Efficient online and Non-Interactive Threshold Signatures with Identifiable Aborts for Identity-Based Signatures in the IEEE P1363 Standard

Yan Jiang[1], Youwen Zhu[a, 1,2], Jian Wang[1] and Yudi Zhang[3]

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
[2] Key Laboratory of Data Protection and Intelligent Management, Sichuan University, Chengdu, China
[3] School of Computing and Information Technology, University of Wollongong, Wollongong, Australia

Abstract. Identity-based threshold signature (IDTS) enables the generation of valid signatures without revealing cryptographic keys in the signing process. While current protocols have achieved much progress in their efficiency, many schemes easily suffer from denial-of-service attacks in which misbehaving parties could keep from generating signatures without being caught. The identifiable abort property is designed to withstand such an attack in some recent IDTS protocols. However, all these schemes require many rounds of interaction for the resulting signature or utilize cryptographic techniques, which have a high complexity. In this study, we put forward a novel IDTS protocol that can achieve identifiable abort and resist arbitrary collusion attacks. Precisely, this ensures that corrupted parties are responsible in case of failure and cannot jointly obtain the input of honest parties. Moreover, we present the ideal IDTS functionality and provide the provable security of the proposed protocol with the global random oracle model. Our scheme has non-interactive signing, compatibility with the offline/online settings, and practical efficiency for the online phase. Finally, we give theoretical analyses and experimental results of our solution, showing that the signing time is less than two milliseconds and that the scheme is suitable for resource-constrained settings.

Keywords: Threshold signatures · Identity-based signatures · Non-interacting online signing · Identifiable aborts

1 Introduction

Recent advances in blockchain technology and the 5G network have led to the growth of resource-constrained but extremely capable Internet of Things (IoT) devices [ZCS23], including sensors, smartphones, and intelligent appliances. According to a prediction by International Data Corporation (IDC), the number of IoT devices will reach 55.7 billion, and these devices are expected to produce nearly 80 zettabytes (ZBs) of data in 2025 [IDC21]. However, these devices are vulnerable to various attacks over wireless communication, such as hijacking attacks, message tampering, and user impersonation.

E-mail: y.jiang@nuaa.edu.cn (Yan Jiang), zhuyw@nuaa.edu.cn (Youwen Zhu), wangjian@nuaa.edu.cn (Jian Wang), zhangyudi007@gmail.com (Yudi Zhang)

[a]The corresponding author is Youwen Zhu.

Providing secure and efficient authentication before utilizing the data is the primary task that IoT technologies must solve.

Digital signatures are a promising approach that offers data authenticity and integrity. Unfortunately, an inherent limitation is that certificate management and issue in a typical public-key infrastructure are costly. To address this problem, the definition of identity-based cryptography introduced by Shamir [Sha84] can simply treat email addresses or phone numbers as personal public keys, and later the work of Boneh and Franklin [BF01] realized the concept in bilinear groups. Since then, identity-based signature (IBS) has been studied [CHC03, Hes02, BLMQ05, PS06] and standardized [ISO18, IEE13]. The identity-based cryptographic techniques using pairings [BLMQ05] (denoted as BLMQ) are provided in the IEEE P1363 standard.

In traditional BLMQ schemes, the storage of the keys in one location represents single points of failure [TANBDV20]. The reason is that the theft or loss of the keys can be disastrous, and the attacker can obtain the signing key in use by users to access services or spend one's money in cryptocurrencies [Lin17]. Therefore, the main challenge is to store these keys in a way that is both easy to use and resistant to theft and loss.

Threshold cryptography [Des94] has gained a way to address this problem. The master key and signing key are distributed among multiply servers/devices, so that if the sets of parties are greater than or equal to security thresholds, one can obtain the resulting productions without revealing secret about them. A threshold BLMQ scheme has been presented by Zhang *et al.*[ZHZ$^+$18] via integrating the BLMQ algorithm within a two-party secure computation and later improved in [HZWC18, FHL$^+$20]. Such protocols allow parties holding one's share of the signing key to generate jointly the BLMQ signature, while escaping reconstruction of the key. However, these protocols easily suffer from denial-of-service attacks in the setting with dishonest majority. In this case, the swindlers could prevent honest parties from generating signatures without being caught.

More importantly, BLMQ is slightly "threshold-unfriendly" since threshold protocols for BLMQ require parties to cooperate in computing the inversion operation without knowing the entire master key. A generic way to invert a secret value is to design a secure multiplicative-to-additive conversion protocol, and then locally compute own shares of the secret value. There exist two common approaches for constructing these protocols, namely the Paillier cryptosystem [GG18] and oblivious transfer protocols [DKLS19]. However, both schemes have high computational and communication overhead, respectively.

One technique for designing efficient multiplication protocols relies on authenticated Beaver triples [DPSZ12] for doing the work, which is precomputed in the offline phase and then used to efficiently obtain the multiplication of two random values during the online phase. Unfortunately, these triples must still be enough to guarantee that the corrupted parties are detected after faults occur. This results in the valid reconstruction of prescribed values.

In this study, we present a threshold version of the IEEE P1363 standard via using authenticated multiplication triples. In particular, these triples are generated by multiple servers in a distributed manner, and are used as materials for online multiplication. The computation of the proposed protocol needs to be detectable and then resists denial of service attacks.

In brief, a summary of the contributions in this paper is provided below:

- We propose a threshold protocol for BLMQ with identifiable abort and collusion resistance, which not only collaboratively generates the resulting signatures but also provides efficient and non-interacting online signing.

- We propose a commitment-based identifiable mechanism that allows reliable players to identify misbehaving players after faults occur. Specifically, the parties commit to input shares and check corrupted parties due to the verification of commitment schemes. Further, to resist collusion attacks between the parties, we have designed two distributed protocols with different security thresholds.

- We realize an ideal identity-based threshold signature functionality in the framework of universal composability (UC) and formally prove the security of the proposed solution with the global random oracle model. And that the presented protocol withstands adaptive corruptions of parties by relying on the "single inconsistent party" technique and reliable erasure assumption.

The remainder of this paper is as follows. In Section 2, we propose a relevant description of identity-based threshold signature schemes, and we provide background information in Section 3. In Section 4, we give several ideal functionalities and security models, and we design a secure multiplication protocol required by our work in Section 5. We give specific construction and correctness, formally provide the provable secure analysis, and discuss experimental findings in Section 6. We summarize this paper in Section 7.

2 Related work

We briefly show previous works on IDTS protocols, identifiable aborts and the preprocessing model. Comparison with the related schemes is shown in table 1.

Table 1: Comparison with the related schemes

Scheme	F1	F2	F3	F4	F5	F6	F7
Baek and Zheng [BZ04]	✗	✗	✗	✗	✓	✗	–
Chen *et al.*[CZKK04]	✗	✓	✗	✗	✓	✗	–
Xiong *et al.*[XQL10]	✗	✗	✗	✗	✓	✗	–
He *et al.*[HZWC18]	✗	✗	✗	✗	✓	✗	–
Feng *et al.*[FHL+20]	✗	✗	✗	✗	✗	✗	–
Jiang *et al.*[JZWL23]	✓	✓	✗	✗	✓	✗	2
Gennaro and Goldfeder [GG20]	✓	✓	✗	✗	✓	✓	2
Canetti *et al.*[CGG+20]	✓	✓	✓	✓	✓	✓	2
Liang and Chen [LC24]	✓	✓	✓	✓	✓	✓	2
Castagnos *et al.*[CCL+23]	✓	✓	✓	✗	✓	✓	2
Abram *et al.*[ANO+22]	✓	✓	✗	✗	✗	✓	–
This work	✓	✓	✓	✓	✓	✓	1

✓: Satisfy the requirement; ✗: Not satisfy the requirement; –: Not exist the feature; F1: Fully distributed protocols; F2: Collusion resistance; F3: Adaptive corruptions; F4: UC; F5: Identifiable aborts; F6: Non-interacting online signing; F7: Rounds in identification mechanism.

Identity-based threshold signature (IDTS). The definition of IDTS was firstly presented by Baek and Zheng [BZ04], which uses the pairing techniques to construct provably secure scheme. Subsequent works [CZKK04, XQL10] later improved upon the security of their solutions, and particularly the security assumption, for which the study of Xiong *et al.*[XQL10] does not take into consideration in the setting with random oracle model. These works are viewed as the extensions of IBS [Hes02, CHC03, PS06], which are “threshold-friendly" in the distributing environment. To obtain the inversion operation of the private key, it is indeed challenging to construct a threshold version of the BLMQ scheme. He *et al.*[HZWC18] proposed secure distributed signing protocols for the two-party setting. Later, Feng *et al.*[FHL$^+$20] extended this scheme [HZWC18] to the multiparty case, which used the Elliptic curve operations to invert a secret value. Currently, Jiang *et al.*[JZWL23] has added identifiability to threshold BLMQ, obtaining new functionality in a similar setting. However, the resulting protocols neither provide the non-interacting signing property nor resist the adversary that corrupts the parties during arbitrary time.

Threshold signatures with identifiable abort. The concept of secure computation with identifiable aborts was proposed in a preprocessing model by [IOZ14], where a trusted party provides some correlated values. Recently, researchers have paid attention to threshold signatures with identifiable abort. Gennaro and Goldfeder [GG20] presented one round threshold ECDSA with the identifiability of corrupted parties, and the same property supports for the work [CGG$^+$20] to construct more efficient solutions with four communication rounds. Additionally, Liang and Chen [LC24] gave a threshold protocol for SM2 signing scheme with a similar technique described in [CGG$^+$20]. Also, Castagnos *et al.*[CCL$^+$23] presented a threshold ECDSA protocol in the precomputing model, which replaced the Paillier encryption with the class group. These works require heavy computation for identification mechanism, which involves "complicated" zero-knowledge proofs.

Threshold signature with the offline/online phase. The crucial idea of Even *et al.*[EGM89] can split the signing process into two parts such that expensive computation is able to execute in the offline setting prior to knowing the given message, and then used by the online phase where each party locally computes its signature share after the message is known. The notion of non-interacting signing for threshold ECDSA has been proposed by [GG20] and [CGG$^+$20]. Their works propose a non-interacting step that signature generation boils down to a message in the preprocessing model. In subsequent work, [LC24, CCL$^+$23] independently described how to combine the offline/online technique and identifiable aborts for threshold signature protocols where either a cheater in the online phase will be identified in multiple rounds or the last online part of the resulting signature needs to consider the conversion of decrypting ciphertext. The online step of the signing process is still burdensome and not ideal, even though it is generally practical. Another, more efficient approach with silent preprocessing was given in [ANO$^+$22] where the authors relied on pseudorandom correlation generators of [BCG$^+$19]. In this case, however, the honest players cannot detect the misbehaving players after faults occur.

3 Preliminaries

3.1 Bilinear Groups

Three circle groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, which are the same order q, have additive operation for the first two groups and multiplicative operation for the last one group. Moreover, the generators of the first two groups are P and Q, respectively. The function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is computable on the bilinear mapping and provides the properties as follows:

- Bilinear property: $e(\alpha \cdot P, \beta \cdot Q) = e(P, Q)^{\alpha\beta}$, where $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$, $\alpha, \beta \in \mathbb{Z}_q$;
- Non-degenerate property: $\exists P \in \mathbb{G}_1$, $\exists Q \in \mathbb{G}_2$, $e(P, Q) \neq 1$.
- Computable property: $e(R, S)$ is a computable value, where $R \in \mathbb{G}_1$, $S \in \mathbb{G}_2$.

3.2 BLMQ

Barreto et al. [BLMQ05] proposed the BLMQ scheme that was recognized as the IEEE P1363 standardization. This scheme is constructed on the bilinear groups (see Section 3.1 for a full description). Moreover, two hash functions H_1 and H_2, which reach the security level of cryptography, have the mapping relations $\{0,1\}^* \to \mathbb{Z}_q^*$ and $\{0,1\}^* \times \mathbb{G}_T \to \mathbb{Z}_q^*$. The BLMQ scheme consists of the four algorithms as seen below:

- $\mathsf{Setup}(1^\lambda) \to (s, Q_{pub})$: The private-public key pair is $(s, Q_{pub} = s \cdot Q)$, and the parameter params is $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, q, P, Q, g, Q_{pub}, H_1, H_2)$.
- $\mathsf{Extract}(\mathsf{params}, s, ID) \to d_{ID}$: The signing key is $d_{ID} = \dfrac{1}{H_1(ID) + s} \cdot P$.
- $\mathsf{Sign}(\mathsf{params}, d_{ID}, m) \to (h, S)$: Given the message m, compute the signature s in the following way:
  1. Select a random value r and output a temporary variable $u = g^r$.
  2. Compute the value of the hash function H_2, namely $h = H_2(m||u)$.
  3. Output the signature $S = (r + h) \cdot d_{ID}$.
- $\mathsf{Verify}(\mathsf{params}, m, ID, (h, S))$: Given the message m, the identity ID, and the signature (h, S), verify whether the signature (h, S) is valid for the message m.

For the correctness and security of the BLMQ scheme, refer to [BLMQ05].

4 Definition of security

4.1 The ideal identity-based threshold signature functionality

The functionality consists of the key generation, extracting, signing, verification, and corruption. The key generation runs only once, and the remaining functions can operate arbitrary numbers with appropriate inputs. In ideal execution, the functionality works as follows. Once all KGCs are activated, a verification algorithm $\mathcal{V}$ is supposed to be from an ideal simulator $\mathcal{S}$. The functionality records only the registered values and does not require any checks during the registration phase. Then, the functionality returns a signature σ from $\mathcal{S}$ once these devices appeal to the functionality to get a signature σ for a given message m. Moreover, the functionality stores σ in an entry to represent a valid signature for m. Finally, the functionality must check a tuple (m, ID, σ) and then outputs 0/1, denoting whether an entry (m, ID, σ) is correct or the resulting value of what $\mathcal{V}$ computes. The functionality is depicted in Fig. 1.

4.2 Security model

Universally composable framework. As shown in [Can01], we define the universally composable framework using a real and ideal execution. In practice, $\mathcal{Z}$ sends generated input and reads the output of the protocols on behalf of the external environment. A protocol π is UC-secure if it securely implements the functionality $\mathcal{F}$. This means that for $\mathcal{Z}$, an interacting of π with $\mathcal{A}$ is indistinguishable from that of $\mathcal{F}$ and $\mathcal{S}$.

In the real world, $\mathcal{Z}$ can interact with an adversary $\mathcal{A}$ and a set of parties running a protocol π. Let $REAL_{\pi,\mathcal{A}}^{\mathcal{Z}(z)}$ define the result of $\mathcal{Z}$ in a real execution. In the ideal environment, $\mathcal{Z}$ can interact with a set of parties and a simulator $\mathcal{S}$ controlling the corrupted players. Furthermore, these entities can use the functionality $\mathcal{F}$ to perform some operations on a pre-programmed pattern. Let $IDEAL_{\mathcal{F},\mathcal{S}}^{\mathcal{Z}(z)}$ define the result of $\mathcal{Z}$ in an

Ideal Identity-Based Threshold Signature Functionality $\mathcal{F}_{IBTS}$

Key-generation:

1. When a request (keygen, sid) is obtained from KGC $\mathcal{KGC}_i$, parse $sid = (\ldots, \boldsymbol{KGC})$ and $\boldsymbol{KGC} = (\mathcal{KGC}_1, \ldots, \mathcal{KGC}_l)$.
   (a) For $\mathcal{KGC}_i \in \boldsymbol{KGC}$, give $\mathcal{S}$ the request and then store the entry $(\mathsf{keygen}, sid, \mathcal{KGC}_i)$.
   (b) Else, overlook the request.
2. When all entries $(\mathsf{keygen}, sid, \ell)$ are correctly stored for $\mathcal{KGC}_\ell \in \boldsymbol{KGC}$, give $\mathcal{S}$ the message (pubkey, sid). Then, execute things in the following way:
   (a) Once $(\mathsf{pubkey}, sid, Q_{pub}, \mathcal{V})$ has been obtained, store the entry $(sid, Q_{pub}, \mathcal{V})$.
   (b) When a request (pubkey, sid) is obtained from $\mathcal{KGC}_i \in \boldsymbol{KGC}$:
      i. If the entry $(sid, Q_{pub}, \mathcal{V})$ is stored, output $(\mathsf{pubkey}, sid, Q_{pub})$ to $\mathcal{KGC}_i$.
      ii. Otherwise, the message is ignored.

Extracting:

1. When a request $(\mathsf{extract}, ssid, ID)$ is obtained from $\mathcal{D}_i$, parse $ssid = (\ldots, \boldsymbol{KGC}, \boldsymbol{D})$ as well as $\boldsymbol{D} = (\mathcal{D}_1, \ldots, \mathcal{D}_n)$.
   (a) If $\mathcal{D}_i \in \boldsymbol{D}$, give $\mathcal{S}$ the request and then store the entry $(\mathsf{extract}, ssid, ID, \mathcal{D}_i, i)$.
   (b) Else ignore the request.
2. Once a request $(\mathsf{extract}, ssid, ID, j)$ is obtained, store the entry $(\mathsf{extract}, ssid, ID, \mathcal{KGC}_j)$ if $\mathcal{KGC}_j$ is corrupted.
   Else overlook the request.
3. When all entries $(\mathsf{extract}, ssid, ID, j)$ are correctly stored for $\mathcal{D}_j \in \boldsymbol{D}$, give $\mathcal{S}$ the message $(\mathsf{extract}, ssid, ID)$. Then, execute things in the following way:
   (a) Once $(\mathsf{received}, sid, ID, \mathsf{ok}, \mathcal{C})$ has been obtained, operate below:
      i. If $\mathcal{C}$ is malicious, store $\mathcal{C}$ to take for the revealed KGC on ID.
      ii. Else store the first corrupted KGC as the revealed KGC for this private key on ID. "*Identifiability*"
   (b) Upon receiving $(\mathsf{prikey}, sid, ID)$ from $\mathcal{D}_i \in \boldsymbol{D}$, output $(\mathsf{extracted}, sid, \mathcal{D}_i)$ to $\mathcal{D}_i$.

