1-Round Distributed Key Generation with Efficient Reconstruction Using Decentralized CP-ABE

Liang Zhang, Feiyang Qiu, Feng Hao, and Haibin Kan*

***Abstract*—Distributed key generation (DKG) is widely used in multi-party computation and decentralized applications. DKG has two phases, namely sharing and reconstruction. Most of the prior DKG protocols need at least 2 rounds for the sharing phase, in case some party raises a dispute. The existing 1-round DKG protocol [Fouque et al., PKC'01], built based on a publicly verifiable secret sharing (PVSS) scheme, assumes a static adversary model and its reconstruction phase requires $O(n^2)$ communication complexity. Motivated by the observation that a ciphertext-policy attribute-based encryption (CP-ABE) scheme hides secret sharing (SS) in ciphertext, we utilize decentralized CP-ABE to achieve the first adaptively secure 1-round DKG protocol. Firstly, a CP-ABE scheme enables the ciphertexts in DKG to be externally decrypted, making our protocol superior to the PVSS-based DKG protocol in reconstruction. The communication and computation complexities are both lowered to $O(n)$ thanks to the constant-sized decryption key and the proposed batch decryption. The use of CP-ABE also makes our DKG protocol storage-friendly, i.e., the parties store no ciphertext after the sharing phase. Secondly, we add non-interactive zero-knowledge (NIZK) proofs to make the CP-ABE ciphertext publicly verifiable by leveraging the sigma protocol and the Fiat-Shamir heuristic. Thirdly, we demonstrate our protocol's feasibility by presenting a proof-of-concept implementation over Ethereum, which is used as a public channel and a trustworthy computation platform. The implementation is a non-trivial task due to Ethereum's incompatibility with the bilinear mapping group.**

***Index Terms*—distributed key generation, ethereum, smart contract, attribute-based encryption, zero-knowledge proofs, sigma protocol**

Manuscript received April 19, 2021; revised December 19, 2021; accepted January 30, 2022. This work was supported by National Key R & D Program of China (No. 2019YFB2101703), National Natural Science Foundation of China (Grant No. U19A2066), and the Innovation Plan of Shanghai Science and Technology (Nos. 20511102200 and 20222420800). The third author is supported by Royal Society (ICA\R1\180226) and EPSRC (EP/T014784/1).

Liang Zhang and Haibin Kan are with Shanghai Key Laboratory of Intelligent Information Processing, School of Computer Science, Fudan University, Shanghai 200433, China, and with Shanghai Engineering Research Center of Blockchain, Shanghai 200433, China. Haibin Kan is also with Zhuhai Fudan Innovation Research Institute Zhuhai City 518057, China. E-mail: liangzhang19@fudan.edu.cn; hbkan@fudan.edu.cn.

Feiyang Qiu is with the Department of Electrical Engineering, Katholieke Universiteit Leuven, 3001 Leuven, Belgium. E-mail: fqiu@esat.kuleuven.be.

Feng Hao is with the Department of Computer Science, University of Warwick, Coventry, CV4 7AL, U.K. E-mail: feng.hao@warwick.ac.uk.

*Corresponding author: Haibin Kan

I. Introduction

Distributed key generation (DKG) is a cryptographic protocol in which n distributed parties cooperate to generate a common key pair and each of them holds a sub-key pair. Only at least a threshold $t(\leq n)$ of them can calculate the private key, while any subset of fewer than t parties learn nothing about it. DKG is the basis of threshold cryptosystems [1]–[3], which are widely used in distributed protocols, such as identity-based encryption schemes [4], threshold signatures [3] and randomness beacons [5]. Due to its crucial role in cryptography, DKG captures a lot of attention [6]–[12]. Typically, a DKG protocol consists of two phases: *sharing* phase for the (master) public key generation and *reconstruction* phase for the (master) private key calculation. To achieve agreement in the *sharing* phase and enable threshold recovery in the *reconstruction* phase, verifiable secret sharing (VSS) [16] and publicly verifiable secret sharing (PVSS) [19]–[21] are widely used to implement DKG protocols.

VSS enables parties to verify received shares. Usually, this property is achieved by attaching a non-interactive zero-knowledge proof (NIZK) to each share. A line of work [6]–[12] uses VSS to construct DKG. Since the shares are transferred in pairwise private channels, a dealer may dispense incorrect shares to start DoS attacks in the *sharing* phase. If so, honest parties need to handle *disputes* in additional communication rounds.

PVSS incorporates an additional public key encryption (PKE) scheme for VSS, enabling shares to be encrypted and publicly verified. By replacing VSS with PVSS, a DKG protocol can be built on public channels. Fouque et al. [13] propose the first 1-round(-sharing) DKG protocol based on PVSS, which uses the Paillier cryptosystem [25] as the underlying PKE. Their protocol is not *adaptively* secure [26] in both *sharing* and *reconstruction* phase. Furthermore, PVSS schemes suffer from the problem that an encrypted share can only be decrypted by the corresponding shareholder. Consequently, when an external user starts the *reconstruction* phase, which is called *delegatable reconstruction*, PVSS-based DKG protocols all have heavy computation and communication overloads.

In this paper, we propose the first 1-round DKG protocol, achieving adaptive security, public verifiability and efficient delegatable reconstruction at the same time. We investigate applying ciphertext-policy attribute-based encryption (CP-ABE) [29] as the primitive to construct DKG for the first time. We note that both PVSS and

CP-ABE (with NIZK proofs) enable a party to hide a secret among multiple parties with public verification and threshold recovery of the secret. However, CP-ABE has the property of external decryption (i.e., an external user can decrypt the ciphertexts), making it particularly suitable to achieve a DKG with an efficient delegatable *reconstruction* phase. As a result, our protocol has competitive advantages over the existing PVSS-based 1-round DKG protocol [13]. An external user only requires $O(n)$ for both communication and computation complexities in the *reconstruction* phase. Moreover, our protocol is storage-friendly, meaning that no party needs to store ciphertexts and each party costs only $O(1)$ computation complexity during *reconstruction*. Although the *sharing* phase has a verification complexity of $O(n^2)$, it is the same as the PVSS-based protocol [13]. CP-ABE is a versatile primitive and supports generic access control in its ciphertext. We only consider threshold-based access control in our usage. We adopt Rouselakis and Waters's decentralized CP-ABE scheme [30] in the construction of our protocol.

To acquire a concrete public channel, we take advantage of Ethereum [17]. We stress that the public verification is accomplished non-trivially within Ethereum, as will be explained in Section VIII-B.

Our contributions are fourfold.

1) We add NIZK proofs to prove plaintext knowledge of decentralized CP-ABE ciphertext by leveraging the sigma protocol and the Fiat-Shamir heuristic (in Section VI) for the first time.
2) With the decentralized CP-ABE and the NIZK proofs, we implement a publicly verifiable DKG protocol. Moreover, the protocol assumes *adaptive* rushing adversaries and each DKG instance requires only one round in the *sharing* phase.
3) In the *reconstruction* phase, we achieve high efficiency. We lower the computation complexity to $O(n)$ using batch decryption (in Section VII-E). Each decryption key in the CP-ABE is of size $O(1)$, leading to only $O(n)$ communication complexity.
4) To cater for Ethereum's unique cryptographic computation model and achieve better efficiency and compatibility, we convert bilinear mapping group ($\mathbb{G}_T$) elements comparison into calculation on group $\mathbb{G}_1$ (using Equations (12) and (13) in Section VIII-B). Further, to lower the gas cost, we convert expensive computation on $\mathbb{G}_2$ to operations on $\mathbb{G}_1$ and native-supported pairing check (using Equations (14) and (15) in Section VIII-B).

II. Related Work

Different researches take advantage of different cryptographic primitives to implement DKG protocols. To distinguish previous works, we introduce them according to the primitives they use.

VSS is the first primitive to be considered to implement DKG protocols. The basic idea is that each party in a group runs a VSS protocol so that a threshold amount of them are able to recover all the secrets, even if there are malicious or crashed ones. The secrets are summed to construct the distributed secret key. In 1991, Pedersen [6] was the first to propose a threshold cryptosystem that presents a DKG prototype (Joint-Feldman DKG). It contributes significantly to threshold cryptography in the following decades. Gennaro et al. [7] point out that Pedersen's protocol is prone to a special rushing attack, resulting in a bias in the created DKG public key over the keyspace. The attack works as follows: two malicious parties bias the distribution of the last bit of the DKG public key with probability 3/4 rather than 1/2, through bringing ambiguity to qualified parties set $\mathcal{Q}$ [7], [8], [13]. Then, Gennaro et al. propose to prevent this attack by separating the *sharing* phase into two parts: the honest parties define qualified parties $\mathcal{Q}$ in the first part; and they calculate the DKG public key in the second part. Both of Pedersen's and Gennaro et al. DKG protocols are on top of Feldman's VSS protocol [16]. Tomescu et al. [18] present some techniques to scale distributed systems that utilize secret sharing schemes. They improve VSS-based DKG protocols using authenticated multipoint evaluation trees (AMTs) [18] which speed up proving polynomial evaluations. However, it still has high overhead in the worst-case situation that dishonest behaviors exist. Kate and Goldberg [10] propose the first DKG protocol in a semi-synchronous communication model (as Kokoris-Kogias et al. [12] claim). Later, Kate et al. [11] implement their scheme and test on 70 nodes distributed over different countries. It's a trade-off, however. In a synchronous system, such as protocols [6], [13], it tolerates up to $n/2$ malicious colluding adversaries. Kate et al. [10], [11] have to assume $n > 3f + 2x$ parties (n is the total number of parties i.e., $|\mathcal{P}|$, f is the number of parties controlled by the adversary, x is the number of the crashed ones) and have to implement a byzantine agreement protocol along with DKG. Built on asynchronous verifiable secret sharing (AVSS), Kokoris-Kogias et al. [12] propose the first asynchronous distributed key generation (ADKG) protocol, which needs $O(f)$ rounds. They use ADKG to construct a validated asynchronous byzantine agreement (VABA) protocol tolerating $f < n/3$ adversaries.

PVSS or encrypting secret shares is a solution to enhance integrity and privacy in constructing DKG protocols. Fouque et al. [13] propose the first 1-round DKG protocol based on PVSS in 2001. They assume a synchronous network model to resist another rushing attack[1] in the *sharing* phase. However, studies [22], [23] have shown that rushing attacks can also exist in a synchronous network. Each party in their protocol initiates a Paillier cryptosystem to encrypt its secret shares. They add NIZK proofs to Paillier cryptosystem so that the encrypted shares can be verified publicly. The protocol assumes a *static* adversary model where adversaries are fixed in *sharing* and *reconstruction* phases. Neji et al. [8] extend Joint-

[1] A malicious party waits to define its own value until all other parties have published their values.

