ABERand: Effective Distributed Randomness on Ciphertext-Policy Attribute-Based Encryption

Liang Zhang, Haibin Kan, Zening Chen, Ziqi Mao, and Jinjie Gao

***Abstract*—Distributed randomness is very useful for many applications, such as smart contract, the proof-of-stake-based blockchain, elliptic curve generation and lottery. Randomness beacon protocols are proposed, which are aimed at continuously distributed randomness generation. However, a reliable source of distributed randomness is gained with difficulty because of Byzantine behavior which may lead to bias for distributed randomness. These Byzantine behaviors include, but not limited to, the "last actor" problem, DoS attack, and collusion attack. Various cryptography schemes have been used to generate distributed randomness. Current constructions face challenging obstacles due to high communication overheads and collusion problems. Given these barriers, we propose a new protocol that is the first precept to utilize attribute-based encryption for distributed randomness (ABERand). Compared to existing state-of-the-art public distributed randomness protocols, ABERand possesses distinguished scalability, security and efficiency. More specifically, we resolve the "last actor" problem and make ABERand an intensive output randomness beacon with communication complexity $O(n^2)$, computation complexity $O(1)$, verification complexity $O(n)$, and communication complexity $O(n)$ of nodes adding/removing.**

***Index Terms*—distributed randomness, ciphertext-policy attribute-based encryption, last actor, proof-of-stake, blockchain**

I. Introduction

How to generate public distributed random values among mutually distrustful nodes over a distributed network was first proposed by Blum [1] in 1981, thereby introducing coin-tossing protocols. Public randomness beacon, which aims to generate fresh, unpredictable and unbiased random values at certain intervals, was formalized by Rabin [2] in 1983. Many facts have indicated that the process of generating randomness should be more public and transparent to establish credibility and impartiality. For example, transparent machines with balls flying inside which seems to create complete chaos for the final result, are commonly used for national lotteries. However, a method like this is exposed to be unreliable.[1] The teams in NBA that missed the playoffs in the previous year will participate in a lottery to determine the draft order to sign top amateur players. Technologies related to blockchain [3], [4], such as consensus protocols [5], [6] and smart contracts [7], [8], have increased the demand for randomness beacons. These situations indicate that random values should not be predicted before being generated but must be publicized later. Recently, coin-tossing protocols and distributed randomness beacons have received increasing attention because randomness is both a vital component and application of blockchain. It is critical that the random number must be generated reliably.

The rise of blockchain has brought about a new perspective on how to gain and how to use distributed randomness [5], [6], [9], [10], [12]. Permissionless blockchain systems implement consensus protocol by picking one or a subset of participants to pack the new block and announce the network's latest state. Some blockchain systems select leaders via proof of work (PoW) [3], in which candidates solve difficult puzzles. However, the performance of PoW is low, and the computational cost of PoW is high. Proof-of-stake (PoS) [5], [11] protocols, which rely on virtual resources, needs an algorithm to elect a leader for packing the next block. Kiayias et al. [5] pointed out that leader election is a fundamental issue of PoS-based protocols since an adversary may manipulate the result if any entropy is introduced. The generation of manipulation resistant and unpredictable distributed randomness beacon is a solution of leader election for PoS-based protocols. It is of vital importance that distributed randomness beacons possess properties of scalability, availability, unpredictability, bias resistance and public verifiability [13]. In other words, public distributed randomnesses should be unpredictable before generation, should be unbiased at the time of generation, and should be publicly verifiable after generation, even if there are malicious or collusive participants.

A. Background and Motivation

A substantial number of studies related to distributed randomness beacon protocols have been published both in academia and in the industry due to the rise of blockchain. Some protocols, such as national lotteries [14], NIST Randomness Beacon[2], and Lavarand[3], rely on third parties or physical sources of noises. Some other protocols abide by the commit-and-reveal protocol or commitment scheme[4] [17] principle. In the commit-and-reveal protocol, values are chosen and "locked" in the commit phase, and they are verified after the reveal phase. To illustrate the protocols from the perspective of cryptography, they are divided into PoW-based ones [9], [15], hash-based ones [17], [18], publicly verifiable secret

Liang Zhang, Haibin Kan, Zening Chen, Ziqi Mao, and Jinjie Gao are with Shanghai Key Laboratory of Intelligent Information Processing, School of Computer Science, Fudan University, Shanghai 200433, P. R. China, and Fudan-Zhongan Joint Laboratory of Blockchain and Information Security, Shanghai Engineering Research Center of Blockchain, Shanghai Institute of Intelligent Electronics & Systems, Shanghai Institute for Advanced Communication and Data Science, Shanghai, P.R. China, Email: hbkan@fudan.edu.cn

This work was supported by National Natural Science Foundation of China (Grant No. 61672166), the Plan of Shanghai Excellent Academic Leaders (Grant No. 16XD1400200), the Innovation Plan of Shanghai Science and Technology (Grant No. 16JC1402700) and Shanghai Leading Talent Programmes.

sharing (PVSS)-based ones [5], [16], [19], and signature-based ones [20], [21].

PoW-based method. In the traditional method, financial data are regarded as a seed of public random numbers [22]. Heuristically, public random numbers can be obtained as a side-effect of Bitcoin's [9] proof-of-work-based consensus system. Compared to traditional financial data, the bitcoin market is open 24 hours a day, 7 days a week, and no trusted or centralized party is needed to maintain the system. The incentives of bitcoin consensus make every participant scramble to publish the nonce of a new block header. The nonce is therefore used to generate distributed randomness. However, if a miner gets more benefit than the incentives from manipulating the randomness by withholding attack [23], the final randomness will be biased. [9] focused on analyzing the financial cost when a miner withheld the nonce rather than providing an unbiased solution. Pierrot et al. [24] presented a detailed analysis of how a malicious entity could manipulate the random numbers on a public blockchain, even with a limited financial budget and computational power. In a word, the PoW-based method is not strictly bias-resistant.

Hash-based method. Popov's research [25] introduced the principle of the hash-based distributed randomness protocol. To solve the "last actor" problem, wherein the last one has the option never to reveal his/her input, participants are divided into groups, and strong settings are assumed. More precisely, the settings are 1) at least one group contains only honest members, and 2) no group consists entirely of colluding parties. Iterated hash [17] is the basic idea to form a delay function [17], [18], [26], which is also called a slow-time function. The parameter of iteration count c is set so that every participant is deprived of the opportunity to compute an optional output until it is too late to manipulate it. Iterated hash can resolve the "last actor" problem, but it's a problem on how to set the value of c.

PVSS-based method. In 1999, Schoenmakers [27] proposed the publicly verifiable secret sharing (PVSS) scheme. A dealer in PVSS is a temporary participant who distributes shares according to his/her secret. PVSS is a verifiable secret sharing scheme with the property that anyone can verify the validity of the shares distributed by the dealer. It is reported that Ouroboros [5] was the first to adopt PVSS as a provably secure proof-of-stake blockchain protocol. It relies on a combination of PVSS and other cryptographic primitives to obtain the last verifiable randomness. To achieve scalability, RandHound and RandHerd are systems with a failure probability of 0.08%, based on PVSS and collective signing [28]. Scrape optimizes a variant of Schoenmakers's PVSS with an existing bulletin board, thus reducing computation complexity to $O(n^2)$. HydRand [13] was proposed as a standalone, self-contained protocol with a permissioned system model. In PVSS-Based methods, verification is needed and it is costly. What's more, all the participants have to re-run the PVSS protocol when adding/removing participants.

Signature-based method. Algorand [20] builds a distributed ledger by combining a randomized Byzantine agreement protocol [29] and a randomness beacon based on Verifiable Random Function (VRF) [30]. VRF is used to produce unique signatures. However, strictly speaking, the produced randomness is not bias-resistant. Dfinity [21] is a blockchain project, and its core technique lies in a decentralized randomness beacon, which acts as a VRF. The innovative beacon is based on the threshold Boneh-Lynn-Shacham (tBLS) [31] signature and distributed key generation (DKG) [32], [33]. Dfinity allows for only t-out-of-n participants to compute a signature on a message so that a public delayed random value can be obtained even with malicious or colluding participants. However, Dfinity has to rerun DKG when a participant joins or leaves. Besides, the worst communication complexity of running DKG is $O(n^2 \cdot f)$, where f is the number of evil nodes.

B. Our Contributions

We define a totally decentralized threshold multi-authority ciphertext-policy attribute-based encryption(MA CP-ABE, more details in II-D and II-E)) protocol according to the Y. Rouselakis and B. Waters MA CP-ABE construction [34] and put forward ABERand as a public distributed randomness beacon. ABERand uses the totally decentralized threshold MA CP-ABE in a commit-and-reveal scheme, so that no adversaries could succeed to abort the protocol or bias the final result.

Compared to hash-based and other public-key based approaches, ABERand solves the fatal "last actor" problem. And compared to current state-of-the-art methods, ABERand lowers the computation and communication complexity on the whole. The whole network communication complexity is $O(n^2)$ with an asynchronized communication model. The single node computation complexity is only $O(n)$ and the verification complexity of final randomness is only $O(1)$ in ABERand, while the single node computation complexity is only $O(1)$ and the verification complexity of final randomness is only $O(n)$ in ABERand′. The network communication complexity is only $O(n)$ when nodes join or leave.