Signing:

1. When a request $(\mathsf{sign}, ssid, m, ID)$ is obtained from $\mathcal{D}_i$, give $\mathcal{S}$ the request and then store the entry $(\mathsf{sign}, ssid, m, ID, i)$.
2. Once a request $(\mathsf{sign}, ssid, m, ID, j)$ is obtained, store the entry $(\mathsf{sign}, ssid, ID, \mathcal{D}_j)$ if $\mathcal{D}_j$ is malicious.
   Else over the request.
3. When all entries $(\mathsf{sign}, ssid, m, ID, i)$ are correctly stored for $\mathcal{D}_i \in \boldsymbol{D}$, execute things in the following way:
   (a) If one of recorded entries $(ID, \mathcal{D}_i)$ is not stored, then output an error.
   (b) Else, if not all KGCs in $\boldsymbol{KGC}$ are malicious, then send $(\mathsf{sign}, ssid, m, ID)$ to $\mathcal{S}$:
      i. When a request $(\mathsf{signature}, ssid, m, ID, \sigma, \mathcal{C})$ is gained, for $\boldsymbol{C} \subsetneq \boldsymbol{D}$:
         A. If an entry $(ssid, m, ID, \sigma, 0)$ exists, print an error message.
         B. Else, for $\mathcal{V}(m, ID, \sigma) = 1$, store the entry $(ssid, m, ID, \sigma, 1)$.
         C. Else, store $\mathcal{C}$ if $\mathcal{C}$ is malicious.
         D. Else, reveal the identity of the first malicious device. "*Identifiability*"
      ii. When a request $(\mathsf{signature}, ssid, m, ID)$ is sent from $\mathcal{D}_i \in \boldsymbol{D}$:
         A. If the entry $(ssid, m, ID, \sigma, 1)$ exists, give $\mathcal{D}_i$ the message $(\mathsf{signature}, ssid, m, ID, \sigma)$.
         B. Else return the revealed device and KGC.

Figure 1: The ideal identity-based threshold signature functionality $\mathcal{F}_{IBTS}$

Verification:

When a request $(\mathsf{sig\text{-}vrfy}, ssid, m, ID, \sigma, \mathcal{V})$ is obtained from a party $\mathcal{Q}$, give $\mathcal{S}$ the request and execute things in the following way:

(a) If an entry (m, ID, σ, β') exists, mark $\beta = \beta'$.

(b) Otherwise, if not all KGCs in $\boldsymbol{KGC}$ and devices in $\boldsymbol{D}$ are corrupted, and that ID and (m, ID) are never required, mark $\beta = 0$. "*Unforgeability*"

(c) Otherwise, mark $\beta = \mathcal{V}(m, ID, \sigma)$.

Store (m, ID, σ, β) and then give $\mathcal{Q}$ the message $(\mathsf{istrue}, ssid, m, ID, \sigma, \beta)$.

Corruption/Decorruption:

When the request $(\mathsf{corrupt}, \mathcal{KGC}_j)$ or $(\mathsf{corrupt}, \mathcal{D}_j)$ is sent from $\mathcal{S}$, record $\mathcal{KGC}_j$ or $\mathcal{D}_j$ is malicious.

Upon receiving $(\mathsf{decorrupt}, \mathcal{KGC}_j)$ or $(\mathsf{decorrupt}, \mathcal{D}_j)$ from $\mathcal{S}$:

(a) If not all KGCs or devices are corrupted, and there exists an entry that $\mathcal{KGC}_j$ or $\mathcal{D}_j$ is corrupted, then erase it.

(b) Else do nothing.

Figure 1: The ideal identity-based threshold signature functionality $\mathcal{F}_{IBTS}$ (cont.)

ideal execution. In addition, $\mathcal{Z}$ pre-assigns the security parameter λ and random input z to all entities.

Definition 1. A protocol π is said to be UC-secure if π securely implements the functionality $\mathcal{F}$. This means that for the environment $\mathcal{Z}$, the following distributions are indistinguishable below:

$$\{REAL_{\pi,\mathcal{A}}^{\mathcal{Z}(z)}(1^\lambda)\}_{z\in\{0,1\}^*} \approx \{IDEAL_{\mathcal{F},\mathcal{S}}^{\mathcal{Z}(z)}(1^\lambda)\}_{z\in\{0,1\}^*}$$

Communication model. We rely on a synchronous broadcast channel. Moreover, we also make use of point-to-point authenticated channels for associating with each party.

Adversarial model. In this work, we consider a malicious adversary who can adaptively choose a set of corrupted parties during arbitrary time. Further, the security model of the protocol enables an adversary to control up to $l-1$ KGCs and $n-1$ devices. Let l and n be the total membership for servers and devices.

4.3 Ideal functionalities

The security of the proposed protocol is proved by means of some functionalities as defined in the work [CGG$^+$20], including the global random oracle functionality $\mathcal{H}^{\mathrm{g}}$, zero-knowledge module functionality $\mathcal{F}_{\mathbf{zk}}$, and resharing triples and keys functionality $\mathcal{F}_{\mathbf{rtk}}$. Precisely, these proofs are instanced by the Fiat-Shamir heuristic [FS86] on the Σ-protocols. Then all hash values are computed by requesting the random oracle, and the related values are sent by secure channels. We present the full details of these ideal functionalities below.

The global random oracle functionality $\mathcal{H}^{\mathrm{g}}$. We use the global random oracle functionality $\mathcal{H}^{\mathrm{g}}$ to complete the security proof, formally defined in Fig. 2. For all hash requests, the answers of the random oracle model are programmed by the length q, and will output the same answer with identical parameters. All entities have access to the model in both a real life and an ideal world, including an environment $\mathcal{Z}$.

The ideal functionality $\mathcal{H}^{\mathrm{g}}$ of a global random oracle

- Upon receiving a request (query, x) from a party $\mathcal{X}$, perform as follows:
  1. If an entry (x, a) exists, give $\mathcal{X}$ the message (answer, a).
  2. Otherwise, choose $a \leftarrow \{0,1\}^q$, and record (x, a).
     Return (answer, a).

Figure 2: The ideal functionality $\mathcal{H}^{\mathrm{g}}$ of a global random oracle

The zero-knowledge module functionality $\mathcal{F}_{\mathbf{zk}}$. In order to make more efficient use of the zero-knowledge proofs, the Fiat-Shamir technique is employed to reduce the round of communication. In addition, the zero-knowledge module also needs to compute a commitment execution by comparison with the standard zero-knowledge proof. For the details see Fig. 3.

The zero-knowledge module functionality $\mathcal{F}_{\mathbf{zk}}$

- Upon receiving $(\mathsf{com}, \Pi, 1^\lambda)$, parse the request $\Pi = (\mathsf{P}_1, \dots)$ and do in the following way: select carefully τ in the specified range, return $(A = \mathsf{P}_1(\tau, 1^\lambda); \tau)$.
- Upon receiving $(\mathsf{prove}, \Pi, \mathsf{aux}, x; w, \tau)$, parse the request $\Pi = (\mathsf{P}_1, \mathsf{P}_2, \dots)$, compute $A = \mathsf{P}_1(\tau)$, $e = \mathcal{H}(\mathsf{aux}, x, A)$ as well as $z = \mathsf{P}_2(x, w, \tau, e)$, and return (A, e, z).
- Upon receiving $(\mathsf{vrfy}, \Pi, \mathsf{aux}, x, \psi)$, parse the requests $\Pi = (\dots, \mathsf{V}_2)$ as well as $\psi = (A, e', z)$. If $\mathsf{V}_2(x, A, e', z) = 1$ and $e' = \mathcal{H}(\mathsf{aux}, x, A)$, output true for a successful verification. Else if the these equations fail to pass the checks, output false.

Figure 3: The zero-knowledge module functionality $\mathcal{F}_{\mathbf{zk}}$

The resharing triples and keys functionality $\mathcal{F}_{\mathbf{rtk}}$. In the proposed protocol, we will have parties reshare multiplication triples and private keys, and then send these values to the relevant parties. Following the BLMQ scheme, the interconnection of these entities uses a secure channel to communicate with each other. Hence, an extra assumption has yet to be introduced. The full details work in Fig. 4. Note that one can also make use of verifiable secret sharing scheme with adaptive security to send these values, and we leave the method in the future direction.

The resharing triples and keys functionality $\mathcal{F}_{\mathbf{rtk}}$

Parameter: Output tuples chk_i^j.

- Upon receiving $(\text{reshare}, ssid, \ell, i, msg, \mathbb{Z}_q^*)$ from $\mathcal{P}_i$, it samples $\rho_{i,k}^1, \dots, \rho_{i,k}^{n-1} \leftarrow \mathbb{Z}_q^*$, set $\rho_{i,k}^n = y - \sum_{j=1}^{n-1} \rho_{i,k}^j$, $\mathsf{chk}_i^j = (\rho_{i,k}^j)_j$ for $j \in [n]$, where $y \in msg = \{d'_{ID_i}, (\alpha_i, a_i, b_i, c_i, \delta_i, \zeta_i, \chi_i)\}$.
- It sends $(ssid, i, \mathsf{chk}_i^j)$ to the relevant $\mathcal{P}_j$ via a secure channel.

Figure 4: The resharing triples and keys functionality $\mathcal{F}_{\mathbf{rtk}}$

5 Secure multiplication - $\mathcal{F}_{mult}$

5.1 Functionality definition

A multiplication functionality can compute the product of additive shares held by each party $\mathcal{P}_i$. Intuitively, given a_i and b_i s.t. $a = a_1 + \cdots + a_n$ and $b = b_1 + \cdots + b_n$, it can compute $c = \sum_{\ell=1}^{n} c_\ell = (\sum_{\ell=1}^{n} a_\ell) \cdot (\sum_{\ell=1}^{n} b_\ell)$, where c_i is the output values.

Beaver triples [Bea91] are a common approach that if secret sharings of the values a, b and c computed in the offline phase are such that $ab = c$ then this enables a multiplication to be efficiently computed in the online phase. In our work, we need to provide multiplication with identifiable abort. This is done by having a commitment before calculating a multiplication, and then enabling detection for causing the protocol to fail. The multiplication functionality is depicted in Fig. 5.

The Multiplication functionality $\mathcal{F}_{mult}$

Functionality $\mathcal{F}_{mult}$ works with $\mathcal{P}_1, \ldots, \mathcal{P}_k$ as follows ($k = \{l, n\}$):

- Upon receiving $(\mathsf{init}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, q, P, Q, g, \mathbf{KGC}, \mathbf{D})$ from all parties, $\mathcal{F}_{mult}$ stores $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, q, P, Q, g, \mathbf{KGC}, \mathbf{D})$. If the entry is stored, overlook the request.
- When a request $(\mathsf{trigen}, ssid, \ell, i, \alpha_i, a_i, b_i)$ is sent from a party $\mathcal{P}_i$, if $\mathcal{P}_i \in \mathbf{KGC}$, then $\mathcal{F}_{mult}$ records $(ssid, \ell, i, \alpha_i, a_i, b_i)$. Else, overlook the request.
- When all entries $(\mathsf{trigen}, ssid, \ell, j)$ are correctly stored for $\mathcal{P}_j \in \mathbf{KGC}$, $\mathcal{F}_{mult}$ computes $\alpha = \sum_{s=1}^{n} \alpha_i$, $a = \sum_{s=1}^{n} a_i$, $b = \sum_{s=1}^{n} b_i$, and sets $\delta = a\alpha$, $\chi = b\alpha$, $c = ba$. Then, $\mathcal{F}_{mult}$ stores $(ssid, \ell, \delta, \chi, c)$ and sends $(\mathsf{trigen}, ssid)$ to all parties $\mathbf{KGC}$.
- Upon receiving $(\mathsf{mult}, ssid, \ell, i, e_i, d_i)$ from a party $\mathcal{P}_i$, if $\mathcal{P}_i \in \mathbf{KGC}$ (or $\mathbf{D}$), $\mathcal{F}_{mult}$ checks that some $(ssid, \ell, \delta, \chi, c)$ has been recorded. If yes, $\mathcal{F}_{mult}$ sets $\xi = e \cdot d$ for $e = \sum_{s=1} e_s$, $d = \sum_{s=1} d_s$ and sends $(\mathsf{mult}, ssid, \xi)$ to all parties $\mathbf{KGC}$ (or $\mathbf{D}$).
- Upon receiving $(\mathsf{input}, ssid, i, r_i, \rho_i)$ from a party $\mathcal{P}_i$, if $\mathcal{P}_i \in \mathbf{D}$, then $\mathcal{F}_{mult}$ records $(ssid, i, r_i, \rho_i)$. Else, overlook the request.
- When all entries $(\mathsf{input}, ssid, \ell, j)$ are correctly stored for $\mathcal{P}_j \in \mathbf{D}$, then $\mathcal{F}_{mult}$ computes $r = \sum_{s=1}^{n} r_i$, $u = g^r$, stores $(ssid, u)$ and sends $(ssid, \mathsf{input})$ to all parties $\mathbf{D}$.

Figure 5: The Multiplication functionality $\mathcal{F}_{mult}$

5.2 Securely computing $\mathcal{F}_{mult}$

The functionality $\mathcal{F}_{mult}$ includes four subprotocols; one subprotocol to output the key pair, one subprotocol to obtain the correlated random values when the message is not known, one subprotocol to generate the multiplication triple which is applied to efficiently compute in the online phase, and finally, one for computing the product of the related values once user identity or the message is given.

Init. In order to obtain the BLMQ public key Q_{pub} as well as Paillier key pairs $(N_i, p_i, q_i)_{i \in [l]}$, each party $\mathcal{P}_i$ invokes the init procedure and outputs (s_i, p_i, q_i). Moreover, $\sum_{\ell=1}^{l} s_\ell = s$ and $Q_{pub} = s \cdot Q$. Additionally, the parties also collaborate with each other on a random value X, which can be used for a public generator in the Pedersen commitment. Let (A, e, z) denote three messages of the Σ-protocols. Here, we require the parties to commit to $Q_i = s_i \cdot Q$ and A such that the simulator can extract the witness in security proof. The subprotocol works as depicted in Fig. 6.

Input. The Input procedure is to generate a BLMQ nonce u, where each party holds shares $r_1, \ldots, r_n$ of r so that $\sum_{\ell=1}^{n} r_\ell = r$ and $u = g^r$. In addition, every party also

Initialization subprotocol of $\mathcal{F}_{mult}$

Once obtaining a request $(\mathsf{init}, sid, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, q, P, Q, g, \mathbf{KGC}, \mathbf{D}, l)$, every player $\mathcal{P}_i$ executes below:

Round 1:

(a) $\mathcal{P}_i$ samples $s_i \leftarrow \mathbb{Z}_q^*$ and set $Q_i = s_i \cdot Q$. Then, it samples $(A_i, \tau) \leftarrow \mathcal{F}_{\mathbf{zk}}(\mathsf{com}, \Pi^{\mathsf{sch}})$.

(b) $\mathcal{P}_i$ samples $x_i \leftarrow \mathbb{Z}_q^*$ and sets $X_i = x_i \cdot P$. Then, it samples $(B_i, \hat{\tau}) \leftarrow \mathcal{F}_{\mathbf{zk}}(\mathsf{com}, \Pi^{\mathsf{sch}})$.

(c) $\mathcal{P}_i$ samples two 2λ-bit primes p_i', q_i' and satisfies $1 + 2p_i' = p_i$ and $1 + 2q_i' = q_i$. Then, it sets $N_i = p_i q_i$.

(d) $\mathcal{P}_i$ samples rid_i, u_i and computes a hash value $V_i = \mathcal{H}(sid, i, rid_i, Q_i, X_i, A_i, B_i, N_i, u_i)$.

(e) $\mathcal{P}_i$ broadcasts the message (sid, i, V_i).

Round 2:

Once all requests (sid, j, V_j) are obtained, $\mathcal{P}_i$ sends $(sid, i, rid_i, Q_i, X_i, A_i, B_i, N_i, u_i)$ to each party.

Round 3:

(a) When a request $(sid, j, rid_j, Q_j, X_j, A_j, B_j, N_j, u_j)$ is notified, $\mathcal{P}_i$ verifies:

i. $N_j \geq 2^{8\lambda}$.

ii. $\mathcal{H}(sid, j, rid_j, Q_j, X_j, A_j, B_j, N_j, u_j) = V_j$.

(b) $\mathcal{P}_i$ sets $rid = \oplus_j rid_j$, and computes:

i. $\psi_i = \mathcal{F}_{\mathbf{zk}}(\mathsf{prove}, \Pi^{\mathsf{sch}}, (sid, i, rid), Q_i; s_i, \tau)$.

ii. $\hat{\psi}_i = \mathcal{F}_{\mathbf{zk}}(\mathsf{prove}, \Pi^{\mathsf{gen}}, (sid, i, rid), N_i; (p_i, q_i))$

iii. $\tilde{\psi}_i = \mathcal{F}_{\mathbf{zk}}(\mathsf{prove}, \Pi^{\mathsf{sch}}, (sid, i, rid), X_i; x_i, \hat{\tau})$.

(c) $\mathcal{P}_i$ returns $(sid, i, \psi_i, \hat{\psi}_i, \tilde{\psi}_i)$.

Output:

(a) When a request $(sid, j, \psi_j, \hat{\psi}_j, \tilde{\psi}_j)$ is obtained, $\mathcal{P}_i$ interprets $\psi_j = (\overline{A}_j, \dots)$, $\tilde{\psi}_i = (\overline{B}_j, \dots)$, and verifies:

i. If $\overline{A}_j$ is equal to A_j, $\mathcal{F}_{\mathbf{zk}}(\mathsf{vrfy}, \Pi^{\mathsf{sch}}, (sid, j, rid), Q_j, \psi_j)$ outputs true.

ii. $\mathcal{F}_{\mathbf{zk}}(\mathsf{vrfy}, \Pi^{\mathsf{gen}}, (sid, j, rid), N_j, \phi_j) = 1$.

iii. $\overline{B}_j = B_j$, $\mathcal{F}_{\mathbf{zk}}(\mathsf{vrfy}, \Pi^{\mathsf{sch}}, (sid, j, rid), X_j, \tilde{\psi}_j) = 1$.

(b) $\mathcal{P}_i$ outputs $Q_{pub} = \prod_j Q_j$, $X = \prod_j X_j$ and $\boldsymbol{N} = (N_j)_j$.

Error. When one of these verification fails, report the cheater and abort.

