY-AUTHENTICATION-PREVENTING SIGNATURES

In this section, we present technical descriptions of the proposed PoAPS. Firstly, we introduce policy-based verifiable secret sharing (PBVSS) as a building block to design PoAPS. Then we formalize PoAPS by introducing the system components and security model. Finally, we provide a generic construction together with an instantiation.

3.1 Policy-based verifiable secret sharing

Before diving into the details of PoAPS, we first introduce PBVSS as a main building block, which is a generalization of the verifiable secret sharing schemes to that with a policy-based extract structure. Specifically, a PBVSS allows a secret to be shared among several users, in which each share can be verified publicly and individually without the help of other shares. Only the shares satisfying a specific extract structure can recover the secret.

3.1.1 Concrete construction of PBVSS. We show a concrete construction based on Shamir's secret sharing scheme [34] and the tree-based access structure [20]. Define the extract policy based on a tree structure, which is built in a top-down manner. Each non-leaf node is a threshold gate containing a unique polynomial, whose degree is the threshold value minus 1. The secret is embedded in the root node, as the constant term of the polynomial in the root. Each child node contains the shares of the secret in the parent node. The leaf nodes store the final secret shares.

We first introduce some notations in the construction. G is a multiplicative cyclic group of a large prime order p and g is the generator of G. For a node N, $prt(N)$ is the parent of the node N. $num(N)$ denotes the number of children of a node N. $index(N)$ is the specific number with the node N. k_N is the threshold number of node N. d_N is the degree of the polynomial embedded in this node N. We now present the detailed constructions.

- $\mathsf{PSSPolicyGen}(s, \mathsf{desc}) \rightarrow (\Gamma, pp)$. On input the description of a policy, the dealer generates a tree-based structure. And then formalize the tree with the help of the secret s to get the extract policy Γ. To be more specific, insert the detailed parameters of the tree in a top-down manner as follows. Choose a random polynomial $q_R(z)$ for the root R by setting $q_R(0) = s$. The degree of $q_R(z)$ is $d_R = k_R - 1$. The other coefficients of $q_R(z)$ are some randomness chosen by the dealer. Then for non-leaf nodes, generate the embedded polynomials level by level recursively. Specifically, choose a random polynomial $q_N(z)$ for each non-leaf node N, whose degree is $d_N = k_N - 1$, by setting $q_N(0) = q_{prt(N)}(index(N))$ and the other coefficients with random numbers $a_i^{(N)} (1 \leq i \leq d_N)$. And then compute $g^{a_i^{(N)}}$, where $\{a_i^{(N)}\} (1 \leq i \leq d_N)$ are the coefficients in each polynomial. Finally publish all the $\{g^{a_i^{(N)}}\}$ and g^s as the public parameters pp in the system and outputs the policy Γ.
- $\mathsf{PSSShareGen}(s, \Gamma) \rightarrow (s_i, aux_i)$. On input the secret s and the policy Γ, the dealer computes the shares, which are only contained in the leaf nodes of a policy Γ. For each leaf node N, the dealer selects a random $x_i \in Z_p$, and computes the value of the polynomial embedded in the parent node, that is, $s'_i = q_{prt(N)}(x_i)$. The share s_i is returned as (x_i, s'_i). The auxiliary information aux_i is the subset of the public parameters related to the polynomials of the nodes on the path from the current leaf node to the root (Detailed instantiation is shown in sec. 3.6).
- $\mathsf{PSSVerify}(s_i, aux_i, \Gamma) \rightarrow (0/1)$. The verifier holding the secret share in node C and the corresponding auxiliary information (s_i, aux_i) validates the share as follows. First parse x_i and s'_i from s_i and check $g^{s'_i}$ with the help of the auxiliary information. Specifically, the verifier computes
$$g^{s'_i} = \prod_{j=0}^{d_{prt(C)}} g^{a_j^{(prt(C))} x_i^j}, \tag{1}$$
where $g^{a_j^{(prt(C))}} (1 \leq j \leq d_{prt(C)})$ are in aux_i and $g^{a_0^{(prt(C))}}$ can be expressed by calling the polynomials in the upper-level nodes till g^s in root R.
- $\mathsf{PSSReconstruct}(\{(x_i, s_i)\}_{i \subseteq S}, \Gamma)$: If a set of players $S \subseteq P$, where S satisfies the policy, want to recover the secret s with $\{(x_i, s_i)\}$, they are able to reconstruct s iteratively by computing
$$q_N(z) = \sum_{P_i \in S} \Delta_{x_i, s}(z) s_i, \tag{2}$$

until $q_R(0) = s$, where

$$\Delta_{x_i,s}(z) = \prod_{P_i \in S, j \neq i} \frac{z - x_j}{x_i - x_j}$$

denotes the Lagrange coefficient.

3.1.2 *Security analysis of PBVSS.*

In this part, we briefly analyze the security of the proposed PBVSS. To be more specific, we show the proposed PBVSS satisfies secrecy and verifiability.

Secrecy. Our construction of PBVSS is based on Shamir's secret share scheme [34], which satisfies perfect secrecy. We generalize the basic construction into a multi-level tree-structured construction, in which the secret is shared level by level from the root node to the leaf nodes. The root node contains the secret to be shared. For each intermediate node in each level as a threshold gate, the constant of the embedded polynomial is a secret share of its parent node and will be distributed to multiple secret shares in the next level. For each node, the mechanism is degenerated to the basic secret share scheme, in which the shares reveal nothing about the secret in the upper level, including the original secret in the root and thus the secrecy of the secret in the upper level is naturally achieved. The shares in the leaf nodes are recursively generated and the secrecy of the proposed PBVSS is satisfied.

Verifiability. Verifiability can be easily checked by the correctness of PSSVerify. Specifically, equation 3.1.1 can be validated with the public parameters $\{g^{a_i^{(N)}}\}$ and the polynomials embedded in the nodes from the specific leaf to the root by the substitution of $q_N(0) = q_{prt(N)}(index(N))$.

3.2 System components of PoAPS

A PoAPS is a signature scheme with an extraction algorithm that can extract the secret key when the pre-defined policy is broken. The formal system components containing four algorithms are as follows.

Definition 3.1. **(PoAPS).** A policy-authentication-preventing-signature (PoAPS) on a message space $\mathcal{M}$ and a policy $\mathcal{S}$ consists of the following PPT algorithms.