C. Organization

Section II introduces the notation and cryptographic primitives that will be used throughout the paper. Section III briefly gives the idea of a totally decentralized threshold MA CP-ABE. Section IV provides the model of our protocol, followed by concrete construction in Section V. In section VI, we introduce the implementation and performance of our protocols. Section VII, we analyze the security of the protocol. Finally, in section VIII, we conclude with directions for future work.

II. Preliminaries

In this section, we provide a brief review of the background theory on access structures, linear secret-sharing schemes, bilinear maps, CP-ABE, multi-authority CP-ABE, complexity assumption and security model for CP-ABE.

A. Access Structures

Definition 1: [36] Let P = $\{P_1, P_2, ..., P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\{P_1, P_2, ..., P_n\}}$ is monotone if for any B and C, the following holds: if B $\in \mathbb{A}$ and C $\subseteq$ B, then C $\in \mathbb{A}$. An access structure is a collection $\mathbb{A}$ of nonempty subsets of $\{P_1, P_2, ..., P_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{P_1, P_2, ... P_n\}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in A are called the unauthorized sets.

In an ABE system, the roles of the parties are defined by attributes. An access structure in ABE contains the authorized sets of attributes. In our construction, we only consider monotone access structures, which means that when a user acquires more attributes, he will not lose his possible decryption privileges. There are different forms to describe access policies [37], such as minimal form access structures, monotone boolean formulas, threshold-gate access tree and monotone access tree. In this paper, we take advantage of monotone access trees, more precisely, the threshold-gate access tree.

B. Linear Secret Sharing Schemes

Definition 2: [41] Let p be a prime and Ω be the attribute universe. A secret sharing scheme Π with a domain of secrets $\mathbb{Z}_p$ realizing access structures on Ω is linear over $\mathbb{Z}_p$ if:

1. The shares of a secret z $\in \mathbb{Z}_p$ for each attribute form a vector over $\mathbb{Z}_p$.

2. For each access structure $\mathbb{A}$ on Ω, there exists a matrix a matrix A $\in \mathbb{Z}_p^{l \times n}$, called the share-generating matrix, and a function σ that labels the rows of A with attributes from Ω, i.e., σ: [l] $\longrightarrow \Omega$, which satisfy the following: during the generation of the shares, we consider the column vector υ = $(z, r_2, ..., r_n)^{\perp}$, where $r_2, ..., r_n \overset{R}{\longleftarrow} \mathbb{Z}_p$. Then, the vector of l shares of the secret z according to Π is equal to λ = A$\upsilon \in \mathbb{Z}_p^{l \times 1}$. The share λ_j with j $\in$ [l] "belongs" to attribute σ(j).

We will be referring to the pair (A, σ) as the policy of the access structure $\mathbb{A}$.

As shown in [36], any monotone access structure can be achieved by a linear secret sharing scheme. Any monotone access structure can be converted into LSSS matrices. [37]

C. Bilinear Maps and Complexity Assumption

Let $\mathbb{G}_1$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}_1$ and e be a bilinear map, $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$. The bilinear map e has the following properties [40]:

1. Bilinearity: for all μ, $\upsilon \in \mathbb{G}_1$ and a,b $\in \mathbb{Z}_p$, we have $e(\mu^a, \upsilon^b) = e(\mu, \upsilon)^{ab}$.

2. Nondegeneracy: $e(g, g) \neq 1$.

3. Computability: There is an efficient algorithm for computing $e(\mu, \upsilon)\ \forall\ \mu, \upsilon \in \mathbb{G}_1$.

We call $\mathbb{G}_1$ a bilinear group if the group operation in $\mathbb{G}_1$ and the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ are both efficiently computable. The above definition considers the so-called symmetric groups, where the two arguments of the pairing belong to the same group. In general, there exist asymmetric bilinear groups, where $e' : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ and $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are three different groups of prime order p. Several asymmetric instantiations of bilinear groups possess beneficial properties such as faster operations under the same security level and/or easier hashing to group elements.

Definition 3: (q-Decisional Parallel Bilinear Diffie-Hellman Exponent 2 Assumption). [34] Choose a bilinear group G of order p, and a non-degenerate bilinear mapping $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$. Let s,a,$b_1$,$b_2$,...,$b_q \in \mathbb{Z}_p$, $R \in \mathbb{G}_T$ choose at random. Let

$$D = (\mathbb{G}, p, e, g, g^s, \{g^{a^i}\}_{i \in [2q], i \neq q+1}, \{g^{b_j a^i}\}_{(i,j) \in [2q, q], i \neq q+1}, \{g^{s/b_i}\}_{i \in [q]}, \{g^{s a^i b_j / b_{j'}}\}_{(i,j,j') \in [q+1, q, q], j \neq j'})$$

No probabilistic polynomial-time algorithm A can distinguish the tuple $(D, e(g, g)^{sa^{q+1}})$ from the tuple (D, R) with more than a negligible advantage. The advantage of A is

$$\left| Pr[A(D, e(g, g)^{sa^{q+1}}) = 1] - Pr[A(D, R) = 1] \right|$$

where the probability is taken over the random choice of $s, a, b_1, b_2, ..., b_q \in \mathbb{Z}_p, R \in \mathbb{G}_T$, and the random bits consumed by A.

D. CP-ABE

A great variety of ABE schemes have been proposed, and they are mainly divided into two categories: key-policy attribute-based encryption (KP-ABE), such as in [38], [39], and ciphertext-policy attribute-based encryption (CP-ABE), such as in [40]–[43]. In the KP-ABE scheme, an access structure is embedded in the secret key, and the ciphertext is associated with an attribute set. In the CP-ABE scheme, an attribute set is embedded in the secret key, and the ciphertext is associated with an access structure. In our situation, CP-ABE is our preferred choice. A nice feature of CP-ABE is that data can be encrypted without knowledge of the decryptors. Therefore, users can obtain keys for decryption at some future time after the data have been encrypted with some specific policy. CP-ABE is often realized on bilinear maps as described in the previous subsection. CP-ABE usually contains four algorithms: [40]

Setup. The setup algorithm takes security parameters as input and outputs the public parameters PK and a master key MK.

Encrypt(PK, M, (A, σ)). The encryption algorithm takes as input the public parameters PK, a message M, and an access structure (A, σ) over the universe of attributes. And it produces a ciphertext CT.

Key Generation(MK, *S*). The key generation algorithm takes as input the master key MK and a set of attributes *S* that describe the key. It outputs a private key SK.

Decrypt(PK, CT, SK). The decryption algorithm takes the public parameters PK, a ciphertext CT and a private key SK as input. If and only if *S* satisfies the access structure (A, σ) mentioned above, the algorithm successfully outputs the message M.

E. Multi-Authority CP-ABE

Most existing CP-ABE schemes [40]–[43] have only one authority that is responsible for attribute management and key distribution. This one-authority scenario can be problematic since the single authority can issue private keys to every user in the system. If the authority is malicious or compromised, the encrypted data are at risk. A multi-authority (MA) CP-ABE [34] system is comprised of the following five algorithms:

Global Setup(λ) $\longrightarrow$ GP. it takes in the security parameter λ and outputs global parameters GP for the whole system.

Authority Setup(GP, θ) $\longrightarrow$ SK_θ, PK_θ. Each authority θ takes GP as input to produce its secret key and public key pair, SK_θ, PK_θ.

Encrypt(M, (A, σ), GP, $\{\mathrm{PK}_\theta\}$) $\longrightarrow$ CT. The algorithm takes in a message M, an access structure (A, σ), a set of public keys, and the global parameters. It outputs a ciphertext CT.

KeyGen(GID, GP, u, SK_θ) $\longrightarrow$ $\mathrm{K}_{u,\mathrm{GID}}$. The algorithm takes in an identity GID, the global parameters, an attribute u belonging to the authority θ, and the secret key SK_θ for this authority. It produces a key $\mathrm{K}_{u,\mathrm{GID}}$ for this attribute and identity pair (GID, u).

Decrypt(CT, GP, $\{\mathrm{K}_{u,\mathrm{GID}}\}$) $\longrightarrow$ M. The decryption algorithm takes in the global parameters, the ciphertext, and a collection of keys corresponding to the attribute and identity pairs that all have the same fixed identity GID. Only and only if the collection of attributes u satisfies the access structure corresponding to the ciphertext, it outputs the message M.

Definition 4: [34] A multi-authority CP-ABE system is said to be correct if GP is obtained from the global setup algorithm, CT is obtained from the encryption algorithm on the message M, and $\mathrm{K}_{u,\mathrm{GID}}$ is a set of keys obtained from the key generation algorithm for the same identity GID and for a set of attributes satisfying the access structure of the ciphertext, Decrypt(CT, GP, $\mathrm{K}_{u,\mathrm{GID}}$) = M.

III. Decentralized Threshold MA CP-ABE

From [37], we can know that threshold-gate access tree is equal to boolean formulas for an access structure. So a multi-authority CP-ABE is born to be a threshold MA CP-ABE. In some MA CP-ABE schemes, such as [34], [44], [45], multiple authorities jointly manage the attribute sets of the decryptors. [44] realizes a robust and verifiable multi-authority access control system in public cloud storage, but it has a certificate authority (CA). [34], [45] satisfy the scenario in which attributes are obtained from different authorities and guarantee that the secret key cannot be obtained by any authority alone. However, the attribute universe of [45] is restricted. A threshold MA CP-ABE scheme that is also somewhat decentralized [35]. We adapt MA CP-ABE [34] into a totally decentralized threshold MA CP-ABE protocol to produce public distributed randomness.