TABLE I: Comparison of DKG protocols

Protocol	Primitives	Network model	Adversary model	Comm. channel	Public verif.	Verif.	1-Round	Party Comp.	Party Comm.	External user Comp.	External user Comm.	Byzantine nodes
Gennaro et al. [7]	VSS	semi-sync.	static	private	✗	-	✗	$O(1)$	$O(n)$	$O(n)$	$O(n^2)$	$f < n/2$
Neji et al. [8]	PVSS‡ (VSS+$s \cdot h^s$)	sync.	static	public	✗	-	✗	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$f < n/2$
Schindler et al. [9]	VSS+DH exchange	sync.	adaptive	public	✗	-	✗	$O(n)$	$O(n)$	$O(n^2)$	$O(n^2)$	$f < n/2$
Kate et al. [10], [11]	VSS	semi-sync.	static	private	✗	-	✗	$O(1)$	$O(n)$	$O(n)$	$O(n^2)$	$f < (n-2x)/3$
Kokoris-Kogias et al. [12]	AVSS	async.	adaptive	private	✗	-	✗	$O(1)$	$O(n)$	$O(n^2)$	$O(n^2)$	$f < n/3$
Fouque et al. [13]	PVSS (VSS+Paillier)	sync.	static	public	✓	$O(n^2)$	✓	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$f < n/2$
Our work	CP-ABE	sync.	adaptive	public	✓	$O(n^2)$	✓	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$f < n/2$

The **Primitives** column shows the cryptographic primitives (and how to encrypt secrets or shares) a protocol uses. In PVSS‡, the authors claim to have utilized a PVSS protocol. However, the encrypted shares are not publicly verified, violating the concept of a PVSS protocol.
The **Comm. channel** column indicates whether shares are sent through private or public channels. In case of private channels, the secrecy of the data needs to be protected, which requires pre-existing secure channels. In case of public channels, the data can be sent in the clear since by design they do not reveal any secret information. Public bulletin [14], [21], [22], on which data is assumed to be consistent and publicly available, can be used as a public channel. (Blockchain is regarded as an implementation of a public bulletin with the functionality of decentralized smart contract system [17], [22], [24]. Particularly, Ethereum is one of the most representative blockchains and it is used in our implementation.) If a public channel is used in a protocol, we assume parties of the protocol store ciphertexts (or encrypted shares) on the underlying public channel.
The **Public verif.** column represents whether shares are publicly verifiable, following by **Verif.** column which gives the verification complexity of all the encrypted shares. The use of a public channel does not guarantee public verification. For example, Neji et al. [8] and Schindler et al. [9] use a public channel to handle disputes, rather than to verify the encrypted shares.
The **Party Comp.|Comm.** column demonstrates the computation complexity and the communication cost of a party ($\in \mathcal{P}$), to obtain correct original shares and recover MSK, in the *reconstruction* phase. In our protocol, each decryption key is of size $O(1)$ and the key generation algorithm costs $O(1)$ computation complexity. The external user can decrypt aggregated ciphertext with the decryption keys directly. As a comparison, each party in protocols [8], [9], [13] needs to fetch $O(n)$ encrypted shares from the public channel, since the encrypted shares can only be decrypted by herself (costing $O(n)$). Hence, these protocols require an additional communication round, $O(n)$ computation complexity and $O(n)$ communication complexity for each party. In all previous protocols [7]–[13], each party have to send out $O(n)$-sized shares in *reconstruction*.
The **External user Comp.|Comm.** column gives the computation complexity and communication complexity of the external user, who starts the *delegatable reconstruction* phase. In our protocol, the external user incurs $O(n)$ both for communication and computation complexity, because of $O(n)$-sized aggregated ciphertext, n pieces of $O(1)$-sized decryption keys and batch decryption. However, in previous protocols [7]–[13], the external user requests to each party for original shares and verifies them. Thus, these protocols all cost at least $O(n)$ computation complexity and $O(n^2)$ communication messages (i.e., shares) for the external user.
The **Byzantine nodes** column depicts the tolerant number of byzantine parties ($\subseteq \mathcal{P}$) of a DKG protocol. Whether the faulty parties are adaptive is demonstrated in **Adversary model** column.

Feldman DKG with a public communication channel to solve disputes. They claim to utilize PVSS protocol as the primitive. However, they encrypt the shares of a secret like this: $s_{i,j} \cdot h^{s_{i,j}}$, where $s_{i,j}$ is the share from party i to party j and h is a generator of prime order group. Schindler et al. inherit from Neji et al. to handle complaints publicly on Ethereum. They utilize Etheruem as a public communication channel and use smart contracts to verify the complaints. In their protocol, the shares are encrypted using symmetric key encryption. The symmetric key is calculated by the dealer with her private key and each receiver's public key, which is actually a Diffie-Hellman (DH) key exchange protocol. Unfortunately, both Neji et al. [8] and Schindler et al. [9] protocols just provide a publicly verifiable complaint management and do not eliminate the *dispute* round, because the encrypted shares are not publicly verifiable. Parties in both protocols need to verify the shares' validity after decrypting the encrypted shares. If there are malicious parties, honest parties will complain with NIZK proofs in the additional *dispute* round.

Table I demonstrates the comparison of different DKG protocols. Further explanation about the *reconstruction* complexity is given in this paragraph to highlight our contribution. In VSS+encryption [9] or PVSS [8], [13] based protocols, when a user starts the *reconstruction* phase, each party needs to fetch $O(n)$ encrypted shares from the public channel and decrypt them. Next, each party sends the decrypted shares to the user, leading to communication complexity of $O(n^2)$ for the user. Then the user checks the validity of decrypted shares. Fouque et al. [13] cost $O(n)$ exponentiations. Neji et al. [8] cost $O(n)$ exponentiations using membership proof, which is interactive. Protocols [8], [13] achieve $O(n)$ computation complexity due to *static* adversaries, i.e., the qualified parties of *sharing* phase are assumed to behave honestly in *reconstruction* phase. Schindler et al. [9] cost $O(n^2)$ discrete logarithm equality (DLEQ) [15] and Kokoris-Kogias et al. [12] cost $O(n^2)$ exponentiations, as both of them assume *adaptive* adversaries. In the *reconstruction* phase of our protocol, each decryption key size is $O(1)$ from each party and aggregated ciphertext (in Section VII-E) is $O(n)$ from the smart contract, thus the communication complexity is $O(n)$. Each of the decryption key is validated in constant time (in Lemma 2). The user invokes batch decryption with t valid decryption keys, which takes $O(n)$ pairings. Therefore, the computation complexity is $O(n)$.

III. Preliminaries

A. Bilinear Maps

$\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are three groups of prime order q. Let g_1, g_2 be a generator of $\mathbb{G}_1$ and $\mathbb{G}_2$ repectively and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map with following properties:

1) Bilinearity: for all $\mu \in \mathbb{G}_1$, $\upsilon \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, we have $e(\mu^a, \upsilon^b) = e(\mu, \upsilon)^{ab}$.
2) Nondegeneracy: $e(g_1, g_2) \neq 1$.

We call $\mathbb{G}_T$ a bilinear mapping group. Usually, we additionally require e to be efficiently computable.

B. ACP, LSSS and Decentralized CP-ABE

In CP-ABE schemes [29], [30], access structure or access control policy (ACP) [27] is used to enable only qualified users to decrypt a ciphertext. ACP is usually expressed in linear secret sharing scheme (LSSS) [29] when encrypting.

Definition 1 (Access Structure)**.** *Let Ω be an attribute universe. A collection $\mathbb{A} \subseteq 2^{\Omega}$ is monotone if for any B and C: $B \cap \Omega \in \mathbb{A}$ and $B \subseteq C$, then $C \cap \Omega \in \mathbb{A}$. An access structure is a monotone collection $\mathbb{A}$ of nonempty subsets of Ω, i.e., $\mathbb{A} \subseteq 2^{\Omega} \setminus \{\varnothing\}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets.*

Definition 2 (LSSS)**.** *Let p be a prime and Ω be an attribute universe. A linear secret sharing scheme (LSSS) with a domain of secrets $\mathbb{Z}_p$ realizing access structure 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 share-generating matrix $A \in \mathbb{Z}_p^{l \times n}$, 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 $\upsilon = (z, r_2, ..., r_n)^{\perp}$, where $r_2, ..., r_n \xleftarrow{R} \mathbb{Z}_p$. Then, the vector of l shares of the secret z according to the secret sharing scheme is equal to $\lambda = A\upsilon \in \mathbb{Z}_p^{l \times 1}$. The share λ_j with $j \in$ [l] "belongs" to attribute $\sigma(j)$.

(A, σ) is referred as an access control policy (ACP) of the access structure $\mathbb{A}$.

Definition 3 (Decentralized CP-ABE)**.** *[31] A decentralized ciphertext-policy attribute-based encryption (CP-ABE) protocol has multiple attribute authorities. It is a tuple of algorithms defined as follows:*

- $\mathsf{GP} \leftarrow$ *GlobalSetup*(Λ). *it takes in the security parameter Λ and outputs global parameters* GP.
- $(\mathsf{sk}_\theta, \mathsf{pk}_\theta) \leftarrow$ *AuthSetup*(GP, θ). *Each authority P_θ takes* GP *as input to produce its long-term secret and public key pair* $(\mathsf{sk}_\theta, \mathsf{pk}_\theta)$.
- $\mathsf{Key}_{\mathsf{GID},u} \leftarrow$ *KeyGen*$(\mathsf{GID}, \mathsf{GP}, u, \mathsf{sk}_\theta)$. *The algorithm takes in an identity* GID*, the global parameters, an attribute u belonging to the authority P_θ, and the secret key* sk_θ *for this authority. It produces a key* $\mathsf{Key}_{\mathsf{GID},u}$ *for this identity and attribute pair* (GID, u).
- $C \leftarrow$ *Encrypt*$(M, (\mathrm{A}, \sigma), \mathsf{GP}, \{\mathsf{pk}_\theta\})$. *The algorithm takes in a message M, an access control policy (A, σ), a set of public keys, and the global parameters. It outputs a ciphertext C.*
- $M \leftarrow$ *Decrypt*$(C, \mathsf{GP}, \{\mathsf{Key}_{\mathsf{GID},u}\})$. *The algorithm takes in the global parameters, a ciphertext, and a set of keys regarding an identity* GID*. Only if attributes set $\{u\}$ of the keys satisfies the access control policy of the ciphertext, it outputs the message M.*

C. Sigma Protocol and Fiat-Shamir Heuristic

Definition 4 (Sigma protocol)**.** *[35] Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ be a binary relation. A Sigma protocol for $\mathcal{R}$ is a protocol between a prover algorithm* P *and a verifier algorithm* V. P *takes as input a statement $(x, y) \in \mathcal{R}$.* V *only inputs $y \in \mathcal{Y}$ and outputs* accept *or* reject. P *and* V *work as follows:*

1) P *computes a commitment b to* V.
2) V *sends challenge c (a random value) to* P.
3) P *computes and sends a response w to* V*, regarding c.*
4) At last, $\mathrm{V}(y, b, c, w)$ *outputs either* accept *or* reject.

The sequential steps are called a conversation (b, c, w). A Sigma protocol has following properties:

- *Correctness*: The conversation (b, c, w) is always accepted by an honest verifier V.
- *Special Knowledge Soundness*: Given input a statement $y \in \mathcal{Y}$, with two accepting conversations (b, c, w) and (b, c', w') for y, where c $\neq c'$, one can efficiently compute x $\in \mathcal{X}$ such that $(x, y) \in \mathcal{R}$.
- *Special honest verifier zero knowledge (HVZK)*: There exists a polynomial-time *simulator*, which inputs y and c, outputs an accepting conversation (b, c, w) with the same probability distribution as conversations between the honest P and V on input y.

Throughout, we use the notion $\mathsf{PoK}\{(x_1, x_2, \ldots) :$ statements of $x_1, x_2, \ldots\}$ to denote the proofs of knowledge of secrets $x_1, x_2, \ldots$, such that the statements hold.

Definition 5 (Fiat-Shamir Heuristic)**.** *[35] Assume the conversation (b, c, w) be an interactive sigma protocol, and H be a hash function. Define the Fiat-Shamir (FS) non-interactive proof system (Gen,Check) as follows:*

1) P obtains commitment b. Gen calculates the challenge $c = H(y, b)$; and computes w as in sigma protocol. Then P sends (b, c, w) to V.
2) Upon receiving (b, c, w), Check tests the conversation for y, then outputs Yes *or* No.

If the hash function H is modeled as a random oracle, then a sigma protocol with Fiat-Shamir heuristic constructs NIZK proof for the relationship between x and y [36].

D. Blockchain and Ethereum

Blockchain serves as a concrete implementation of the public bulletin or public channel with smart contract execution engine [22]. It reaches consistency on transactions or blocks under its consensus algorithm among distributed nodes scattered over the world. Ethereum is a permissionless blockchain, enabling any programmers to develop decentralized applications. Ethereum allows users to store arbitrary authenticated data, which is tamper-resistant. Ethereum virtual machine (EVM) provides a Turing complete program execution environment, where solidity is the programming language. The program written in solidity is also called smart contract. All the operations in smart contract are publicly verifiable. In order to prevent malicious users from abusing EVM distributed

around the world, miners charge for computation and storage. The total fee (gas), consumed by the transaction invoker, equals the number of gas amount multiplying gas prices.

IV. Model and Definitions

A. Distributed Key Generation Ptotocols

We first give a formal definition of DKG protocols:

Definition 6 (Distributed Key Generation (DKG))**.** *n parties, denoted as $\mathcal{P} = (P_1, P_2, ...P_n)$, jointly generate a key pair* $(\mathsf{MPK}, \mathsf{MSK})$*. The public key* MPK *is produced in the sharing phase. The relevant private key* MSK *can be recovered by at least t honest parties in the reconstruction phase.* MPK *and* MSK *are defined by all parties in $\mathcal{Q}$, where $\mathcal{Q}$ ($\subseteq \mathcal{P}$) is the set of qualified parties and $t \leq |\mathcal{Q}|$.*

The properties of a DKG include **Correctness**, **Secrecy**, **Robustness** and **Liveness**, slightly adapted from the definition of Schindler et al. [9] and Neji et al. [8].
Correctness. A DKG protocol is correct if it satisfies:

(C1): All honest shares define the final DKG secret key MSK.