- $\mathsf{PKeyGen}(1^\lambda) \rightarrow (pk, sk)$. On input a security parameter λ, this algorithm outputs a secret-public key pair (sk, pk) for a signer.
- $\mathsf{PPolicyGen}(sk, \mathsf{desc}) \rightarrow (\Gamma, pp)$. On input the user's secret key and the description of a policy, this algorithm outputs an extract policy Γ, to express the policy to reveal the signer's secret key, and public parameters pp for public verifiablity. The leaf node of Γ is embedded with some subject to express the corresponding rules.
- $\mathsf{PSign}(sk, m, \Gamma) \rightarrow (\sigma)$. On input a secret key sk, a message $m \in \mathcal{M}$ and the policy Γ, this algorithm outputs a signature σ. Note that in the original DAPS [29], m is composed of a subject ($subj$) and a payload (msg). In this model, we follow this setting and the $subj$ of a message is bound with Γ.
- $\mathsf{PVerify}(pk, m, \sigma, \Gamma) \rightarrow (1/0)$. On input the public key pk, the message $m \in \mathcal{M}$, a signature σ and the policy Γ, this algorithm outputs a bit 0/1 to indicate the signature is valid or not.
- $\mathsf{PExtract}(pk, \{m_i\}, \{\sigma_i\}, \Gamma) \rightarrow (sk)$. On input a public key pk and a set of message $\{m_i\} \in \mathcal{M}$, the corresponding signature set $\{\sigma_i\}$ and the policy Γ, this algorithm outputs the secret key sk of the signer, if the messages satisfy the policy $\mathcal{S}$.

Definition 3.2. **(Correctness).** A PoAPS is correct if for all $\lambda \in \mathbb{N}$, for all $(sk, pk) \leftarrow \mathsf{PKeyGen}(1^\lambda)$, for all $(\Gamma, pp) \leftarrow \mathsf{PPolicyGen}(sk, \mathsf{desp})$, for all messages $m \in \{0,1\}^*$ and for all signatures $\sigma \leftarrow \mathsf{PSign}(sk, m, \Gamma)$, we have $\mathsf{PVerify}(pk, m, \sigma, \Gamma) = 1$ with an overwhelming probability.

3.3 Security model

We now introduce the security properties, in which we mainly consider the most critical properties named unforgeability and extractability. We describe the ability of an attacker and the detailed security model between an adversary and a challenger.

Compromising set. Before presenting the other security properties, we first introduce the notion of a compromising set of messages. A compromising set is a set containing message-signature pairs that break the pre-defined policy. The formal definition of a compromising set is as follows.

Definition 3.3. **(Compromising set).** Let Γ be an extract policy defined on k messages. We say a set of k message-signature pairs $\{(m_i, \sigma_i)\}_{i \in [k]}$ are Γ-compromising for some specific pk and Γ, if each signature σ_i is valid on m_i and the k messages-signature pairs satisfy the extract policy Γ.

Unforgeability. Unforgeability prohibits an adversary (not a signer, but can be any observers) from generating some valid signatures to pass the verification. In this case, the signer is honest and the observers are the attackers. The observers are intended to forge some signatures to complete a compromising set and extract the secret. The unforgeability of PoAPS is much similar to that of the standard signature with one difference. Specifically, $\mathcal{A}$ specifies policy description desc, and the challenger $\mathcal{C}$ generates the key pairs and details the policy according to desc. It also maintains a signing list. The adversary $\mathcal{A}$ is allowed to make some queries for the signatures. The difference is, for a fixed secret key sk and policy Γ, $\mathcal{A}$ can make queries to only non-compromising sets of messages. If the current queried message together with the previous queried messages satisfies Γ, $\mathcal{C}$ aborts. The concrete definition of existential unforgeability for PoAPS is shown as follows.

Definition 3.4. **(Unforgeability).** A PoAPS is existentially unforgeable if for all PPT adversaries $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{EUF}}_{\mathrm{PoAPS},\mathcal{A}} = \Pr[\mathsf{Exp}^{\mathsf{EUF}}_{\Sigma,\mathcal{A}}(\lambda) = 1] \leq negl(\lambda),$$

where $\mathsf{Exp}^{\mathsf{EUF}}_{\Sigma,\mathcal{A}}(\lambda)$ is defined in Fig. 2.

Extractability. Extractability states that as long as enough valid message-signature pairs satisfying a policy are collected,

```
$\mathbf{Exp}^{\mathsf{EUF}}_{\mathrm{PoAPS},\mathcal{A}}(\lambda)$:
    $(sk, pk) \leftarrow \mathsf{PKeyGen}(1^\lambda)$.
    $(\Gamma, pp) \leftarrow \mathsf{PPolicyGen}(sk, \mathsf{desc})$.
    $\mathcal{Q} \leftarrow \emptyset$, $\mathcal{P} \leftarrow \emptyset$.
    $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{PSign}(sk,\cdot)}}(pk)$.
        where $\mathcal{O}_{\mathsf{PSign}}$ is on input $m = (subj, msg)$ by $\mathcal{A}$:
            if $\Gamma(\mathcal{P} \cup subj) = 1$, return $\perp$
            else
            $\sigma \leftarrow \mathsf{PSign}(sk, m, \Gamma)$,
            $\mathcal{Q} \leftarrow \mathcal{Q} \cup (m, \sigma)$, $\mathcal{P} \leftarrow \mathcal{P} \cup subj$
        return $\sigma$
    return 1, if the following conditions hold.
        $\mathsf{PVerify}(pk, m^*, \sigma^*, \Gamma) = 1$, $(m^*, \sigma^*) \notin \mathcal{Q}$
    return 0.
```

Figure 2: EUF security for PoAPS

the secret is able to be recovered. In this case, the signer is the attacker that is going to break the pre-defined policy but avoid being traced. Specifically, the signer is intended to generate signatures and the shares that can pass the verification while even if the policy is satisfied, the secret key still cannot be recovered. In the following formal security definition, two cases are considered. In the trusted setup, $\mathcal{A}$ signs the message and publishes the policy with a secret key sk generated by $\mathcal{C}$. $\mathcal{A}$ succeeds if enough message-signature pairs are collected, but extract a secret key sk' different from the secret key sk used in signing messages. In the untrusted setup, the signer can manipulate valid signatures that pass the verification with pk. $\mathcal{A}$ succeeds if enough message-signature pairs are collected, but extract a secret key sk' which is not corresponding to the public key. The concrete definition of extractability for PoAPS is shown as follows.

Definition 3.5. **(Extractability).** A PoAPS is extractable (PSE) (*resp.* PSE* with untrusted setup), if for all PPT adversary $\mathcal{A}$, we have,

$$\mathsf{Adv}^{\mathsf{PSE}}_{\mathrm{PoAPS},\mathcal{A}} = \Pr[\mathsf{Exp}^{\mathsf{PSE}}_{\Sigma,\mathcal{A}}(\lambda) = 1] \le negl(\lambda),$$

where $\mathsf{Exp}^{\mathsf{PSE}}_{\Sigma,\mathcal{A}}(\lambda)$ (*resp.* $\mathsf{Exp}^{\mathsf{PSE}^*}_{\Sigma,\mathcal{A}}(\lambda)$) is defined in Fig. 3 (*resp.* Fig. 4).