A totally decentralized (t,n)-threshold multi-authority CP-ABE in our system is a protocol that everyone is an authority, an encryptor and a decryptor. All participants have equal rights. In order to implement decentralized (t,n)-threshold multi-authority CP-ABE, the five algorithms of MA CP-ABE have to be invoked as follows: **Global Setup** algorithm is invoked once in a public way; each participant invokes **Authority Setup** algorithm once for his/her key pair; all participants, as encryptors, use a specified kind of access structure that is confirmed by them to encrypt text in **Encrypt** algorithm; each participant deliver his/her key generated in **KeyGen** algorithm after **Encrypt** algorithm; Anyone collects t-out-of-n keys could successfully invoke **Decrypt** algorithm and obtain all the original message. Concrete examples and more details are described in the ABERand protocol section V.

The totally decentralized protocol works well in the commit-and-reveal scheme to produce public distributed randomness, because the **Encrypt** algorithm could "lock" values in the commit phase, the **KeyGen** algorithm and the **Decrypt** could reveal values in the reveal phase. The totally decentralized (t,n)-threshold multi-authority CP-ABE makes a commit-and-reveal scheme atomic, thus solving the "last actor" problem.

IV. System and Threat model

We assume an asynchronous system model with a connected network that honest nodes have path(s) to each other. We target the number of participants in the system n and presuppose that there is an upper bound time for both calculation and communication. f $(< n/2)$ is the maximum number of malicious nodes that may cause Byzantine failures or disobey our protocol. Malicious nodes, also called adversaries, can send their preset data through collaboration. A node is considered to be honest if it abides by the protocol; otherwise, it is considered to be faulty. We assume that the goal of the adversary is to bias or DoS-attack the protocol. Each message of the protocol is signed by the sender and a middleman. A middleman is one who forwards a message in a peer-to-peer network. The counterparts only accept and act upon a message if it is correctly signed. If the adversary could disobey or abort the protocol by preventing the ciphertexts from being decrypted, then bias is introduced into final result.

V. ABERand Protocol

A. System Overview

Figure 1 is the flowchart that summarizes the overview of our protocol. During system setup, global parameters are calculated in a public way. A participant generates his/her own private and public key pair according to global parameters. All participants have to exchange their public keys, which are aggregated when encrypting a message, while secret keys are stored locally. Each epoch of ABERand beacon is separated into 8 phases. Each participant takes a random value as input and generates the ciphertext in the 1st phase. In the 2nd phase, participants share their own ciphertexts. Then, all participants require the ciphertext list from counterparts in the 3rd phase. Next, all participants calculate common valid ciphertexts in the 4th phase. In the 5th phase, participants generate keys for the current epoch and broadcast keys in the 6th phase. In the 7th phase, participants decrypt all valid ciphertexts and obtain

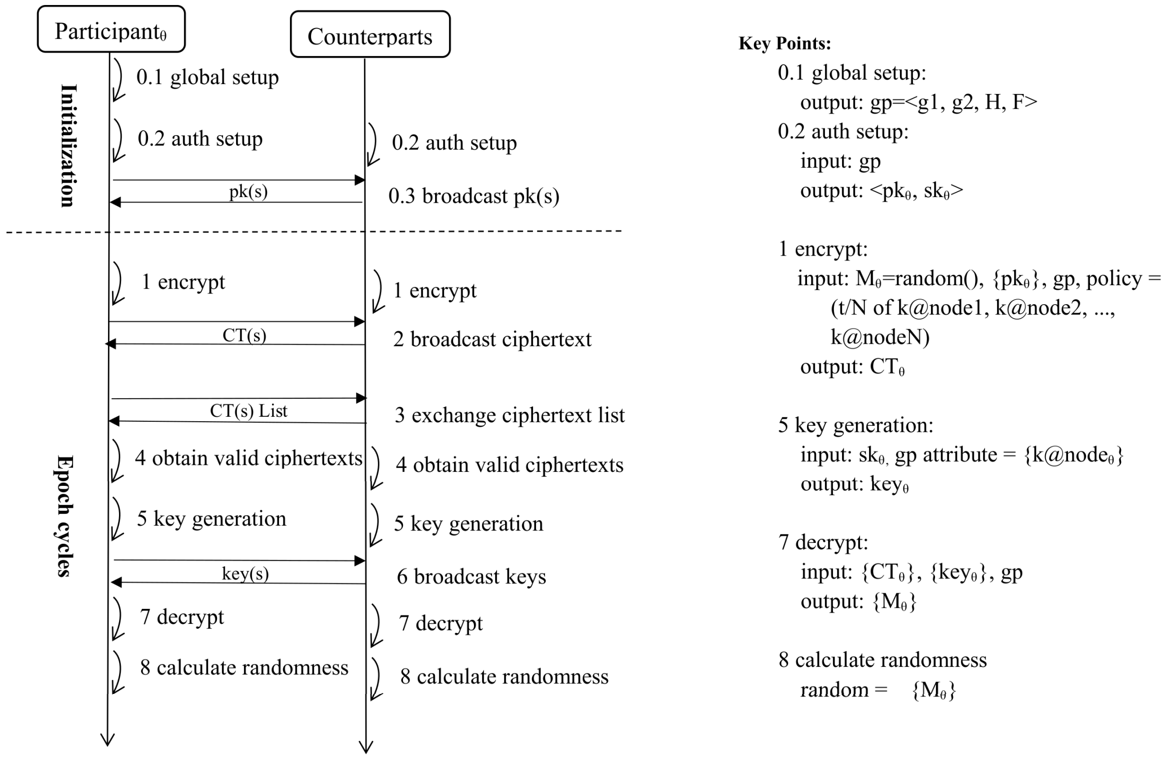

Fig. 1. System overview

a final random number by calculating eXclusive-OR (XOR) of all messages in the last phase. Agreement on the random number is automatic because of the correctness property of the CP-ABE scheme. It is obvious that a peer-to-peer network is utilized in the 2nd, 3rd, and 6th phases.

Same as CP-ABE [34], the threshold MA CP-ABE implementation have attributes in the form of "[attribute–id]@[authority–id]". To distinguish the attributes and to avoid collusion, the "attribute–id" and the "authority-id" are case-sensitive alphanumeric strings. The mapping T mentioned in Section V-D only extracts the part after the @ of the attribute string. For simplicity, all the attributes in ABERand follow the format of "k@nodeId", where k is the current epoch number, and "nodeId" is the authority's identity. Actually, attributes in ABERand are necessary for three aspects. Firstly, "[attribute–id]" represents the unique epoch number as a common-sense according to our protocol; Secondly, "[attribute–id]@[authority–id]" is used in access structure which makes threshold cryptography possible. Thirdly, attributes are designed in this way so that ABERand is a totally decentralized protocol. In traditional public-key encryption schemes without attributes, malicious participants have the opportunity to hide their private key which will bias or abort distributed randomness beacon.

B. Signature Scheme

To simplify the construction for the signature, the "key generate", "sign", and "verify" functions of our signature scheme are defined as:[5] [31]

key generate: The key generation algorithm uses $sk \xleftarrow{R} \mathbb{Z}_p$ as a secret key, where p is the prime order of a bilinear group $\mathbb{G}$.

sign: Given the private key *sk* and some message m, we compute the signature by hashing the bit string m as h = H(m) and output the signature $sign = h^x$

verify: Given a signature *sign* and a public key g^{sk}, we verify that $e(sign, g) = e(H(m), g^{sk})$, where g is the generator of the bilinear group $\mathbb{G}$.

C. Data Structure in Communication

In real application environments, the peer-to-peer network is very complicated due to the free joining and exiting of nodes and the existence of malicious nodes. To guarantee the transferred and forwarded data is valid, a sender or a middleman will sign data. To describe the data structure, we simplify it as <message, ts1, ts2, pk1, pk2, sig1, sig2>. Specifically, "message" is the original data that need to be sent, "sig1" is the sender's signature, "pk1" is the sender's public key, and "ts1" is the timestamp when "sig1" is generated. "sig2" is the last middleman's signature, pk2 is the last middleman's public key, and "ts2" is the timestamp when "sig2" is generated. If the nodes are directly connected, which means no middleman exists during the data transmission, "ts2" will be -1, and "sig2" will be *null.*

We emphasize that the signing process is noninteractive. Below is a JSON struct, which shows an example of the data structure in communication.

```
{
    “message”:   “1234567890”,
    “ts1”:   1546823623,
    “ts2”:   -1,
    “pk1”:   “WJCP4GdQx19lhTLbKxELsL5gv0K...”,
    “pk2”:   “null”,
    “sig1”:   “eETZr8mpiQyqC...”,
    “sig2”:   “null”,
}
```

D. Details of the Protocol

System Initialization. The system invokes the **Global Setup** algorithm in a public way and sets the initial round variable k to 1 (k increases by 1 at the end of every round). The global setup algorithm takes as input the security parameter λ and chooses a suitable bilinear group $\mathbb{G}$ of prime order p with generator g. It also defines a function H that maps global identities GID to elements of $\mathbb{G}$ and another function F that maps strings, interpreted as attributes, to elements of $\mathbb{G}$. Both of these functions will be modeled as random oracles in the security proof. T: U $\longrightarrow$ U$_\theta$ is a publicly computable mapping function that maps each attribute to a unique authority θ.