(C1)': There is an efficient process for honest parties to output the unique secret key MSK.

(C2): All honest parties have the same public key $\mathsf{MPK} = h^{\mathsf{MSK}}$, where h is a generator of prime order group $\mathbb{G}_1$ and MSK is the unique secret key defined by **(C1)**.

(C3): The secret key MSK is uniformly distributed, and MPK is uniformly distributed that is generated by h.
Secrecy. The secret key MSK is confidential to anyone or any f colluding group.
Robustness. Even with f malicious parties, any t honest parties could efficiently calculate the private key.
Liveness. No f adversaries could prevent the honest parties to complete the protocol successfully.

B. Communication Model

We assume that the n parties have shared their own long-term CP-ABE public keys ($\{(\mathsf{pk}_\theta, \mathsf{sk}_\theta)\}_{\theta \in \mathcal{P}}$, generated by the *AuthSetup* algorithm) to each other (i.e., the parties are authenticated). We define a round as a procedure that one party sends some data out and gets some other information back (similar to Fouque et al. [13]). We do not require any private channel between two parties. We leverage a public communication channel. The data submitted to the public channel is encrypted using the decentralized the CP-ABE *Encrypt* algorithm, guaranteeing data privacy. Also, we follow previous protocols [8], [9], [13], all of which use a public channel, to assume a synchronous network model.[2] Our protocol sets an expiration time $\Delta\tau$ for the parties to terminate the DKG protocol when no response is received within $\Delta\tau$.

[2]Fouque et al. directly adapt their protocol to be asynchronous with an incorruptible but honest third party in Appendix 8.1 [13]. However, they give no discussion on the implementation of the underlying asynchronous public channel. There also exists analysis on blockchain in asynchronous network [22]. Nevertheless, the research is based on a strong and unrealistic assumption that all messages arrive within certain rounds. Therefore, we claim our protocol, taking advantage of a blockchain, to be synchronous.

C. Adversarial Model

We allow adversaries to submit arbitrary messages to the public channel in the *sharing* phase to calculate MPK. Also, adversaries could start a DoS attack to bias or abort the protocol. Honest parties abide by the proposed protocol, forming the final qualified set $\mathcal{Q}$ in the *sharing* phase. We assume there are at most f malicious parties. The worst case is that they collude with each other privately. Namely, one adversary controls f parties. He could adaptively behave according to the honest ones, including initiating rushing attacks. To guarantee liveness/availability, honest parties should be more than corrupted ones, meaning $f < n/2$. The adversaries in the *sharing* phase and in the *reconstruction* phase could be different sets, which means that we allow *adaptive* adversaries. We set the threshold value to $t = n/2 + 1$ in our implementation.

To give a more intuitive picture on *adaptive* adversary $\mathcal{A}$, we consider the following three cases.
Case 1:
Step 1. In the *sharing* phase, the adversary $\mathcal{A}$ chooses f parties $\mathcal{M}_1$ to corrupt. $\mathcal{A}$ forces them to follow an arbitrary protocol of his choice.
Step 2. Other parties obey the *sharing* phase, in which each of them choose a random number and encrypts it using the decentralized CP-ABE scheme [30].
Step 3. $\mathcal{A}$ tries to prevent the *sharing* phase to be accomplished or bias the DKG public key MPK.
Case 2:
Step 1. Before the *reconstruction* phase, the adversary $\mathcal{A}$ continue to control the f parties of $\mathcal{M}_1$ and knows DKG public key MPK.
Step 2. $\mathcal{A}$ tries to obtain information about the secret key MSK.
Case 3:
Step 1. In the *reconstruction* phase, the adversary $\mathcal{A}$ loses the control of $\mathcal{M}_1$ and chooses another f parties $\mathcal{M}_2$ to corrupt. $\mathcal{A}$ can actively control their behaviors.
Step 2. Other parties parties obey the *reconstruction* phase, in which each of them issues a CP-ABE decryption key and sends it to the external user.
Step 3. $\mathcal{A}$ tries to prevent the recovery of MSK.

V. Protocol Overview

We leverage decentralized CP-ABE scheme to construct a DKG protocol (Definition 6) with a 1-round *sharing* phase to generate a DKG public key $\mathsf{MPK} := h^{\mathsf{MSK}}$, where h is a generator of $\mathbb{G}_1$, and an efficient *reconstruction* phase to calculate the DKG private key MSK. A DKG instance is composed of the *sharing* phase and *reconstruction* phase.

Figure 1 gives an overview of our protocol. To avoid too much detail of blockchain, we introduce a global functionality $\mathcal{G}_{\mathsf{verify}}$ in Figure 2 to model our verification smart

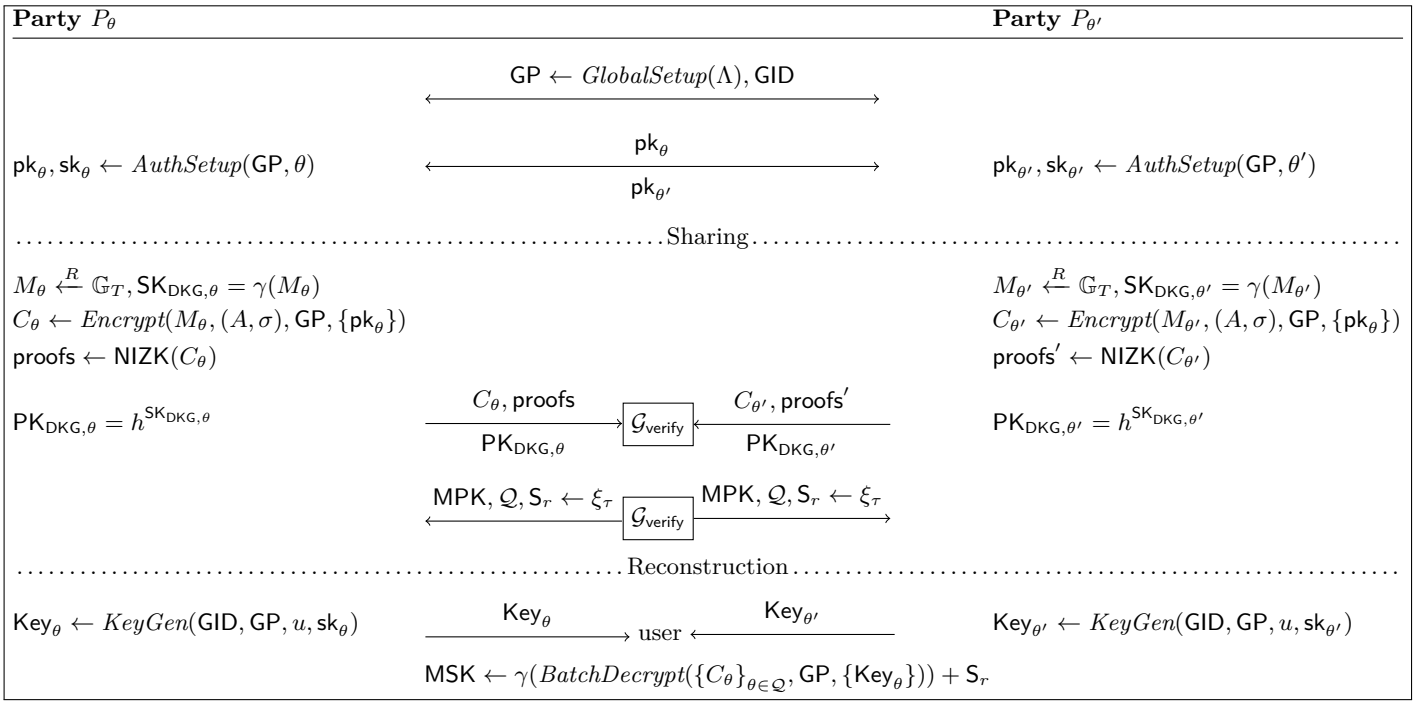

Fig. 1: Protocol overview

contract. Each party P_θ (with identifier PID_θ) randomly chooses a secret M_θ ($\in \mathbb{G}_T$) and encrypts it to get the ciphertext C_θ. The party's sub public key of the DKG is calculated as $\mathsf{PK}_{\mathsf{DKG},\theta} = h^{\mathsf{SK}_{\mathsf{DKG},\theta}}$, where $\mathsf{SK}_{\mathsf{DKG},\theta} = \gamma(M_\theta)$ and γ is an efficient function $\gamma : \mathbb{G}_T \to \mathbb{Z}_p^N$. The detailed implementation of γ is introduced in Section VIII. Each party then sends $\mathsf{PK}_{\mathsf{DKG},\theta}$, C_θ and an associating NIZK proof proofs to the smart contract. Each party can submit multiple times within the timeout, but only the last one is used. The smart contract can access a global clock and initializes with a timeout parameter $\Delta\tau$. It checks the validity of the submitted $(\mathsf{PK}_{\mathsf{DKG},\theta}, C_\theta, \mathsf{proofs})$ tuple. Such a tuple is considered to be honest if and only if it passes the check according to Equations (4). Once timeout occurs or the smart contract receives submissions from all parties, it calculates and outputs the master DKG public key as $\mathsf{MPK} = \prod_{\theta \in \mathcal{Q}} \mathsf{PK}_{\mathsf{DKG},\theta} \cdot h^{\mathsf{S}_r}$, where S_r is a random value to avoid rushing attack and $\mathcal{Q}$ ($|\mathcal{Q}| \geq t$) denotes the set of parties that submitted honest tuple.

Once an external user sends a request to recover the master DKG private key MSK, each of the parties generates a decryption key Key_θ using *KeyGen* and sends it to the user. If the external user receives at least t decryption keys, he/she is capable of decrypting all the honest ciphertexts; Finally, the master private key MSK is calculated as $\sum_{\theta \in \mathcal{Q}} \mathsf{SK}_{\mathsf{DKG},\theta} + \mathsf{S}_r$. Distributed storage via smart contract enables each party not to store encrypted shares or ciphertexts, reducing total communication complexity in the *reconstruction* phase. Moreover, to accelerate the calculation of MSK in *reconstruction* phase, we use batch decryption, which is denoted as *BatchDecrypt* and introduced in Section VII-E.

Functionality $\mathcal{G}_{\mathsf{verify}}$

The global functionality $\mathcal{G}_{\mathsf{verify}}$ (in session sid) interacts with a global clock and stores a buffer $\mathsf{buf} = \varepsilon$, a value $\alpha = h^0$, a value $c = 0$ and a set $\mathcal{Q} = \varnothing$. Let ξ_τ be a randomness source from a beacon [5], [42] or a verifiable delay functions (VDF) [43]. Set τ_0 (i.e., the protocol start time) to be the current time getting from the global clock.

- Upon receiving $(\mathsf{submit}, \mathsf{sid}, \mathsf{PID}_\theta, C_\theta, \mathsf{PK}_{\mathsf{DKG},\theta}, \mathsf{proofs}_\theta)$ from P_θ, get the current time τ and check if $\tau - \tau_0 \geq \Delta\tau$, if yes, send $(\mathsf{timeout}, \mathsf{sid}, \mathsf{PID}_\theta)$ to P_θ and the adversary $\mathcal{A}$. Otherwise, set $\mathsf{buf} = (\mathsf{PID}_\theta, C_\theta, \mathsf{PK}_{\mathsf{DKG},\theta}, \mathsf{proofs}_\theta)$. Then check proofs_θ through Equations (4). If passed, set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \mathsf{PID}_\theta$, $\alpha \leftarrow \alpha \cdot \mathsf{PK}_{\mathsf{DKG},\theta}$, $c \leftarrow c + 1$ and send $(\mathsf{accept}, \mathsf{sid}, \mathsf{PID}_\theta)$ to P_θ and $\mathcal{A}$. If not, send $(\mathsf{reject}, \mathsf{sid}, \mathsf{PID}_\theta)$ to P_θ and $\mathcal{A}$. At last, send $(\mathsf{sid}, \mathsf{buf})$ to $\mathcal{A}$ and then set $\mathsf{buf} \leftarrow \varepsilon$.
- Upon receiving $(\mathsf{query}, \mathsf{sid}, \mathsf{PID}_\theta)$ from P_θ, return $(\mathsf{sid}, \alpha, \mathcal{Q})$.
- Upon receiving $(\mathsf{eval}, \mathsf{sid})$ from an external user, get the current time τ and check if $c = n$ or $\tau - \tau_0 \geq \Delta\tau$. If yes, calculate $\mathsf{S}_r \leftarrow \xi_\tau$, $\mathsf{MPK} \leftarrow \alpha \cdot h^{\mathsf{S}_r}$ and send $(\mathsf{sid}, \mathcal{Q}, \mathsf{MPK}, \mathsf{S}_r)$ to all parties.