3.4 Generic constructions of PoAPS

In this section, we show that a PoAPS can be obtained in a generic way by using a secure digital signature scheme with a PBVSS scheme. The basic idea is as follows. The policy is described in the tree-based extract structure, whose leaf nodes are bound with some subjects and non-leaf nodes are threshold gates. The signer first inserts the parameters to detail the policy with the secret key. When signing a message, the message-signature pair (m_i, σ_i) with a secret share of the secret key will be bound to the leaf node. If sufficient shares are collected, which means the extracting condition is met, the secret key can be revealed via the extract algorithm.

```
$\mathbf{Exp}^{\mathsf{PSE}}_{\mathrm{PoAPS},\mathcal{A}}(\lambda)$:
    $(sk, pk) \leftarrow \mathsf{PKeyGen}(1^\lambda)$.
    $(\Gamma, pp) \leftarrow \mathsf{PPolicyGen}(sk, \mathsf{desc})$.
    $(\sigma_1, \cdots, \sigma_k) \leftarrow \mathcal{A}(sk, pk, \Gamma)$.
    $sk' \leftarrow \mathsf{Extract}(pk, (\sigma_1, \cdots, \sigma_k), \Gamma)$.
    return 1, if for $i \in k$, the followings hold.
        $\mathsf{PVerify}(pk, m_i, \sigma_i, \Gamma) = 1$
        $(m_i, \sigma_i)$ are compromising
        $sk' \neq sk$
    return 0.
```

Figure 3: PSE security for PoAPS

```
$\mathbf{Exp}^{\mathsf{PSE}^*}_{\mathrm{PoAPS},\mathcal{A}}(\lambda)$:
    $(pk, \Gamma, \sigma_1, \cdots, \sigma_k) \leftarrow \mathcal{A}(1^\lambda, \mathsf{desc})$.
    $sk' \leftarrow \mathsf{Extract}(pk, (\sigma_1, \cdots, \sigma_k), \Gamma)$.
    return 1, if for $i \in k$, the followings hold.
        $\mathsf{PVerify}(pk, m_i, \sigma_i, \Gamma) = 1$
        $(m_i, \sigma_i)$ are compromising
        $sk'$ is not the secret key corresponding to $pk$
    return 0.
```

Figure 4: PSE* security for PoAPS

To be more specific, let DSig = ($\mathsf{DKeyGen}$, DSign, $\mathsf{DVerify}$) be a EUF-CMA digital signature and PBVSS = ($\mathsf{PSSPolicyGen}$, $\mathsf{PSSShareGen}$, $\mathsf{PSSVerify}$, $\mathsf{PSSReconstruct}$) be a secure PBVSS scheme. The detailed construction of PoAPS = ($\mathsf{PKeyGen}$, $\mathsf{PPolicyGen}$, PSign, $\mathsf{PVerify}$, $\mathsf{PExtract}$) is shown in Fig. 5.

Remark: We note that, in the current construction, a signer can insert more than one signature for one leaf node in Γ, if $\mathsf{DSign}(sk, m')$ is a normal secure signature scheme. If some applications require only one signature in one leaf node, the double-authentication-preventing signature scheme [29] can be applied.

3.5 Security analysis of PoAPS

Theorem 3.6. *If there is an adversary $\mathcal{A}$ who can break the unforgeability of our proposed PoAPS scheme, then the challenger $\mathcal{C}$ can construct another algorithm $\mathcal{S}'$ to break the unforgeability of the underlying digital signature or the secrecy of the PBVSS with an overwhelming probability.*

Proof. We show our proof in the selective-policy model, where the challenged policy is defined in advance. W.l.o.g, $\mathcal{A}$ selects a specific Γ to challenge upon, which is a tree-based structure with the subject embedded in the leaf nodes. We describe our proof in a sequence of games shown as follows.
Game 0. Game 0 is the same as the original game defined in section 3.3.
Game 1. Game 1 is the same as Game 0 with one difference. $\mathcal{C}$ maintains a singed list $\mathcal{Q}$ for ever issued queries by $\mathcal{A}$. If $\mathcal{A}$

<u>$\mathsf{PKeyGen}$</u>$(1^\lambda) \to (pk, sk)$. Call $\mathsf{DKeyGen}(1^\lambda)$ to setup the system and return the public and secret key pair (pk, sk).

<u>$\mathsf{PPolicyGen}$</u>$(sk, \mathsf{desc}) \to (\Gamma, pp)$. On input the application-based policy description desc, call $\mathsf{PSSPolicyGen}(sk, \mathsf{desc})$ to generate the structure Γ and public parameters pp.

<u>PSign</u>$(sk, m, \Gamma) \to (\sigma)$. To sign a message $m = (subj, msg)$, firstly check which subject in the leaf node of tree, m is related to. Then, to generate a secret share in the corresponding leaf node, the signer calls $\mathsf{PSSShareGen}(sk, \Gamma)$ with $subj$ as the random input[1], and returns the share s and the corresponding aux. Then with $m' = subj \| msg \| s \| aux$, call $\mathsf{DSign}(sk, m')$ to return σ'. The final signature is $\sigma = \sigma' \| subj \| msg \| s \| aux$.

<u>$\mathsf{PVerify}$</u>$(pk, m, \sigma, \Gamma) \to (0/1)$. The verifier first parses $m' = subj \| msg \| s \| aux$ from σ and calls $\mathsf{DVerify}(pk, m', \sigma')$ to check the validity of the signature. If the signature is invalid, return 0 and aborts. Otherwise, parse s and aux in the signature and check the validity of the shares s by calling $\mathsf{PSSVerify}(s, aux, \Gamma)$. If both callings return 1 simultaneously, this algorithm outputs 1; otherwise, it outputs 0 and aborts.

<u>$\mathsf{PExtract}$</u>$(pk, \{(m_i, \sigma_i, \Gamma)\}) \to (s)$. If the message-signature pairs satisfy the policy, then with the shares embedded in the signatures, call $\mathsf{PSSReconstruct}(\{(x_i, s'_i)\}_{i \subseteq S}, \Gamma)$ to recover the signing key sk.

Figure 5: Generic construction of PoAPS

submits a (m, σ) pair that (1) is not in $\mathcal{Q}$ and (2) can pass the verify algorithm, $\mathcal{C}$ declares failure and aborts.

Analysis. $\mathcal{C}$ randomly chooses a public key y and forwards it to $\mathcal{A}$. $\mathcal{A}$ specifies some policy description desc_Γ and issues the subjects $\{subj\}$ it is going to query. $\mathcal{C}$ details Γ by assigning polynomials to each node and simulates the public parameters besides y.