The initial number of participants is n; thus, the number of honest participants t should satisfy the demand $t \geq 1 + n/2$. Each of the n participants, with nodeId$_\theta$ as its identity, invokes the **Authority Setup** algorithm and acquires key pairs {PK$_\theta$, SK$_\theta$}. The authority setup algorithm chooses two random exponents, α_θ,$y_\theta \xleftarrow{R} \mathbb{Z}_p$, and calculates [34]

$$\text{PK}_\theta = (e(g,g)^{\alpha_\theta}, g^{y_\theta}),\ \text{SK}_\theta = (\alpha_\theta, y_\theta).$$

All $\{\text{PK}_\theta\}_{for\ all\ \theta}$ are public, while $\{\text{SK}_\theta\}_{for\ all\ \theta}$ are private. We denote Δt as the upper bound overhead of one round of communication when sharing data among participants.

Therefore, the global parameters of ABERand are GP = $\{k,\ \mathbb{G}_1,\ g1,\ \mathbb{G}_2,\ g2,$ H, F, T, t, n, Δt, $\{PK_\theta\}_{for\ all\ \theta}$, $\{nodeId_\theta\}_{for\ all\ \theta}\}$. $g1$, $g2$ are the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.

Encrypt and Broadcast. The **Encrypt** algorithm takes as input a message M, an access structure (A, σ) with A $\in$ Z$^{l\times n}$, the public keys of the relevant authorities, and the global parameters. The function ρ : [l] as $\rho(\cdot)$ = T($\sigma(\cdot)$) to realize the mapping of rows to authorities. The algorithm first creates vectors $\boldsymbol{\upsilon} = (z, \upsilon_2, \cdots, \upsilon_n)^\perp$ and $\boldsymbol{\omega} = (0, \omega_2, \cdots, \omega_n)^\perp$, where $z, \upsilon_2, \cdots, \upsilon_n, \omega_2, \cdots, \omega_n \xleftarrow{R} \mathbb{Z}_p$. We let λ_x denote the share of z corresponding to row x, i.e., $\lambda_x = \boldsymbol{A}_x\boldsymbol{\upsilon}$, and ω_x denote the share of 0, i.e., $\omega_x = \boldsymbol{A}_x\boldsymbol{\omega}$, where $\boldsymbol{A}_x$ is the x-th row of A. For each row x of A, it chooses a random $t_x \xleftarrow{R} \mathbb{Z}_p$. The ciphertext is computed as: [34]

$$C_0 = Me(g,g)^z,$$

$$\{C_{1,x} = e(g,g)^{\lambda_x} e(g,g)^{\alpha_{\rho(x)} t_x},\ C_{2,x} = g^{-t_x},$$

$$C_{3,x} = g^{y_{\rho(x)} t_x} g^{\omega_x},\ C_{4,x} = F(\sigma(x))\}$$

Each epoch starts when all participants begin to generate their own random numbers. We use t_0 to denote the current time. The random number will be the input of the encrypt algorithm. The access structure for the random number is constructed as “t-of-n $\{k@\text{nodeId}_\theta\}_{for\ all\ \theta}$” so that the output ciphertext can be decrypted when t-out-of-n participants combine their secret keys with attribute $k@\text{nodeId}_\theta$ in the future. After generating ciphertext CT_θ privately, all participants broadcast their ciphertexts in the form of the data structure mentioned in section V-C to each other via the peer-to-peer network in the following period Δt.

Broadcast and Validate Ciphertexts. After Δt, the current moment is t_1 (= t_0 + Δt). At the time of t_1, all parties received the data lists containing $\{CT_\theta\}_{for\ all\ \theta}$. They first verify the signatures to guarantee that all ciphertexts are not modified. Even when the data are verified, there is still the probability that different participants hold different ciphertexts lists because of a malicious party’s forgery. Therefore, in the next Δt, all participants broadcast their sorted ciphertexts list to each other, such that honest parties can obtain the maximum amount of truly valid ciphertexts by comparing all sorted lists. For example, there are 5 parties, in which party1 is malicious in the “Encrypt and Broadcast” phase. Party1 generates two different random values and sends different ciphertexts to different parties, i.e., “122456” to party2 and “123456” to others. Figure 2 demonstrates how to obtain the final valid ciphertexts in a node. In this case, all nodes compare all the ciphertexts lists and acquire [“789012”, “345678”, “567890”, “678901”] as the valid one in the end. Suppose that party1 continues to behave maliciously, and it sends [“122456”, “789012”, “345678”, “567890”, “678901”] to party2 and sends [“123456”, “789012”, “345678”, “567890”, “678901”] to the others. However, party2 will exchange the ciphertexts list with some other participant and receive different versions of party1’s ciphertext. Once this happens, it means that party1 has behaved maliciously or errors have occurred in the transmission; then, party2 will abandon the ciphertext from party1. The other nodes will behave like party2. The overhead of broadcasting the ciphertexts list among all participants via the asynchronous peer-to-peer network is another Δt; then, the current time is t_2 (= t_1 + Δt).

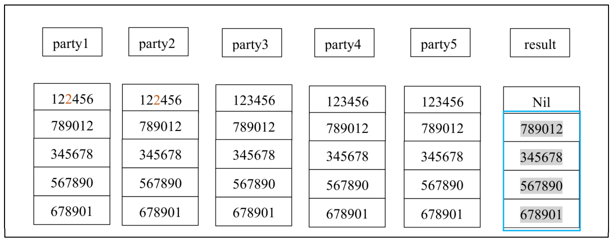

Fig. 2. The lists in the blue box are the valid ciphertexts

Generate and Broadcast Keys. The **KeyGen** algorithm takes as input a global identifier GID, the identifier $_\theta$ of the authority, the attribute u, and the authority’s secret key and the global parameters. It should be the case that $u \in T^{-1}(\theta)$, i.e., the attribute is controlled by the specific authority. The algorithm first chooses a random $t \xleftarrow{R} \mathbb{Z}_p$, and then it outputs the secret key: [34]

$$SK_{\text{GID},u} = \{K_{\text{GID},u} = g^{\alpha_\theta} H(\text{GID})^{y_\theta} F(u)^t, \\ K'_{\text{GID},u} = g^t\}$$

From t_0 to t_1, all participants have encrypted their random values and shared them; from t_1 to t_2, all participants compare all the ciphertexts lists to eliminate invalid ones. For example, at the time of t_2, participant$_\theta$/authority$_\theta$ starts to generate his/her key with the attribute "k@nodeId$_\theta$". Then, in the next Δt, $SK_{\text{GID},u}$ are exchanged in the form of the data structure mentioned in section V-C among the participants; subsequently, t_3 (= t_2 + Δt). At the moment t_3, as long as more than t-out-of-n of the total nodes behave honestly and have shared their real keys, they can decrypt the valid ciphertexts generated at t_0.

To accelerate the verification and to check out whether the key is valid, we demand he/she to send two extra fields along with the key. The first one is the original random number M_θ generated at t_0, so that anyone could verify the decryption with computation complexity $O(1)$. The second one is a ciphertext whose original message is "k" and whose access structure is "1-of-1 k@node$_\theta$". The key in this way could be proved to be valid.

Decrypt ciphertexts. Let (A, σ) be the access structure of the ciphertext. If the decryptor has the secret keys $\{K_{\text{GID},\sigma(x)}, K'_{\text{GID},\sigma(x)}\}$ for a subset of rows $\boldsymbol{A}_x$ of A such that (1,0,$\cdots$,0) is in the span of these rows, then for each such row x, he/she invokes the **Decrypt** algorithm by computing the following: [34]

$$C_{1,x} \cdot e(K_{\text{GID},\sigma(x)},\ C_{2,x}) \cdot e(H(\text{GID}),\ C_{3,x}) \cdot \\ e(K'_{\text{GID},\sigma(x)},\ C_{4,x}) = e(g,\ g)^{\lambda_x} e(H(\text{GID}),\ g)^{\omega_x}$$

The decryptor then calculates constants $c_x \in \mathbb{Z}_p$ such that $\sum_x c_x \boldsymbol{A}_x$ = (1,0,$\cdots$,0) and computes:

$$\prod_x (e(g,g)^{\lambda_x} e(H(\text{GID}),g)^{\omega_x})^{c_x} = e(g,g)^z$$

This is true because $\lambda_x = \boldsymbol{A}_x \boldsymbol{v}$ and $\omega_x = \boldsymbol{A}_x \boldsymbol{\omega}$, where $(1,0,\cdots,0)\boldsymbol{v}$ = z and $(1,0,\cdots,0)\boldsymbol{\omega}$ = 0. The message can then be obtained as follows:

$$\text{M} = C_0/e(g,g)^z$$

After collecting at least t honest participants' keys, every participant can run the **Decrypt** algorithm. Since key_θ is associated with attribute "k@node$_\theta$", with at least t reliably generated keys, the decrypt algorithm runs soundly for everyone.

Generate Random Number. Given a set S of pseudorandom numbers in the range [0,b] that are uniformly distributed over the range and independent, the XORing calculation on them produces a new pseudorandom number in [0,b]. b is a power of 2.

