Robot: Robust Threshold BBS+ in Two Rounds

Guofeng Tang[1], Tian Qiu[2⋆], Bowen Jiang[1], Haiyang Xue[1], Guomin Yang[1], Man Ho Au[3], Robert H. Deng[1], and Kwok-Yan Lam[2]

[1] Singapore Management University, tang.guofeng789@gmail.com, bowen.jiang.2024@phdcs.smu.edu.sg,{haiyangxue,gmyang,robertdeng}@smu.edu.sg
[2] Nanyang Technological University, qtautumn6@gmail.com,kwokyan.lam@ntu.edu.sg
[3] The Hong Kong Polytechnic University, man-ho-allen.au@polyu.edu.hk

Abstract. The BBS+ signature scheme is a widely used foundation for anonymous credential systems. It is favored for its support of selective disclosure and its efficiency in proving credential possession. However, in traditional settings, credentials are typically issued by a single authority, creating a single point of failure and potential security risk. This limitation can be mitigated by adopting a distributed variant, known as the threshold BBS+ scheme.

In this work, we present *Robot*, the first two-round threshold BBS+ signature scheme. Robot is round-minimal and achieves robustness, ensuring that every signing execution successfully completes as long as there exist $t+1$ parties behaving honestly. To achieve this, we employ a threshold verifiable random function (TVRF) to robustly generate the public nonces within a single round. Specifically, we utilize an efficient DDH-based TVRF construction, which not only provides our scheme with a round advantage but also enhances its overall performance. Then, by carefully invoking the threshold Castagnos-Laguillaumie and threshold ElGamal homomorphic encryptions, we complete all remaining non-linear operations within the second round.

Asymptotically, Robot achieves a constant per-party upload communication and linear computation overhead with respect to the number of signers. Compared with the four-round robust scheme of Wong et al. (NDSS'24, WMC24), which has the same asymptotic complexity, Robot achieves a smaller constant communication cost (2.02 KB vs. 3.23 KB) and nearly halves the runtime. Compared with the three-round robust scheme of Tang and Xue (S&P'25, TX25), which has linear communication overhead, Robot exhibits better communication and computational efficiency when the number of signers is five or more.

1 Introduction

In 2006, Au, Susilo, and Mu introduced the BBS+ signature [2], a short and efficient digital signature scheme based on bilinear pairings. It allows a vector of messages to be signed at once, and the signature size is independent of the number of messages. One of its core features is called *selective disclosure*, which means a user can selectively disclose some messages while proving that it has a valid signature, without leaking any other information. This feature allows it to serve as anonymous credentials [18], and it has been used in many other privacy-preserving protocols, such as Direct Anonymous Attestation (DAA [10]), k-Times Anonymous Authentication (k-TAA [2]), Enhanced Privacy ID (EPID [9]), etc. In recent years, it has also gained increasing attention from academia [48], industry [39,41], and active standardization initiatives by W3C Verifiable Credentials Group [31] and IETF [38].

However, the conventional deployment of BBS+ signatures in credential systems relies on a single issuer/signer, which introduces vulnerabilities such as a single point of failure and key compromise. To mitigate these risks, threshold BBS+ signatures have been studied [25]. It extends the standard BBS+ scheme by distributing the signing key among multiple authorities so that no single party can sign alone; instead, at least $t+1$ out of n parties must cooperate to generate a valid signature. Thus, it can significantly increase the security and availability of secret keys.

In threshold (or multi-party) signature schemes, the number of communication *rounds* is a crucial metric. High round complexity often results in significant latency, limiting the practicality of such protocols. Another important metric affecting practical deployment is *robustness* (also known

⋆ Co-first Author.

as guaranteed output delivery [1]). It can guarantee that the protocol always produces a valid signature as long as a sufficient number of parties remain honest. In contrast to security with abort (where parties abort upon malicious behavior) or identifiable abort (where the adversary can be identified after abort), robustness maximizes the guarantee that the computation of honest parties is not wasted. Although achieving robustness requires an honest-majority setting ($n \geq 2t+1$) [19] and a broadcast channel [20], it aims for the "best-of-both-worlds" security: it is secure with abort under a dishonest majority and robust under an honest majority.