Define satisfied node as $Sat(N)$ and unsatisfied node as $UnSat(N)$, where $Sat(N)$ is the node that the subtree $\mathcal{T}_N$ with root N is satisfied with the queried subjects and $UnSat(N)$ is the node where the subtree is not met. For $Sat(N)$, define $\mathsf{PolySat}(\mathcal{T}_N, subj, \lambda_N)$ to set up the polynomials in the satisfied node N, which takes the sub-tree $\mathcal{T}_N$, the queried subject set, and an integer λ_N as input. Denote the degree of the polynomial as d_N. Specifically, from the root R, for the first $Sat(N)$, set $q_N(0) = \lambda_N$ and choose the other d_N randomness as the coefficients to fix the polynomial q_N. And then set polynomials for each child node N' of N recursively as $\mathsf{PolySat}(\mathcal{T}_{N'}, subj, q_N(index(N')))$ until the leaf nodes. Publish the public parameters in $Sat(N)$ as $g^{a_i^{(N)}}$, where $a_i^{(N)}$ are the coefficients in the polynomials of satisfied node N. For the leaf nodes i in $Sat(N)$, the shares are $q_{prt(i)}(H(subj))$.

[1]The subject $subj$ is uniquely mapped to an integer, e.g., by using a collision resistance hash function

And the auxiliary information is set accordingly with the randomly chosen coefficients. For $UnSat(N)$, the parameters should be simulated using randomness, the public key and the Lagrange Interpolation. Define $\mathsf{PolyUnSat}(\mathcal{T}_N, subj, g^{\lambda_N})$ to set up polynomials in unsatisfied node N, which takes the sub-tree $\mathcal{T}_N$, the queried subject set $subj$ and an element g^{λ_N} as input. For an unsatisfied node N, there are at most d_N satisfied children nodes N'. Let h_N be the number of satisfied children nodes of N. For the satisfied child node N' of N, choose a random $\lambda_{N'}$ and set $q_N(index(N')) = \lambda_{N'}$. Choose $d_R - h_R$ randomness to fix q_N. The root node must be an unsatisfied node. Thus, $\mathcal{C}$ firstly define the polynomial in node R. $q_R(0)$ is implicitly set as x, where $y = g^x$. For the satisfied child node N, call $\mathsf{PolySat}(\mathcal{T}_N, subj, \lambda_N)$ to determine the other coefficients. Otherwise, call $\mathsf{PolyUnSat}(\mathcal{T}_N, subj, g_N^{\lambda})$ to implicitly fix the polynomial until the parent nodes of the leaf nodes. Thus for these nodes, $g^{a_i^{(N)}}$, where $a_i^{(N)}$ are the coefficients for the node N, can be published as part of the public parameters. Now we simulate the polynomial for the parent node of the leaf nodes in the $UnSat(N)$. For such parent node P of a leaf node, $g^{q_P(0)}$ is implicitly fixed with its index and the polynomial in its parent node. To simulate other coefficients, choose random secret shares for the queried subject set as $\{(subj_i, s_i)\}_{(1 \le i \le h_P)}$, where h_P is the number of the queried leaf node under node P. Choose $d_P - h_P$ randomness to determine the polynomial in node P, where d_P is the degree of the polynomial in node P. Thus, with these d_P secret share $\{(subj_i, s_i)\}$ and $g^{q_P(0)}$, with the equation 3.1.1, the public parameters in node P can be simulated. The simulation of Γ is perfect with the secrecy of the underlying PBVSS. Thus, the above setting is identical from $\mathcal{A}$'s view.

$\mathcal{C}$ maintains a secret share list $\mathcal{P}$, which records the subject in $\{subj\}$ and the corresponding secret shares. After the simulation, $\mathcal{C}$ completes the secret share list $\mathcal{P}$ with the settings above. Now, $\mathcal{A}$ adaptively issues a query with message $m = (subj_m, msg_m)$ on its will, but is restricted to the subjects in $\{subj\}$. $\mathcal{C}$ first responds with the secret share and then simulates the signature on m. To simulate the secret share, $\mathcal{C}$ checks the corresponding subject $subj_m$ the message m is related to. If $subj_m \notin \{subj\}$, $\mathcal{C}$ aborts. Otherwise, $\mathcal{C}$ looks up the list $\mathcal{P}$ and finds the corresponding record s_m as the secret share. And return the subset of the public parameters on the path from the leaf node to the root as the auxiliary information aux_m.To simulate the signature, $\mathcal{C}$ checks whether m is in the signed list $\mathcal{Q}$. If m is in $\mathcal{Q}$, $\mathcal{C}$ responds with the corresponding signature. Otherwise, $\mathcal{C}$ checks whether m together with the other messages in $\mathcal{Q}$ satisfy the policy Γ. If so, $\mathcal{C}$ aborts. Otherwise, $\mathcal{C}$ answers signing queries by calling the challenger $\mathcal{C}'$ in the underlying signature scheme with $msg_m \| subj_m \| s_m \| aux_m$ and y as the input and returns the signature σ' to $\mathcal{A}$. Then $\mathcal{C}$ updates (m, σ') in $\mathcal{Q}$.

The setting in Game 1 is identical with that in Game 0 from $\mathcal{A}$'s view. If $\mathcal{A}$ causes $\mathcal{C}$ to abort with non-negligible probability, then we can use $\mathcal{A}$ to construct a sub-algorithm

$\mathcal{S}'$ that can break the unforgeability of the underlying digital signature scheme. □

THEOREM 3.7. *If there is an adversary $\mathcal{A}$ who breaks the extractability of our proposed PoAPS under PSE* model, then we can construct another algorithm $\mathcal{B}$ to break the correctness of the underlying digital signature or the verifiability of the policy-based verifiable secret sharing scheme.*

PROOF. We show a heuristic analysis of extractability in the following two steps. Firstly, if a message-signature pair (m, σ) in PoAPS can pass the verify algorithm, then a valid secret share can be parsed from σ. Secondly, perfect verifiability guarantees that the secret key can be extracted from the secret sharing scheme if sufficient shares are collected in a recursive way. Thirdly, according to the PBVSS in 3.1, pk is bound with the root R of the policy, which guarantees that the shares allow the correct sk corresponding to pk to be extracted. □

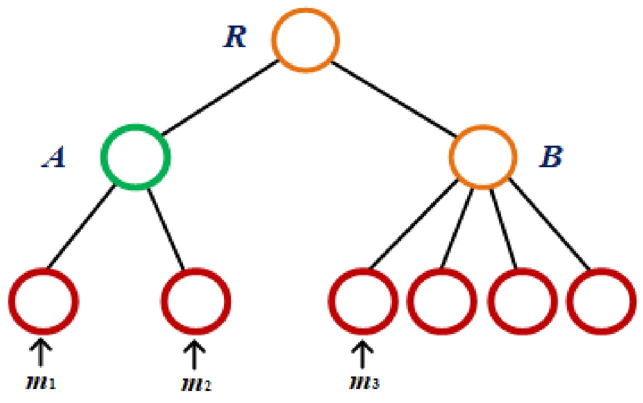

Figure 6: Policy instance in the instantiation

3.6 Instantiation of PoAPS

We show an instantiation with the extract structure in Fig. 6 together with ECDSA signature and our proposed PoAPS, which is perfectly compatible to blockchain system. ECDSA = (EKeyGen, ESign, EVerify) denote the ECDSA signature and PBVSS = (PSSPolicyGen, PSSShareGen, PSSVerify, PSSReconstruct) be a secure PBVSS scheme. We show an instantiation with the policy specified in Fig. 6. This is a three-level tree-based structure with the root node R. W.l.o.g, we define the root node R and node A as AND gates, and B as a 3-out-of-4 gate. The details of the instantiation of PoAPS are as follows.