Proof. It is easy to observe that the map $x \leftrightarrow x \oplus c$ is a bijection over the range [0,b] regardless of the value of the constant c. Thus, if x is uniformly distributed over the range, then so is $x \oplus c$. Since this is true for any constant c, it is also true even if c itself is a random variable, as long as it does not depend on x.

Algorithm 1 Obtain final valid ciphertext list

```
1: procedure getFinalCTs(CTs)
2:     ctsNumber = CTs.length
3:     partiesNumber = CTs[0].length
4:     for all $i \in$ partiesNumber do
5:         flag $\leftarrow$ true
6:         test $\leftarrow CTs[0][i]$
7:         for all $j \in$ ctsNumber do
8:             if test! $= CTs[j][i]$ then flag $\leftarrow$ false
9:         if flag == true then FinalCTs.append(test)
       return FinalCTs
```

Algorithm 2 ABERand on a node$_\theta$

```
1: procedure ABERand
2:     if global setup not done then
3:         global_parameters = GlobalSetup()
4:     public_key$_\theta$, private_key$_\theta$ = AuthSetup()
5:     broadcast public_key$_\theta$
6:     for all public_key $\in$ all received public_key do
7:         pass $\leftarrow$ check signature of public_key
8:         if pass then
9:             public_key.append(public_key)
10:    while true do
11:        CT$_\theta$ $\leftarrow$ Encrypt(RANDOM, policy, public_keys)
12:        broadcast CT$_\theta$
13:        for all ct $\in$ all received CTs do
14:            pass $\leftarrow$ check signature of ct
15:            if pass then
16:                CTs.append(ct)
17:        broadcast CTs list$_\theta$
18:        for all cts $\in$ all received CTs list do
19:            pass $\leftarrow$ check signature of cts
20:            if pass then
21:                ValidCTs.append(cts)
22:        finalCTs $\leftarrow$ getFinalCTs(ValidCTs)
23:        KEY$_\theta$ $\leftarrow$ KeyGen(public_keys, attributes)
24:        broadcast KEY$_\theta$
25:        for all key $\in$ all received KEY do
26:            pass $\leftarrow$ check signature of key
27:            if pass then
28:                KEYs.append(key)
29:        for all CT $\in$ finalCTs do
30:            RANDOM $\leftarrow$ Decrypt(CT, KEYs)
31:            RANDOMs append RANDOM
32:        random $\leftarrow \oplus\{RANDOMs\}$
33:        broadcast and record random
```

By now, i.e., t_3, the random values $\{M'_\theta\}$ independently generated by participants are obtained with the decrypt algorithm. All participants calculate exclusive-or for these random values and will obtain the same random value for the current epoch of ABERand, i.e., $res = \oplus\{M'_\theta\}$.

All the broadcasted messages are recorded for public verification. The public verification could be done instantly just by comparing res with $\oplus\{M_\theta\}$.

Nodes Join/Leave. ABERand is a scalable protocol that permits participants to join and leave freely. When a participant joins, he/she runs the authority setup algorithm according to the global parameters, keeps the secret key, and broadcasts the public key. Meanwhile, the algorithm collects the counterparts' public keys. Then, the t and n in the global parameters are updated to $t \leftarrow 1+(n+1)/2$, $n \leftarrow n+1$. The new participant will participate in the next epoch to distribute his/her random value, which will finally be used in the XOR calculation. When a node in the system leaves, if the time of leaving is in $t_0 \sim t_2$, his/her random value will be not included in current epoch's XOR calculation; if the time of leaving is in $t_2 \sim t_3$, it will have no impact on the final randomness of the current epoch. Then set $t \leftarrow 1+(n-1)/2$, $n \leftarrow n-1$.

The join/leave event that happens in the connected network costs communication complexity of $O(n)$, and nodes in the network hardly need to do anything.

Pseudocode. Algorithm 1 "getFinalCTs" takes ciphertexts lists and then outputs common ones by comparing the input list. Then, Algorithm 2 "ABERand" follows, which produces distributed verifiable randomness:

Optimization. As mentioned in the above subsections, the generating process of distributed randomness is divided into a series of steps, i.e., $t_0 \sim t_1$, $t_1 \sim t_2$, and $t_2 \sim t_3$. Although each step is dependent on the previous steps, two continuous randomness occurrences are independent. Specifically, with pipelining optimization, ABERand can produce randomness with any small interval. Pipelining does not reduce the time that it takes to generate a random number, but it increases the number of steps that can be processed simultaneously and eliminates the delay between the generation of two random values, thus increasing throughput. Figure 3 shows how to optimize the randomness beacon in a pipelining way. In the pipelining design, every random number is delivered by the interval Δt, instead of 3Δt. To reduce storage and communication overhead, when broadcasting ciphertext lists in the period $t_1 \sim t_2$, we only broadcast ciphertext hash lists.

VI. Implementation and Evaluation

Framework. We implemented dectralized threshold MA CP-ABE in Charm [46], which is a framework for constructing cryptographic schemes and protocols. It is written in Python language. The Charm framework relies on the gmp (GNU multiple precision) arithmetic library and the PBC (pairing-based cryptography) library written in C language. We implemented the peer-to-peer network in which "ip:port" was used to represent a node. All our experiments were executed on 8 cores of an Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00 GHz with 32 GB RAM running Linux Ubuntu 4.10.0 and Python 3.6.7.

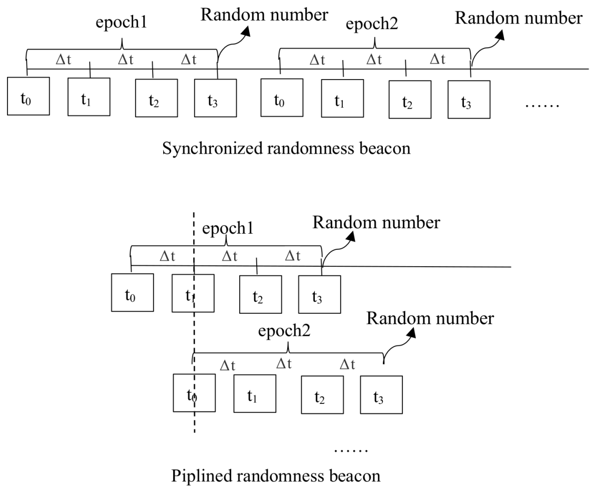

Fig. 3. Optimize randomness beacon in a pipelining way

Implementation Details. We chose supersingular symmetric EC group "SS512", which is an asymmetric group in our experiment. Namely, we had three groups, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, and the pairing e was a function from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$. The GID was set to the epoch k as a virtual decryptor. We initiated from 2 to 24 nodes to test the randomness beacon efficiency, in which nodes were full connected. The nodes in the network mainly responded to "NODECONNECTED", which represented a new connection event, and "NODEMESSAGE", which informed the node that new data are arriving.

A. Properties of ABERand

Liveness/Availability: For honest parties, the protocol successfully produces the final random output accurately. In an asynchronous system, if a participant takes action based on the actions of others, he/she will be able to evaluate multiple options to make a favorable decision or even abort the procedure. However, even a malicious participant initiates attacks mentioned above, it has no chance to break the liveness of ABERand. Since there are a sufficient number of honest parties, the (t, n)-threshold access structure will guarantee the finality and availability of ABERand.

Bias resistance: Under our setting, adversaries cannot influence the value of the random output in a meaningful way. As discussed in the discussion of liveness/availability, ABERand is a coherent protocol. Even if f $(< n/2)$ malicious parties try every means available to modify or abort the protocol, the final random output represents an unbiased, uniformly random value. They can only gain parts of the inputs for the XOR operation; however, the XOR calculation makes sure that the output is randomly distributed with more than one honest random input.

Unpredictability: Based on the security and effectiveness of MA CP-ABE and the properties of the XOR calculation, the final results cannot be predicted unless all n randomnesses are revealed. The adversary cannot construct the result without