Fig. 2: Functionality $\mathcal{G}_{\mathsf{verify}}$ for the verification contract

In case malicious parties deviate from the protocol, the dishonest behaviors are detected and do not impact

the correctness. In the *sharing* phase, if someone submits invalid C_θ or $\mathsf{PK}_{\mathsf{DKG},\theta}$, the smart contract can recognize it. In the *reconstruction* phase, if party $P_\theta \in \mathcal{Q}$ behaves maliciously, a subset of more than t honest parties can recover M_θ through decrypting C_θ jointly.

In this paper, the global parameters are instantiated by Rouselakis and Waters's decentralized CP-ABE scheme [30] and the global functionality $\mathcal{G}_{\mathsf{verify}}$ is implemented on Ethereum (refer to Section VIII).

VI. Proofs of Plaintext Knowledge

A. Adding NIZK Proofs

In this section, we demonstrate how to construct zero-knowledge proofs of plaintext knowledge for the Rouselakis and Waters's decentralized CP-ABE [30] scheme. In their construction, an authority P_θ (also an encryptor in our protocol) generates a long-term key pair, where private key $\mathsf{sk}_\theta = (\alpha_\theta, y_\theta)$ and public key $\mathsf{pk}_\theta = (e(g_1,g_2)^{\alpha_\theta}, g_2^{y_\theta})$. Without causing confusion, sometimes we omit the subscript θ of P_θ, M_θ, C_θ and $\mathsf{PK}_{\mathsf{DKG},\theta}$. With all authorities' public keys, a party P encrypts a message M by *Encrypt* algorithm and gets a ciphertext $C = (C_0, C_1, C_2, C_3, C_4) = (C_0, \{C_{1x}, C_{2x}, C_{3x}, C_{4x}\}_{x\in[l]})$ which is calculated as the Equation (1) (refer to [30]). In the equation, $[l]$ is the set of rows of LSSS matrix A generated from access control policy acp (refer to Section VII-A for details). g_1, g_2 are generators of group $\mathbb{G}_1$ and $\mathbb{G}_2$, defined in Section III-A. e is a bilinear mapping from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$. ρ is a function that maps rows of LSSS matrix to authorities. F is a function that maps an arbitrary string to a point of $\mathbb{G}_1$. λ_x and ω_x are the shares of $z \xleftarrow{R} \mathbb{Z}_p$ and 0, respectively, for each row x of A. k_x is randomly chosen from $\mathbb{Z}_p$ for each row x.

$$C = \begin{cases} C_0 = Me(g_1,g_2)^z, \\ \Big\{C_{1x} = e(g_1,g_2)^{\lambda_x} e(g_1,g_2)^{\alpha_{\rho(x)} k_x}, \\ C_{2x} = g_2^{-k_x}, \\ C_{3x} = g_2^{y_{\rho(x)} k_x} g_2^{\omega_x}, \\ C_{4x} = F(\sigma(x))^{k_x} \Big\}_{x\in[l]} \end{cases} \tag{1}$$

For each party, the sub DKG public key $\mathsf{PK}_{\mathsf{DKG}}$ is defined as:

$$\mathsf{PK}_{\mathsf{DKG}} \coloneqq h^{\gamma(M)} \tag{2}$$

h is randomly chosen generator of $\mathbb{G}_1$. To prove the knowledge of M in C, i.e., $\mathsf{PoK}\{(M) : C = \mathit{Encrypt}(M, (A,\sigma), \mathsf{GP}, \{\mathsf{pk}_\theta\})\}$, the prover chooses random values z' from $\mathbb{Z}_p$, M' from $\mathbb{G}_T$, then calculates C' and $\mathsf{PK}'_{\mathsf{DKG}}$ as follows:

$$C' = \begin{cases} C'_0 = M' e(g_1,g_2)^{z'}, \\ \Big\{C'_{1x} = e(g_1,g_2)^{\lambda'_x} e(g_1,g_2)^{\alpha_{\rho(x)} k'_x}, \\ C'_{2x} = g_2^{-k'_x}, \\ C'_{3x} = g_2^{y_{\rho(x)} k'_x} g_2^{\omega'_x}, \\ C'_{4x} = F(\sigma(x))^{k'_x} \Big\}_{x\in[l]} \end{cases} \tag{3}$$

$$\mathsf{PK}'_{\mathsf{DKG}} \coloneqq h^{\gamma(M')}$$

Then, by Fiat-Shamir heuristic, he/she calculates $c_1 = H_1(C, C')$, $c_2 = H_2(C, C')$, $M_1 = \frac{M'}{M^{c_1}}$, $M_2 = (\gamma(M') - c_2\gamma(M)) \bmod q$, $\tilde{z} = z' - c_1 z$, $\{\tilde{k_x} = k'_x - c_1 k_x$, $\tilde{\lambda_x} = \lambda'_x - c_1\lambda_x$, $\tilde{\omega_x} = \omega'_x - c_1\omega_x\}_{x\in[l]}$, where $H_1, H_2 : \{0,1\}^* \to \mathbb{Z}_p$ are different hash functions. P sends all these values as a conversation $((C, \mathsf{PK}_{\mathsf{DKG}}, C', \mathsf{PK}'_{\mathsf{DKG}}), (c_1, c_2), (M_1, M_2, \tilde{z}, \{\tilde{k_x}, \tilde{\lambda_x}, \tilde{\omega_x}\}_{x\in[l]}))$ to the verification contract (i.e., the verifier V). V invokes algorithm *Check* and it outputs Yes if and only if the following equations hold:

$$\begin{aligned} C'_0 &\stackrel{?}{=} M_1 e(g_1,g_2)^{\tilde{z}} C_0^{c_1} \\ \Big\{C'_{1x} &\stackrel{?}{=} e(g_1,g_2)^{\tilde{\lambda_x}} e(g_1,g_2)^{\alpha_{\rho(x)}\tilde{k_x}} C_{1x}^{c_1} \\ C'_{2x} &\stackrel{?}{=} g_2^{-\tilde{k_x}} C_{2x}^{c_1} \\ C'_{3x} &\stackrel{?}{=} g_2^{y_{\rho(x)}\tilde{k_x}} g_2^{\tilde{\omega_x}} C_{3x}^{c_1} \\ C'_{4x} &\stackrel{?}{=} F(\sigma(x))^{\tilde{k_x}} C_{4x}^{c_1} \Big\}_{x\in[l]} \\ \mathsf{PK}'_{\mathsf{DKG}} &\stackrel{?}{=} h^{M_2} \mathsf{PK}_{\mathsf{DKG}}^{c_2} \\ \tilde{z} &\stackrel{?}{=} \mathit{interpolate}(\{\sigma(x), \tilde{\lambda_x}\}_{x\in[l]}) \\ 0 &\stackrel{?}{=} \mathit{interpolate}(\{\sigma(x), \tilde{\omega_x}\}_{x\in[l]}) \end{aligned} \tag{4}$$

Otherwise, it outputs No. *interpolate* presents the Lagrange polynomial interpolation [32], where any t out of $\{\sigma(x), \lambda_x\}_{x\in[l]}$ (or $\{\sigma(x), \omega_x\}_{x\in[l]}$) recovers $\tilde{z}$ (or 0). The *interpolate* in the verification phase prevents a malicious party from constructing an invalid access control policy.

B. Security Analysis

In this section, we prove the *correctness*, the *special knowledge soundness* and the *special HVZK* of our sigma protocol construction given by Section VI-A.

1) Correctness. The protocol in the previous subsection is correct if the *Check* algorithm of Fiat-Shamir heuristic always output Yes. By checking all the verification equations (4) from right to left, we prove the correctness as follows:

$$\begin{aligned} M_1 e(g_1,g_2)^{\tilde{z}} C_0^{c_1} &= \frac{M'}{M^{c_1}} (Me(g_1,g_2)^z)^{c_1} e(g_1,g_2)^{z' - c_1 z} \\ &= M' e(g_1,g_2)^{z'} = C'_0 \end{aligned}$$

$$\begin{aligned} &e(g_1,g_2)^{\tilde{\lambda_x}} e(g_1,g_2)^{\alpha_{\rho(x)}\tilde{k_x}} C_{1x}^{c_1} = e(g_1,g_2)^{\lambda'_x - c_1\lambda_x}. \\ &e(g_1,g_2)^{\alpha_{\rho(x)}(k'_x - c_1 k_x)} (e(g_1,g_2)^{\lambda_x} e(g_1,g_2)^{\alpha_{\rho(x)} k_x})^{c_1} \\ &= e(g_1,g_2)^{\lambda'_x} e(g_1,g_2)^{\alpha_{\rho(x)} k'_x} = C'_{1x} \end{aligned}$$

$$g_2^{-\tilde{k_x}} C_{2x}^{c_1} = g_2^{-(k'_x - c_1 k_x)} g_2^{-c_1 k_x} = g_2^{-k'_x} = C'_{2x}$$

$$\begin{aligned} g_2^{y_{\rho(x)}\tilde{k_x}} g_2^{\tilde{\omega_x}} C_{3x}^{c_1} &= g_2^{y_{\rho(x)}(k'_x - c_1 k_x)} g_2^{\omega'_x - c_1\omega_x} (g_2^{y_{\rho(x)} k_x} g_2^{\omega_x})^{c_1} \\ &= g_2^{y_{\rho(x)} k'_x} g_2^{\omega'_x} = C'_{3x} \end{aligned}$$

$$\begin{aligned} F(\sigma(x))^{\tilde{k_x}} C_{4x}^{c_1} &= F(\sigma(x))^{(k'_x - c_1 k_x)} (F(\sigma(x))^{k'_x})^{c_1} \\ &= F(\sigma(x))^{k'_x} = C'_{4x} \end{aligned}$$

$$h^{M_2}\mathsf{PK}_{\mathsf{DKG}}^{c_2} = h^{\gamma(M')-c_2\cdot\gamma(M))}(h^{\gamma(M)})^{c_2} = h^{\gamma(M')} = \mathsf{PK}'_{\mathsf{DKG}}$$

The last two verification operations related to *interpolate* in Equations (4) is correct due to the linearity of LSSS.

2) Special Knowledge Soundness. To prove it's efficient to calculate M with two accepting conversations, let's write C_0 in this form $C_0 = r^m q^z$, where $r = e(g_1', g_2'), q = e(g_1, g_2)$ and g_1', g_1 are the generators of $\mathbb{G}_1$, g_2', g_2 are the generators of $\mathbb{G}_2$, m is the value which makes $M = r^m$. Then r, q are points on $\mathbb{G}_T$. In this notation, we get the same form as Okamoto's protocol [33]. No one, except the encryptor, has knowledge about m, however, it has no impact on the deduction of knowledge soundness proof. Suppose two conversations (C_{0t}, c_{11}, $(M_{1t} = r^{m_1}, z_1)$) and (C_{0t}, c_{12}, $(M_{2t} = r^{m_2}, z_2)$) are accepted for C_0 in the sigma protocol. Then we have:

$$r^{m_1}q^{z_1} = C_{0t}C_0^{c_{11}} \text{ and } r^{m_2}q^{z_2} = C_{0t}C_0^{c_{12}}$$

Since $c_{12} \neq c_{11}$, let's denote $\Delta c = c_{12} - c_{11}$, $\Delta m = m_2 - m_1$, $\Delta z = z_2 - z_1$. By dividing the two equations, we have

$$r^{\Delta m}q^{\Delta z} = C_0^{\Delta c} = r^{m\Delta c}q^{z\Delta c}$$
$$\Rightarrow m = \Delta m \cdot \Delta c^{-1}, z = \Delta z \cdot \Delta c^{-1}$$
$$\Rightarrow M = r^{\Delta m\cdot\Delta c^{-1}} = (r^{m_2-m_1})^{\Delta c^{-1}} = \frac{M_{2t}}{M_{1t}}^{\Delta c^{-1}}$$

In the above demonstration, we put C_0 in a simplified form in which we introduce an unknown variable m. We show that m does not affect the result of obtaining M and z in an efficient computation from the deduction. In the same way, k_x in C_{2x}, λ_x in C_{1x} and ω_x in C_{3x} also can be effectively calculated with similar two conversations. Because C_{2x} is the same with Schnorr's protocol [34], and C_0, C_{1x}, C_{3x} are the same with Okamoto's protocol [33].