Stored State. Store the following: rid, $\boldsymbol{N} = (N_1, \dots, N_l)$, X, and s_i, x_i, p_i, q_i.

Figure 6: Initialization subprotocol of $\mathcal{F}_{mult}$

computes and stores their ephemeral nonce ρ_i, and all auxiliary values $\rho_1 \cdot P, \dots, \rho_n \cdot P$, which will be used to detect cheaters during signing generation. The details are provided in Fig. 7.

TriGen. The TriGen procedure is to generate multiplication triples with the help of a multiplication-to-addition (MTA) conversion and Paillier encryption[GGI19]. Such a triple consists of random values (a_i, b_i, c_i) such that $ab = c$. The values are authenticated using a linear MAC scheme from [DPSZ12] to prevent cheating. Moreover, these MACs are verified without reconstructing the MAC key. In order to see the reason for causing the subprotocol to fail, the players are required to reveal the plaintext of the related encryptions.

Input subprotocol of $\mathcal{F}_{mult}$

Once getting a request $(\mathsf{input}, ssid, i, r_i, \rho_i)$ for $\mathcal{P}_i \in \mathbf{D}$, each party $\mathcal{P}_i$ implements below:

Round 1:

(a) $\mathcal{P}_i$ samples $r_i \leftarrow \mathbb{Z}_q^*$ and set $R_i = g^{r_i}$. Then, it samples $(A_i, \tau) \leftarrow \mathcal{F}_{\mathbf{zk}}(\mathsf{com}, \Pi^{\mathsf{sch}})$.

(b) $\mathcal{P}_i$ samples a random value $\rho_i \leftarrow \mathbb{Z}_q^*$ to set $Y_i = \rho_i \cdot P$. Then, it samples $(B_i, \hat{\tau}) \leftarrow \mathcal{F}_{\mathbf{zk}}(\mathsf{com}, \Pi^{\mathsf{sch}})$.

(c) $\mathcal{P}_i$ samples rid'_i, ς_i, and set a hash value $V_i = \mathcal{H}(ssid, i, rid'_i, R_i, Y_i, A_i, B_i, \varsigma_i)$.

(d) $\mathcal{P}_i$ broadcasts the message $(ssid, i, V_i)$.

Round 2:

Once all requests $(ssid, j, V_j)$ are notified, $\mathcal{P}_i$ sends $(ssid, i, rid'_i, R_i, Y_i, A_i, B_i, \varsigma_i)$ to every party.

Round 3:

(a) When a request $(ssid, j, rid'_j, R_j, Y_j, A_j, B_j, \varsigma_j)$ is obtained from $\mathcal{P}_j$, $\mathcal{P}_i$ verifies:

  i. $\mathcal{H}(ssid, j, rid'_j, R_j, Y_j, A_j, B_j, \varsigma_j) = V_j$.

(b) $\mathcal{P}_i$ sets $rid' = \oplus_j rid'_j$, and computes:

  i. $\psi_i = \mathcal{F}_{\mathbf{zk}}(\mathsf{prove}, \Pi^{\mathsf{sch}}, (ssid, i, rid'), R_i; r_i, \tau)$.

  ii. $\hat{\psi}_i = \mathcal{F}_{\mathbf{zk}}(\mathsf{prove}, \Pi^{\mathsf{sch}}, (ssid, i, rid'), Y_i; \rho_i, \hat{\tau})$.

(c) $\mathcal{P}_i$ sends the message $(ssid, i, \psi_i, \hat{\psi}_i)$.

Output:

(a) When a request $(ssid, j, \psi_j, \hat{\psi}_j)$ is notified, $\mathcal{P}_i$ interprets $(\psi_j = (\overline{A}_j, \dots)$, $\hat{\psi}_i = (\overline{B}_j, \dots)$, and verifies:

  i. If $\overline{A}_j$ is equal to A_j, $\mathcal{F}_{\mathbf{zk}}(\mathsf{vrfy}, \Pi^{\mathsf{sch}}, (ssid, j, rid'), R_j, \psi_j)$ outputs true.

  ii. If $\overline{B}_j$ is equal to B_j, $\mathcal{F}_{\mathbf{zk}}(\mathsf{vrfy}, \Pi^{\mathsf{sch}}, (ssid, j, rid'), Y_j, \hat{\psi}_j)$ outputs true.

(b) $\mathcal{P}_i$ outputs $u = \prod_j R_j = g^r$ and $\boldsymbol{Y} = (Y_1, \dots, Y_n)$.

Error. When one of these verification fails, report the cheater and abort.

Stored State. Store the following: rid', $\boldsymbol{Y} = (Y_1, \dots, Y_n)$, and u, r_i, ρ_i.

Figure 7: Input subprotocol of $\mathcal{F}_{mult}$

Then, they verify whether the prescribed equations are correct on the plaintext and the proven public values. In addition, the BLMQ scheme and Paillier encryption work on different groups, and thus each party is required to prove their inputs in a certain range via a zero-knowledge proof. Here, we get around the proofs of the complex statements by switching to prove the discrete logarithm problem of these inputs which is efficiently computable. The global view of the subprotocol is presented in Fig. 8.

Mult The multiplication procedure is the core content to realize the functionality $\mathcal{F}_{mult}$. After the completion of TriGen phase, each party $\mathcal{P}_i$ obtains triples (a_i, b_i, c_i) as well as shares of the MACs on these values. Thus, $\mathcal{P}_i$ can use the triples on random values e_i, d_i to generate an additive share of $e \cdot d$. In addition, each party verifies the correctness of $e \cdot d$ that will rely on whether the sum of the prescribed MACs is 0. Unfortunately, the parties cannot detect which party is to be blamed if the verification fails. We address this by having every party compute a Pedersen commitment on the additive share; the parties then involve calls to hash function and publish the decommitment values. The computation of these commitments is straightforward according to these decommitment

TriGen subprotocol of $\mathcal{F}_{mult}$

Upon input $(\mathsf{TriGen}, ssid, \ell, i, \alpha_i, a_i, b_i)$ to $\mathcal{P}_i$, where $ssid = (\dots, sid, rid, rid', \boldsymbol{N}, X, \boldsymbol{Y})$, each party $\mathcal{P}_i$ executes below:

Round 1:

(a) $\mathcal{P}_i$ samples $\rho_i, \varrho_i, k_i \leftarrow \mathbb{Z}^*_{N_i}$, and sets $K_i = \mathsf{enc}_i(\alpha_i; \rho_i)$, $\hat{K}_i = \mathsf{enc}_i(a_i; \varrho_i)$, $\tilde{K}_i = \mathsf{enc}_i(b_i; k_i)$.

(b) $\mathcal{P}_i$ selects $\eta_i \leftarrow \{0,1\}^\lambda$, and sets $\Gamma_i = \alpha_i \cdot P$, $\hat{\Gamma}_i = a_i \cdot P$, $\tilde{\Gamma}_i = b_i \cdot P$, $G_i = \mathcal{H}(ssid, i, \Gamma_i, \hat{\Gamma}_i, \tilde{\Gamma}_i, \eta_i)$.

(c) $\mathcal{P}_i$ broadcasts $(ssid, i, K_i, \hat{K}_i, \tilde{K}_i, G_i)$.

Round 2:

(a) When receiving $(ssid, j, K_j, \hat{K}_j, \tilde{K}_j, G_j)$ from $\mathcal{P}_j$, and $\mathcal{P}_i$ computes:

   i. If j is not equal to i, then set $\psi'_i = \mathcal{F}_{\mathbf{zk}}(\mathsf{prove}, \Pi^{\mathsf{sch}^*}, (ssid, i), (G_i, \Gamma_i, \hat{\Gamma}_i, \tilde{\Gamma}_i, P); (\alpha_i, a_i, b_i, \eta_i))$.

(b) $\mathcal{P}_i$ sends $(ssid, i, \Gamma_i, \hat{\Gamma}_i, \tilde{\Gamma}_i, \psi'_i)$ to all.

Round 3:

(a) When the request $(ssid, j, \Gamma_j, \hat{\Gamma}_j, \tilde{\Gamma}_j, \psi'_j)$ is obtained from $\mathcal{P}_j$, $\mathcal{P}_i$ verifies $\mathcal{F}_{\mathbf{zk}}(\mathsf{vrfy}, \Pi^{\mathsf{sch}^*}, (ssid, j), (G_j, \Gamma_j, \hat{\Gamma}_j, \tilde{\Gamma}_j, P); \psi'_j) = 1$.

(b) For each $j \neq i$, $\mathcal{P}_i$ samples $r_{i,j}, \theta_{i,j}, \hat{r}_{i,j}, \hat{\theta}_{i,j}, \tilde{r}_{i,j}, \tilde{\theta}_{i,j}, \nu'_{i,j}, \hat{\nu}'_{i,j}, \tilde{\nu}'_{i,j} \leftarrow \mathbb{Z}^*_{N_j}$ and computes:

   i. $D_{j,i} = (K_j)^{a_i} \cdot \mathsf{enc}_j(-\nu'_{i,j}, \theta_{i,j})$, $F_{j,i} = \mathsf{enc}_i(\nu'_{i,j}, r_{i,j})$ and $I_{j,i} = \nu_{i,j} \cdot P$, where $\nu_{i,j} = \nu'_{i,j} \bmod q$.

   ii. $\hat{D}_{j,i} = (K_j)^{b_i} \cdot \mathsf{enc}_j(-\hat{\nu}'_{i,j}, \hat{\theta}_{i,j})$, $\hat{F}_{j,i} = \mathsf{enc}_i(\hat{\nu}'_{i,j}, \hat{r}_{i,j})$ and $\hat{I}_{j,i} = \hat{\nu}_{i,j} \cdot P$, where $\hat{\nu}_{i,j} = \hat{\nu}'_{i,j} \bmod q$.

   iii. $\tilde{D}_{j,i} = (\hat{K}_j)^{b_i} \cdot \mathsf{enc}_j(-\tilde{\nu}'_{i,j}, \tilde{\theta}_{i,j})$, $\tilde{F}_{j,i} = \mathsf{enc}_i(\tilde{\nu}'_{i,j}, \tilde{r}_{i,j})$ and $\tilde{I}_{j,i} = \tilde{\nu}_{i,j} \cdot P$, where $\tilde{\nu}_{i,j} = \tilde{\nu}'_{i,j} \bmod q$.

   iv. $\psi_{j,i} = \mathcal{F}_{\mathbf{zk}}(\mathsf{prove}, \Pi^{\mathsf{sch}^*}, (ssid, i), (I_{j,i}, \hat{I}_{j,i}, \tilde{I}_{j,i}, P); (\nu_{i,j}, \hat{\nu}_{i,j}, \tilde{\nu}_{i,j}))$

(c) $\mathcal{P}_i$ sends $(ssid, i, D_{j,i}, F_{j,i}, I_{j,i}, \hat{D}_{j,i}, \hat{F}_{j,i}, \hat{I}_{j,i}, \tilde{D}_{j,i}, \tilde{F}_{j,i}, \tilde{I}_{j,i}, \psi_{j,i})$ to every $\mathcal{P}_j$.

Round 4:

(a) When the request $(ssid, j, D_{i,j}, F_{i,j}, I_{i,j}, \hat{D}_{i,j}, \hat{F}_{i,j}, \hat{I}_{i,j}, \tilde{D}_{i,j}, \tilde{F}_{i,j}, \tilde{I}_{i,j}, \psi_{i,j})$ is notified, $\mathcal{P}_i$ checks $\mathcal{F}_{\mathbf{zk}}(\mathsf{vrfy}, \Pi_i^{\mathsf{sch}^*}, ssid, j, I_{i,j}, \hat{I}_{i,j}, \tilde{I}_{i,j}, P;$
$\psi_{i,j}) = 1$.

(b) For $j \neq i$, $\mathcal{P}_i$ sets $\mu_{i,j} = \mathsf{dec}_i(D_{i,j}) \bmod q$, $\hat{\mu}_{i,j} = \mathsf{dec}_i(\hat{D}_{i,j}) \bmod q$, $\tilde{\mu}_{i,j} = \mathsf{dec}_i(\tilde{D}_{i,j}) \bmod q$, and checks $\mu_{i,j} \cdot P + I_{i,j} = \alpha_i \cdot \hat{\Gamma}_j$, $\hat{\mu}_{i,j} \cdot P + \hat{I}_{i,j} = \alpha_i \cdot \tilde{\Gamma}_j$, $\tilde{\mu}_{i,j} \cdot P + \tilde{I}_{i,j} = a_i \cdot \tilde{\Gamma}_j$.

(c) $\mathcal{P}_i$ sets $\hat{\Gamma} = \prod_\ell \hat{\Gamma}_\ell$, $\tilde{\Gamma} = \prod_\ell \tilde{\Gamma}_\ell$ and computes in the following way:

   i. $\delta_i = a_i\alpha_i + \sum_{j \neq i}(\mu_{i,j} + \nu_{i,j}) \bmod q$, $\Sigma_i = \delta_i \cdot P - \alpha_i \cdot \hat{\Gamma}$, $J_{j,i} = \mu_{i,j} \cdot P$ $(j \neq i)$.

   ii. $\zeta_i = b_i\alpha_i + \sum_{j \neq i}(\hat{\mu}_{i,j} + \hat{\nu}_{i,j}) \bmod q$, $\hat{\Sigma}_i = \zeta_i \cdot P - \alpha_i \cdot \tilde{\Gamma}$, $\hat{J}_{j,i} = \hat{\mu}_{i,j} \cdot P$ $(j \neq i)$.

   iii. $c_i = b_i a_i + \sum_{j \neq i}(\tilde{\mu}_{i,j} + \tilde{\nu}_{i,j}) \bmod q$, $\tilde{\Sigma}_i = c_i \cdot P - a_i \cdot \tilde{\Gamma}$, $\tilde{J}_{j,i} = \tilde{\mu}_{i,j} \cdot P$ $(j \neq i)$.

(d) $\mathcal{P}_i$ samples $\epsilon_i, \iota_i \leftarrow \{0,1\}^\lambda$, and sets $\Phi_i = c_i \cdot P$, $U_i = \mathcal{H}(ssid, i, \Phi_i, \epsilon_i)$, $W_i = \mathcal{H}(ssid, i, \Sigma_i, \hat{\Sigma}_i, \tilde{\Sigma}_i, \iota_i)$.

(e) $\mathcal{P}_i$ broadcasts $(ssid, i, U_i, W_i)$ and sends $(ssid, i, J_{j,i}, \hat{J}_{j,i}, \tilde{J}_{j,i})$ to $\mathcal{P}_j$.

Round 5:

(a) When a request $(ssid, j, U_j, W_j, J_{i,j}, \hat{J}_{i,j}, \tilde{J}_{i,j})$ is obtained from all $\mathcal{P}_j$, $\mathcal{P}_i$ checks:

   i. For $j \neq i$, $J_{i,j} + I_{j,i} = a_i \cdot \Gamma_j$, $\hat{J}_{i,j} + \hat{I}_{j,i} = b_i \cdot \Gamma_j$ and $\tilde{J}_{i,j} + \tilde{I}_{j,i} = b_i \cdot \hat{\Gamma}_j$.

(b) $\mathcal{P}_i$ computes $\hat{\psi}_i = \mathcal{F}_{\mathbf{zk}}(\mathsf{prove}, \Pi^{\mathsf{sch}}, (ssid, i), (U_i, \Phi_i, P); c_i, \epsilon_i)$.

(c) $\mathcal{P}_i$ sends $(ssid, i, \Sigma_i, \hat{\Sigma}_i, \tilde{\Sigma}_i, \iota_i, \Phi_i, \hat{\psi}_i)$ to each $\mathcal{P}_j$.

Figure 8: TriGen subprotocol of $\mathcal{F}_{mult}$

Round 6:

(a) Upon receiving $(ssid, j, \Sigma_j, \hat{\Sigma}_j, \tilde{\Sigma}_j, \iota_j, \Phi_j, \hat{\psi}_j)$ from all $\mathcal{P}_j$, $\mathcal{P}_i$ verifies:

  i. $W_j = \mathcal{H}(ssid, j, \Sigma_j, \hat{\Sigma}_j, \tilde{\Sigma}_j, \iota_j)$.

  ii. $\mathcal{F}_{\mathbf{zk}}(\mathsf{vrfy}, \Pi^{\mathsf{sch}}, (ssid, j), (U_j, \Phi_j, P); \hat{\psi}_j) = 1$

(b) $\mathcal{P}_i$ verifies $\sum_k \Sigma_k = 0$, $\sum_k \hat{\Sigma}_k = 0$ and $\sum_k \tilde{\Sigma}_k = 0$. (If the first equation fails, then report the cheaters by sending α_i, a_i, δ_i to all and checking $\Gamma_j = \alpha_j \cdot P$, $\hat{\Gamma}_j = a_j \cdot P$ as well as $\delta_j \cdot P = \alpha_j a_j \cdot P + \sum_{\ell \neq j}(J_{\ell,j} + I_{\ell,j})$. The similar method is also used to check for the latter two equations.)

(c) For every $j \neq i$, $\mathcal{P}_i$ samples $\overline{r}_{i,j}, \overline{\theta}_{i,j}, \overline{\nu}'_{i,j} \leftarrow \mathbb{Z}^*_{N_j}$ and computes:

  i. $\overline{D}_{j,i} = (K_j)^{c_i} \cdot \mathsf{enc}_j(-\overline{\nu}'_{i,j}, \overline{\theta}_{i,j})$, $\overline{F}_{j,i} = \mathsf{enc}_i(\overline{\nu}'_{i,j}, \overline{r}_{i,j})$ and $\overline{I}_{j,i} = \overline{\nu}_{i,j} \cdot P$, where $\overline{\nu}_{i,j} = \overline{\nu}'_{i,j} \bmod q$.

  ii. $\overline{\psi}_{j,i} = \mathcal{F}_{\mathbf{zk}}(\mathsf{prove}, \Pi^{\mathsf{sch}}, (ssid, i), (\overline{I}_{j,i}, P); \overline{\nu}_{i,j})$.

(d) $\mathcal{P}_i$ sends $(ssid, i, \overline{D}_{j,i}, \overline{F}_{j,i}, \overline{I}_{j,i}, \overline{\psi}_{j,i})$ to $\mathcal{P}_j$.