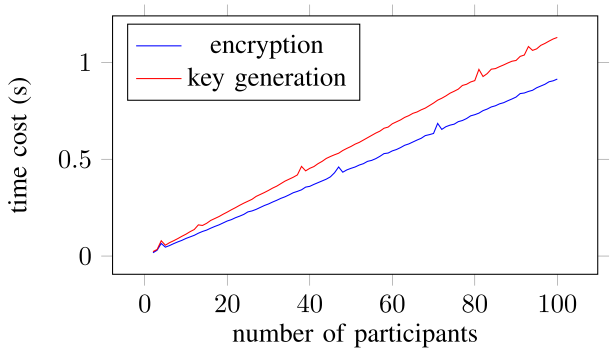

Fig. 4. Encryption and key generation cost

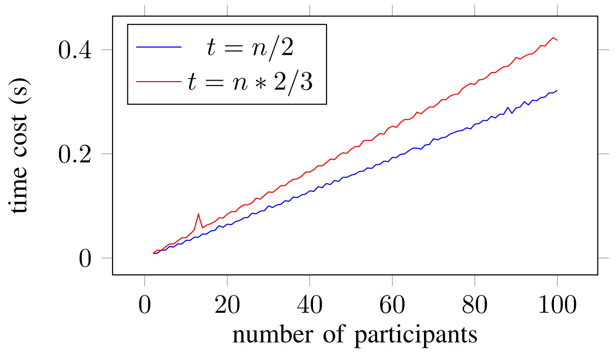

Fig. 5. Decryption cost

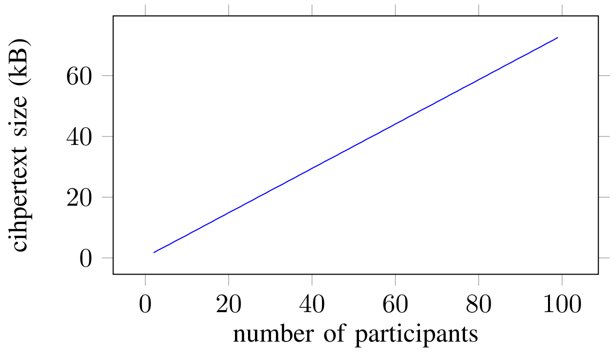

Fig. 6. Ciphertext sizes against number of AAs

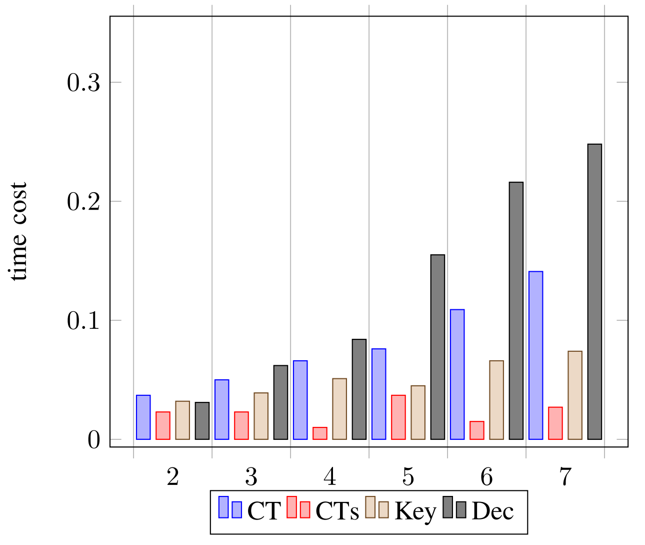

Fig. 7. Cost of different steps with different nodes

CT: cost of generating and broadcasting cihpertexts;
CTs: cost of broadcasting ciphertext lists;
Key: cost of generating and broadcasting keys;
Dec: cost of decryptions.

the keys generated by the honest nodes. No party can learn anything about the final random output before the keys are generated. ABERand ensures that the output remains unknown to all until keys are broadcasted when the participants begin to decrypt all the ciphertexts.

Public verifiability: Public distributed randomness beacon construction must be able to garner public belief in its fairness. When having got the final randomness, any third party must be able to verify that it is correctly generated. The broadcasted data are public, which can be used to replay the protocol and verify the result. If an adversary can forge the verification process, the adversary must have owned t-out-of-n of the secret keys in the system, violating the assumption that at most f ($< n/2$) nodes are controlled by the adversary. Therefore, any party could verify the legality of the random output with computation complexity $O(1)$ in ABERand and $O(n)$ in ABERand$'$.

Scalability: ABERand scales well when nodes join or leave. As is described in V-D, it costs only communication complexity of $O(n)$. For PVSS-based or BLS-based methods, communication complexity of at least $O(n^2)$ is required.

B. Performance Analysis

For a distributed randomness beacon, the frequency of producing a random value is critical. We mostly focus on the time consumption in our experiment and analyze the bottleneck parts that prevent our protocol from speeding up. We first provide a chart depicting the overhead of the decentralized threshold multi-authority CP-ABE scheme. Then, we implement the scheme in a real peer-to-peer network environment and provide charts describing the overhead of each step in our protocol. Finally, we analyze the experimental data and draw conclusions.

Figure 4 shows that the encryption and key generation time cost increase linearly with the amount of attribute authorities. Figure 5 shows the decryption time cost comparison between $t = n/2$ and $t = n * 2/3$. Figure 6 shows that the ciphertext size increases linearly with the amount of attribute authorities. All of the above are tested on one core of the CPU group. Figure 7 shows the histogram of the comparison of the time cost of different steps in ABERand. We can conclude that the “Dec” is the most time-consuming step in ABERand if ciphertexts are decrypted one after one. However, these decryptions can be done simultaneously if a node has enough computation resources.

Since original random number is along with key in “Gener-

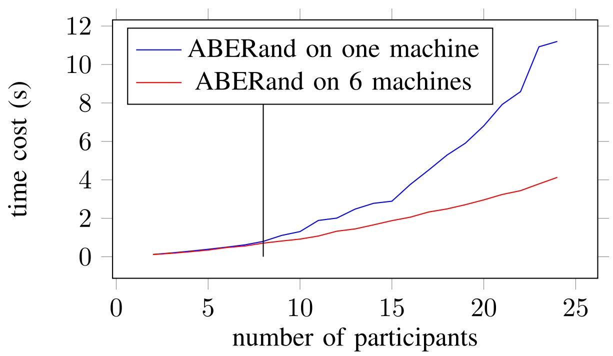

Fig. 8. Total cost of ABERand randomness

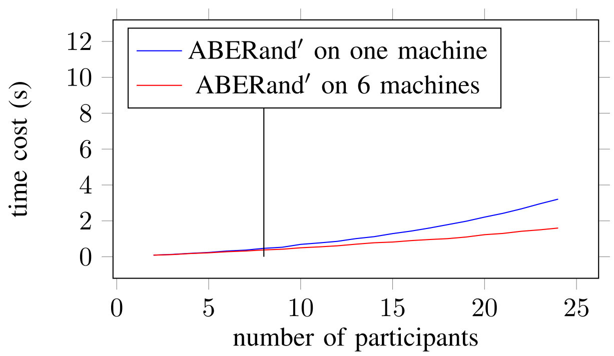

Fig. 10. Total cost of ABERand′ randomness

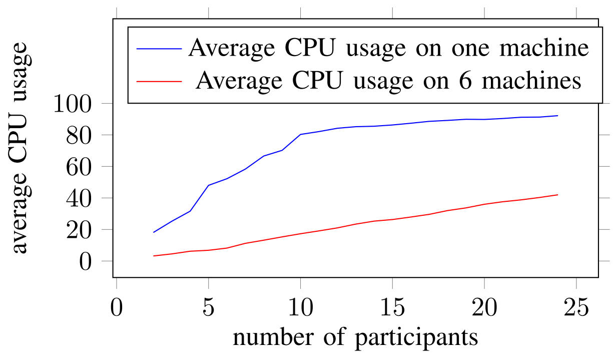

Fig. 9. CPU usage of ABERand randomness

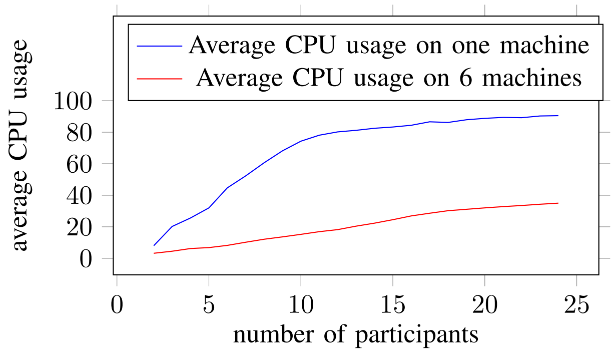

Fig. 11. CPU usage of ABERand′ randomness

ate and Broadcast Keys” phase, see more in V-D. An exciting optimization is that the original random number is directly used for the final randomness and decryption is only used for verification. Thus, the computation complexity is O(1), and the verification complexity is O(n). We call this protocol ABERand′. Figure 10 and figure 11 depict the performance of ABERand′, and the results show great improvement compared to figure 8 and figure 9.

For a participant, ABERand receives $n-1$ ciphertexts, $n-1$ ciphertext lists, and $n-1$ keys and performs n times of decryptions. In the broadcast environment, one node receives data from $n-1$ counterparts, so the total communication overhead is (n^2), and the one node computation overhead is $O(n)$. Decryptions are dense for a node, but could be calculated in parallel. With multiple nodes on one physical machine, there will not be much difference if pipelining methods are considered in our testing experiment.

When the group size is 100, a node in an epoch produces ∼8MB data which could be further compressed. To get the final beacon value, it takes ∼0.9s to encrypt, ∼1.1s to generate key and ∼0.3s × 100 to decrypt. To optimize, the decryptions could be done in parallel. Because of limited system resources, deviation exists when conducting experiments in a fully connected network. To test ABERand, we prepare 2 running environments, i.e., 2 to 24 nodes run on one physical machine and on 6 physical machines. All 8 cores of the CPU groups of a machine are used in the above two situations.

Figure 8 provides the results to compare the frequency of the distributed randomness in the 2 different environments. It is evident that when the number of participants is ≤ 8 (8 is the number of cores of a machine), the overhead of ABERand in the two situations is nearly the same. It also implies that performance is much better if one machine runs only one node. Figure 9 shows the average CPU usage of a machine in the 2 different environments.

C. *Comparison*

With Hash-Based. Hash-Based Method is a basic idea to implement the commit-and-reveal protocol to resist tampering. However, it has the “last actor” problem. When a participant receives all counterparts’ original randoms, he/she could compute the final beacon results. If the result is little of an advantage to him/her, he/she could abort the commit-and-reveal protocol, which is fatal to distributed randomness beacons. ABERand is a multiple-step protocol. However, it has no problem of the “last actor” problem because of threshold cryptography.