- $\mathsf{PKeyGen}(1^\lambda) \to (x, y)$. Call $\mathsf{EKeyGen}(1^\lambda)$ to set up the system with some security parameter λ and return a key pair (x, y).
- $\mathsf{PPolicyGen}(x, \mathsf{desc}) \to (\Gamma, pp)$. On input the application-based policy description as the tree in Fig. 6, generate the extract policy Γ and the public parameters pp by calling $\mathsf{PSSPolicyGen}(x, \mathsf{desc})$. Specifically, the polynomial for R is $f_R(z) = a_1^{(R)} z + x$, where $a_1^{(R)}$ is a random number and x is the secret key. The polynomial for A is $f_A(z) = a_1^{(A)} z + a_0^{(A)}$, where $a_1^{(A)}$ is a random number and $a_0^{(A)}$ is $f_R(index_A)$. For node B, the degree of the polynomial should be 2. Thus, the polynomial is $f_B(z) = a_2^{(B)} z^2 + a_1^{(B)} z + a_0^{(B)}$, where $a_2^{(B)}$ and $a_1^{(B)}$ are random numbers and $a_0^{(A)}$ is $f_R(index_B)$. The public parameters are $pp = (g^{a_1^{(R)}}, g^{a_1^{(A)}}, g^{a_2^{(B)}}, g^{a_1^{(B)}}, y)$.
- $\mathsf{PSign}(x, m, \Gamma) \to (\sigma)$. To sign a message $m = (subj, msg)$, first check the message is binding to some subject in the leave node N. Then generate a secret share for the leaf node by calling $\mathsf{PSSShareGen}(x, \Gamma)$ with the random input as $H(subj)$ and return the share s and the corresponding aux. To generate a signature, m is set as $m' = subj \| msg \| s \| aux$ and calls $\mathsf{ESign}(x, m')$ to return $\sigma' = (r, s)$. Then the signature is $\sigma = \sigma' \| subj \| msg \| s \| aux$.
- $\mathsf{PVerify}(y, m, \sigma, \Gamma) \to (0/1)$. The verifier first parses $m' = subj \| msg \| s \| aux$ from σ and calls $\mathsf{EVerify}(y, m', \sigma')$ to check the validity of the signature and then parses the embedded s and aux to validate the shares s by calling $\mathsf{PSSVerify}(s, aux, \Gamma)$. If the two callings return 1 simultaneously, it outputs 1; otherwise, output 0.
- $\mathsf{PExtract}\ (y, \{(m_i, \sigma_i)\}, \Gamma) \to (s)$. If the message-signature pairs satisfy the policy, then with the shares embedded in the signature, call the $\mathsf{PSSReconstruct}(\{(x_i, s_i)\}_{i \subseteq S}, \Gamma)$ to recover the signing key x.

3.7 Extension to a designated beneficiary

The protocol in the previous sections is with public verifiability and public extractability for the secret. However, some scenarios require public verifiability with private extractability by a designated beneficiary. Thus, in this section, we introduce PoAPS with public verifiability and private extractability by a designated beneficiary integrating to the blockchain.

For the system model of the PoAPS with a designated beneficiary, besides the system roles in PoASS, another beneficiary is also involved. To be more specific, the signature can be verified by anyone in the system, but only a designated beneficiary can recover the secret when the structure is satisfied. To achieve the new security requirement, some new techniques are required, including public-key encryption schemes [15] and zero-knowledge proofs [19, 31]. The basic idea of the PoAPS with public verifiability and private extractability is as follows. The signer and the beneficiary hold public-secret key pairs (pk_s, sk_s) and (pk_b, sk_b), respectively. Upon receiving a message m, the signer binds m by some subject in the structure Γ. And then the signer generates a secret share s according to the Γ and encrypts the share with an encryption scheme with the beneficiary's public key pk_b and gets $e = Enc_{pk_b}(s \| aux)$. And then generate a zero-knowledge proof to prove that e is a correct form of encryption under pk_b and the corresponding plaintext is a valid secret share that can be verified with pk_s. After that, the signer sets $m = m\|subj\|e$ and signs the message m to obtain a signature. Upon receiving the signature, all the users can check the validity of the signature and the embedded encrypted shares while gets nothing about the share. Only the beneficiary can decrypt e with the corresponding secret key sk_b. After collecting all the shares, the beneficiary can extract the secret key.

To be more specific, we show the detailed construction on top of the proposed PoAPS in Fig. 7 with the help of ElGamal encryption [15] and zero-knowledge proofs [19, 31].

<u>PKeyGen</u>$(1^\lambda) \rightarrow (x, y)$. The signer calls $\mathsf{EKeyGen}(1^\lambda)$ to set up the system with some security parameter λ and return a key pair (x_s, y_s). The beneficiary calls $\mathsf{EKeyGen}$ (1^λ) to generate a key pair (x_b, y_b).

<u>PPolicyGen</u>$(x_s, \mathsf{desp}) \rightarrow (\Gamma, pp)$. Call $\mathsf{PSSPolicyGen}(x_s, \mathsf{desp})$ to generate an application-based extract structure and the necessary public parameters pp.

<u>PSign</u>$(x_s, m, \Gamma) \rightarrow (\sigma)$. To sign a message m, the signer needs to do the following steps.

(1) First check the message is binding to some subjects in the leaf node N.

(2) Then generate a secret share for the leaf node by calling $\mathsf{PSSShareGen}(x_s, \Gamma)$ and return the share s and the corresponding aux.

(3) To encrypt the share, the signer first chooses a random number α, and calculates

$$E = (A, B) = (h^\alpha, s^{-1} \cdot y_b^\alpha).$$

(4) To generate a zero-knowledge proof for the encryption, the signer needs to prove

$$SPK\{(\alpha) : V^B = g^{y^\alpha}\}.$$

The instantiation of such proof can be easily obtained following the techniques in [27].