3) Special HVZK. A *simulator* could generate a conversation in arbitrary order, unlike the real conversation between P and V. We retake C_0 of ciphertext C in the form of $C_0 = r^m q^z$ and the conversation in the form of (C_0, c, (M, z)). The simulator randomly chooses $M \xleftarrow{R} \mathbb{G}_T$, $z \xleftarrow{R} \mathbb{Z}_p$, and on input $C_0 \in \mathbb{G}_T$ and c $\in \mathcal{C}$ ($\mathcal{C}$ is the challenge space), then computes:

$$C_{0t} \leftarrow Mq^z/C_0^c$$

and the output is (C_{0t}, $(M,\ z)$). It's evident that $(C_{0t}, c, (M, z))$ always represents an accepting conversation, as required.

Now we prove that the output of the simulator has the exact distribution when c is randomly chosen. In a real conversation, c, M, z are mutually independent where c is uniformly distributed over $\mathbb{Z}_p$, M is uniformly distributed over $\mathbb{G}_T$, z is uniformly distributed over $\mathbb{Z}_p$. With the equation $Mq^z = C_{0t}C_0^c$, C_{0t} is uniquely determined then. It's easy to observe that the real conversation and the simulator have the same output distribution.

Similarly, other parts (C_1, C_2, C_3, C_4) of ciphertext C and $\mathsf{PK}_{\mathsf{DKG}}$ can be proved to have the same output distribution with the simulator.

VII. Construction Details

In this section, we give the construction of our DKG protocol based on Rouselakis and Waters's decentralized CP-ABE scheme [30]. In our protocol, each party is considered to be an *attribute authority* and an *encryptor* at the same time. The decentralized CP-ABE *GlobalSetup* algorithm can be invoked by any trustless user or a verifiable delay function [43] and outputs global parameters GP. Each party P_θ in group $\mathcal{P}$ invokes *AuthSetup* to obtain its long-term private key sk_θ=(α_θ, y_θ) and public key pk_θ= $(e(g_1, g_2)^{\alpha_\theta}, g_1^{y_\theta})$. GP and $\{\mathsf{pk}_\theta\}_{\theta\in\mathcal{P}}$ are public available for each party.

A. Access Control Policy in Our Usage

Access control policy can be expressed in different kinds of forms [28], such as LSSS matrices, boolean formulae, threshold access trees and AND-OR-gate access trees. The latter three are more intuitive and readable than LSSS matrix. While LSSS matrix is more often used to specify the policies in the ciphertext. A host of CP-ABE protocols directly use LSSS matrix when encrypting, including Rouselakis and Waters's protocol. Beimel [27] has proved that a LSSS matrix can express any monotone access control policy. Liu et al. [28] demonstrate how to efficiently convert threshold form into a LSSS matrix. We take advantage of the (t, n)-threshold form when considering threshold cooperation in our DKG protocol.

Suppose a group with n parties in our DKG protocol, denoted by $\mathcal{P} = (P_1, P_2, ...P_n)$. Each of them is both an attribute authority and an encryptor. Attribute strings are in the form of "$\mathsf{GID@PID}_\theta$" following Rouselakis and Waters's construction [30], in which "GID" is the identifier of decryptor and "PID_θ" is the identifier of an attribute authority. "$\mathsf{GID@PID}_\theta$" means that attribute authority P_θ can generate a decryption key with attribute string "$\mathsf{GID@PID}_\theta$" for a virtual decryptor "GID". In our proposed DKG protocol, each party P_θ sets the access control policy $\mathsf{acp} \leftarrow$ "t of $(\mathsf{GID@PID}_1, \mathsf{GID@PID}_2, ..., \mathsf{GID@PID}_n)$" to obtain a ciphertext. A ciphertext C is encrypted using *Encrypt* algorithm with acp as input. If at least t attribute authorities, each of them takes $\mathsf{GID@PID}_\theta$ as input, generate decryption keys using *KeyGen*, then the decryptor can decrypt C successfully. The decryption keys are combined in *Decrypt* algorithm by the external user who owns "GID". Since each DKG instance has a unique session sid and a unique GID, we could set $\mathsf{sid} = \mathsf{GID}$ in the implementation.

B. Sharing Ciphertext

In the *sharing* phase, party P_θ calculates the ciphertext using Rouselakis and Waters's protocol as following (refer to algorithm (1)):

$$M_\theta \xleftarrow{R} \mathbb{G}_T$$
$$\mathsf{acp} = \text{“}t \text{ of } (\mathsf{GID@PID}_1, \mathsf{GID@PID}_2, ..., \mathsf{GID@PID}_n)\text{”}$$
$$(\mathrm{A}_\theta, \sigma_\theta) \leftarrow \mathsf{acp}$$
$$C_\theta \leftarrow \mathit{Encrypt}(M_\theta, (\mathrm{A}_\theta, \sigma_\theta), \mathsf{GP}, \{\mathsf{pk}_\theta\}_{\theta\in\mathcal{P}}) \quad (5)$$

That means party P_θ randomly chooses a value M_θ from $\mathbb{G}_T$, transfers the threshold access tree (acp) into LSSS matrix, and encrypts M_θ under the LSSS matrix to generate ciphertext C_θ. Together with C_θ, party P_θ calculates $\mathsf{PK}_{\mathsf{DKG},\theta}$ as (2) demonstrates.

$$\mathsf{PK}_{\mathsf{DKG},\theta} = h^{\mathsf{SK}_{\mathsf{DKG},\theta}} = h^{\gamma(M_\theta)} \tag{6}$$

then he/she calculates (C'_θ, $\mathsf{PK}'_{\mathsf{DKG},\theta}$) according to (3) in the same way. Finally, party P_θ sets $\mathsf{proofs}_\theta = (C'_\theta, \mathsf{PK}'_{\mathsf{DKG},\theta}, \tilde{M}_\theta, \tilde{z_\theta}, \tilde{t_{\theta x}}, \tilde{\lambda_{\theta x}}, \tilde{\omega_{\theta x}})$ and then sends (submit, GID, PID_θ, C_θ, $\mathsf{PK}_{\mathsf{DKG},\theta}$, proofs_θ) to global functionality $\mathcal{G}_{\mathsf{verify}}$. That indicates the party P_θ commits to value M_θ in C_θ.

Lemma 1. *It is impossible for adversaries to break the ciphertext of Rouselakis and Waters's protocol or extract M_θ from an honest party in the sharing phase.*

Proof. Rouselakis and Waters's ciphertext C_θ keeps the secrecy of M_θ in our scenario. The proof proofs_θ generated by sigma protocol reveals no information about the plaintext M_θ. Since the threshold $t = n/2 + 1$, less than $n/2 (< t)$ malicious parties cannot learn useful information about the plaintext M_θ in the *sharing* phase. Besides, to deduce M_θ from $\mathsf{PK}_{\mathsf{DKG},\theta} = h^{\gamma(M_\theta)}$ is hard based on the discrete logarithm assumption.

C. Calculating MPK

In an upper time bound $\Delta\tau$, all parties have submitted their ciphertexts and proofs. The equations in (4) are checked in smart contract to find out all the well-constructed ciphertexts. Those who pass the verification form a qualified set $\mathcal{Q}$ ($\mathcal{P} - \mathcal{M}_1 \subseteq \mathcal{Q} \subseteq \mathcal{P}$). Once all parties have submitted or the timeout passes, any external party can send ($\mathsf{eval}, \mathsf{GID}$) to $\mathcal{G}_{\mathsf{verify}}$ and the MPK of the DKG is calculated as follows:

$$\mathsf{MPK} \leftarrow \prod_{\theta \in \mathcal{Q}} \mathsf{PK}_{\mathsf{DKG},\theta} \cdot h^{\mathsf{S}_r} = \prod_{\theta \in \mathcal{Q}} h^{\gamma(M_\theta)} \cdot h^{\mathsf{S}_r}$$

D. Reconstruction of MSK

In *reconstruction* phase, the master secret key MSK is defined as below:

$$\mathsf{MSK} \coloneqq \sum_{\theta \in \mathcal{Q}} \gamma(M_\theta) + \mathsf{S}_r$$

We assume *adaptive* adversaries, so that a party P_θ is in $\mathcal{Q}$ does not guarantee this party is honest or always available. Therefore, if anyone (in or out of the protocol), called a *user*, wants to recover M_θ, we require he/she to send requests to each party, including those that are not in $\mathcal{Q}$. Each party P_θ invokes *KeyGen* (with $O(1)$ computation complexity) to generate Key_θ as follows:

$$\begin{gathered} u \leftarrow \text{“}\mathsf{GID}@\mathsf{PID}_\theta\text{”} \\ d \xleftarrow{R} \mathbb{Z}_p \\ \mathsf{Key}_\theta = \mathsf{Key}_{\mathsf{GID},u} \leftarrow \mathit{KeyGen}(\mathsf{GID}, \mathsf{GP}, u, \mathsf{sk}_\theta) \\ = \{K_{\mathsf{GID},u} = g_1^{\alpha_\theta} H(\mathsf{GID})^{y_\theta} F(u)^d, K'_{\mathsf{GID},u} = g_2^d\} \end{gathered} \tag{7}$$

H is a public available function that hashes an attribute to a unique authority. Also, we prove that our protocol can detect "bad" decryption keys when there are malicious parties (by Lemma 2). When the user receives at least $\lfloor n/2 \rfloor + 1$ honest keys, he/she can merge these keys together and execute the *Decrypt* algorithm to obtain M_θ following Rouselakis and Waters's construction [30]. The algorithm first computes for each such row x of the LSSS matrix:

$$\begin{gathered} C_{1x} \cdot e(K_{\mathsf{GID},\sigma(x)},\ C_{2x}) \cdot e(H(\mathsf{GID}),\ C_{3x}) \cdot \\ e(K'_{\mathsf{GID},\sigma(x)},\ C_{4x}) = e(g_1, g_2)^{\lambda_x} e(H(\mathsf{GID}), g_2)^{\omega_x} \end{gathered} \tag{8}$$

Then, it calculates constants c_x such that $\sum_x c_x A_x = (1, 0, \cdots, 0)$ and it computes:

$$\prod_x (e(g_1, g_2)^{\lambda_x} e(H(\mathsf{GID}), g_2)^{\omega_x})^{c_x} = e(g_1, g_2)^z \tag{9}$$

Finally, M_θ is obtained:

$$M_\theta = M = C_0 / e(g_1, g_2)^z \tag{10}$$

Lemma 2. *When reconstructing MSK, a party can behave dishonestly by generating a "bad" decryption key. However, this won't affect the reconstruction phase and the dishonesty is discovered as soon as it submits the key.*

Proof. In Section VII-D, a party P_θ takes in "$\mathsf{GID}@\mathsf{PID}_\theta$" and invokes *KeyGen* algorithm to issue a decryption key Key_θ. To ensure the validity of Key_θ, we require the party to encrypt "$1_{\mathbb{G}_T}$" using access control policy "1 of ($\mathsf{GID}@\mathsf{PID}_\theta$)", and submit the ciphertext C_{test} along Key_θ. In this manner, the decryption key Key_θ can be proved to be valid if it can be used to decrypt C_{test} successfully. The decryption costs $O(1)$, since only one attribute is in the access control policy. By Lemma 1, we can infer that no forged decryption key can decrypt the ciphertext with high probability. Since party P_θ has bound its unique identity PID_θ in *AuthSetup* algorithm, it will soon be discovered if decryption for C_{test} fails. With at least $n - f (>= t)$ honest parties (i.e., $\mathcal{P} - \mathcal{M}_2$), the combined decryption keys will successfully decrypt all $\{M_\theta\}_{\theta \in \mathcal{Q}}$ through invoking *Decrypt* algorithm.

A corrupted party P_θ may launch rushing attack: biasing MPK by selectively submitting its $\mathsf{PK}_{\mathsf{DKG},\theta}$ or not. We address the rushing attack problem by introducing a random value $\mathsf{S}_r \leftarrow \xi_\tau$ from the smart contract $\mathcal{G}_{\mathsf{verify}}$, where ξ_τ denotes the randomness beacon [5], [42]. To ensure unpredictability, S_r is defined as the first randomness after the time τ. With either beacon or VDF, the value of S_r is also unbiased and publicly verifiable. Thus, the DKG key pairs are calculated as: $\mathsf{MPK} = \prod_{\theta \in \mathcal{Q}} \mathsf{PK}_{\mathsf{DKG},\theta} \cdot h^{\mathsf{S}_r}$ and $\mathsf{MSK} = \sum_{\theta \in \mathcal{Q}} \gamma(M_\theta) + \mathsf{S}_r$.