With PVSS-Based. One-shot PVSS is a commit-and-reveal protocol that produces a “manipulated” public verifiable randomness with communication complexity $O(n^2)$. PVSS-based projects [5], [16], [19] are mainly pragmatic rather than theoretical. In PVSS-Based methods, verification is needed and costly. In ABERand, verification is instantly accomplished when randomness is generated under the security of MA CP-ABE [34], since ciphertexts, access structures and keys

Protocol	Comm. model	Availability/ Faulty prop.	Comm. Complexity	Unpredi-ctability	Bias-resistant	Comp. complexity	Add/remove participant	Verification Comp.	Trusted dealer/DKG	Byzantine nodes
Algorand	syn.	1e-12	O(cn)	√	×	O(c)	?	O(1)	No	< n/2
Dfinity	syn.	1e-17	~O(cn)	√	√	O(c)	O(n^2)	O(1)	Yes	< n/2
Ouroboros	syn.	√	O(n^3)	√	√	O(n^3)	O(n^3)	O(n^3)	No	< n/2
Scrape	syn.	√	O(n^3)	√	√	O(n^2)	O(n^2)	O(n^2)	No	< n/2
RandHound	syn.	0.08%	~O(c^2 n)	√	√	~O(c^2 n)	O(n^2)	~O(c^2 n)	No	< n/3
RandHerd	syn.	0.08%	?	√	√	O(c^3 log n)	O(n^2)	O(c^3 log n)	Yes	< n/3
PoW	syn.	√	O(n)	√	×	very high	O(1)	O(1)	No	< n/2
Hydrand	syn.	√	O(n^2)	√	√	O(n^2)	O(n^2)	O(n^2)	No	< n/3
ABERand	asyn.	√	O(n^2)	√	√	O(n)	O(n)	O(1)	No	< n/2
ABERand′	asyn.	√	O(n^2)	√	√	O(1)	O(n)	O(n)	No	< n/2

Fig. 12. Detailed comparison based on Hydrand's [13]
The "Comm. Complexity" column gives the total network communication cost. The "Comp. complexity", "Add/remove participant" and "Verification Comp." columns show the computation cost in one node with one processor. The "Byzantine nodes" column gives the permitted maximum amount of malicious nodes.

are public to all at the end of each epoch. In addition to lower communication costs, ABERand is more scalable, bias-resistant and solves the "last actor" problem. The honest majority can always decrypt all ciphertexts in each epoch, even when malicious ones collude or hide their keys. So the DoS attack will not occur. RandHound[13] and RandHerd[13] use PVSS and other cryptographic primitives, such as BFT and Collective Signature[24] to support scalability. They lowered communication complexity to $O(c^2 \cdot n)$ and $O(c^2 \cdot logn)$ where c is the group size. However, it has the "self-DoS" problem, and it has a high faulty probability of 0.08%.

With BLS-Based. Dfinity [21] combines DKG and BLS to produce distributed randomness. DKG requires the communication complexity of at least $O(n^2)$. When f evil nodes exist, f "complaints" [33] will be broadcasted which will delay the protocol, and the worst communication complexity of running DKG is $O(n^2 \cdot f)$. For this reason, in our opinion, to lower the communication cost and improve efficiency, participants are divided into small groups by threshold relay protocol [21]. Groups in Dfinity[6] are fixed in each epoch, which brings the faulty probability of 10^{-17}. Parameters, such as group size, epoch interval, may further affect the availability when members leave since it will take long (e.g., one week) to update a group. DKG of Dfinity uses Joint-Feldman verifiable secret sharing [33] which was proved to be insecure [33] in some situations. The security of such a system (DKG + tBLS) needs to be more discussed. Compared to BLS-based protocol, ABERand has these advantages: It is a standalone randomness beacon independent of any blockchain or public bulletin board; Each epoch could generate randomness simultaneously; No DKG protocol is needed and a participator's public key and private key pair is once and for all; It only takes communication complexity of O(n) when adding/removing a participant, and there is little impact on others.

When considering adding/removing participants, the participants in PVSS-based or BLS-based methods have to re-run the PVSS or DKG protocol, both of which cost at least $O(n^2)$, while the participants in ABERand hardly needs to run a cryptography scheme. Figure 12 gives a detailed comparison of various distributed randomness beacon mentioned in section I.

VII. Security

Possible attacks on ABERand are analyzed below, which proves that ABERand is a secure and sound protocol.

A. Collusion Attack

To prevent collusion attacks, we prove that ABERand is secure in two aspects.

On the one hand, from the perspective of the MA CP-ABE scheme [34], the global identities are "tied" together with the various attributes that belong to a specific user so that they cannot be successfully combined with another user's attributes in the decryption. More specifically, the encryption algorithm blinds the message M with $e(g_1, g_1)^s$, where g_1 is a generator of the subgroup G_{p_1}, and s is a randomly chosen value in $\mathbb{Z}_N$. The value s is then split into shares λ_x according to the LSSS matrix, and the value 0 is split into shares ω_x. The decryptor must recover the blinding factor $e(g_1, g_1)^s$ by pairing their keys for the attribute and identity pairs (i, GID) with the ciphertext elements to obtain the shares of s. In doing so, the decryptor will introduce terms of the form $e(g_1, H(\text{GID}))^{\omega_x}$. If the decryptor has a satisfying set of keys with the same identity GID, these additional terms will be canceled from the final result since the ω_xs are shares of 0. If two users with different identities GID and GID′ attempt to collude and combine their keys, then there will be some terms of the form $e(g_1, H(\text{GID}))^{\omega_x}$ and some terms of the form $e(g_1, H(\text{GID}'))^{\omega_x{}'}$, and these will not cancel with each other, thereby preventing the recovery of $e(g_1, g_1)^s$.

On the other hand, according to the ABERand protocol specification, the number of malicious parties $f < n/2$.

Therefore, in the "worst case", these malicious parties collude with each other in advance to decrypt the ciphertexts. Honest parties abide by the protocol and set the threshold t, which is higher than $n/2$. Since the threshold $t > f$, f malicious parties fail to decrypt ciphertexts even if they share their keys in advance, thus preventing collusion when producing distributed randomness.

B. Attacks on ABERand

ABERand has five stages in general when considering the interaction among nodes. Participants join or leave the group at the 1st stage; Participants broadcast ciphertext at the 2nd stage; Participants broadcast the ciphertext list at the 3rd stage; Participants broadcast keys at the 4th stage; Participants compute the final randomness at the 5th stage. There is a timeout Δt for each stage.

Join/Leave Attack: When a participant joins, it will participate in the next epoch with no impact on others; If a participant leaves at the 1st stage, it contributes none to the final beacon randomness; If it leaves at later stages, its randomness will be obtained through the decryption by the honest ones. It neither bias the result to any one's favor nor aborts the whole process.

Key Attack: In the 4th stage, since participant θ sends a ciphertext whose original message is "k" and whose access structure is "1-of-1 k@node$_\theta$". Others could check out whether the key is valid through decrypting the ciphertext. If the key is failed to decrypt the associated ciphertext, it will not be used in the decryption step. With an honest majority, the whole process will succeed.

Message(Ciphertext/Ciphertext List) Attack: In all stages, ciphertexts are indistinguishable based on MA CP-ABE. Any message can be verified to be un-tampered with public key and signature in the message; Honest ones are connected so that no malicious one could cheat them by broadcasting different ciphertexts or ciphertext lists; Any malicious participant could send a forged ciphertext/ciphertext list, thus leading to failure of decryption at the 4th stage. It is a situation the same as Join/Leave Attack.

Public Key Attack: At the first stage, a malicious participant broadcasts a fake public key which is similar to the situation that a participant loses his/her private key. Without a private key, the participant will not produce a key in the 4th stage. It is a situation the same as Join/Leave Attack.

Randomness Attack: Malicious participants may collude to decrypt the honest parties' ciphertext in advance, however, they will fail because f is less than $n/2$. Moreover, they cannot stop honest ones from obtaining randomness in each epoch. Since all the broadcasted information is public, anyone outside the system can verify the result instantly.

DoS Attack: Malicious participants at any stage, could deny issuing any information. If they initiate DoS attack at the 1st and 2nd stage, it is a situation the same as Join/Leave Attack; If they attack at later stages, it is a situation the same as Message Attack or Key Attack.

Sybil Attack: With "nodeId" as each node's public unique identity, all nodes are registered and recognized in our protocol. No adversary could create a majority of colluding nodes ($f >= n/2$) in our setting to start a sybil attack [47].

VIII. Concluding remarks

In this paper, we adapt Rouselakis's multi-authority CP-ABE to the decentralized (t, n)-threshold multi-authority CP-ABE approach, in which t-out-of-n participants generate keys for decrypting ciphertexts, and propose a scalable distributed randomness beacon, named ABERand. Hash-based protocol and traditional public-key based protocol will suffer the "last actor" problem which leads to bias or breaks liveness of a beacon. ABERand addresses the problem, primarily because an honest majority of authorities will publicize their generated keys at a later time according to the protocol. Other protocols either introduce bias or have higher computation and communication complexity for the beacon. While (t, n)-threshold multi-authority CP-ABE in our protocol is decentralized and highly efficient to construct a distributed randomness beacon.