Round 7:

(a) When the request $(ssid, j, \overline{D}_{i,j}, \overline{F}_{i,j}, \overline{I}_{i,j}, \overline{\psi}_{i,j})$ is notified, $\mathcal{P}_i$ verifies $\mathcal{F}_{\mathbf{zk}}(\mathsf{vrfy}, \Pi^{\mathsf{sch}}_i, (ssid, j), (\overline{D}_{i,j}, K_i, \overline{F}_{i,j}, \Phi_j, \overline{I}_{i,j}; \overline{\psi}_{i,j}) = 1$

(b) If j is not equal to i, $\mathcal{P}_i$ sets $\overline{\mu}_{i,j} = \mathsf{dec}_i(\overline{D}_{i,j})$ and then checks $\overline{\mu}_{i,j} \cdot P + \overline{I}_{i,j} = \alpha_i \cdot \Phi_j$.

(c) $\mathcal{P}_i$ computes $\Phi = \prod_\ell \Phi_\ell$, $\chi_i = c_i \alpha_i + \sum_{j \neq i}(\overline{\mu}_{i,j} + \overline{\nu}_{i,j})$, $\overline{\Sigma}_i = \chi_i \cdot P - \alpha_i \cdot \Phi$, $\overline{J}_{j,i} = \overline{\mu}_{i,j} \cdot P$ $(j \neq i)$.

(d) $\mathcal{P}_i$ selects randomly ϖ_i and computes the hash value $V_i = \mathcal{H}(ssid, i, \overline{\Sigma}_i, \varpi_i)$.

(e) $\mathcal{P}_i$ broadcasts $(ssid, i, V_i)$ and sends $(ssid, i, \overline{J}_{j,i})$.

Round 8:

(a) Once all requests $(ssid, j, V_j, \overline{J}_{j,i})$ are accepted, $\mathcal{P}_i$ checks $\overline{J}_{i,j} + \overline{I}_{j,i} = c_i \cdot \Gamma_j$ for j is not equal to i.

(b) $\mathcal{P}_i$ sends $(ssid, i, \overline{\Sigma}_i, \varpi_i)$ to all.

Output:

(a) Upon receiving $(ssid, j, \overline{\Sigma}_j, \varpi_j)$ from $\mathcal{P}_j$, $\mathcal{P}_i$ verifies $V_j = \mathcal{H}(ssid, j, \overline{\Sigma}_j, \varpi_j)$.

(b) $\mathcal{P}_i$ verifies $\sum_k \overline{\Sigma}_k = 0$ (As used in **Round 6**, check and then report the corrupted parties if the equation fails).

(c) $\mathcal{P}_i$ outputs $(\ell, \alpha_i, a_i, b_i, c_i, \delta_i, \zeta_i, \chi_i)$.

Error. When one of these verification fails, report the cheater and abort.

Stored State. Store the modulus $\boldsymbol{N}$ and the private keys (p_i, q_i).

Figure 8: TriGen subprotocol of $\mathcal{F}_{mult}$ (cont.)

values and random point X in $\mathbb{G}_1$. The view of the subprotocol is provided in Fig. 9.

6 Our Protocol

We present here our scheme for universally composable and non-interactive BLMQ with identifiable aborts. The scheme includes the key generation, pre-extracting, extracting, pre-signing, and signing. Let $\mathcal{P}_i$ be the party who executes these phases. Moreover, we use l and n as the number of KGCs and devices, respectively.

Mult subprotocol of $\mathcal{F}_{mult}$

Upon input $(\mathsf{mult}, ssid, \ell, i, e_i, d_i)$ to $\mathcal{P}_i$, if an entry $(\ell, \alpha_i, a_i, b_i, c_i, \delta_i, \zeta_i, \chi_i)$ is stored, and then it proceeds as follows:

Round 1:

(a) $\mathcal{P}_i$ sets $\mu_i = e_i - a_i$, and $\nu_i = d_i - b_i$.

(b) $\mathcal{P}_i$ sends the message $(ssid, i, \mu_i, \nu_i)$ to other players.

Round 2:

(a) Once a request $(ssid, j, \mu_j, \nu_j)$ is notified from $\mathcal{P}_j$, $\mathcal{P}_i$ sets $\mu = \sum \mu_j$ and $\nu = \sum \nu_j$.

(b) $\mathcal{P}_i$ sets $\xi_i = c_i + \mu b_i + \nu a_i + \mu\nu/k \bmod q$ and $\varkappa_i = \chi_i + \mu\zeta_i + \nu\delta_i + \mu\nu\alpha_i \bmod q$, where $k \in \{l, n\}$.

(c) $\mathcal{P}_i$ sets $\overline{r}_i \leftarrow \mathbb{Z}_q^*$ and $\overline{\Gamma}_i = \xi_i \cdot P + \overline{r}_i \cdot X$.

(d) $\mathcal{P}_i$ sends $(ssid, i, \xi_i \cdot P)$ to all.

Round 3:

(a) When a request $(ssid, j, \xi_j \cdot P)$ is obtained from $\mathcal{P}_j$, $\mathcal{P}_i$ sets $\xi \cdot P = \sum \xi_j \cdot P$ and $\Lambda_i = \varkappa_i \cdot P - \alpha_i \xi \cdot P$.

(b) $\mathcal{P}_i$ samples $\varsigma_i \leftarrow \{0,1\}^\lambda$ and sets $M_i = \mathcal{H}(ssid, i, \Lambda_i, \varsigma_i, \overline{r}_i, \overline{\Gamma}_i)$.

(c) $\mathcal{P}_i$ broadcasts the message $(ssid, i, M_i)$ to all other players.

Round 4:

Once all requests $(ssid, j, M_j)$ are notified, $\mathcal{P}_i$ sends $(ssid, i, \Lambda_i, \varsigma_i, \overline{r}_i, \overline{\Gamma}_i)$ to other players.

Output:

When all requests $(ssid, j, \Lambda_j, \varsigma_j, \overline{r}_j, \overline{\Gamma}_j)$ are obtained, $\mathcal{P}_i$ verifies:

i. $M_j = \mathcal{H}(ssid, j, \Lambda_j, \varsigma_j, \overline{r}_j, \overline{\Gamma}_j)$.

ii. $\sum \Lambda_j = 0$.

  A. If the verification fails, check $\overline{\Gamma}_j = \xi_j \cdot P + \overline{r}_j \cdot X$, for $j \neq i$, and report the cheater.

  B. Else output ξ_i.

Error. When one of these verification fails, report the cheater and abort.

Erase $(\tau, \alpha_i, a_i, b_i, c_i, \delta_i, \zeta_i, \chi_i)$.

Stored State. Store ξ_i, $\{\overline{r}_j, \overline{\Gamma}_j\}_j$.

Figure 9: Mult subprotocol of $\mathcal{F}_{mult}$

6.1 The protocol for $\mathcal{F}_{IBTS}$

Given $\mathcal{F}_{mult}$, it is simple to generate Beaver triples for securely computing $\mathcal{F}_{IBTS}$. Particularly, such a triple allows one to multiply two given values without carrying out the (inefficient) multiplication protocol. Thus, parties are able to select random values d_i, and use $\mathcal{F}_{mult}$ to get (a_i, b_i, c_i) such that $c = a \cdot b$, to reveal $\mu_i = (s_i + \mathcal{H}(ID)/l) - a_i$, $\nu_i = d_i - b_i$, and finally open $\delta_i = c_i + \mu b_i + \nu a_i + \mu\nu$ to obtain $\delta = \sum \delta_i = (s + \mathcal{H}(ID)) \cdot d$. By bringing this value out in the open, every party locally computes $\delta^{-1} = d^{-1} \cdot (s + \mathcal{H}(ID))^{-1}$ and then multiplies this value with d_i to obtain $d_i \cdot \delta^{-1}$. The resulting value $d_i \delta^{-1} \cdot P$ is an additive share of the private key $d_{ID} = (s + \mathcal{H}(ID))^{-1} \cdot P$.

To achieve non-interacting signing, BLMQ signing requires computing $(h = \mathcal{H}(m||g^r)$, $S = (r + h) \cdot d_{ID})$ with one round message. Thus, it is necessary for parties to precompute some tuples $(u = g^r, \rho_i, \xi'_i, \xi''_i, \xi'''_i)$ where ξ', ξ'' and ξ''' are respectively additive shares of $r \cdot \rho$, $(r+k) \cdot \rho$ and $d'_{ID} \cdot (r+k)$ for randomly choosing values r_i, k_i and ρ_i. Upon receiving a request to sign a message m, each party locally sets $v_i = \xi'_i + h \cdot \rho_i$ for $h = \mathcal{H}(m||u)$ and broadcasts. Observe that all parties can compute $v = \sum v_i = \xi' + h\rho = (r+h)\rho$, $\tau = v/(\sum \xi''_i) = (r+h)/(r+k)$ and then obtain $(h, S = \tau(\sum \xi'''_i) \cdot P = d'_{ID}(r+h) \cdot P)$ which is the signature needed. The details are presented in Fig. 10.

Parameter: $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, q, P, Q, g)$.

Key generation: When the request $\mathsf{keygen}(sid)$ is notified, each player $\mathcal{P}_i$ works below ($\mathcal{P}_i \in \boldsymbol{KGC}$):

1. $\mathcal{P}_i$ sends (init, sid) to $\mathcal{F}_{mult}$, and receives back $(\mathsf{init}, sid, (s_i, x_i, p_i, q_i), (Q_{pub}, X, \boldsymbol{N}))$.
2. *Output:* Q_{pub} is considered to be a global key.

Pre-Extracting: When a request $\mathsf{pre\text{-}extract}(ssid)$ is obtained, each player $\mathcal{P}_i$ executes in the following ways ($\mathcal{P}_i \in \boldsymbol{KGC}$):

1. $\mathcal{P}_i$ sends $(\mathsf{TriGen}, ssid, \ell, i)$ to $\mathcal{F}_{mult}$, and receives back $(\mathsf{TriGen}, ssid, (\ell, \alpha_i, a_i, b_i, c_i, \delta_i, \zeta_i, \chi_i))$.
2. *Output:* $\mathcal{P}_i$ locally stores the tuples $\{(\ell, \alpha_{i,\ell}, a_{i,\ell}, b_{i,\ell}, c_{i,\ell}, \delta_{i,\ell}, \zeta_{i,\ell}, \chi_{i,\ell})\}_{\ell \in [L]}$.

Extracting: When a request $\mathsf{extract}(ssid, ID)$ is notified, each player $\mathcal{P}_i$ implements below ($\mathcal{P}_i \in \boldsymbol{KGC}$):

1. $\mathcal{P}_i$ sends the ideal functionality $\mathcal{F}_{mult}$ the message $(\mathsf{mult}, ssid, \ell, i, e_i, d_i)$, and gets $(\mathsf{mult}, ssid, \ell, i, \xi_i, \{\overline{r}_j, \overline{\Gamma}_j\}_j)$ (note that $\xi = e \cdot d$ and $e_i = s_i + \mathcal{H}(ID)/l \bmod q$).
2. $\mathcal{P}_i$ broadcasts ξ_i to all $\mathcal{P}_j$.
3. $\mathcal{P}_i$ checks $\overline{\Gamma}_j = \xi_j \cdot P + \overline{r}_j \cdot X$ and computes $d'_{ID_i} = (\sum_{j=1}^{l} \xi_j)^{-1} d_i \bmod q$.
4. $\mathcal{P}_i$ sends $(\mathsf{reshare}, ssid, \ell, i, \mathbb{Z}_q^*)$ to $\mathcal{F}_{rtk}$.
5. $\mathcal{P}_j$ receives $(\mathsf{reshare}, ssid, i, \mathsf{chk}_i^j)$from $\mathcal{F}_{rtk}$ (note that $\mathcal{P}_j \in \boldsymbol{D}$).
6. $\mathcal{P}_j$ computes $y_j = \sum_{i=1}^{l} \rho_{i,k}^j \bmod q$ (note that $k \in [8]$).
7. *Output:* $\mathcal{P}_j$ locally stores the tuples $\{(d'_{ID_{j,\ell'}}, \alpha_{j,\ell'}, a_{j,\ell'}, b_{j,\ell'}, c_{j,\ell'}, \delta_{j,\ell'}, \zeta_{j,\ell'}, \chi_{j,\ell'})\}_{\ell' \in [L']}$

Pre-Signing: Upon input $\mathsf{pre\text{-}signing}(ssid)$, $\mathcal{P}_i$ works as follows ($\mathcal{P}_i \in \boldsymbol{D}$):

1. $\mathcal{P}_i$ sends the ideal functionality $\mathcal{F}_{mult}$ the message $(\mathsf{input}, ssid||1, ssid||2, i)$, and obtains $(\mathsf{input}, ssid, i, u, r_i, \rho_i, \boldsymbol{Y})$ (note that $u = g^r$ and $r = \sum_j r_j$).
2. $\mathcal{P}_i$ sends $(\mathsf{mult}, ssid||1, ssid||2, ssid||3)$ to $\mathcal{F}_{mult}$ ((r_i, ρ_i), $(r_i + k_i, \rho_i)$ and $(d'_{ID_i}, r_i + k_i)$ are the input values of the command, respectively).
3. $\mathcal{P}_i$ receives $(\mathsf{mult}, (ssid||1, \xi'_i, \{\overline{r}'_j, \overline{\Gamma}'_j\}_j), (ssid||2, \xi''_i, \{\overline{r}''_j, \overline{\Gamma}''_j\}_j), (ssid||3, \xi'''_i, \{\overline{r}'''_j, \overline{\Gamma}'''_j\}_j))$ from $\mathcal{F}_{mult}$ (note that $\xi' = r \cdot \rho$, $\xi'' = (r+k) \cdot \rho$ and $\xi''' = d'_{ID} \cdot (r+k)$).
4. *Output:* $\mathcal{P}_i$ locally stores $(u, \rho_i, \{\overline{r}'_j, \overline{\Gamma}'_j, \overline{r}''_j, \overline{\Gamma}''_j, \overline{r}'''_j, \overline{\Gamma}'''_j, Y_j\}_j, \xi'_i, \xi''_i, \xi'''_i)$.

Signing: Upon input $\mathsf{sign}(ssid, m)$, $\mathcal{P}_i$ works as follows ($\mathcal{P}_i \in \boldsymbol{D}$):

1. $\mathcal{P}_i$ computes $h = \mathcal{H}(m||u)$ and $v_i = \xi'_i + h \cdot \rho_i$, and broadcasts $(ssid, i, v_i, \xi''_i, \xi'''_i)$.
2. $\mathcal{P}_i$ computes $v = \sum v_i = \xi' + h\rho$, $\xi'' = \sum \xi''_i$, $\xi''' = \sum \xi'''_i$ and $S' = v\xi'''/\xi''$, and check that $(h, S' \cdot P)$ is a correct signature.
   (a) If the above fails, check $v_j \cdot P = (\overline{\Gamma}'_j - \overline{r}'_j \cdot X) + h \cdot Y_j$, $\overline{\Gamma}''_j = \xi''_i \cdot P + \overline{r}''_j \cdot X$ and $\overline{\Gamma}'''_j = \xi'''_i \cdot P + \overline{r}'''_j \cdot X$, for $j \neq i$.
   (b) Else, output $(\mathsf{signature}, ssid, i, m, h, S' \cdot P)$.

Figure 10: Securely computing $\mathcal{F}_{IBTS}$

Correctness. Observe that ξ is the product of $e = \sum_{\ell=1}^{l}(s_\ell + \mathcal{H}(ID)/l)$ and $d = \sum_{\ell=1}^{l} d_\ell$, and so $d'_{ID} \cdot P = \sum_{\ell=1}^{l} d_\ell \cdot \xi^{-1} \cdot P = d \cdot (d \cdot (s + \mathcal{H}(ID)))^{-1} \cdot P = (s + \mathcal{H}(ID))^{-1} \cdot P$ is the private key as required.

Furthermore, observe that $h = (\mathcal{H}(m||u))$, $v_i = \xi'_i + h \cdot \rho_i$, $\sum \xi'_i = r \cdot \rho$, $\sum \xi''_i = (r+k) \cdot \rho$, and $\sum \xi'''_i = d'_{ID} \cdot (r+k)$. We have $S' = \frac{\sum_{\ell=1}^{n} v_\ell}{\xi''} \xi''' = \frac{\xi' + h\rho}{(r+k)\rho} d'_{ID} \cdot (r+k) = d'_{ID}(r+h)$. Thus, the signature $(h, S' \cdot P)$ is valid for a given message m.

6.2 Proof of security of protocol IBTS

Next, we give an overview of the provable security for the protocol π_{IBTS}. The proof follows by contradiction that if π_{IBTS} does not implement $\mathcal{F}_{IBTS}$, then a PPT algorithm could see the distribution between the Paillier ciphertexts *or* a PPT faker could break the existential unforgeability of the BLMQ signature scheme. Furthermore, the protocol π_{IBTS} can resist collusion attacks between different KGCs and devices. Informally, this means that an attacker cannot output valid signatures on unrequested messages, even if the attacker can corrupt any parties with less than the security threshold.

Theorem 1. *Assuming that the Paillier scheme is semantic security and the BLMQ signature is existential unforgeability, then the protocol shown in Fig. 10 is able to securely implement the functionality $\mathcal{F}_{IBTS}$ described in Fig. 1.*

Corollary 1. *If the protocol shown in Fig. 10 can securely implement the functionality $\mathcal{F}_{IBTS}$, then the protocol described in Fig. 10 is secure against a static and malicious adversary who controls up to both $l-1$ KGCs and $n-1$ devices.*

Proof. An ideal simulator $\mathcal{S}$ shown in Fig. 1 denotes the number of corrupted KGCs and devices, namely that ideal functionality $\mathcal{F}_{IBTS}$ operates under many collusive attackers. Further, two different security thresholds guarantee that an attacker needs to break all KGCs/devices to get the entire keys, and thus the proposed protocol can resist the collusion attack between multiple KGCs and devices. □

6.2.1 Proof of theorem 1

Here, we give a description of formally proving the theorem 1 in the following lemmas.