E. Batch Decryption in Reconstruction

In our protocol, the decryption keys, as equations (7) produce, remain unchanged for all the ciphertexts in the *Decrypt* algorithm. It's easy to notice that these ciphertexts can be aggregated into one single ciphertext by

multiplication, of which the corresponding plaintext is exactly MSK. The aggregation takes $O(n^2)$ "add" operations on $\mathbb{G}_T$, which is negligible, as Table III shows. Indeed, this technique benefits from the property of homomorphism of the decentralized CP-ABE [30]. We term it batch decryption, which enables a user to invoke *Decrypt* algorithm only once and highly lowers the computation complexity from $O(n^2)$ to $O(n)$. The batch decryption algorithm $\mathsf{MSK} \leftarrow \mathit{BatchDecrypt}(\{C_\theta\}_{\theta\in\mathcal{Q}}, \mathsf{GP}, \{\mathsf{Key}_\theta\})$ runs as follows:

$$\tilde{C} = \left(\prod_{\theta\in\mathcal{Q}} C_0, \left\{\prod_{\theta\in\mathcal{Q}} C_{1x}, \prod_{\theta\in\mathcal{Q}} C_{2x}, \prod_{\theta\in\mathcal{Q}} C_{3x}, \prod_{\theta\in\mathcal{Q}} C_{4x}\right\}_{x\in[l]}\right)$$

$$\mathit{Decrypt}(\tilde{C}, \mathsf{GP}, \{\mathsf{Key}_\theta\}) \rightarrow \sum_{\theta\in\mathcal{Q}} M_\theta \tag{11}$$

$$\mathsf{MSK} = \gamma(\sum_{\theta\in\mathcal{Q}} M_\theta) + \mathsf{S}_r = \sum_{\theta\in\mathcal{Q}} \gamma(M_\theta) + \mathsf{S}_r$$

Any user who collects at least t valid decryption keys can invoke *BatchDecrypt* to reconstruct MSK. The total communication complexity remains $O(n)$, since each decryption key is of size $O(1)$ and the aggregated ciphertext is $O(n)$. Therefore, both the communication complexity and computation complexity are $O(n)$ in *reconstruction* phase.

VIII. Implementation and Evaluation

A. Implementation Environment

Our protocol is composed of an offline (local) part and an online (smart contract) part. Because in *sharing* phase, parties only interact with a smart contract, there is no need to implement a P2P network. In the offline part, we implement our decentralized CP-ABE in Python language based on the *py_ecc* library[3], which has an implementation of elliptic curve crypto (ECC) curve "bn_128". The offline part is executed on MacBook Air macOS 10.15.4, 1.6 GHz Dual-Core Intel Core i5 CPU, 16 GB RAM, Python 3.7.7. We also run Ganache 2.3.0[4] as a full node to handle the smart contract, which is compiled by solidity codes compiler v0.5.17.

Elements in $\mathbb{G}_T$ of curve "bn_128" are with embedding degree=12 [38]. Thus, the map function γ is defined as follows: $M_\theta \rightarrow [M_{\theta,0}, M_{\theta,1}, ..., M_{\theta,11}]$, where $M_{\theta,j}$ $(j \in [0, 11])$ is an element in $\mathbb{Z}_p$. Then, $h^{\gamma(M_\theta)}$ is calculated as $[h^{M_{\theta,0}}, h^{M_{\theta,1}}, ..., h^{M_{\theta,11}}]$. Our experiment has shown the correctness of map function γ.

To our knowledge, we are the first to implement decentralized CP-ABE [30] on the "bn_128" curve. It is interesting to notice that verification costs on a personal computer (PC) and that on Ethereum differ tremendously, which indicates that Ethereum has an entirely different computation model for some cryptographic primitives.

[3]py_ecc: https://github.com/ethereum/py_ecc, Accessed: 2021-11-13

[4]Ganache: https://www.trufflesuite.com/ganache, Accessed: 2021-11-13

B. Adaption of Verification on Ethereum

Ethereum improvement proposals (EIP) are standards for APIs and core protocols that are used on Ethereum. EIP-197 [39] provides optimal Ate pairing check on the elliptic curve "bn_128" where $\mathbb{G}_1$ is defined by $Y^2 = X^3+3$ and $\mathbb{G}_2$ is defined by $Y^2 = X^3+3/(i+9)$. EIP-196 [40] has demonstrated that operations on $\mathbb{G}_1$ is also pre-built on Ethereum. However, C_0 and C_1 are group elements of $\mathbb{G}_T$ (i.e., the bilinear mapping group on *bn_128*), which is not provided by Ethereum solidity. Due to this obstacle, we reform the ciphertext and fine-tune the verification process without loss of correctness and security. We denote C_0 and C_{1x} as a tuple with two elements. The first element is a point on $\mathbb{G}_1$ and the second one is g_2, so that the pairing verification is equal to point comparison on $\mathbb{G}_1$.

$$\begin{aligned} C_0 = e(g_1,g_2)^m e(g_1,g_2)^z = e(g_1^{m+z}, g_2) \\ C_0 \xrightarrow{\text{denoted as}} (g_1^{m+z}, g_2) \\ C_{1x} = e(g_1,g_2)^{\lambda_x} e(g_1,g_2)^{\alpha_{\rho(x)}k_x} \\ C_{1x} \xrightarrow{\text{denoted as}} (g_1^{\lambda_x+\alpha_{\rho(x)}k_x}, g_2) \end{aligned} \tag{12}$$

Then the verification for C_0 and C_{1x} in equation (4) is converted into following equations.

$$\begin{aligned} C_0' &\stackrel{?}{=} M_1 e(g_1,g_2)^{\tilde{z}} C_0^{c_1} \\ &\Updownarrow \\ g_1^{(m'+z')} &\stackrel{?}{=} g_1^{\tilde{m}} g_1^{\tilde{z}} (g_1^{m+z})^{c_1} \\ C_{1x}' &\stackrel{?}{=} e(g_1,g_2)^{\tilde{\lambda_x}} e(g_1,g_2)^{\alpha_{\rho(x)}\tilde{k_x}} C_{1x}^{c_1} \\ &\Updownarrow \\ g_1^{\lambda_x'+\alpha_{\rho(x)}k_x'} &\stackrel{?}{=} g_1^{\tilde{\lambda_x}} g_1^{\alpha_{\rho(x)}\tilde{k_x}} (g_1^{\lambda_x+\alpha_{\rho(x)}k_x})^{c_1} \end{aligned} \tag{13}$$

In equations (12) and (13), m, m' are randomly chosen by the encryptor from $\mathbb{Z}_p$ to get $M = e(g_1,g_2)^m$ and $M' = e(g_1,g_2)^{m'}$, respectively. $\tilde{m} = m' - mc_1$, which is also sent to the smart contract.

Since C_{2x} and C_{3x} in equation (4) are elements of group $\mathbb{G}_2$ which is not well-supported on Ethereum. (See experimental results about the gas consumption in Table III.) We propose a method to verify C_{2x} and C_{3x} efficiently with operations on $\mathbb{G}_1$ (EIP-196) and pairing check (EIP-197). By changing the base in $C_{2x} = g_2^{-k_x}$ and $C_{3x} = g_2^{y_{\rho(x)}k_x} g_2^{\omega_x}$ from g_2 to g_1, we let $\hat{C}_{2x} \coloneqq g_1^{-k_x}$ and $\hat{C}_{3x} \coloneqq g_1^{y_{\rho(x)}k_x} g_1^{\omega_x}$. Then the smart contract verifies the $\hat{C}_{2x}$ and $\hat{C}_{3x}$ by checking the following equations:

$$\begin{aligned} \hat{C}_{2x}' &\stackrel{?}{=} g_1^{-\tilde{k_x}} \hat{C}_{2x}^{c_1} \\ \hat{C}_{3x}' &\stackrel{?}{=} g_1^{y_{\rho(x)}\tilde{k_x}} g_1^{\tilde{\omega_x}} \hat{C}_{3x}^{c_1} \end{aligned} \tag{14}$$

$$\begin{aligned} e(g_1, C_{2x}) &\stackrel{?}{=} e(\hat{C}_{2x}, g_2) \\ e(g_1, C_{3x}) &\stackrel{?}{=} e(\hat{C}_{3x}, g_2) \end{aligned} \tag{15}$$

Equations (15) are to prove that C_{2x} and $\hat{C}_{2x}$ (as well as C_{3x} and $\hat{C}_{3x}$) have a same exponent, but with different bases. Our experiment shows the method is far more gas-efficient than direct operations on $\mathbb{G}_2$ [37].

Lemma 3. *Under the discrete logarithm assumption and assumption of the decentralized CP-ABE scheme [30], an unauthorized party cannot pass as an honest party.*

Proof. We use non-interactive Sigma protocol to prove plaintext of secret value M_θ in Rouselakis and Waters's protocol in Section VI. Rouselakis and Waters's protocol is static secure based on q-DPBDHE2 assumption [30]. We reform part of the ciphertext (C_0 and C_1) by denoting element on $\mathbb{G}_T$ as elements on $\mathbb{G}_1$ and $\mathbb{G}_2$ in implementation, described in equation (12). It is hard to know $m+z$ with (g_1, g_1^{m+z}) due to discrete logarithm assumption. It is also hard to know g_1^m from g_1^{m+z}, where z is a random value. So the reformation maintains the privacy of the plaintext. Also, the reformation does not impact the correctness of the decentralized CP-ABE scheme, since C_0 and C_1 on $\mathbb{G}_T$ can be re-calculated. Therefore, an unauthorized party cannot pass the verification Equations (4) executed in the smart contract as an honest party.

C. Performance and Cost

Table II introduces the number of operations in the decentralized CP-ABE algorithms, which is deduced from Equations (1), (2), (7), (8), (9), and (10). The notation $\text{Add}_{\mathbb{G}_i}$ or $\text{Multiply}_{\mathbb{G}_i}$ in the table denotes the multiplication or exponentiation on group $\mathbb{G}_i$. Though the off-chain operations (decentralized CP-ABE algorithms) are trivial for the 1-round DKG protocol, we evaluate the group operation cost on "bn_128" curve. Table III compares gas consumption and time cost about group operations on PC and Ethereum. Obviously, curve "ss_512" is much more efficient than curve "bn_128", since it is based on well-known GMP (the Gnu MultiPrecision) library[5] and PBC (Pairing-Based Cryptography) library[6] on PC. We also observe that the computation cost of $\mathbb{G}_2$ using implementation [37] on Ethereum is too much costly.

TABLE II: Number of operations of the decentralized CP-ABE algorithms

Algorithm	$\text{Add}_{\mathbb{G}_1}$	$\text{Multiply}_{\mathbb{G}_1}$	$\text{Add}_{\mathbb{G}_2}$	$\text{Multiply}_{\mathbb{G}_2}$	$\text{Add}_{\mathbb{G}_T}$	$\text{Multiply}_{\mathbb{G}_T}$	Pairing
Encrypt	–	$n+1$	n	$3n$	$n+1$	$3n$	–
KeyGen	2	3	–	1	–	–	–
Decrypt	–	–	–	–	$3n$	$4n$	n

TABLE III: Comparison of group operation cost on PC (second) and Ethereum (gas)

Platform	Curve	$\text{Add}_{\mathbb{G}_1}$	$\text{Multiply}_{\mathbb{G}_1}$	$\text{Add}_{\mathbb{G}_2}$	$\text{Multiply}_{\mathbb{G}_2}$	$\text{Add}_{\mathbb{G}_T}$	$\text{Multiply}_{\mathbb{G}_T}$	Pairing
PC	ss_512	1.4×10^{-5}	0.002	1.2×10^{-5}	0.002	1.7×10^{-5}	0.002	0.0009
PC	bn_128	3×10^{-5}	0.01	1.6×10^{-4}	0.04	8×10^{-4}	0.19	0.7
Ethereum	bn_128	500	40000	**30,000**	**2,000,000**	–	–	80,000

Therefore, we transform operations on $\mathbb{G}_2$ and $\mathbb{G}_T$ into operations on $\mathbb{G}_1$ and paring check. Table IV depicts the original verification cost (on PC) of the ciphertext in equations (4). Table V demonstrates the times of verification operations of each part in the ciphertext on Ethereum, as equations (13), (14) and (15) demonstrate. In the tables, C_0, C_1, C_2, C_3, C_4, $\mathsf{PK}_{\mathsf{DKG}}$ are different parts described in equations (1) and (2). Recall that the notation $\text{Add}_{\mathbb{G}_i}$ and $\text{Multiply}_{\mathbb{G}_i}$ are multiplication and exponentiation on group $\mathbb{G}_i$. "ModPow" performs modulus division on a big number raised to the power of another big number. "Pairing" is the bilinear mapping from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$ on PC and is the pairing check (defined in EIP-197) on Ethereum. From Table IV and V, we can conclude that C_0 and $\mathsf{PK}_{\mathsf{DKG}}$ verification only cost a constant number of gas, and C_1, C_2, C_3, C_4 verification cost linearly with increasing numbers of parties.