Our main technique is to ensure that ciphertexts can only be decrypted in the future, which prevents malicious participants from obtaining information in advance. Meanwhile, malicious nodes could not abort or bias the protocol with threshold cryptography. Therefore, it is a viable idea to construct a new public-key cryptography scheme in which the public key is generated before the private key in a distributed system. To lower the communication cost and to improve the performance in a real production environment, sharding [48] is a good idea to be adopted for future work.

Notes

[1]http://en.wikipedia.org/wiki/1980_Pennsylvania_Lottery_scandal
[2]https://www.nist.gov/programs-projects/nist-randomness-beacon
[3]https://en.wikipedia.org/wiki/Lavarand
[4]https://en.wikipedia.org/wiki/Commitment_scheme
[5]https://en.wikipedia.org/wiki/Boneh-Lynn-Shacham
[6]https://dfinity.org/pdf-viewer/pdfs/viewer?file=../library/threshold-relay-blockchain-stanford.pdf

References

[1] M. Blum. Coin flipping by telephone. In Allen Gersho, editor, CRYPTO '81, volume ECE Report 82-04, pages 11–15. U.C. Santa Barbara, Dept. of Elec. and Computer Eng., 1981.
[2] M. O. Rabin. Transaction protection by beacons. J. Comput. Syst. Sci., 27(2):256–267, 1983.
[3] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
[4] Garay J., Kiayias A., Leonardos N. (2015) The Bitcoin Backbone Protocol: Analysis and Applications. In: Oswald E., Fischlin M. (eds) Advances in Cryptology - EUROCRYPT 2015. EUROCRYPT 2015. Lecture Notes in Computer Science, vol 9057. Springer, Berlin, Heidelberg.
[5] Kiayias, A., Konstantinou, I., Russell, A., David, B.M., Oliynykov, R. (2016). A Provably Secure Proof-of-Stake Blockchain Protocol. IACR Cryptology ePrint Archive, 2016, 889.
[6] S. Azouvi, P. McCorry, and S. Meiklejohn. Winning the caucus race: Continuous leader election via public randomness. arXiv preprint arXiv:1801.07965, 2018.
[7] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. white paper, 2014.
[8] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In 2016 IEEE Symposium on Security and Privacy, pages 839–858. IEEE Computer Society Press, May 2016.

[9] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. Cryptology ePrint Archive, Report 2015/1015, 2015. http://eprint.iacr.org/2015/1015.

[10] M. Milutinovic, W. He, H. Wu, M. Kanwal, "Proof of luck: An efficient blockchain consensus protocol", Proc. 1st Workshop Syst. Softw. Trusted Execution (SysTEX), pp. 1-6, 2016.

[11] King, S., Nadal, S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 2012

[12] Albert Kwon, David Lu and Srinivas Devadas. XRD: Scalable Messaging System with Cryptographic Privacy. Cryptology ePrint Archive, 2019. http://arxiv.org/abs/1901.04368.

[13] Schindler, Philipp, Aljosha Judmayer, Nicholas Stifter and Edgar R. Weippl. HydRand : Efficient Continuous Distributed Randomness. (2019).

[14] Thomas Baignères and Cécile Delerablée and Matthieu Finiasz and Louis Goubin and Tancrède Lepoint and Matthieu Rivain. Trap Me If You Can – Million Dollar Curve. (2015)

[15] I. Bentov, A. Gabizon, and D. Zuckerman. Bitcoin beacon. https://arxiv.org/pdf/1605.04559v2, 2016. Accessed: 2016-06-06.

[16] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford. Scalable Bias-Resistant Distributed Randomness. In Security and Privacy (SP), 2017 IEEE Symposium on, pages 444–460. IEEE, 2017. Accessed: 2017-08-20.

[17] A. K. Lenstra and B. Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366, 2015.

[18] B. Bunz, S. Goldfeder, and J. Bonneau. Proofs-of-delay and randomness beacons in Ethereum. In S&B '17: Proceedings of the 1st IEEE Security & Privacy on the Blockchain Workshop, April 2017. Accessed: 2017-08-21.

[19] I. Cascudo and B. David. Scrape: Scalable randomness attested by public entities. http://eprint.iacr.org/2017/216, 2017. Accessed: 2017-03-24.

[20] Yossi Gilad and Rotem Hemo and Silvio Micali and Georgios Vlachos and Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. (2017)

[21] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY Technology Overview Series, Consensus System. CoRR, Vol. abs/1805.04548 (2018). arxiv: 1805.04548 http://arxiv.org/abs/1805.04548

[22] Jeremy Clark and Urs Hengartner. On the use of financial data as a random beacon. Proceedings of the 2010 international conference on Electronic voting technology/workshop on trustworthy elections, p.1-8, August 09-10, 2010, Washington, DC

[23] S. Bag, S. Ruj and K. Sakurai, "Bitcoin Block Withholding Attack: Analysis and Mitigation," in IEEE Transactions on Information Forensics and Security, vol. 12, no. 8, pp. 1967-1978, Aug. 2017. doi: 10.1109/TIFS.2016.2623588

[24] C. Pierrot and B. Wesolowski (2016). Malleability of the blockchain's entropy. Cryptography and Communications, 10, 211-233.

[25] S. Popov (2016). On a decentralized trustless pseudo-random number generation algorithm. IACR Cryptology ePrint Archive, 2016, 228.

[26] Dan Boneh, Joseph Bonneau, Benedikt Bunz, and Ben Fisch. Verifiable delay functions.In CRYPTO 2018, 2018.

[27] Schoenmakers B. (1999) A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting. In: Wiener M. (eds) Advances in Cryptology — CRYPTO' 99. CRYPTO 1999. Lecture Notes in Computer Science, vol 1666. Springer, Berlin, Heidelberg

[28] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In 37th IEEE Symposium on Security and Privacy, May 2016.

[29] S. Micali. (2017). Byzantine Agreement , Made Trivial.

[30] Micali, Silvio, Michael O. Rabin and Salil P. Vadhan. "Verifiable Random Functions." FOCS (1999).

[31] Boneh D., Lynn B., Shacham H. (2001) Short Signatures from the Weil Pairing. In: Boyd C. (eds) Advances in Cryptology — ASIACRYPT 2001. ASIACRYPT 2001. Lecture Notes in Computer Science, vol 2248. Springer, Berlin, Heidelberg

[32] Pedersen T.P. (1991) A Threshold Cryptosystem without a Trusted Party. In: Davies D.W. (eds) Advances in Cryptology — EUROCRYPT '91. EUROCRYPT 1991. Lecture Notes in Computer Science, vol 547. Springer, Berlin, Heidelberg

[33] Gennaro R., Jarecki S., Krawczyk H., Rabin T. (1999) Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In: Stern J. (eds) Advances in Cryptology — EUROCRYPT '99. EUROCRYPT 1999. Lecture Notes in Computer Science, vol 1592. Springer, Berlin, Heidelberg

[34] Rouselakis Yannis, Waters Brent. (2015). Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption. IACR Cryptology ePrint Archive. 2015. 315-332. 10.1007978-3-662-47854-7_19.

[35] H. Lin, Z. Cao, X. Liang, and J. Shao. Secure threshold multi authority attribute based encryption without a central authority. In INDOCRYPT, pages 426–436, 2008.

[36] A. Beimel. Secure schemes for secret sharing and key distribution. Ph.D. dissertation, Israel Institute of Technology, Technion, Haifa, Israel, 1996.

[37] Liu, Z. J. et al. Efficient Generation of Linear Secret Sharing Scheme Matrices from Threshold Access Trees. (2014).

[38] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. in Proc. 13th ACM Conf. Comput. Commun. Security, 2006, pp. 89–98.

[39] N. Attrapadung, B. Libert, and E. Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. in Proc. 14th Int. Conf. Practice Theory Public Key Cryptography, 2011, pp. 90–108.

[40] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. in Proc. IEEE Symp. Security Privacy, 2007, pp. 321–334.

[41] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. in Proc. 14th Int. Conf. Practice Theory Public Key Cryptography, 2011, pp. 53–70.

[42] V. Goyal, A. Jain, O. Pandey, and A. Sahai. Bounded ciphertext policy attribute based encryption. in Proc. 35th Int. Colloquium Automata, Lang. Programm., 2008, pp. 579–591.

[43] R. Bobba, H. Khurana, and M. Prabhakaran. Attribute-sets: A practically motivated enhancement to attribute-based encryption. in Proc. 14th Eur. Symp. Res. Comput. Security, 2009, pp. 587–604.

[44] Li W, Xue K, Xue Y, Hong J. (2015). TMACS: a robust and verifiable threshold multi-authority access control system in public cloud storage. IEEE Trans Inf Forensics Secur 10(1):55–68

[45] Lewko A, Waters B (2011). Decentralizing attribute-based encryption. In: Advances in cryptology–EUROCRYPT 2011, Springer, NewYork, pp 568–588

[46] Joseph A. Akinyele, Matthew Green, and Avi Rubin. Charm: A framework for rapidly prototyping cryptosystems. Cryptology ePrint Archive, Report 2011/617, 2011. http://eprint.iacr.org/.

[47] J. R. Douceur. The Sybil attack. In 1st International Workshop on Peer-to-Peer Systems (IPTPS), Mar. 2002.

[48] Loi Luu , Viswesh Narayanan , Chaodong Zheng , Kunal Baweja , Seth Gilbert , Prateek Saxena, A Secure Sharding Protocol For Open Blockchains, Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, October 24-28, 2016, Vienna, Austria [doi: 10.1145/2976749.2978389]