Prior Works. Existing threshold BBS+ schemes rely on multiplicative-to-additive conversion [25,47], threshold Castagnos–Laguillaumie encryption [49], or pseudorandom correlation generators [26].

To the best of our knowledge, the round-minimal construction is due to Doerner et al. [25], who presented the first three-round threshold BBS+ (called DKLST23 hereafter). DKLST23 is based on Multiplicative-to-Additive (MtA) conversion (also known as Oblivious Linear Evaluation) using oblivious transfer (OT). The OT-based protocol can achieve very fast running time, but it cannot provide robustness and incurs rapidly growing linear per-party communication (over $49.7n$ KB for n parties). Recently, Tang and Xue [47] explored robustness within three rounds (denoted by TX25 hereafter). It leverages another type of MtA using Castagnos-Laguillaumie (CL) homomorphic encryption [16]. Since it requires $O(n^2)$ MtA executions for n parties, and robust protocols involve checking the correctness of each MtA instance, TX25 requires $O(n^2)$ per-party computation.

Inspired by the construction of MPC from threshold homomorphic encryption (THE) in [8], Wong et al. [49] (hereafter WMC24) develop a *robust* threshold BBS+ scheme based on threshold CL encryption, augmented with suitable zero-knowledge proofs (ZKPs). Compared with the two MtA-based works above, WMC24 achieves $O(1)$ per-party communication but requires four rounds.

Another threshold BBS+ scheme, proposed by Faust et al. [26], is based on a pseudorandom correlation generator (PCG). Although their scheme is non-interactive, it requires a trusted setup to distribute the PCG seed, whereas the other works do not. We do not discuss it further, as our focus is on threshold signatures without a trusted setup. As a concurrent work, Wu et al. [51] proposed a two-round, two-party BBS+ construction. However, their approach cannot be straightforwardly extended to the general t-out-of-n threshold setting.

Motivation. A comparison of state-of-the-art threshold BBS+ schemes without a trusted setup is provided in Table 1, covering the number of rounds, robustness, and communication and computation costs. In summary, existing (general t-out-of-n) threshold BBS+ schemes without a trusted setup require at least three rounds, even when robustness is not achieved.

Thus, in this paper, we investigate whether a threshold BBS+ scheme can be constructed with *fewer rounds* while achieving *robustness* and incurring *low communication* overhead.

1.1 Contributions

Schemes	Type	Rounds	Robust	Comm.	Comp.
DKLST23 [25]	OT-MtA	3	✗	$O(n)$	$O(n)$
WMC24 [49]	THE	4	✓	$O(1)$	$O(n)$
TX25 [47]	CL-MtA	3	✓	$O(n)$	$O(n^2)$
Ours	**THE, TVRF**	**2**	✓	$\mathbf{O(1)}$	$\mathbf{O(n)}$

Table 1: Comparisons of Threshold BBS+ with trustless setup

As illustrated in Table 1, we present a round-minimal threshold BBS+ construction *Robot*, which has the following properties:

- It requires only two rounds of communication.

- It achieves "best-of-both-worlds": security with abort under a dishonest majority and robustness under an honest majority.
- It offers scalability, featuring constant sending communication cost and linear computational complexity for each party.