TABLE IV: Original verification operations on PC

Part	$\text{Add}_{\mathbb{G}_1}$	$\text{Multiply}_{\mathbb{G}_1}$	ModPow	$\text{Add}_{\mathbb{G}_2}$	$\text{Multiply}_{\mathbb{G}_2}$	$\text{Add}_{\mathbb{G}_T}$	$\text{Multiply}_{\mathbb{G}_T}$
C_0	–	–	–	–	–	2	2
C_1	–	–	–	–	–	$2n$	$2n$
C_2	–	–	–	n	$2n$	–	–
C_3	–	–	–	$2n$	$3n$	–	–
C_4	n	$2n$	$3n$	–	–	–	–
$\mathsf{PK}_{\mathsf{DKG}}$	1	2	–	–	–	–	–
interpolate	–	–	–	–	–	–	–

TABLE V: New verification operations on PC or Ethereum

Part	$\text{Add}_{\mathbb{G}_1}$	$\text{Multiply}_{\mathbb{G}_1}$	ModPow	Pairing
C_0	2	2	–	–
C_1	$2n$	$3n$	–	–
C_2	n	$2n$	–	n
C_3	$2n$	$3n$	–	n
C_4	n	$2n$	$3n$	–
$\mathsf{PK}_{\mathsf{DKG}}$	1	2	–	–
interpolate	–	–	–	–

We compare the verification cost of curve "bn_128" on PC in Figure 3 between Table IV and Table V when $n = 10$. We observe that Table V costs much more time than Table IV on PC, since bilinear mapping (rather than pairing check) costs significantly. To point out, Ethereum has a unique cryptographic computation model, which only supports pairing check, eliminating the possibility to compute the gas consumption in Table IV. Fortunately, we accomplish the verification on Ethereum according to Table V. Furthermore, by eliminating operations on $\mathbb{G}_2$ and $\mathbb{G}_T$, we lower the gas consumption by tens of times regarding Table III.

We estimate the calculation cost in smart contract to show the practicality of our protocol. Smart contract checks the validity of each ciphertext, no matter how many malicious parties there are. Accordingly, we have no worst gas cost, unlike Schindler et al. [9]. We set up the whole system with $t = n/2 + 1$, where t is the threshold in CP-ABE usage. The contract deployment costs 3,229,937 gas. Figure 4 gives an estimation of gas consumption in verifying different parts of a ciphertext. We see that C_2 and C_3 are the most costly parts. From Table V and Figure 4, we conclude that "Paring" is more costly than "Add", "Multiply" and "ModPow" on Ethereum, which keeps consistency with Table III. The *interpolate* algo-

[5]GMP: https://gmplib.org/, Accessed: 2021-11-13
[6]PBC: https://crypto.stanford.edu/pbc/, Accessed: 2021-11-13

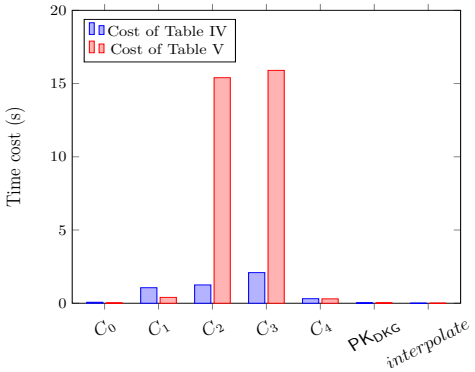

Fig. 3: Verification cost (s) comparison on PC when n=10

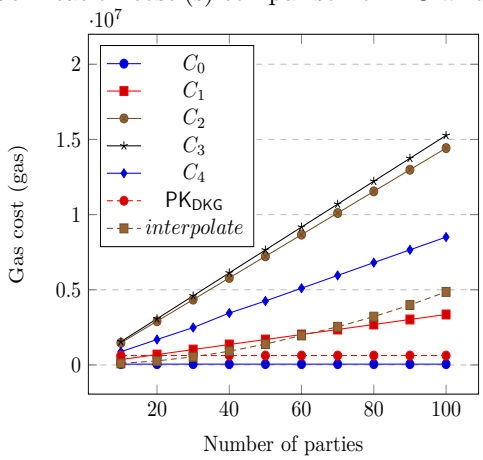

Fig. 4: Verification cost (gas) of different parts

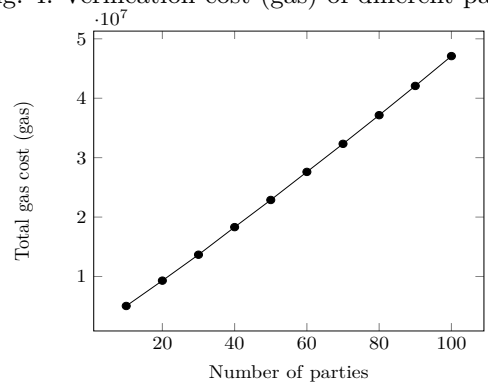

Fig. 5: Total gas cost cost when increasing parties

rithm costs little time on PC, which is shown in Figure 3. Due to the high cost of multiplication on big numbers in solidity, *interpolate* algorithm costs 4,859,768wei when $n = 100$ on Ethereum. Figure 5 shows that the total gas consumption increases linearly with the numbers of parties. Figure 5 implies that individual gas cost is 46 million when $n = 100$. As a comparison, the verification complexity of Schindler et al. protocol [9] is $O(n)$ and gas consumption is less than our implementation. However, it trades round complexity for verification cost—it is not 1-round in sharing phase. Furthermore, their encrypted shares are not publicly verifiable. But indeed, we can further lower the verification cost to $O(n \log n)$ by replace the sigma-protocol-style zero-knowledge proofs with polynomial commitment schemes [41], which we leave for future work.

We lower the computation cost to $O(n)$ with batch decryption in the *reconstruction* phase, detailed in Section VII-E. Figure 6 shows the experimental results of curve "bn_128" on PC that multiple decryptions (one by one) costs $O(n^2)$ time with the number of parties, while batch decryption costs linearly with the number of parties.

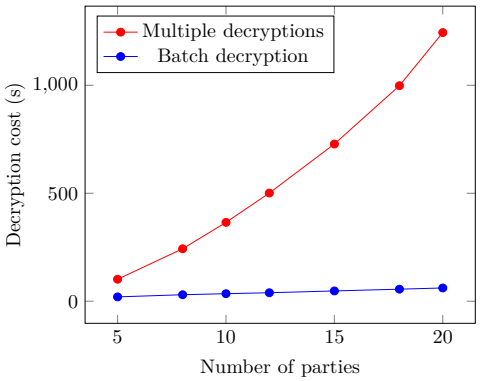

Fig. 6: Cost of multiple decryptions v.s. batch decryption

IX. Security Analysis

In this section, we show that our protocol satisfies the requirements of a secure DKG defined in Section IV-A. We show it formally in our main theorem.

Theorem 1 (Main Theorem)**.** *Given global functionality $\mathcal{G}_{\mathsf{verify}}$, there exists a secure distributed key generation protocol Π under rushing adaptive adversary $\mathcal{A}$.*

Proof. With Claim 1 to Claim 6, by Definition 6 we have this theorem stand.

Lemma 4. *The party P_θ could commit an invalid ciphertext in the sharing phase, but it can be discovered when verifying Equations* (4) *in smart contract.*

Proof. The malicious parties could do anything to deviate our protocol. A malicious one could provide forged ciphertext C, fake proofs or $\mathsf{PK}_{\mathsf{DKG}}$ in the *sharing* phase. However, after submitting the C, proofs and $\mathsf{PK}_{\mathsf{DKG}}$ to blockchain, the smart contract will throw exceptions and the state of blockchain will fall back when verifying the equations in (4).

Claim 1 (**Correctness(C1)**)**.** *All honest shares (from $\mathcal{Q}$) define the unique DKG secret key* MSK.

Proof. By Lemma 3, we prove that malicious parties cannot pass as honest parties. In addition, an adversary controlling the f parties $\mathcal{M}_1$ cannot impact the behaviors of honest parties due to the independence of each party. By Lemma 4, our protocol can detect whether a ciphertext is valid or not in the smart contract. Parties, submitting valid ciphertexts, form the qualified set $\mathcal{Q}$ ($|\mathcal{Q}| \geq n/2+1$). All ciphertexts $\{C_\theta\}_{\theta \in \mathcal{Q}}$ (and the corresponding plaintexts

$\{M_\theta\}_{\theta\in\mathcal{Q}}$) are tamper-resistant at the end of *sharing* phase. Thus, the shared secret key MSK is unique and defined by $\sum_{\theta\in\mathcal{Q}} \gamma(M_\theta) + \mathsf{S}_r$.

Claim 2 (**Correctness(C1)'**). *There is an efficient process for honest parties to output the secret key* MSK.

Proof. All valid ciphertexts have the same threshold access tree in the encryption algorithm (refer to Section VII-B). An external user can decrypt all ciphertexts in $\mathcal{Q}$ if collecting $t = n/2 + 1$ honest decryption keys (refer to Section VII-D). We also improve the efficiency by incorporating batch decryption (refer to Section VII-E). Thus, it is efficient to output the secret key MSK.

Claim 3 (**Correctness(C2)**). *All qualified parties obtain the public key* $\mathsf{MPK} = h^{\mathsf{MSK}}$, *where* MSK *is the unique secret key guaranteed by Claim 1.*

Proof. The public key MPK is calculated as $\mathsf{MPK} = h^{\mathsf{MSK}} = h^{\sum_{\theta\in\mathcal{Q}} \gamma(M_\theta)+\mathsf{S}_r} = \prod_{\theta\in\mathcal{Q}} h^{\gamma(M_\theta)} \cdot h^{\mathsf{S}_r}$. $\mathsf{PK}_{\mathsf{DKG},\theta} = h^{\gamma(M_\theta)}$ and h^{S_r} are public and determined at the end of *sharing* phase and the validity of these values are verified by the verifying contract according to global functionality $\mathcal{G}_{\mathsf{verify}}$. Therefore, all parties obtains the same value MPK determined by the qualified parties $\mathcal{Q}$ and a publicly verifiable random value h^{S_r}.

Claim 4 (**Correctness(C3)**). *The secret key* MSK *is uniformly distributed, and then* MPK *is uniformly distributed that is generated by* h.

Proof. Only parties who have submitted well-constructed ciphertexts are qualified, denoted as $\mathcal{Q}$ ($\mathcal{P} - \mathcal{M}_1 \subseteq \mathcal{Q} \subseteq \mathcal{P}$). $\mathcal{Q}$ is tamper-resistant at the end of *sharing* phase, i.e., when timeout event is triggered by global functionality $\mathcal{G}_{\mathsf{verify}}$ or n well-constructed CP-ABE ciphertexts are collected. We assume *adaptive* adversary model, where a party P_i in $\mathcal{Q}$ may behave dishonestly in the *reconstruction* phase. In this case, M_i can be recovered by at least t honest parties (i.e., $\mathcal{P} - \mathcal{M}_2$) as described in Section VII-D. The secret key MSK is defined by summing $\{\mathsf{SK}_{\mathsf{DKG},\theta}\}$ up, where M_θ is randomly chosen by party P_θ. We eliminate the bias by incorporating unpredictable but publicly verifiable entropy S_r from randomness beacons when a dishonest party $P_\theta \in \mathcal{Q}$ does not choose M_θ randomly or when there is a rushing adversary. Thus, $\mathsf{MSK} = \sum_{\theta\in\mathcal{Q}} \mathsf{SK}_{\mathsf{DKG},\theta} + \mathsf{S}_r$ is uniformly distributed in $\mathbb{G}_T$. As a consequence, MPK is calculated as $\prod_{\theta\in\mathcal{Q}} h^{\mathsf{SK}_{\mathsf{DKG},\theta}} \cdot h^{\mathsf{S}_r}$ in the *sharing* phase, is also uniformly distributed generated by h.