Lemma 1. *If $\mathcal{F}_{IBTS}$ is not securely implemented by π_{IBTS} with the ideal functionality $\mathcal{H}^{\mathrm{g}}$ of a global random oracle, then an environment $\mathcal{Z}$ can fake a valid signature for unasked messages with the functionality $\mathcal{H}^{\mathrm{g}}$.*

Proof. The simulations described in Appendix A are well-founded, so the claim is straightforward. □

Lemma 2. *If $\mathcal{Z}$ is able to forge valid signatures in a real execution of π_{IBTS}, then there exist two algorithms $\mathcal{R}_1$ and $\mathcal{R}_2$ such that given oracle access to $\mathcal{Z}$, those algorithms can win the events below.*

1. *$\mathcal{R}_1$ outputs "success" in the indistinguishable security experiment for Paillier with probability at least $\geq 1/2$.*
2. *$\mathcal{R}_2$ outputs "success" in the existential unforgeability experiment for BLMQ with non-negligible probability.*

Proof. Let $\mathcal{Z}$ be an environment. Moreover, $\mathcal{Z}$ is able to fake valid signatures in the real execution of π_{IBTS}. $T \in \mathsf{poly}(\lambda)$ denotes the maximum number of executing the pre-extracting phase prior to occurring the forgery. In addition, $N^1, \ldots, N^T$ stand for the corresponding Paillier public keys during the simulation settings. Analogously, (Q_{pub}, s) represents the BLMQ key pair. For the processes $\mathcal{R}_1$ and $\mathcal{R}_2$, take the experiments into consideration below:

> **Experiment A.** Given a set of inputs (Q_{pub}, s) and $\big(N^k, C^k = \mathsf{enc}_{N^k}(1)\big)_{k=1,\ldots,T}$, $\mathcal{R}_1$ interacts with $\mathcal{Z}$ and then returns the output.
>
> **Experiment B.** Given a set of inputs (Q_{pub}, s) and $\big(N^k, C^k = \mathsf{enc}_{N^k}(0)\big)_{k=1,\ldots,T}$, $\mathcal{R}_1$ interacts with $\mathcal{Z}$ and then returns the output.
>
> **Experiment C.** Given the BLMQ public key Q_{pub}, $\mathcal{R}_2$ interacts with $\mathcal{Z}$ and then returns the output.

The process $\mathcal{R}_1$ acts on behalf of uncorrupted players in experiment A. In the meantime, $\mathcal{R}_1$ also interacts with an environment $\mathcal{Z}$ below. $\mathcal{R}_1$ starts at the beginning of selecting the BLMQ private key s and secret keys of uncorrupted players. Next, $Q_{pub} = s \cdot Q$ is considered to be the BLMQ public key (this is done by rewinding $\mathcal{Z}$). Then, $\mathcal{R}_1$ in the pre-extracting phase (or the pre-signing phase) follows the commands for the uncorrupted players messages except that a random honest party $\mathcal{P}_{\overline{b}}$ (or $\mathcal{P}_{\widehat{b}}$). For the specific party $\mathcal{P}_{\overline{b}}$ (or $\mathcal{P}_{\widehat{b}}$), the reduction will choose the public key from $(N^1, \ldots, N^T)$, say N^t, and convert the $\mathcal{P}_{\overline{b}}$'s (or $\mathcal{P}_{\widehat{b}}$'s) ciphertexts to a related ciphertext C^t. Further, the security proofs of all $\mathcal{P}_{\overline{b}}$ (or $\mathcal{P}_{\widehat{b}}$) are emulated by making use of the random oracle ability to rewind the environment $\mathcal{Z}$. For the extracting, pre-signing and signing phase, the same trick is employed to simulate the behavior of the specific player $\mathcal{P}_{\overline{b}}$ (or $\mathcal{P}_{\widehat{b}}$).

If one is encrypted for the ciphertext C^t, the distribution of $\mathcal{R}_1$ interacting with $\mathcal{Z}$ is indistinguishable, namely that it is indistinguishable from that of an interaction of real protocol between honest parties and $\mathcal{Z}$. Otherwise, this distribution is “distinguishable" since the ciphertexts of the specific party are encryptions of zero.

Claim. *If there exists an environment $\mathcal{Z}$ which (τ, ϵ)-forges valid signatures in an execution of Fig. 10, then the output of experiment A is $(\tau \cdot l \log l \cdot n \log n, \epsilon^3 - 1/\mathsf{poly}(\lambda))$-successful with a call to $\mathcal{Z}$.*

Remark 1. Note that the executing time increased by $l \log l \cdot n \log n$ times, and the probability of a valid forgery is reduced from ϵ to ϵ^3. As a result of the rewinding technique, it is necessary for the security loss in the experiments. Precisely, one need to provide $\tau \cdot l \log l \cdot n \log n$ in time for rewinding $\mathcal{Z}$ (whenever $\mathcal{Z}$ conjectures about $\mathcal{P}_{\overline{b}}$ (or $\mathcal{P}_{\widehat{b}}$) at any moment), as well as ϵ^3 in probability for rewinding the zero-knowledge proofs.

Claim. *Assume that the Paillier encryption is semantic security. If $\mathcal{Z}$ is a forger that (τ, ϵ)-forges valid signatures in experiment A, then the output of experiment B is $(\tau, \epsilon - 1/\mathsf{poly}(\lambda))$-successful with a call to $\mathcal{Z}$.*

Remark 2. Given oracle access to the functionality $\mathcal{H}^{\mathrm{g}}$ and the BLMQ public key, the process $\mathcal{R}_2$ emulates the distribution between $\mathcal{Z}$ and the uncorrupted players. The simulation is similar to the strategy of $\mathcal{R}_1$ in the key generation, pre-extracting and extracting phase, except that the differences are as follows:

- In order to acquire a value u in $\mathbb{G}_T$, the pre-signing simulation needs to invoke the functionality $\mathcal{H}^{\mathrm{g}}$ of the oracle.
- The signing simulation carries out a query to the oracle on a message m and u to get the signature (h, S).

Claim. *If $\mathcal{Z}$ is a forger that (τ, ϵ)-forges valid signatures in experiment B, then the output of experiment C is (τ, ϵ)-successful with a call to $\mathcal{Z}$.*

$\square$

6.3 Efficiency and experimental results

We present the theoretical complexity analysis and running time for our protocol.

6.3.1 Theoretical complexity

The $\mathcal{F}_{mult}$ protocol includes Init, Input, TriGen as well as Mult. In addition, it requires a call to $\mathcal{F}_{\mathbf{rtk}}$. The cost analysis of each subprotocol is provided in Table 2. We also remark that the computations of $\mathcal{F}_{\mathbf{rtk}}$ is implemented during the pre-extracting phase, which is connected via secure channels as would the original BLMQ algorithm. Thus, the cost of $\mathcal{F}_{\mathbf{rtk}}$ is not counted here.

Table 2: Theoretical analyses of all overheads in our protocols.

Protocol	Rounds	Computation	Communication (kilobytes)
Init of $\mathcal{F}_{mult}$	3	$(l+1)\times(2\boldsymbol{G}_1+2\boldsymbol{G}_2+11\boldsymbol{N})$	$24\kappa+12\lvert N\rvert$ (3.84 KB)
Input of $\mathcal{F}_{mult}$	3	$n\times(2\boldsymbol{G}_T+2\boldsymbol{G}_1)+2\boldsymbol{G}_T+\boldsymbol{G}_1$	48κ (1.54 KB)
TriGen of $\mathcal{F}_{mult}$	8	$l\times(8\boldsymbol{N}+38\boldsymbol{G}_1)-5\boldsymbol{N}-10\boldsymbol{G}_1$	$l\times(16\lvert N\rvert+20\kappa)+15\kappa-10\lvert N\rvert$ $\big((4.74l-2.08)\text{KB}\big)$
Mult of $\mathcal{F}_{mult}$	4	$4\boldsymbol{G}_1$	11κ (0.35 KB)
Totals			
KeyGen of $\mathcal{F}_{IBTS}$	3	$(l+1)\times(2\boldsymbol{G}_1+2\boldsymbol{G}_2+11\boldsymbol{N})$	3.84 KB
Pre-Ext of $\mathcal{F}_{IBTS}$	8	$l\times(8\boldsymbol{N}+38\boldsymbol{G}_1)-5\boldsymbol{N}-10\boldsymbol{G}_1$	$(4.74l-2.08)$ KB
Ext of $\mathcal{F}_{IBTS}$	5	$4\boldsymbol{G}_1$	0.38 KB
Pre-Sign of $\mathcal{F}_{IBTS}$	4	$n\times(2\boldsymbol{G}_T+2\boldsymbol{G}_1)+2\boldsymbol{G}_T+12\boldsymbol{G}_1$	2.59 KB
Signing of $\mathcal{F}_{IBTS}$	1	$T_e+\boldsymbol{G}_T+\boldsymbol{G}_1$	3κ (0.10 KB)

• Theoretical analyses of all overheads in our scheme. T_e denotes the operation over bilinear pairing. $\boldsymbol{G}_1$, $\boldsymbol{G}_2$, $\boldsymbol{G}_T$ and $\boldsymbol{N}$ denote computing exponentiation in the bilinear groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and composite group $\mathbb{Z}_N$. Group members (denoted $\lvert\mathbb{Z}_q^*\rvert$) and values on $\mathbb{Z}_N$ (denoted $\lvert N\rvert$) are considered to be the communication overhead, including things that are sent to every other party. These values are based on security parameter $\lambda = 128$, and thus Paillier modulus is $N = 2048$ and the size of group elements is $\kappa = \lvert\mathbb{Z}_q^*\rvert = 256$. For bilinear groups, $\lvert\mathbb{G}_1\rvert$, $\lvert\mathbb{G}_2\rvert$ and $\lvert\mathbb{G}_T\rvert$ are respectively 2, 4, 12 factor of the $\mathbb{Z}_q^*$ element bit-length κ.

The $\mathcal{F}_{IBTS}$ protocol consists of KeyGen, Pre-Extracting, Extracting, Pre-Signing, and Signing. The KeyGen and Pre-Extracting phase involve respectively one call to Init and TriGen of $\mathcal{F}_{mult}$. The Extracting phase includes a call to Mult of $\mathcal{F}_{mult}$ and $\mathcal{F}_{\mathbf{rtk}}$. The Pre-Signing phase consists of a call to Input of $\mathcal{F}_{mult}$, and three parallel calls to Mult. This process needs to run in order and hence the total cost can be summarized. See Table 2 for a specific description.

6.3.2 Experimental results

We choose the BN254 group for an element with bit length 256 as the finite field for 128-bit security level, namely $\log\lvert\mathbb{Z}_q^*\rvert = 256$. Hence, $\lvert\mathbb{G}_1\rvert = 512$, $\lvert\mathbb{G}_2\rvert = 1024$, and $\lvert\mathbb{G}_T\rvert = 3072$. 256 and 4 bytes correspond to the length of the Paillier modulus and user identity. For the operations of a server, we make use of Alibaba Cloud to emulate the behaviors, which are implemented in Ubuntu 20.04.03 LTS. Moreover, the properties of terminal emulation are an Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.5GHz as well as 32 GB of RAM. The simulations of users are instanced on Google Pixel 3 XL with Octa-core and 4GB of RAM, running Android 12.0 (Pie).

We ran experiments with three parties, i.e. $l = n = 3$; every subprotocol repeated 1000 times and then we took an average. As plotted in Fig. 11, the Pre-Signing and Signing time are about 19ms and 1.6ms, respectively, and thus are very practical for IoT devices. In fact, we take advantage of the online/offline technique that the expensive operations (e.g., exponentiation execution in Paillier cryptosystem) can be computed by servers in the offline phase, and then the corrected values for multiplicative work are efficiently involved during the online phase. For the Pre-Extracting phase, the computational overhead is less

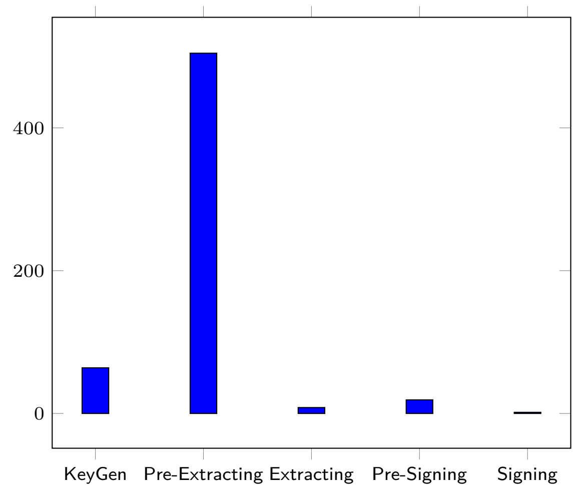

Figure 11: The executing time in milliseconds for each subprotocol in our scheme.

than one second. Particularly, this process can be run many times before knowing the user identity, and generate a mass of corrected values to output the corresponding private key and signature. The costs of KeyGen and Extracting are 64ms and 8ms, respectively, and both subprotocols only need to be done once. Therefore, the total computation is very efficient for deploying IoT devices in real life.

7 Conclusion

We presented an efficient and non-interacting threshold BLMQ protocol with identifiable aborts and collusion resistance. The solution started by employing the Beaver triples to construct a secure multiplicative-to-additive conversion and then converting an inverse operation to a linear combination while carrying heavy computation into the offline phase. Furthermore, we proposed a commit-open way to detect the multiplicative shares in case of faults, which does not introduce additional communication rounds and can locally verify the relation between the shares and commitments. The security proof is loose since it incurs a reduction loss of $\binom{n}{t}$, where n and t denote the number of total parties and signers, respectively. An interesting point for future work is to utilize a random salt to obtain a tight security reduction.

References

[ANO$^+$22] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ecdsa via pseudorandom correlation generators. In *IEEE Symposium on Security and Privacy*, pages 2554–2572. IEEE, 2022.

[BCG$^+$19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent ot extension and more. In *Annual International Cryptology Conference*, pages 489–518. Springer, 2019.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual International Cryptology Conference*, pages 213–229. Springer, 2001.

[BLMQ05] Paulo SLM Barreto, Benoît Libert, Noel McCullagh, and Jean-Jacques Quisquater. Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 515–532. Springer, 2005.

[BZ04] Joonsang Baek and Yuliang Zheng. Identity-based threshold signature scheme from the bilinear pairings. In *Proceedings of the International Conference on Information Technology: Coding and Computing*, volume 1, pages 124–128. IEEE, 2004.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[CCL+23] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold ec-dsa revisited: Online/offline extensions, identifiable aborts proactive and adaptive security. *Theoretical Computer Science*, 939:78–104, 2023.

[CGG+20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1769–1787, 2020.

[CHC03] Jae Cha Choon and Jung Hee Cheon. An identity-based signature from gap diffie-hellman groups. In *International Workshop on Public Key Cryptography*, pages 18–30. Springer, 2003.

[CZKK04] Xiaofeng Chen, Fangguo Zhang, Divyan M Konidala, and Kwangjo Kim. New id-based threshold signature scheme from bilinear pairings. In *International Conference on Cryptology in India*, pages 371–383. Springer, 2004.

[Des94] Yvo G Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994.

[DKLS19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ecdsa from ecdsa assumptions: the multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE, 2019.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual International Cryptology Conference*, pages 643–662. Springer, 2012.

[EGM89] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. In *Annual International Cryptology Conference*, pages 263–275. Springer, 1989.

[FHL+20] Qi Feng, Debiao He, Zhe Liu, Ding Wang, and Kim-Kwang Raymond Choo. Distributed signing protocol for ieee p1363-compliant identity-based signature scheme. *IET Information Security*, 14(4):443–451, 2020.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Annual International Cryptology Conference*, pages 186–194. Springer, 1986.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1179–1194, 2018.

[GG20] Rosario Gennaro and Steven Goldfeder. One round threshold ecdsa with identifiable abort. Cryptology ePrint Archive, Paper 2020/540, 2020. https://eprint.iacr.org/2020/540.

[GGI19] Rosario Gennaro, Steven Goldfeder, and Bertrand Ithurburn. Fully distributed group signatures, 2019. https://www.orbs.com/wp-content/uploads/2019/04/Crypto_Group_signatures-2.pdf.

[Hes02] Florian Hess. Efficient identity based signature schemes based on pairings. In *International Workshop on Selected Areas in Cryptography*, pages 310–324. Springer, 2002.

[HZWC18] Debiao He, Yudi Zhang, Ding Wang, and Kim-Kwang Raymond Choo. Secure and efficient two-party signing protocol for the identity-based signature scheme in the ieee p1363 standard for public key cryptography. *IEEE Transactions on Dependable and Secure Computing*, 17(5):1124–1132, 2018.

[IDC21] IDC. Future of industry ecosystems: Shared data and insights. https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights/, 2021.

[IEE13] IEEE 1363.3-2013. Ieee 1363.3-2013-ieee standard for identity-based cryptographic techniques using pairings. https://standards.ieee.org/standard/1363-2000.html, 2013.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *Annual International Cryptology Conference*, pages 369–386. Springer, 2014.

[ISO18] ISO/IEC 14888-3:2018. It security techniques — digital signatures with appendix — part 3: Discrete logarithm based mechanisms. https://www.iso.org/standard/76382.html, 2018.

[JZWL23] Yan Jiang, Youwen Zhu, Jian Wang, and Xingxin Li. Fully distributed identity-based threshold signatures with identifiable aborts. *Frontiers of Computer Science*, 17(5):175813, 2023.

[LC24] Huiqiang Liang and Jianhua Chen. Non-interactive sm2 threshold signature scheme with identifiable abort. *Frontiers of Computer Science*, 18(1):181802, 2024.

[Lin17] Yehuda Lindell. Fast secure two-party ecdsa signing. In *Annual International Cryptology Conference*, pages 613–644. Springer, 2017.

[PS06] Kenneth G Paterson and Jacob CN Schuldt. Efficient identity-based signatures secure in the standard model. In *Australasian Conference on Information Security and Privacy*, pages 207–222. Springer, 2006.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Annual International Cryptology Conference*, pages 47–53. Springer, 1984.

[TANBDV20] Luís T. A. N. Brandão, Michael Davidson, and Apostols Vassilev. Roadmap toward criteria for threshold schemes for cryptographic primitives. https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8214A.pdf, 2020.

[XQL10] Hu Xiong, Zhiguang Qin, and Fagen Li. Identity-based threshold signature secure in the standard model. *Int. J. Netw. Secur.*, 10(1):75–80, 2010.