(5) To generate a signature, m is set as $m = m\|subj\|s\|aux\|E\|\pi$, where π is the proof, and calls $\mathsf{ESign}(x_s, m)$ to return a signature $\sigma = (r, s)$.

<u>PVerify</u>$(y, m, \sigma) \rightarrow (0/1)$. The verifier first calls $\mathsf{EVerify}(y, m, \sigma)$ to check the validity of the signature and then parses the embedded m_i, s_i and aux_i to check the validity of the shares s_i by calling $\mathsf{PSSVerify}(s_i, aux_i)$. Verify the embedded zero-knowledge proof. If the all the verification return 1 simultaneously, it outputs 1; otherwise, output 0.

<u>PExtract</u>$(y_s,\ x_b,\ \{(m_i, \sigma_i)\})$. If the message-signature pairs satisfy the policy, then decrypt the shares embedded in the signature σ_i as $s_i = A^{x_b}/B$, call the $\mathsf{PSSReconstruct}(\{(x_i, s_i)\}_{i \subseteq S})$ to recover the signing key x_s.

Figure 7: PoAPS with designated beneficiary

Remarks. We show three remarks for the proposed PoAPS.

(1) A blockchain-based non-equivocation with public verifiability and private extractability can be designed by integrating the deposit mechanism borrowed from [37] together with the above PoAPS by the designated beneficiary.

(2) The proofs of forgeability and private extractability can be obtained directly from the underlying PoAPS (c.f. section 3.5).

(3) Additionally, this protocol also needs to satisfy another security requirement, *privacy*, which shows the ciphertexts do not reveal the information about the encrypted shares. This property indicates that even if collecting enough transactions (satisfying the extract structure) containing encrypted shares, the outsiders are not able to recover the corresponding secret key. In the model, this can be reduced to the security of the underlying encryption scheme.

4 POLICY-BASED NON-EQUIVOCATION IN BLOCKCHAIN

Putting everything together, in this section, we introduce policy-based non-equivocation in the blockchain.

Three entities are involved in the system, as shown in Fig. 8, namely an authority, a set of users and a blockchain. We show the way to achieve non-equivocation in the blockchain by integrating the proposed PoAPS. The workflow is as follows.

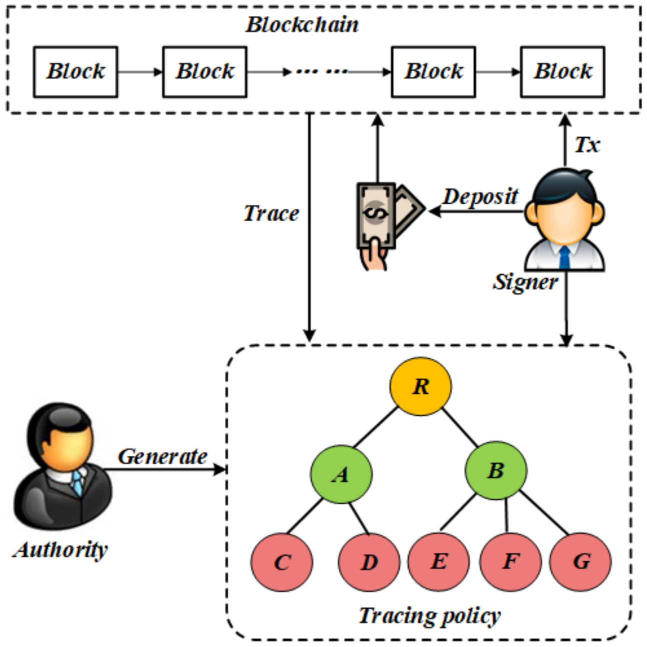

Figure 8: System model of policy-based non-equivocation in blockchain

- <u>*Setup.*</u> This phase involves the authority and the users. The authority is not a fully trusted party, who can be offline after initiation. The authority initiates the whole system and specifies the policy with some policy description desc that the users in the system need to follow. Each participant who involves in the system calls $\mathsf{PKeyGen}(1^\lambda)$ to return a key pair (pk_U, sk_U). The user who conducts transactions in the system details the extract structure according to the pre-defined policy by calling $\mathsf{PPolicyGen}(sk_U, \mathsf{desc})$ and return Γ and pp. Γ is stored in blockchain publicly. The user who engages in the system creates deposit v in time T on the blockchain.

- *Sign.* In this phase, we suppose user A as a signer, who will be penalized in case of equivocation, and user B as a verifier, who can detect the misbehaviour of A by validating the transactions. A generates a transaction with necessary information and signs the transaction by calling $\mathsf{PSign}(sk_A, tx, \Gamma)$, where tx contains the transaction details. The transaction will be validated by the miners by calling $\mathsf{PVerify}(pk_A, tx, \sigma, \Gamma)$. If the transaction is invalid, it will be discarded. Otherwise, the transaction will be published on the blockchain with the consensus protocol, which is publicly accountable by all the users in the system. B collects the transactions from the blockahin and the embedded shares for future extractability.
- *Extract.* In this phase, if B detects equivocation according to Γ, B calls $\mathsf{PExtract}(pk, \{m_i\}, \{tx_i\}, \Gamma)$ to recover A's secret key in the deposit and transfers the money to B's account. The users in the system compete to extract the secret and only the first one who works it out will get the reward. If A honestly performs in the system, A can redeem the deposit after time T without any loss.

5 IMPLEMENTATION

We evaluate the proposal in this section. Firstly, we theoretically analyze the computation complexity and the communication overhead of the proposed PoAPS. Then we test the time costs of these algorithms. The gas costs of the functions are also provided together with the actual monetary costs. We present the implementation result in this section.

5.1 Complexity analysis

We analyze the computation complexity and the communication overhead of the proposed PoAPS based on ECDSA [21] and the proposed PBVSS (section 3.1). We focus only on the expensive operations, including the exponentiation and multiplication in the group and ignore the cheap ones. The results are listed in Table 1.

The parameters in Table 1 are explained as follows. n is the number of the nodes in the extract structure (tree). h is the height of the extract structure (tree). d is the degree in a tested polynomial. exp_G and exp_P are the exponentiation operations in group G and Z_p, respectively. $mult_G$ and $mult_P$ are the multiplication operations in the group G and Z_p, respectively.