Claim 5 (**Secrecy**). *The secret key* MSK *is confidential to anyone or any* f *colluding group.*

Proof. The secret key MSK is defined as the sum of random secret values $\gamma(M_\theta)$. Malicious parties have to acquire all the M_θ by decrypting all the ciphertexts from $\mathcal{Q}$. However, $|\mathcal{Q}| - |\mathcal{M}_1| \geq t - f > 0$. Thus, obtaining all $\{M_\theta\}_{\theta\in\mathcal{Q}}$ amounts to break the cryptosystem of decentralized CP-ABE and solve discrete logarithm problem. By Lemma 1, the possibility that colluding or malicious adversaries gain useful information regarding an honest ciphertext is negligible. Therefore, the secret key MSK is confidential to anyone or any $f(< n/2)$ colluding group (i.e., $\mathcal{M}_1$ in the *sharing* phase) before *reconstruction*.

Lemma 5. *Our protocol allows adaptive adversaries, meaning the adversaries could be different sets in the generating* MPK *phase and in the reconstructing* MSK *phase, under the assumption of* $n/2 + 1$ *honest parties.*

Proof. By Lemma 2, each party can prove the validity of its decryption key with an associated ciphertext. As long as t honest decryption keys are combined, the ciphertexts of honest parties $\mathcal{Q}$ can be decrypted and MSK is guaranteed to be obtained. Parties invoke *Encrypt* algorithm in constructing MPK as encryptors and inovke *KeyGen* in recovering MSK as attribute authorities. The two algorithms are invoked independently and the t honest parties, who guarantee the availability, could be different in each algorithm. Therefore, we allow *adaptive* adversaries in generating MPK and recovering MSK.

Claim 6 (**Robustness**). *Even with* f *malicious parties, any* $t > n/2$ *honest parties could calculate the private key* MSK *efficiently.*

Proof. We have assumed that there exists $f < n/2$ malicious nodes (i.e., $\mathcal{M}_2$ in the *reconstruction* phase), meaning the number of the honest parties is greater than the threshold t, i.e., $n - f >= t$. All the ciphertexts have the same threshold access tree "t of $(\mathsf{GID@PID}_1, \mathsf{GID@PID}_2, ..., \mathsf{GID@PID}_n)$", that is used in encryption algorithm (see more in Section VII-A). The threshold value t is set to $n/2 + 1$ ($> f$). In Section VII-D, we demonstrate that by invoking *KeyGen* and *Decrypt* algorithms, parties can calculate MSK efficiently. By Lemma 2, we prove that our protocol keeps robustness with $f(< n/2)$ malicious parties. In addition, Lemma 5 demonstrates the f malicious parties could be *adaptive*.

Claim 7 (**Liveness**). *No* f *adversaries could prevent the honest parties to complete the protocol successfully.*

Proof. In global functionality $\mathcal{G}_{\mathsf{verify}}$, we assume that the *sharing* phase ends before a fixed amount of time (i.e., $\Delta\tau$) passes. The set of qualified parties $\mathcal{Q}$ who have submitted well-constructed ciphertexts is determined at the end of the *sharing* phase. Eventually, the protocol will produce a DKG public key MPK. By Lemma 4, our protocol keeps running with f malicious parties who may submit forged ciphertexts. By Lemma 2, we prove that f malicious parties cannot prevent the reconstruction of the secret key MSK. Therefore, our protocol has the property of liveness.

In summary, by Claim 1, it can be inferred that the adversary $\mathcal{A}$ controlling $\mathcal{M}_1$ cannot prevent the protocol

from obtaining the qualified parties $\mathcal{Q}$. Thus, $\mathcal{A}$ has negligible advantage in **Case 1** (refer to Section IV-C) to abort the *sharing* phase. By Claim 4, we prove that an entropy S_r from randomness beacons helps resolve rushing adversary attack. Hence, $\mathcal{A}$ is unable to bias the output of *sharing* phase in **Case 1**. By Claim 5, we prove that MSK keeps private to anyone even though $\mathcal{A}$ controls f parties $\mathcal{M}_1$. Therefore, $\mathcal{A}$ will fail to obtain information about MSK in **Case 2**. By Claim 3 and Claim 6, we prove that MSK can be recovered effectively without the participation of the f parties $\mathcal{M}_2$ (i.e., passive adversaries). Further, by Lemma 2, Lemma 5 and Claim 7, we prove that MSK can be recovered even the f adversaries publish "bad" decryption (i.e., active adversaries). Thus, $\mathcal{A}$ gains no advantage in **Case 3**.

X. Conclusion

In this paper, we propose the first adaptively secure 1-round DKG protocol with efficient delegatable reconstruction. To make our protocol non-interactive, we incorporate a sigma-protocol-style NIZK proofs for CP-ABE ciphertext. With smart contract verifying the proofs, we eliminate the *dispute* round. As a consequence, we only need one round to generate a distributed public key. Compared with the existing 1-round DKG protocol, we assume an *adaptive* adversary model and achieve only $O(n)$ communication complexity in *reconstruction*. Also, we do not require any space for parties to store ciphertext. These advantages originate from the use of the decentralized CP-ABE scheme, which provides long-term key pairs and enables external decryption. Besides, we lower the computation complexity to $O(n)$ by introducing batch decryption. Moreover, we give concrete experimental results to demonstrate the verification cost, taking Ethereum smart contract to implement our system. In the future, we plan to further research on cryptographic operations on Ethereum while respecting its unique computational model and lower the verification cost as described in Section VIII-C.

References

[1] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," in *Proceedings of Crypto*, California, USA, 1989, pp. 307-315.

[2] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold DSS signatures," in *Proceedings of EUROCRYPT*, Saragossa, Spain, 1996, pp. 354-371.

[3] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Proceedings of ASIACRYPT*, Gold Coast, Australia, 2001, pp. 514–532.

[4] D. Boneh and M.K. Franklin, "Identity-based encryption from the Weil pairing. in *Proceedings of Crypto*, California, USA, 2001, pp. 213–229.

[5] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series: Consensus system," [Online]. Available: http://arxiv.org/abs/1805.04548, 2018.

[6] T.P. Pedersen, "A threshold cryptosystem without a trusted party," in *Proceedings of EUROCRYPT*, Brighton, UK, 1991, pp. 221-242.

[7] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Journal of Cryptology*, vol. 20, pp. 51–83, May 2006.

[8] W. Neji, K. Blibech, and N.B. Rajeb, "Distributed key generation protocol with a new complaint management strategy," *Security and communication networks*, vol. 9, no. 17, pp. 4585–4595, Nov. 2016.

[9] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "Distributed key generation with Ethereum smart contracts," [Online]. Available: https://eprint.iacr.org/2019/985, 2019.

[10] A. Kate and I. Goldberg, "Distributed key generation for the internet," in *Proceedings of 29th IEEE International Conference on Distributed Computing Systems*, Montreal, QC, 2009, pp. 119–128.

[11] A. Kate, Y. Huang, and I. Goldberg, "Distributed key generation in the wild," [Online]. Available: https://eprint.iacr.org/2012/377, 2019.

[12] E. Kokoris-Kogias, A. Spiegelman, and D. Malkhi, "Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures," in *Proceedings of ACM CCS*, Virtual Event USA, 2020, pp. 1751–1767.

[13] P.A. Fouque and J. Stern, "One Round Threshold Discrete-Log Key Generation without Private Channels," in *Proceedings of PKC*, Cheju Island, Korea, 2001, pp. 300-316.

[14] S. Rafaeli, The electronic bulletin board: A computer-driven mass medium. *Computers and the Social Sciences*, 1984, pp. 123-136.

[15] S.S.M. Chow, C. Ma, and J. Weng, "Zero-Knowledge Argument for Simultaneous Discrete Logarithms," *Algorithmica*, vol. 64, pp. 246–266, Nov. 2011.

[16] P. Feldman, "A Practical Scheme for Non-interactive Verifiable Secret Sharing," in *Proceedings of Foundations of Computer Science*, Washington DC, USA, 1987, pp. 427–438.

[17] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf, 2017.

[18] A. Tomescu, R. Chen, Y. Zheng, I. Abraham, B. Pinkas, G.G. Gueta, and S. Devadas, "Towards scalable threshold cryptosystems." in *Proceedings of IEEE Symposium on Security and Privacy*, CA, USA, 2020, pp. 877-893.

[19] M. Stadler, "Publicly verifiable secret sharing," in *Proceedings of EUROCRYPT*, Saragossa, Spain, 1996, pp. 190–199.

[20] B. Schoenmakers, "A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting," in *Proceedings of CRYPTO*, California, USA, 1999, pp. 148-164.

[21] I. Cascudo and B. David, "SCRAPE: Scalable randomness attested by public entities," in *Proceedings of ACNS*, Kanazawa, Japan, 2017, pp. 537–556.

[22] R. Pass, L. Seeman and A. Shelat, "Analysis of the Blockchain Protocol in Asynchronous Networks," in *EUROCRYPT 2017*, Paris, France, 2017, pp. 643-673.

[23] J. Garay, A. Kiayias, N. Leonardos, "The bitcoin backbone protocol: analysis and applications," In *EUROCRYPT 2015*, 2015, pp. 281–310.

[24] A. Kosba, A. Miller, E. Shi, Z. Wen and C. Papamanthou, "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts," in *2016 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, 2016, pp. 839-858.

[25] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Cryptology - EUROCRYPT'99*, Prague, Czech Republic, 1999, pp. 223-238.

[26] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, "Adaptive Security for Threshold Cryptosystems," *CRYPTO' 99*, California, USA, 1999, 98-116.

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

[28] Z. Liu, Z. Cao, and D.S. Wong, "Efficient Generation of Linear Secret Sharing Scheme Matrices from Threshold Access Trees," [Online]. Available: https://eprint.iacr.org/2010/374, 2010.

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

[30] Y. Rouselakis and B. Waters, "Effcient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption," in *Proceedings of FC 2015 (LNCS)*, San Juan, Puerto Rico, 2015, pp. 315–332.

[31] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proceedings of EUROCRYPT*, Tallinn, Estonia, 2011, pp. 568–588.
[32] J. Berrut and L. Trefethen, "Barycentric Lagrange Interpolation," *In SIAM Review*, vol. 46, pp. 501–517, 2004.
[33] T. Okamoto, "Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes," in *Proceedings of Crypto*, California, USA, 1992, pp. 31–53.
[34] C. Schnorr, "Efficient Signature Generation by Smart Cards," *Journal of Cryptology*, vol. 4, pp. 161–174, Jan. 1991.
[35] D. Boneh and V. Shoup, "A graduate course in applied cryptography," [Online]. Available: http://toc.cryptobook.us/book.pdf, version 0.5, 2020.
[36] M. Ciampi, G. Persiano, L. Siniscalchi and I. Visconti, "A Transform for NIZK Almost as Efficient and General as the Fiat-Shamir Transform Without Programmable Random Oracles," In *Theory of Cryptography, TCC*, Tel Aviv, Israel, 2016, pp. 83-111.
[37] M. Al-Bassam, "Implementation of elliptic curve operations on G2 for alt_bn128 in Solidity," [Online]. Available: https://github.com/musalbas/solidity-BN256G2, 2020.
[38] P.S.L.M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Proceedings of Selected Areas in Cryptography (SAC)*, ON, Canada, 2006, pp. 319-331.
[39] V. Buterin and C. Reitwiessner, "EIP-197: Precompiled contracts for optimal Ate pairing check on the elliptic curve alt_bn128," [Online]. Available: https://eips.ethereum.org/EIPS/eip-197, 2017.
[40] C. Reitwiessner, "EIP-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128," [Online]. Available: https://eips.ethereum.org/EIPS/eip-196, 2017.
[41] B. Bünz, B. Fisch, and A. Szepieniec. "Transparent SNARKs from DARK compilers." in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, Cham, 2020, pp. 677-706.
[42] J. Bonneau, J. Clark, and S. Goldfeder. "On Bitcoin as a public randomness source." [Online]. Available: https://eprint.iacr.org/2015/1015, 2015.
[43] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. "Verifiable delay functions." in *Advances in Cryptology – CRYPTO 2018. CRYPTO 2018*, Santa Barbara, CA, USA, 2018, pp. 757-788.