[ZCS23] Mingwu Zhang, Yu Chen, and Willy Susilo. Decision tree evaluation on sensitive datasets for secure e-healthcare systems. *IEEE Trans. Dependable Secur. Comput.*, 20(5):3988–4001, 2023. doi:10.1109/TDSC.2022.3219849.

[ZHZ+18] Yudi Zhang, Debiao He, Sherali Zeadally, Ding Wang, and Kim-Kwang Raymond Choo. Efficient and provably secure distributed signing protocol for mobile devices in wireless networks. *IEEE Internet of Things Journal*, 5(6):5271–5280, 2018.

A Simulators

Here, we give an overview of the UC-simulation and two independent reductions. Moreover, one corresponds to the semantic security of the Paillier encryption ($\mathcal{R}_1$) and the other corresponds to the unforgeability of standard BLMQ ($\mathcal{R}_2$) The details of the simulators are provided as follows.

A.1 UC simulator

In the ideal process, the ideal adversary computes messages for uncorrupted parties as required, carries out the commands of the protocol for each phase, and works below.

1. The simulator gives the functionality the global parameters as well as the BLMQ verification algorithm at end of key generation.
2. Upon receiving a signing request, the simulator submits the functionality the valid pair (h, S).
3. Once $\mathcal{Z}$ corrupts a player, the simulator sends the functionality the request.
4. The simulator reports the identity of cheaters as specified by the functionality in case of aborts.

A.2 Paillier distinguisher $\mathcal{R}_1$

Let $\mathcal{R}_1$ be an adversary, which can see the distribution of ciphertexts for the Paillier encryption. $T \in \mathsf{poly}(\lambda)$ denotes the maximum number of executing the pre-extracting phase prior to occurring the forgery. $(N^1, C^1), \ldots, (N^T, C^T)$ and (Q_{pub}, s) are respectively the key-pair for BLMQ signature scheme, as well as the public keys and ciphertexts for Paillier encryption scheme. ctr is considered to be a counter and the initial value is 0. $\boldsymbol{L}$ denotes a list that the simulator maintains, storing the queries and answers. The interaction of $\mathcal{R}_1$ with $\mathcal{Z}$ is described as follows.

Oracle calls. When a request $(\mathsf{query}, x) = (\mathsf{query}, sid^*, rid^*, \ldots)$ is notified, execute below:

1. If (sid^*, rid^*) is equal to (sid, rid), respond the message $(\mathsf{answer}, a = \mathcal{H}(x))$.
2. Otherwise, if $x = ([sid, j, \psi])$ is obtained in a rewinding process, set the random oracle, derive the private keys p, q, and store the corresponding tuple to $\boldsymbol{L}$.

3. Otherwise, if (x, a) belongs to $\boldsymbol{L}$, respond (answer, a). Then, select a random value a, respond (answer, a), as well as store the tuple (x, a) to $\boldsymbol{L}$.

Key generation. $\mathcal{Z}$ injects the command $(\mathsf{keygen}, sid = (\ldots, \boldsymbol{KGC}), i)$ into $\mathcal{P}_i$, and controls the set of corrupted parties $\boldsymbol{C} \subsetneq \boldsymbol{KGC}$. Run $\mathcal{S}^1(sid, \boldsymbol{C}, \boldsymbol{L}, Q_{pub})$ and get outputs $\bar{b}$, $\boldsymbol{L}$, rid, $\{X_k, N_k\}_{k\in\boldsymbol{KGC}}$, and $\{(p_k, q_k), x_k, s_k\}_{k\neq\bar{b}}$. Output $s_{\bar{b}} = s - \sum_{j\neq\bar{b}} s_j$.

Pre-Extracting. $\mathcal{Z}$ injects the command $(\mathsf{pre\text{-}extract}, ssid = (\ldots, sid, \boldsymbol{D}), \ell, i)$ into $\mathcal{P}_i$, and controls the set of corrupted parties $\boldsymbol{C} \subsetneq \boldsymbol{KGC}$. Sample $\alpha_{\bar{b}}, a_{\bar{b}}, b_{\bar{b}} \leftarrow \mathbb{Z}_q^*$, and p_0, q_0 such that $N_0 = p_0q_0$. Let $C_0 = (1+N_0)\rho^{N_0} \bmod N_0^2$ in which ρ is randomly chosen from the group $\mathbb{Z}_{N_0}^*$. Set $\mathsf{pre\text{-}ext}_0 = (\alpha_{\bar{b}}, a_{\bar{b}}, b_{\bar{b}}, N_0, C_0)$, $\mathsf{ctr} = \mathsf{ctr} + 1$, $\mathsf{pre\text{-}ext} = (\alpha_{\bar{b}}, a_{\bar{b}}, b_{\bar{b}}, N^{\mathsf{ctr}}, C^{\mathsf{ctr}})$ and work as follows.

1. If $|\boldsymbol{KGC}\backslash\boldsymbol{C}| = 1$, invoke $\mathcal{S}^2(ssid, \boldsymbol{C}, \boldsymbol{L}, \bar{b}, \mathsf{pre\text{-}ext}_0)$. Once finished, check that no cheaters are identified. And then, rewind $\mathcal{Z}$ to the beginning procedure of $\mathcal{S}^2$, reselect the random value $\rho_{\bar{b}}$ and take $(ssid, \boldsymbol{C}, \boldsymbol{L}, \mathsf{pre\text{-}ext})$ as its input, as well as set the tuple $(\alpha_j, a_j, b_j, c_j, \delta_j, \zeta_j, \chi_j)_{j\neq\bar{b}}$.

2. Else, run $\mathcal{S}^2(ssid, \boldsymbol{C}, \boldsymbol{L}, \bar{b}, \mathsf{pre\text{-}ext})$ for the above cases.

Extracting. The environment $\mathcal{Z}$ injects the command $(\mathsf{extract}, ssid, \ell, ID, i)$ into $\mathcal{P}_i$, and controls $\boldsymbol{C} \subsetneq \boldsymbol{KGC}$. Sample $d_{\bar{b}} \leftarrow \mathbb{Z}_q^*$, as well as output $\boldsymbol{e}^{\backslash\bar{b}} = (e_j)_{j\neq\bar{b}}$, $\boldsymbol{x}^{\backslash\bar{b}} = (x_j)_{j\neq\bar{b}}$ and $\mathsf{aux} = (e_{\bar{b}}, d_{\bar{b}})$, where $e_k = s_k + \mathcal{H}(ID)/l$ for every $k \in [l]$.

1. Invoke $\mathcal{S}^3(ssid, \boldsymbol{C}, \boldsymbol{L}, \bar{b}, \boldsymbol{e}^{\backslash\bar{b}}, \boldsymbol{x}^{\backslash\bar{b}}, \mathsf{aux})$ for the extracting process and obtain ξ_i, $\{\overline{r}_j, \overline{\Gamma}_j\}_j$. Broadcast ξ_i to all and set $d'_{ID_{\bar{b}}} = (\sum \xi_i)^{-1} d_{\bar{b}} \bmod q$.

2. Hand over $\{(ssid, i, d'_{ID_i}, \alpha_i, a_i, b_i, c_i, \delta_i, \zeta_i, \chi_i)\}_{i\notin\boldsymbol{C}}$ to $\mathcal{Z}$, and obtain $\{d'_{ID_j}\}_{j\notin\boldsymbol{C}}$.

Pre-Signing. The environment $\mathcal{Z}$ injects the command $(\mathsf{pre\text{-}signing}, ssid, i)$ into $\mathcal{P}_i$, and controls the set of corrupted parties $\boldsymbol{C} \subsetneq \boldsymbol{D}$. Sample $S_0 \leftarrow \mathbb{G}_1$, $h_0 \leftarrow \mathbb{Z}_q^*$, and let $u_0 = e(S, Q_{ID})e(P, Q)^{-h_0}$, $Q_{ID} = \mathcal{H}(ID) \cdot Q + Q_{pub}$. Run $\mathcal{S}^4(ssid, \boldsymbol{C}, \boldsymbol{L}, u_0)$ and get outputs $\widehat{b}$, $\boldsymbol{L}$, rid', $\{Y_k\}_{k\in\boldsymbol{D}}$ and $\{r_k, \rho_k\}_{k\neq\widehat{b}}$. Sample $r_{\widehat{b}}, k_{\widehat{b}}, \rho_{\widehat{b}} \leftarrow \mathbb{Z}_q^*$, as well as output $\boldsymbol{r}^{\backslash\widehat{b}} = (r_j)_{j\neq\widehat{b}}$, $\boldsymbol{\rho}^{\backslash\widehat{b}} = (\rho_j)_{j\neq\widehat{b}}$, and $\mathsf{aux} = (r_{\widehat{b}}, k_{\widehat{b}}, \rho_{\widehat{b}})$. Then, make a call to the simulator $\mathcal{S}^3(ssid, \boldsymbol{C}, \boldsymbol{L}, \widehat{b}, \boldsymbol{e}^{\backslash\widehat{b}}, \boldsymbol{d}^{\backslash\widehat{b}}, \boldsymbol{x}^{\backslash\bar{b}}, \mathsf{aux})$ for the $\mathsf{pre\text{-}signing}$.

Signing. The environment $\mathcal{Z}$ injects the command $(\mathsf{sign}, ssid, m, i)$ into $\mathcal{P}_i$, and controls the set of corrupted parties $\boldsymbol{C} \subsetneq \boldsymbol{D}$.

1. Retrieve u and $\{\xi'_i, \xi''_i, \xi'''_i, \rho_i\}_{i\notin\boldsymbol{C}}$, set $h = \mathcal{H}(m||u)$.

2. Hand over $\{(ssid, i, v_i = \xi'_i + h \cdot \rho_i, \xi''_i, \xi'''_i)\}_{i\notin\boldsymbol{C}}$.

Dynamic Corruptions.

1. If the environment $\mathcal{Z}$ controls a player $\mathcal{P}_i$ that belongs to the set $\overline{\boldsymbol{H}}$ or $\widehat{\boldsymbol{H}}$, expose the secret state of the parties.

2. Else, rewind $\mathcal{Z}$ to the (†) step of $\mathcal{S}^1$ or $\mathcal{S}^4$, and then erase all relevant entries recorded in $\boldsymbol{L}$.

A.3 BLMQ forger $\mathcal{R}_2$

Let $\mathcal{R}_2$ a forger that can output valid signatures on messages that are not required. In addition, we denote the BLMQ public key by Q_{pub}. Let ctr be a counter which is initialized to 0. $\boldsymbol{L}$ stands for a list that the simulator maintains, storing the queries and answers. The interaction of $\mathcal{R}_2$ with $\mathcal{Z}$ is described as follows.

Oracle calls. When a request $(\mathsf{query}, x) = (\mathsf{query}, sid^*,\ rid^*,\ \dots)$ is notified, execute below:

1. If (sid^*, rid^*) is equal to (sid, rid), respond the message $(\mathsf{answer}, a = \mathcal{H}(x))$.
2. Otherwise, if $x = ([sid, j, \psi])$ is obtained in a rewinding process, set the random oracle, derive the private keys p, q, and store the corresponding tuple to $\boldsymbol{L}$.
3. Otherwise, if (x, a) belongs to $\boldsymbol{L}$, respond (answer, a). Then, select a random value a, respond (answer, a), as well as store the tuple (x, a) to $\boldsymbol{L}$.

Key generation. $\mathcal{Z}$ injects the command $(\mathsf{keygen}, sid = (\dots, \boldsymbol{KGC}), i)$ into $\mathcal{P}_i$, as well as controls the set of corrupted parties $\boldsymbol{C} \subsetneq \boldsymbol{KGC}$. Run $\mathcal{S}^1(sid, \boldsymbol{C}, \boldsymbol{L}, Q_{pub})$ and get outputs $\overline{b}$, $\boldsymbol{L}$, rid, $\{X_k, N_k\}_{k \in \boldsymbol{KGC}}$, $\{(p_k, q_k), x_k\}_{k \in \boldsymbol{KGC}}$ and $\{s_k\}_{k \neq \overline{b}}$.

Pre-Extracting. The environment $\mathcal{Z}$ injects the command $\big(\mathsf{pre\text{-}extract}, ssid = (\dots, sid, \boldsymbol{D}), \ell, i\big)$ into $\mathcal{P}_i$, and controls the set of corrupted parties $\boldsymbol{C} \subsetneq \boldsymbol{KGC}$. Run $\mathcal{S}^2(ssid, \boldsymbol{C}, \boldsymbol{L}, \perp)$. If $\mathcal{S}^2$ ends, rewind $\mathcal{Z}$ to the beginning procedure of $\mathcal{S}^2$, as well as reselect $\rho_{\overline{b}}$. Set $\overline{b}$ and $(\alpha_j, a_j, b_j, c_j, \delta_j, \zeta_j, \chi_j)_{j \neq \overline{b}}$.

Extracting. The environment $\mathcal{Z}$ injects the command $(\mathsf{extract}, ssid, \ell, ID, i)$ into $\mathcal{P}_i$, and controls $\boldsymbol{C} \subsetneq \boldsymbol{KGC}$. Sample $\alpha, a, b \leftarrow \mathbb{Z}_q^*$, as well as output $\boldsymbol{e}^{\setminus \overline{b}} = (e_j)_{j \neq \overline{b}}$, $\boldsymbol{x}^{\setminus \overline{b}} = (x_j)_{j \neq \overline{b}}$, $\mathsf{aux} = (\alpha, a, b)$, where $e_k = s_k + \mathcal{H}(ID)/l$ for every $k \neq \overline{b}$.

1. Invoke $\mathcal{S}^3(ssid, \boldsymbol{C}, \boldsymbol{L}, \boldsymbol{e}^{\setminus \overline{b}}, \boldsymbol{x}^{\setminus \overline{b}}, \mathsf{aux})$, and obtain ξ_i. Broadcast ξ_i to all and set $d'_{ID_{\overline{b}}} = (\sum \xi_i)^{-1} d_{\overline{b}} \bmod q$.
2. Hand over $\{(ssid, i, d'_{ID_i}, \alpha_i, a_i, b_i, c_i, \delta_i, \zeta_i, \chi_i)\}_{i \notin \boldsymbol{C}}$ and obtain $\{d'_{ID_j}\}_{j \notin \boldsymbol{C}}$.

Pre-Signing. The environment $\mathcal{Z}$ injects the command $(\mathsf{pre\text{-}signing}, ssid, i)$ into $\mathcal{P}_i$, and controls the set of corrupted parties $\boldsymbol{C} \subsetneq \boldsymbol{D}$.

1. Call to the BLMQ oracle to get a point $u \in \mathbb{G}_T$. Run $\mathcal{S}^4(ssid, \boldsymbol{C}, \boldsymbol{L}, u)$ and obtain $\widehat{b}$, $\boldsymbol{L}$, rid', $\{Y_k\}_{k \in \boldsymbol{D}}$ and $\{r_k, \rho_k\}_{k \neq \widehat{b}}$.
2. Sample $\alpha, a, b \leftarrow \mathbb{Z}_q^*$, set $\mathsf{aux} = (\alpha, a, b)$ and run $\mathcal{S}^3(ssid, \boldsymbol{C}, \boldsymbol{L}, \mathsf{aux})$.

Signing. The environment $\mathcal{Z}$ injects the command $(\mathsf{sign}, ssid, m, i)$ into $\mathcal{P}_i$, and controls the set of corrupted parties $\boldsymbol{C} \subsetneq \boldsymbol{D}$.

1. Retrieve $(ssid, \ell, \eta_1^7, \eta_2^7, \eta_3^7)$ and $(ssid, \ell, u, \xi', \xi'_i, \xi'', \xi''_i,\ \xi''', \xi'''_i, \rho_i)$, for i belongs to $\widehat{\boldsymbol{H}}$.
2. In order to obtain signature (h, S), make use of a call to the BLMQ oracle. For $\mathcal{P}_i$ belongs to $\widehat{\boldsymbol{H}}$, v_i is considered to be prescribed value, as well as submit $(ssid, i, v_i, \xi''_i, \xi'''_i)$. For $\mathcal{P}_{\widehat{b}}$, set $v_{\widehat{b}} = (\xi' - \eta_1^7) + h\rho_{\widehat{b}}$, $\xi''_{\widehat{b}} = \xi'' - \eta_2^7$, $\xi'''_{\widehat{b}} = \xi''' - \eta_3^7$ and submit $(ssid, \widehat{b}, v_{\widehat{b}}, \xi''_{\widehat{b}}, \xi'''_{\widehat{b}})$.

Dynamic Corruptions.

1. If $\mathcal{Z}$ controls $\mathcal{P}_i \in \overline{\boldsymbol{H}}$ or $\mathcal{P}_i \in \widehat{\boldsymbol{H}}$, then expose the secret state of the parties.
2. Else, rewind $\mathcal{Z}$ to the (†) step of $\mathcal{S}^1$ or $\mathcal{S}^4$, and then erase all relevant entries recorded in $\boldsymbol{L}$.

B Standalone simulators

In this section, we denote the ZK-simulator of Π^{prt} by $\mathcal{S}^{\mathsf{prt}}$, where $\mathsf{prt} \in \{\mathsf{sch}, \mathsf{sch}^*\}$.

B.1 Initialization simulator ($\mathcal{S}^1$)

The simulator $\mathcal{S}^1(sid, \boldsymbol{C}, \boldsymbol{L}, Q_{pub})$ takes something as input, including a session identifier sid, a query-answer list $\boldsymbol{L}$, a group of players $\boldsymbol{C} \subsetneq \boldsymbol{KGC}$, the BLMQ public key Q_{pub}, and works below.

Round 1. Set $\mathsf{exp} = 0$ and select $\{V_i\}_{i \notin \boldsymbol{C}}$. Then, give $\mathcal{Z}$ the message (sid, i, V_i), where each $\mathcal{P}_i$ does not belong to $\boldsymbol{C}$.