Algorithm	Computation complexity	Communication overhead
PKeygen	exp_G	$O(n)\lvert G\rvert$
PPolicyGen	$O(n)mult_p$	$O(n)\lvert Z_p\rvert$
PSign	$exp_G + (d+2)mult_p$	$3\lvert Z_p\rvert + O(h)\lvert G\rvert$
PVerify	$2mult_p + 2exp_G +$ $mult_G + O(h)exp_p$	1
PExtract	$O(n)(mult_p + exp_p)$	$\lvert Z_p\rvert$

Table 1: The theoretical analysis of each algorithm in PoAPS

5.2 Implementation results.

We implement the algorithms to evaluate the time consumption. The projects are written in C++ language with Miracl Library [33] to achieve the cryptographic operations and the Barreto-Naehrig curve [14] was chosen in the evaluation. The evaluation of the algorithms was first conducted on a laptop with Intel(R) Core(TM) i7-8550 CPU @1.8 GHz Win 10 operating system. Then we test the algorithms in a Raspberry Pi to simulate a thin client. The Raspberry Pi 3 Model B+ is equipped with 64-bit quad-core ARM v7 processor rev 4 and 1 GB LPDDR2 SDRAM. The operating system is Ubuntu Mate 16.04.6 LTS. We implement the extract structure in Fig. 6 and the time costs for each algorithm are shown in the table below.

PoAPS		
Algorithm	Laptop execution time	Raspberry Pi execution time
PKeygen	1 ms	9.113 ms
PPolicyGen	0.002 ms	0.075 ms
PSign	2 ms	12 ms
PVerify	18 ms	79.26 ms
PExtract	28 ms	98 ms
Designated beneficiary		
Algorithm	Laptop execution time	Raspberry Pi execution time
PKeygen	1 ms	9.3 ms
PPolicyGen	0.001 ms	0.083 ms
PSign	18 ms	92 ms
PVerify	30 ms	160 ms
PExtract	34 ms	124 ms

Table 2: The time costs of each algorithms in PoAPS

As we can see from Table 2, the execution on Raspberry Pi is much slower than that on a laptop, as the processing ability of an Raspberry Pi is lower than that of a laptop. Among the algorithms, PPolicyGen is the cheapest, as the cost to initialize an access structure is for generating some polynomials with random coefficients without other time-consuming operations. The most expensive algorithm is PExtract, as this algorithm needs to traverse all the nodes from the leaves to the root level by level. Nevertheless we need to note that the sub-tree on the same level can be processed in parallel, which would save much time. While the sub-tree at different levels can only be conducted serially. For example, in our example, the extraction in subtree A and B can be conducted meanwhile, which saves 6 ms. In this way, the time cost of PExtract based on a multi-level tree structure can be approximately estimated. We provide a benchmark of the time cost of PExtract algorithm on a single-level structure, which is shown in Fig. 9. We can tell from this figure that the time cost in a single-level structure is linear with the number of nodes, which also complies with the theoretical analysis, as the more nodes in the level, the more exponent operations are needed. And the algorithm is practical in the sense that for 30 nodes, it costs 173 ms.

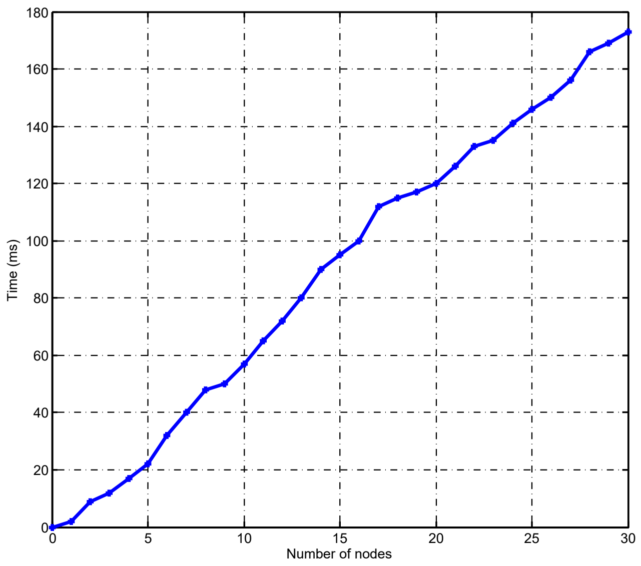

Figure 9: Time costs of PExtract algorithm on laptop in single level

We then test the efficiency of the construction with designated beneficiaries. As we can see from Table 2, the time costs for PKeyGen and PPolicyGen are almost the same as those in the publicly verifiable one. PSign, PVerify and PExtract are more costly since more operations are needed to achieve privacy, which is in accordance with the empirical analysis. The additional overhead in PSign is to encrypt the share and generate the zero-knowledge proofs, as the proof is to prove witness for double discrete logarithms. And the overhead for PVerify and PExtract are to verify a zero-knowledge proof and decrypt the ciphertext, respectively. The additional overhead is 16 ms, 12 ms and 6 ms, which is acceptable.

Algorithm	Gas costs	USD costs
Verify a signature	2199 gas	0.0079 USD
Verify a share	1415 gas	0.0051 USD

Table 3: The gas costs in Ethereum

For the implementation in blockchain, we use Solidity [11] to test the gas costs of the algorithms based on the ECC library by A. Olofsson to achieve the cryptographic computation and evaluate the corresponding actual USD costs. The gas price is 0.02 Ether per million gas and the Ether price is \$180.75 (Jan. 2020. https://coinmarketcap.com/), which is acceptable.

6 OTHER APPLICATION EXAMPLES

In this section, we show more examples in which our system provides tracing/extract ability by the equivocation conditions.

Accountable delegation of signing ability. In traditional signing delegation schemes, the original signer issues an authorization to the proxy signer. The proxy signer can sign on behalf of the original signer when the original signer is unavailable. It is reasonable that the original signer makes some constraints to limit the proxy signer's signing ability, in a way that if the proxy signer violates the rules, it can be traced by revealing the proxy signing key and the proxy thus can not sign anymore. Specifically, the original signer issues an authorization together with a tracing policy to the proxy signer. The proxy signer makes deposits with his/her secret key. The proxy signer also needs to initiate the tracing policy with the secret key and make it publicly verifiable (i.e., put it on the blockchain). Then the proxy signer can sign (i.e., generate transactions) instead of the original signer. The proxy signer will be punished by losing all the deposit if he/she does not follow the policy specified by the original signer.

Disincentivizing contractual (policy-based) spending. Double-spending is one of the most common equivocation in e-cash and cryptocurrencies, in which it requires a single digital currency cannot be spent more than once. Policy-based spending, with a policy linking to the currency, is essential for organizations to guide the investment tradeoff [2]. For example, the money raised for the charity should be regulated by some rule. If the charitable spending breaks the rule, then the secret key of the money can be revealed and the money will be confiscated by the government. In a cryptocurrency system, the developer could specify spending rules based on a tree-based policy structure, in which more than double-spending can be considered. Then by applying the mechanisms provided in

[2]https://russellinvestments.com/-/media/files/ca/en/insights/institutions/understanding-effects-of-spending-policies.pdf?la=en

Sect. 4, it satisfies more general regulation requirements in the system.