Round 2. When obtaining (sid, j, V_j) for all $\mathcal{P}_j \in \boldsymbol{C}$ (†), do as follows and add the corresponding values to $\boldsymbol{L}$:

1. If exp is not equal to 0, all values are set according to the protocol, and output $\{(sid, i, rid_i, Q_i, X_i, A_i, B_i, N_i, u_i)\}_{i \notin \boldsymbol{C}}$.
2. Otherwise, select $\mathcal{P}_{\overline{b}} \leftarrow \boldsymbol{KGC} \backslash \boldsymbol{C}$, as well as set $\overline{H} = \boldsymbol{KGC} \backslash \boldsymbol{C} \cup \{\mathcal{P}_{\overline{b}}\}$, and execute below:
   (a) For $\mathcal{P}_i$ that belongs to $\overline{H}$, compute all entries according to the protocol and output $(sid, i, rid_i, Q_i, X_i, A_i, B_i, N_i, u_i)$.
   (b) For $\mathcal{P}_{\overline{b}}$ that is carefully selected, output $Q_{\overline{b}} = Q_{pub} - \sum_{j \neq \overline{b}} Q_j$. Then, run the simulator of the zero knowledge proof to get $\psi_{\overline{b}} = (A_{\overline{b}}, \dots) \leftarrow \mathcal{S}^{\mathsf{sch}}(Q_{\overline{b}}, \dots)$, and sample $X_{\overline{b}} \leftarrow \mathbb{G}_1$ and set $\tilde{\psi} = (B_{\overline{b}}, \dots) \leftarrow \mathcal{S}^{\mathsf{sch}}(X_{\overline{b}}, \dots)$. Furthermore, submit $(sid, \overline{b}, rid_{\overline{b}}, Q_{\overline{b}}, X_{\overline{b}}, A_{\overline{b}}, B_{\overline{b}}, N_{\overline{b}}, u_{\overline{b}})$.

Round 3. Once all requests $(sid, j, rid_j, Q_j, X_j, A_j, B_j, N_j, u_j)$ are correctly stored, output $\{\psi_j, \hat{\psi}_j, \tilde{\psi}_j\}_{j \in \boldsymbol{C}}$. Moreover, select $rid = \oplus_j rid_j$, as well as output $(sid, i, \psi_i, \hat{\psi}_i, \tilde{\psi}_i)_{i \notin \boldsymbol{C}}$. In addition, append the corresponding values to $\boldsymbol{L}$.

Output. If exp is not equal to 0, let exp become 1 and return (†) from **round 2** and discard the values stored in the list $\boldsymbol{L}$ from that moment on. Otherwise, extract $\{s_j, p_j, q_j, x_j\}_{j \notin \boldsymbol{C}}$, and then set $\overline{b}, \boldsymbol{L}, rid, \{s_k, p_k, q_k, x_k\}_{k \neq \overline{b}}$.

B.2 Triple-generation simulator ($\mathcal{S}^2$)

The simulator $\mathcal{S}^2(ssid, \boldsymbol{C}, \boldsymbol{L}, \overline{b}, \mathsf{pre\text{-}ext})$ takes something as input, including the session identifier $ssid$, a query-answer list $\boldsymbol{L}$, a group of malicious players $\boldsymbol{C} \subsetneq \boldsymbol{KGC}$, as well as additional information $\mathsf{pre\text{-}ext} = \perp$ or $\mathsf{pre\text{-}ext} = (\alpha_{\overline{b}}, a_{\overline{b}}, b_{\overline{b}}, N^*, C)$.

Round 1. For $\mathcal{P}_i$ that belongs to $\overline{\boldsymbol{H}}$, all values are set according to the protocol and send $(ssid, i, K_i, \hat{K}_i, \tilde{K}_i, G_i)$. For $\mathcal{P}_{\overline{b}}$, sample $G_{\overline{b}}$ randomly and set as follows:

1. If $\mathsf{pre\text{-}ext} = \perp$, set $N_{\overline{b}} = N^*$, $K_{\overline{b}} = \mathsf{enc}_{\overline{b}}(0)$, $\hat{K}_{\overline{b}} = \mathsf{enc}_{\overline{b}}(0)$ and $\tilde{K}_{\overline{b}} = \mathsf{enc}_{\overline{b}}(0)$. And then, sample $\Gamma_{\overline{b}}, \hat{\Gamma}_{\overline{b}}, \tilde{\Gamma}_{\overline{b}} \leftarrow \mathbb{G}_1$.
2. If $\mathsf{pre\text{-}ext} \neq \perp$, set $N_{\overline{b}} = N^*$, $K_{\overline{b}} = C^{\alpha_{\overline{b}}} \cdot \mathsf{enc}_{\overline{b}}(0)$, $\hat{K}_{\overline{b}} = C^{a_{\overline{b}}} \cdot \mathsf{enc}_{\overline{b}}(0)$, $\tilde{K}_{\overline{b}} = C^{b_{\overline{b}}} \cdot \mathsf{enc}_{\overline{b}}(0)$, $\Gamma_{\overline{b}} = \alpha_{\overline{b}} \cdot P$, $\hat{\Gamma}_{\overline{b}} = a_{\overline{b}} \cdot P$ and $\tilde{\Gamma}_{\overline{b}} = b_{\overline{b}} \cdot P$. Next, give $(ssid, \overline{b}, K_{\overline{b}}, \hat{K}_{\overline{b}}, \tilde{K}_{\overline{b}}, G_{\overline{b}})$, as well as store the corresponding entries.

Round 2. When a request $(ssid, j, K_j, \dots)$ is notified, execute below:

1. For $\mathcal{P}_i$ that belongs to $\overline{\boldsymbol{H}}$, all values are set according to the protocol, as well as output $(ssid, i, \Gamma_i, \hat{\Gamma}_i, \tilde{\Gamma}_i, \psi_i')$.
2. For $\mathcal{P}_{\overline{b}}$, invoke the zero-knowledge simulators $\psi_{\overline{b}}' \leftarrow \mathcal{S}^{\mathsf{sch}^*}(G_{\overline{b}}, \Gamma_{\overline{b}}, \hat{\Gamma}_{\overline{b}}, \tilde{\Gamma}_{\overline{b}}, P, \dots)$. Then, hand over $(ssid, \overline{b}, \Gamma_{\overline{b}}, \hat{\Gamma}_{\overline{b}}, \tilde{\Gamma}_{\overline{b}}, \psi_{\overline{b}}')$, as well as store the related calls.

Round 3. When a request $(ssid, j, \Gamma_j, \hat{\Gamma}_j, \tilde{\Gamma}_j, \psi_j')$ is notified, retrieve (α_j, a_j, b_j) and work below:

1. For $\mathcal{P}_i$ that belongs to $\overline{\boldsymbol{H}}$, all values are set according to the protocol, as well as output $(ssid, i, D_{j,i}, F_{j,i}, I_{j,i}, \hat{D}_{j,i}, \hat{F}_{j,i}, \hat{I}_{j,i}, \tilde{D}_{j,i}, \tilde{F}_{j,i}, \tilde{I}_{j,i}, \psi_{j,i})$.
2. For $\mathcal{P}_{\overline{b}}$, choose random tuples $\{\mu_{\ell,\overline{b}}, \hat{\mu}_{\ell,\overline{b}}, \tilde{\mu}_{\ell,\overline{b}}\}_{\ell \neq \overline{b}}$, as well as output $D_{\ell,\overline{b}} = \mathsf{enc}_\ell(\mu_{\ell,\overline{b}})$, $\hat{D}_{\ell,\overline{b}} = \mathsf{enc}_\ell(\hat{\mu}_{\ell,\overline{b}})$, $\tilde{D}_{\ell,\overline{b}} = \mathsf{enc}_\ell(\tilde{\mu}_{\ell,\overline{b}})$, and
   (a) If $\mathsf{pre\text{-}ext} = \perp$, set $F_{\ell,\overline{b}} = \mathsf{enc}_{\overline{b}}(0)$, $\hat{F}_{\ell,\overline{b}} = \mathsf{enc}_{\overline{b}}(0)$, $\tilde{F}_{\ell,\overline{b}} = \mathsf{enc}_{\overline{b}}(0)$, and $I_{\ell,\overline{b}}, \hat{I}_{\ell,\overline{b}}, \tilde{I}_{\ell,\overline{b}} \leftarrow \mathbb{G}_1$.
   (b) If $\mathsf{pre\text{-}ext} \neq \perp$, set $F_{\ell,\overline{b}} = C^{\alpha_\ell a_{\overline{b}} - \mu_{\ell,\overline{b}}} \cdot \mathsf{enc}_{\overline{b}}(0)$, $\hat{F}_{\ell,\overline{b}} = C^{\alpha_\ell b_{\overline{b}} - \hat{\mu}_{\ell,\overline{b}}} \cdot \mathsf{enc}_{\overline{b}}(0)$, $\tilde{F}_{\ell,\overline{b}} = C^{a_\ell b_{\overline{b}} - \tilde{\mu}_{\ell,\overline{b}}} \cdot \mathsf{enc}_{\overline{b}}(0)$, $I_{\ell,\overline{b}} = (\alpha_\ell a_{\overline{b}} - \mu_{\ell,\overline{b}}) \cdot P$, $I_{\ell,\overline{b}} = (\alpha_\ell b_{\overline{b}} - \hat{\mu}_{\ell,\overline{b}}) \cdot P$ and $\tilde{I}_{\ell,\overline{b}} = (a_\ell b_{\overline{b}} - \tilde{\mu}_{\ell,\overline{b}}) \cdot P$.

   Then, for $j \neq \overline{b}$, invoke the zero-knowledge simulators $\psi_{j,\overline{b}} \leftarrow \mathcal{S}^{\mathsf{sch}^*}(I_{j,\overline{b}}, \hat{I}_{j,\overline{b}}, \tilde{I}_{j,\overline{b}}, P, \dots)$. Next, hand over $(ssid, \overline{b}, D_{j,\overline{b}}, F_{j,\overline{b}}, I_{j,\overline{b}}, \hat{D}_{j,\overline{b}}, \hat{F}_{j,\overline{b}}, \hat{I}_{j,\overline{b}}, \tilde{D}_{j,\overline{b}}, \tilde{F}_{j,\overline{b}}, \tilde{I}_{j,\overline{b}}, \psi_{j,\overline{b}})$, as well as store the corresponding values.

Round 4. When a request $(ssid, j, D_{i,j}, F_{i,j}, I_{i,j}, \dots)$ is notified for $j \in \boldsymbol{C}$, $i \notin \boldsymbol{C}$, retrieve $\{\mu_{j,\overline{b}}, \hat{\mu}_{j,\overline{b}}, \tilde{\mu}_{j,\overline{b}}, \nu_{j,\overline{b}}, \hat{\nu}_{j,\overline{b}}, \tilde{\nu}_{j,\overline{b}}\}_{j \neq \overline{b}}$ from $\{F_{\overline{b},j}, \hat{F}_{\overline{b},j}, \tilde{F}_{\overline{b},j}\}_{j \in \boldsymbol{C}}$, as well as work below:

1. If $\mathsf{pre\text{-}ext} = \perp$, check that $\mathsf{dec}(D_{\overline{b},j}) = \nu_{j,\overline{b}} \bmod q$, $\mathsf{dec}(\hat{D}_{\overline{b},j}) = \hat{\nu}_{j,\overline{b}} \bmod q$ and $\mathsf{dec}(\tilde{D}_{\overline{b},j}) = \tilde{\nu}_{j,\overline{b}} \bmod q$ for $j \neq \overline{b}$, sample $\alpha, a, b \leftarrow \mathbb{Z}_q^*$, as well as compute:
   (a) $\eta^0 = \sum_{j \neq \overline{b}}(\alpha_j)$, $\eta^1 = \sum_{j \neq \overline{b}}(a_j)$ and $\eta^2 = \sum_{j \neq \overline{b}}(b_j)$.
   (b) $\eta^3 = \sum_{j,i \neq \overline{b}} a_i \alpha_j + \sum_{j \neq \overline{b}} \left(\mu_{j,\overline{b}} + \nu_{j,\overline{b}}\right)$.
   (c) $\eta^4 = \sum_{j,i \neq \overline{b}} b_i \alpha_j + \sum_{j \neq \overline{b}} \left(\hat{\mu}_{j,\overline{b}} + \hat{\nu}_{j,\overline{b}}\right)$.
   (d) $\eta^5 = \sum_{j,i \neq \overline{b}} b_i a_j + \sum_{j \neq \overline{b}} \left(\tilde{\mu}_{j,\overline{b}} + \tilde{\nu}_{j,\overline{b}}\right)$.
   (e) $\alpha_{\overline{b}} = \alpha - \eta^0$, $a_{\overline{b}} = a - \eta^1$ and $b_{\overline{b}} = b - \eta^2$.

(f) For $\ell \neq \bar{b}$, $\mu_{\bar{b},\ell} = \alpha_{\bar{b}} a_\ell - \nu_{\ell,\bar{b}}$, $\hat{\mu}_{\bar{b},\ell} = \alpha_{\bar{b}} b_\ell - \hat{\nu}_{\ell,\bar{b}}$ and $\tilde{\mu}_{\bar{b},\ell} = a_{\bar{b}} b_\ell - \tilde{\nu}_{\ell,\bar{b}}$; $J_{\ell,\bar{b}} = \mu_{\bar{b},\ell} \cdot P$, $\hat{J}_{\ell,\bar{b}} = \hat{\mu}_{\bar{b},\ell} \cdot P$ and $\tilde{J}_{\ell,\bar{b}} = \tilde{\mu}_{\bar{b},\ell} \cdot P$.

(g) $\delta_{\bar{b}} = a\alpha - \eta^3$, $\zeta_{\bar{b}} = b\alpha - \eta^4$, and $c_{\bar{b}} = ba - \eta^5$.

(h) $\Sigma_{\bar{b}} = \delta_{\bar{b}} \cdot P - \alpha_{\bar{b}} \cdot (\hat{\Gamma}_{\bar{b}} + \eta^1 \cdot P)$, $\hat{\Sigma}_{\bar{b}} = \zeta_{\bar{b}} \cdot P - \alpha_{\bar{b}} \cdot (\tilde{\Gamma}_{\bar{b}} + \eta^2 \cdot P)$ and $\tilde{\Sigma}_{\bar{b}} = c_{\bar{b}} \cdot P - a_{\bar{b}} \cdot (\tilde{\Gamma}_{\bar{b}} + \eta^2 \cdot P)$ (this step requires rewinding $\mathcal{Z}$ to item (a) of round 1, satisfying that $\alpha P = \Gamma_{\bar{b}} + \eta^0 P$, $aP = \hat{\Gamma}_{\bar{b}} + \eta^1\ P$ and $bP = \tilde{\Gamma}_{\bar{b}} + \eta^2 P$, and then proceeds as prescribed).

Next, sample (U_i, W_i), and send $\{(ssid, i, U_i,\ W_i, J_{j,i}, \hat{J}_{j,i}, \tilde{J}_{j,i})\}_{i \notin \boldsymbol{C}}$, submit $(ssid, \bar{b}, J_{j,\bar{b}}, \hat{J}_{j,\bar{b}}, \tilde{J}_{j,\bar{b}})$.

2. Else, check that $\mathsf{dec}(D_{\bar{b},j}) = \alpha_{\bar{b}} \cdot \mathsf{dec}(\hat{K}_j) - \nu_{j,\bar{b}}$, $\mathsf{dec}(\hat{D}_{\bar{b},j}) = \alpha_{\bar{b}} \cdot \mathsf{dec}(\tilde{K}_{\bar{b}}) - \hat{\nu}_{j,\bar{b}}$ and $\mathsf{dec}(\tilde{D}_{\bar{b},j}) = a_{\bar{b}} \cdot \mathsf{dec}(\tilde{K}_{\bar{b}}) - \tilde{\nu}_{j,\bar{b}}$, and compute

   (a) $\delta_{\bar{b}} = \alpha_{\bar{b}} a_{\bar{b}} + \sum_{j \neq \bar{b}} \big((\alpha_{\bar{b}} a_j - \nu_{j,\bar{b}}) + (\alpha_j a_{\bar{b}} - \mu_{j,\bar{b}})\big)$, $\zeta_{\bar{b}} = \alpha_{\bar{b}} b_{\bar{b}} + \sum_{j \neq \bar{b}} \big((\alpha_{\bar{b}} b_j - \hat{\nu}_{j,\bar{b}}) + (\alpha_j b_{\bar{b}} - \hat{\mu}_{j,\bar{b}})\big)$ and $c_{\bar{b}} = a_{\bar{b}} b_{\bar{b}} + \sum_{j \neq \bar{b}} \big((a_{\bar{b}} b_j - \tilde{\nu}_{j,\bar{b}}) + (a_j b_{\bar{b}} - \tilde{\mu}_{j,\bar{b}})\big)$; $\Sigma_{\bar{b}} = \delta_{\bar{b}} \cdot P - \alpha_{\bar{b}} \cdot \hat{\Gamma}$, $\hat{\Sigma}_{\bar{b}} = \zeta_{\bar{b}} \cdot P - \alpha_{\bar{b}} \cdot \tilde{\Gamma}$ and $\tilde{\Sigma} = c_{\bar{b}} \cdot P - a_{\bar{b}} \cdot \tilde{\Gamma}$.

   (b) For $\ell \neq \bar{b}$, $J_{\ell,\bar{b}} = (\alpha_{\bar{b}} a_\ell - \nu_{\ell,\bar{b}}) \cdot P$, $\hat{J}_{\ell,\bar{b}} = (\alpha_{\bar{b}} b_\ell - \hat{\nu}_{\ell,\bar{b}}) \cdot P$ and $\tilde{J}_{\ell,\bar{b}} = (a_{\bar{b}} b_\ell - \tilde{\nu}_{\ell,\bar{b}}) \cdot P$.

   Then, sample (U_i, W_i) and send $\{(ssid, i, U_i,\ W_i, J_{j,i}, \hat{J}_{j,i}, \tilde{J}_{j,i})\}_{i \notin \boldsymbol{C}, j \in \boldsymbol{KGC}}$.

Round 5. Once a request $(ssid, j, U_j, W_j, J_{i,j}, \hat{J}_{i,j}, \tilde{J}_{i,j})$ is notified, and execute below:

1. If $\mathsf{pre\text{-}ext} = \bot$, check that $J_{\bar{b},j} = \mu_{j,\bar{b}} \cdot P$, $\hat{J}_{\bar{b},j} = \hat{\mu}_{j,\bar{b}} \cdot P$ and $\tilde{J}_{\bar{b},j} = \tilde{\mu}_{j,\bar{b}} \cdot P$ for $j \neq \bar{b}$.