Internet censorship. Internet censorship is to control the resources published or accessed on the Internet by the regulators or the administrators of the web site. Suppose an online content sharing web site, in which each user has a public-secret key pair when they registered, is allowed their users to upload and share their data. The subscribers can pay for the material to share the data or donate to the publisher if they like the material the publisher shares. The reward will be put in the account controlled by the secret key. Each time the publisher uploads a content, a signature should be generated for future regulation. The web site management team defines the uploading conditions based on some tracing structure. If any user breaks the rules, the secret key will be revealed, and the account will be placed in the blacklist. The money in their account can be transferred to the user who finds the violation first as a reward.

Policy-based insurance pension scheme. This example is for the condition with public verifiability and a designated beneficiary. Suppose a private pension scheme is designed by an insurance company. The scheme is a contributory one based on a policy structure where there are many conditions, for example, the age of the participation, the health condition, a fixed amount of the salaries received, etc. When each condition is met, the insurance company generates a signature for the participants. If the pre-defined policy is met, then the participant can retrieve the account and acquire the corresponding pension inside.

7 CONCLUSION

Non-equivocation is a basic security requirement for distributed systems including blockchain. In this paper, we came up with some new primitives to deal with equivocation in blockchain-based applications. Specifically, we presented PoAPS on top of PBVSS with a black-box construction and an instantiation based on ECDSA. Security proofs are presented to demonstrate the unforgeability and extractability. We also integrated our proposal with blockchain-based cryptocurrencies to detail the punishment by losing the deposit. We evaluated the algorithms to test the time costs and the gas costs, which show the practicality of the scheme. Some potential applications were also discussed. Future work includes exploring more expressive policy to design more general spending rules.

8 ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers who led to the improvement of this paper. Yong Yu is supported by the National Natural Science Foundation of China under Grant 61872229, Grant U19B2021, the Key Research and Development Program of Shaanxi under Program 2020ZDLGY09-06 and the Blockchain Core Technology Strategic Research Program of Ministry of Education of China under Grant 2020KJ01030.

REFERENCES

[1] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J Freedman. 2016. Blockstack: A global naming and storage system secured by blockchains. In *2016 USENIX Annual Technical Conference*. 181–194.

[2] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. 2014. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 443–458.

[3] Henry Aspegren. 2018. *b_verify: scalable non-equivocation for verifiable management of data*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[4] Michael Backes, Fabian Bendun, Ashish Choudhury, and Aniket Kate. 2014. Asynchronous MPC with a strict honest majority using non-equivocation. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*. 10–19.

[5] Mihir Bellare, Bertram Poettering, and Douglas Stebila. 2017. Deterring certificate subversion: efficient double-authentication-preventing signatures. In *IACR International Workshop on Public Key Cryptography*. Springer, 121–151.

[6] Iddo Bentov and Ranjit Kumaresan. 2014. How to use bitcoin to design fair protocols. In *Annual Cryptology Conference*. Springer, 421–439.

[7] Dan Boneh, Sam Kim, and Valeria Nikolaenko. 2017. Lattice-based DAPS and generalizations: Self-enforcement in signature schemes. In *International Conference on Applied Cryptography and Network Security*. Springer, 457–477.

[8] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* 3 (2014), 37.

[9] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[10] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. 1985. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. IEEE, 383–395.

[11] Chris Dannen. 2017. *Introducing Ethereum and Solidity*. Springer.

[12] David Derler, Sebastian Ramacher, and Daniel Slamanig. 2018. Generic double-authentication preventing signatures and a post-quantum instantiation. In *International Conference on Provable Security*. Springer, 258–276.

[13] David Derler, Sebastian Ramacher, and Daniel Slamanig. 2018. Short double-and n-times-authentication-preventing signatures from ECDSA and more. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 273–287.

[14] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. 2007. Implementing cryptographic pairings over Barreto-Naehrig curves. In *International Conference on Pairing-Based Cryptography*. Springer, 197–207.

[15] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory* 31, 4 (1985), 469–472.

[16] Amos Fiat and Moni Naor. 1993. Broadcast encryption. In *Annual International Cryptology Conference*. Springer, 480–491.

[17] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 186–194.

[18] Matthias Fitzi, Juan Garay, Shyamnath Gollakota, C Pandu Rangan, and Kannan Srinathan. 2006. Round-optimal and efficient verifiable secret sharing. In *Theory of Cryptography Conference*. Springer, 329–342.

[19] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM Journal on computing* 18, 1 (1989), 186–208.

[20] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. 2006. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*. Acm, 89–98.

[21] Jonathan Katz. 2010. *Digital signatures*. Springer Science & Business Media.

[22] Aggelos Kiayias and Qiang Tang. 2013. How to keep a secret: leakage deterring public-key cryptosystems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 943–954.

[23] Ranjit Kumaresan and Iddo Bentov. 2014. How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014*

ACM SIGSAC Conference on Computer and Communications Security. ACM, 30–41.

[24] Jinyuan Li, Maxwell Krohn, David Mazieres, and Dennis Shasha. 2004. SUNDR: Secure untrusted data repository. In *USENIX Symposium on Operating Systems Design and Implementation*, Vol. 4. 9–25.

[25] Yannan Li, Guomin Yang, Willy Susilo, Yong Yu, Man Ho Au, and Dongxi Liu. 2019. Traceable monero: Anonymous cryptocurrency with enhanced accountability. *IEEE Transactions on Dependable and Secure Computing* (2019), 10.1109/TDSC.2019.2910058.

[26] Satoshi Nakamoto et al. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[27] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*. Springer, 129–140.

[28] Bertram Poettering. 2018. Shorter double-authentication preventing signatures for small address spaces. In *International Conference on Cryptology in Africa*. Springer, 344–361.

[29] Bertram Poettering and Douglas Stebila. 2014. Double-authentication-preventing signatures. In *European Symposium on Research in Computer Security*. Springer, 436–453.

[30] Bertram Poettering and Douglas Stebila. 2017. Double-authentication-preventing signatures. *International Journal of Information Security* 16, 1 (2017), 1–22.

[31] Charles Rackoff and Daniel R Simon. 1991. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Annual International Cryptology Conference*. Springer, 433–444.

[32] Tim Ruffing, Aniket Kate, and Dominique Schröder. 2015. Liar, liar, coins on fire!: Penalizing equivocation by loss of bitcoins. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 219–230.

[33] Michael Scott. 2003. Multiprecision integer and rational arithmetic C/C++ library (MIRACL). *URL: http://www. shamus. ie* (2003).

[34] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[35] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. 2016. Keeping authorities "honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*. Ieee, 526–545.

[36] Alin Tomescu and Srinivas Devadas. 2017. Catena: Efficient non-equivocation via bitcoin. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 393–409.

[37] Xingjie Yu, Michael Thang Shiwen, Yingjiu Li, and Robert Deng Huijie. 2017. Fair deposits against double-spending for bitcoin transactions. In *2017 IEEE Conference on Dependable and Secure Computing*. IEEE, 44–51.