2. Else, check that $J_{\bar{b},j} + I_{j,\bar{b}} = a_{\bar{b}} \cdot \Gamma_j$, $\hat{J}_{\bar{b},j} + \hat{I}_{j,\bar{b}} = b_{\bar{b}} \cdot \Gamma_j$ and $\tilde{J}_{\bar{b},j} + \tilde{I}_{j,\bar{b}} = b_{\bar{b}} \cdot \hat{\Gamma}_j$. Then, set $\Phi_i = c_i \cdot P$ for $i \in \overline{\boldsymbol{H}}$, hand over $\{(ssid, i, \Sigma_i, \hat{\Sigma}_i, \tilde{\Sigma}_i, \iota_i, \Phi_i, \hat{\psi}_i)\}_{i \in \overline{\boldsymbol{H}}}$ and store the related value, where ι_i is generated as required.

Round 6. Once a request $(ssid, j, \Sigma_j, \hat{\Sigma}_j, \tilde{\Sigma}_j)$ is notified, verify $\sum_\ell \Sigma_\ell = 0$, $\sum_\ell \hat{\Sigma}_\ell = 0$ and $\sum_\ell \tilde{\Sigma}_\ell = 0$. If the first equation fails, open the related values $\{\alpha_i, a_i, \delta_i\}_{i \in \overline{\boldsymbol{H}}}$.

1. For $\mathcal{P}_i$ that belongs to $\overline{\boldsymbol{H}}$, all values are set according to the protocol, output $\{(ssid, i, \overline{D}_{j,i}, \overline{F}_{j,i}, \overline{I}_{j,i}, \overline{\psi}_{j,i})\}_{j \in \boldsymbol{KGC}, i \in \boldsymbol{H}}$.

2. For $\mathcal{P}_{\bar{b}}$, sample $\{\overline{\mu}_{\ell,\bar{b}}\}_{\ell \neq \bar{b}}$, and set $\overline{D}_{\bar{b}}(\overline{\mu}_{\ell,\bar{b}})$, and do

   (a) If $\mathsf{pre\text{-}ext} = \bot$, set $\overline{F}_{\ell,\bar{b}} = \mathsf{enc}_{\bar{b}}(0)$ and $\overline{I}_{\ell,\bar{b}} \leftarrow \mathbb{G}_1$.

   (b) Else, set $\overline{F}_{\ell,\bar{b}} = C^{\alpha_\ell c_{\bar{b}} - \overline{\mu}_{\ell,\bar{b}}} \cdot \mathsf{enc}(0)$ and $\overline{I}_{\ell,\bar{b}} = (\alpha_\ell c_{\bar{b}} - \overline{\mu}_{\ell,\bar{b}}) \cdot P$.

   Then, for $j \neq \bar{b}$, invoke the zero-knowledge simulators $\overline{\psi}_{j,\bar{b}} \leftarrow \mathcal{S}^{\mathsf{sch}}(\overline{I}_{j,\bar{b}}, P, \dots)$. Furthermore, hand over $(ssid, \bar{b}, \overline{D}_{j,\bar{b}}, \overline{F}_{j,\bar{b}}, \overline{I}_{j,\bar{b}}, \overline{\psi}_{j,\bar{b}})$ for j is not equal to $\bar{b}$, and store the corresponding values.

Round 7. When a request $(ssid, j, \overline{D}_{i,j}, \overline{F}_{i,j}, \overline{I}_{i,j}, \overline{\psi}_{i,j})$ is notified from $j \in \boldsymbol{C}$, retrieve $(\alpha_j, a_j, b_j, c_j, a, b)$ as well as $\{\overline{\mu}_{j,\bar{b}}, \overline{\nu}_{j,\bar{b}}\}_{j \neq \bar{b}}$ from $\{\overline{F}_{\bar{b}}\}_{j \in \boldsymbol{C}}$, and do and add the related values to $\boldsymbol{L}$:

1. If $\mathsf{pre\text{-}ext} = \bot$, check that $\mathsf{dec}(\overline{D}_{\bar{b},j}) = \overline{\nu}_{j,\bar{b}} \bmod q$ for j is not equal to $\bar{b}$, and compute

   (a) $\eta^6 = \sum_{j, i \neq \bar{b}} \alpha_i c_j + \sum_{j \neq \bar{b}} \big(\overline{\mu}_{j,\bar{b}} + \overline{\nu}_{j,\bar{b}}\big)$, $\chi_{\bar{b}} = \alpha ab - \eta^6$; $\overline{\Sigma}_{\bar{b}} = \chi_{\bar{b}} \cdot P - \alpha_{\bar{b}} \cdot ab \cdot P$.

(b) For $\ell \neq \bar{b}$, $\overline{\mu}_{\bar{b},\ell} = c_{\bar{b}}\alpha_\ell - \overline{\nu}_{j,\bar{b}}$ and $\overline{J}_{\ell,\bar{b}} = \overline{\mu}_{\bar{b},\ell} \cdot P$.

Then, sample V_i and send $\{(ssid, i, V_i, \overline{J}_{j,i})\}_{i \notin \boldsymbol{C}}$, Next, output $(ssid, \bar{b}, \overline{J}_{j,\bar{b}})$, for j is equal to $\bar{b}$.

2. Else, check that $\mathsf{dec}(\overline{D}_{\bar{b},j}) = b_{\bar{b}} \cdot \mathsf{dec}(\hat{K}_j) - \overline{\nu}_{j,\bar{b}}$, for j is equal to $\bar{b}$, as well as compute

(a) $\chi_{\bar{b}} = \alpha_{\bar{b}}c_{\bar{b}} + \sum_{j \neq \bar{b}} \left((\alpha_{\bar{b}}c_j - \overline{\nu}_{j,\bar{b}}) + (\alpha_j c_{\bar{b}} - \overline{\mu}_{j,\bar{b}})\right)$ and $\overline{\Sigma}_{\bar{b}} = \chi_{\bar{b}} \cdot P - \alpha_{\bar{b}} \cdot \Phi$.

(b) For $\ell \neq \bar{b}$, $\overline{J}_{\ell,\bar{b}} = (\alpha_{\bar{b}}c_\ell - \overline{\nu}_{\ell,\bar{b}}) \cdot P$.

Then, sample V_i and send $\{(ssid, i, V_i, \overline{J}_{j,i})\}_{i \notin \boldsymbol{C}, j \in \boldsymbol{KGC}}$.

Round 8. Once a request $(ssid, j, V_j, \overline{J}_{i,j})_{j \in \boldsymbol{C}, i \notin \boldsymbol{C}}$ is notified, and execute below:

1. If $\mathsf{pre\text{-}ext} = \perp$, check that $\overline{J}_{\bar{b},j} = \overline{\mu}_{j,\bar{b}} \cdot P$.

2. Else, check that $\overline{J}_{\bar{b},j} + \overline{I}_{j,\bar{b}} = c_{\bar{b}} \cdot \Gamma_j$. Then, output $\{(ssid, i, \overline{\Sigma}_i, \varpi_i)\}_{i \in \overline{\boldsymbol{H}}}$.

Output. Once a request $(ssid, j, \overline{\Sigma}_j, \varpi_j)$ is notified, verify that $\sum_\ell \overline{\Sigma}_\ell = 0$. If the verification fails, open the related values $\{\alpha_i, c_i, \chi_i\}_{i \in \overline{\boldsymbol{H}}}$.

1. If $\mathsf{pre\text{-}ext} = \perp$, output $(ssid, \ell, \eta^0, \ldots, \eta^6)$ and $(ssid, \ell, \alpha_i, a_i, b_i, c_i, \delta_i, \xi_i, \chi_i)_{i \in \overline{\boldsymbol{H}}}$.

2. Else, output $(ssid, \ell, \alpha_i, a_i, b_i, c_i, \delta_i, \xi_i, \chi_i)_{i \notin \boldsymbol{C}}$.

B.3 Multiplication simulator ($\mathcal{S}^3$)

The simulator $\mathcal{S}^3(ssid, \boldsymbol{C}, \boldsymbol{L}, \mathsf{rep}, \boldsymbol{e}^{\backslash\mathsf{rep}}, \boldsymbol{d}^{\backslash\mathsf{rep}}, \boldsymbol{x}^{\backslash\mathsf{rep}}, \mathsf{aux})$ takes something as input, including the session identifier $ssid$, a query-answer list $\boldsymbol{L}$, a group of corrupted players $\boldsymbol{C} \subsetneq \boldsymbol{KGC}$ or $\boldsymbol{C} \subsetneq \boldsymbol{D}$, an index $\mathsf{rep} \in \{\bar{b}, \widehat{b}\}$ and $\boldsymbol{e}^{\backslash\mathsf{rep}} = (e_j)_{j \neq \mathsf{rep}}$, $\boldsymbol{d}^{\backslash\mathsf{rep}} = (d_j)_{j \neq \mathsf{rep}}$, $\boldsymbol{x}^{\backslash\mathsf{rep}} = (x_j)_{j \neq \mathsf{rep}}$ such that $\mathcal{P}_{\mathsf{rep}} \notin \boldsymbol{C}$ and additional information $\mathsf{aux} = (\alpha, a, b)$ or $\mathsf{aux} = (e_{\mathsf{rep}}, d_{\mathsf{rep}})$. Let exp' be a symbol, initialized as 0.

Round 1. If exp' is not equal to 0, all values are set according to the protocol, as well as output $\{(ssid, i, \mu_i, \nu_i)\}_{i \notin \boldsymbol{C}}$ to $\mathcal{Z}$. Otherwise, retrieve $(ssid, \ell, \eta^0, \ldots, \eta^6)$ and $\{(ssid, \ell, \alpha_i, a_i, b_i, c_i, \delta_i, \xi_i, \chi_i)\}_{i \in \overline{\boldsymbol{H}} \text{ or } \widehat{\boldsymbol{H}}}$, and do:

1. If $\mathsf{aux} = (\alpha, a, b)$, set

$$\begin{cases} e_{\mathsf{rep}} = a + u - \sum_\ell e_\ell, & a_{\mathsf{rep}} = a - \eta^1 \\ d_{\mathsf{rep}} = b + \nu - \sum_\ell d_\ell, & b_{\mathsf{rep}} = b - \eta^2 \\ \mu_{\mathsf{rep}} = e_{\mathsf{rep}} - a_{\mathsf{rep}}, & \nu_{\mathsf{rep}} = d_{\mathsf{rep}} - b_{\mathsf{rep}} \end{cases}$$

Then, output $(ssid, i, \mu_i, \nu_i)_{i \notin \boldsymbol{C}}$.

2. Else, set

$$\begin{cases} e = e_{\mathsf{rep}} + \sum_\ell e_\ell, & a_{\mathsf{rep}} = e - \mu - \eta^1 \\ d = d_{\mathsf{rep}} + \sum_\ell d_\ell, & b_{\mathsf{rep}} = d - \nu - \eta^2 \\ \mu_{\mathsf{rep}} = e_{\mathsf{rep}} - a_{\mathsf{rep}}, & \nu_{\mathsf{rep}} = d_{\mathsf{rep}} - b_{\mathsf{rep}} \end{cases}$$

Then, output $(ssid, i, \mu_i, \nu_i)_{i \notin \boldsymbol{C}}$.

Round 2. Once a request $(ssid, j, \mu_j, \nu_j)$ is notified, and execute below:

1. If $\mathsf{aux} = (\alpha, a, b)$, set $\eta^7 = \xi_{\boldsymbol{C}} = \eta^5 + \mu\eta^2 + \nu\eta^1$ and $\varkappa_{\boldsymbol{C}} = \eta^6 + \mu\eta^4 + \nu\eta^3 + \mu\nu\eta^0$.
   (a) If $\sum_\ell \mu_\ell$ is equal to μ and $\sum_\ell \nu_\ell = \nu$,
      i. Set $\xi_{\mathsf{rep}} = ab - \eta^5 + \mu(b - \eta^2) + \nu(a - \eta^1)$ and $\varkappa_{\mathsf{rep}} = \alpha ab - \eta^6 + \mu(\alpha b - \eta^4) + \nu(\alpha a - \eta^3) + \mu\nu(\alpha - \eta^0) \bmod q$.
      ii. Submit $(ssid, i, \xi_i \cdot P)_{i \notin \boldsymbol{C}}$.
   (b) Else, sample $\xi \leftarrow \mathbb{Z}_q^*$, and set $\xi_{\mathsf{rep}} = \alpha\xi - \xi_{\boldsymbol{C}} - \mu\nu$, $\varkappa_{\mathsf{rep}} = \alpha\xi - \varkappa_{\boldsymbol{C}}$. Then, output $(ssid, i, \xi_i \cdot P)_{i \notin \boldsymbol{C}}$.
2. Otherwise, set $\xi = ed$ and send $(ssid, i, \xi_i \cdot P)_{i \notin \boldsymbol{C}}$.

Round 3. Once a request $(ssid, j, \xi_j \cdot P)$ is notified, sample $\{M_i\}_{i \notin \boldsymbol{C}}$, as well as output $(ssid, i, M_i)$, for i does not belong to $\boldsymbol{C}$.

Round 4. Once a request $(ssid, j, M_j)_{j \in \boldsymbol{C}}$ is notified, and execute below:

1. If $\mathsf{aux} = (\alpha, a, b)$ and $\sum_\ell \mu_\ell = \mu$ and $\sum_\ell \nu_\ell = \nu$,
   (a) Set $\xi = (a + \mu)(b + \nu)$, $\Lambda_{\mathsf{rep}} = \varkappa_{\mathsf{rep}} \cdot P - (\alpha - \eta^0)\xi \cdot P$.
   (b) Send $\{(ssid, i, \Lambda_i, \varsigma_i, \overline{r}_i, \overline{\Gamma}_i)\}_{i \notin \boldsymbol{C}}$ and add the corresponding values to $\boldsymbol{L}$.
2. Else, send $(ssid, i, \Lambda_i, \varsigma_i, \overline{r}_i, \overline{\Gamma}_i)$ for $i \notin \boldsymbol{C}$.

Output. When receiving $(ssid, j, \Lambda_j, \varsigma_j, \overline{r}_j, \overline{\Gamma}_j)_{j \in \boldsymbol{C}}$, verify $\sum_\ell \Lambda_\ell = 0$. If the equation fails, check that $\overline{\Gamma}_j = \xi_j \cdot P + \overline{r}_j \cdot X$ for $j \neq i$; Else, work below:

1. If aux is equal to (α, a, b), output $(ssid, \ell, \eta^7)$ and $(ssid, \ell, \xi, \xi_i)$ for $i \in \overline{\boldsymbol{H}}$ or $\widehat{\boldsymbol{H}}$ and $\{\overline{r}_j, \overline{\Gamma}_j\}_j$.
2. Else, set $\xi = ed$ and output $(ssid, \ell, \xi, \xi_i)_{i \notin \boldsymbol{C}}$.

B.4 Input simulator ($\mathcal{S}^4$)

The simulator $\mathcal{S}^4(ssid, \boldsymbol{C}, \boldsymbol{L}, u)$ takes something as input, including the session identifier $ssid$, a query-answer list $\boldsymbol{L}$, a group of malicious parties $\boldsymbol{C} \subsetneq \boldsymbol{D}$, the nonce u and works as follows.

Round 1. Set $\mathsf{exp}'' = 0$, and then select $\{V_i\}_{i \notin \boldsymbol{C}}$, as well as send $(ssid, i, V_i)$, for i does not belong to $\boldsymbol{C}$.

Round 2. Once a request $(ssid, j, V_j)$ is notified (†), work below and add the corresponding values to $\boldsymbol{L}$:

1. If exp'' is equal to 0, all values are set according to the protocol, as well as output $\{(ssid, i, rid'_i, R_i, Y_i, \varsigma_i)\}_{i \notin \boldsymbol{C}}$.
2. Otherwise, sample $\mathcal{P}_{\widehat{b}} \leftarrow \boldsymbol{D} \backslash \boldsymbol{C}$, set $\widehat{H} = \boldsymbol{D} \backslash \boldsymbol{C} \cup \{\mathcal{P}_{\widehat{b}}\}$, as well as work below:
   (a) For $\mathcal{P}_i$ that belongs to $\widehat{H}$, all values are set according to the protocol, and submit $(ssid, i, rid'_i, R_i, Y_i, \varsigma_i)$.
   (b) For special party $\mathcal{P}_{\widehat{b}}$ that is carefully chosen, compute $R_{\widehat{b}} = u / \prod_{j \neq \widehat{b}} R_j$. Then, run ZK simulator $\psi_{\overline{b}} = (A_{\widehat{b}}, \dots) \leftarrow \mathcal{S}^{\mathsf{sch}}(R_{\widehat{b}}, \dots)$, sample $Y_{\widehat{b}} \leftarrow \mathbb{G}_1$, and set $\widehat{\psi} = (B_{\widehat{b}}, \dots) \leftarrow \mathcal{S}^{\mathsf{sch}}(Y_{\widehat{b}}, \dots)$. Moreover, hand over $(ssid, \widehat{b}, rid'_{\widehat{b}}, R_{\widehat{b}}, Y_{\widehat{b}}, A_{\widehat{b}}, B_{\widehat{b}}, \varsigma_{\widehat{b}})$ to $\mathcal{Z}$.

Round 3. Once all requests $(ssid, j, rid'_j, R_j, Y_j, A_j, B_j, \varsigma_j)$ are obtained, store $\{\psi_j, \hat{\psi}_j\}_{j \in \boldsymbol{C}}$ into $\boldsymbol{E}''$, as well as execute below :

1. Put $rid' = \oplus_j rid'_j$, and submit $(ssid, i, \psi_i, \hat{\psi}_i)_{i \notin \boldsymbol{C}}$.
2. Store the related values.

Output. If exp'' is equal to $= 0$, let exp'' become 1, and return to (†) from round 2 and discard the values stored into $\boldsymbol{L}$ from that moment on. Otherwise, extract $\{r_j, \rho_j\}_{j \notin \boldsymbol{C}}$, and then output $\widehat{b}, \boldsymbol{L}, \boldsymbol{Y}, rid', \{r_k, \rho_k\}_{k \neq \widehat{b}}$.