We achieve this by employing a threshold verifiable random function (TVRF) to generate unbiased public nonces in a single round, and utilizing threshold homomorphic encryption (THE) to handle the non-linear operations in BBS+ signing, followed by another round. Each of these two processes—generating unbiased nonces and handling the non-linear operations—requires at least two rounds in prior works. By leveraging these simple yet effective tools with careful design, we accomplish both processes in just two rounds in total. Robustness is ensured by the threshold structures of the VRF and homomorphic encryptions. Specifically, we adopt a Decisional Diffie-Hellman (DDH) based VRF scheme over elliptic curve (EC) groups, which is significantly more efficient than using THE to generate public nonces, as in prior works (such as WMC24 and TX25). Meanwhile, for non-linear operations, we leverage both the threshold CL [8] and threshold ElGamal homomorphic encryptions in a well-designed manner.

We implement our signing protocol and provide benchmarking comparisons with existing works in Section 6. As shown in Tables 2 and 3, our constant per-party upload communication is only 2.02 KB, compared to 3.23 KB in WMC24, which likewise achieves constant communication. Compared to another robust work TX25, Robot achieves asymptotically better communication and computation complexity, and in practice, it starts to show a clear advantage once the number of signers reaches five. In particular, compared to the OT-based non-robust DKLST23, Robot provides robustness and achieves significantly better communication efficiency, albeit at the cost of much slower computation due to the CL operations. Refer to Section 6 for a detailed comparison.

1.2 Technical Overviews

A BBS+ signature for a (hashed) message $m \in \mathbb{Z}_q$ consists of three components (A, e, s), where the group element $A \in \mathbb{G}_1$ takes the form of $A = \frac{G_1 + sH_0 + mH_1}{x+e}$ with (G_1, H_0, H_1) as public parameters, and (e, s) are two signing nonces in $\mathbb{Z}_q$ (with a more formal definition in Section 2.3). The computation of A involves a non-linear operation $(x+e)^{-1} \mod q$ for the shared secret key x, which is the main difficulty.

To handle an inversion of a secret value $(x+e)^{-1}$, the prior works adopt techniques developed for threshold ECDSA [36,30,53,52,46]. It requires another shared secret nonce $\gamma \leftarrow\$ \mathbb{Z}_q$, and computes the multiplication $\gamma(x+e)$ in a distributed manner. After opening $\gamma(x+e)$, the A-component is generated via $A = \frac{\gamma(G_1 + sH_0 + mH_1)}{\gamma(x+e)}$. The previous approaches for multiplication fall into two main paradigms: those based on MtA (e.g., DKLST23 and TX25) and those based on threshold CL (e.g., WMC24).

Challenges of achieving two rounds. Based on the BBS+ equation, any threshold BBS+ protocol must involve the following specific steps:

1. Generate (e, s) in a distributed manner.
2. With e and (x, γ)'s shares, generate $\gamma(x+e)$'s shares; with s, m and γ's shares, generate $\gamma(G_1 + sH_0 + mH_1)$'s shares.
3. Open these shares and assemble $\gamma(x+e)$ and $\gamma(G_1 + sH_0 + mH_1)$.

Regarding these steps, we make the following *observation*: Steps 2 and 3 must occur after Step 1, and all steps require communication among signers. Therefore, achieving a two-round protocol is possible only if Step 1 can be carried out within one round, and Steps 2 and 3 can likewise be finished within another single round. However, we find that both of these tasks come with their own technical challenges, which prevent existing threshold BBS+ techniques from achieving them.

Challenge 1: one-round Step 1. We first emphasize that, for Step 1, even though e and s are public nonces, they cannot be directly derived from a random oracle with public input. Doing so would make (e, s) values that the adversary can obtain in advance, which is inconsistent with the definition of BBS+ signatures, where these values should be chosen by the signer or jointly by multiple signers in threshold settings. Instead, DKLST23 adopts the commit-and-release paradigm

to generate (e, s) in a distributed manner, which requires two rounds and lacks robustness. This is because if a malicious party refuses to reveal its shares e_i, s_i, the signing protocol must abort; otherwise, the adversary could bias the final values of e and s. While the threshold CL technique (as in WMC24) or threshold ElGamal technique (as in TX25) can provide robustness, applying them in Step 1 still demands two rounds of communication. That is, Step 1 cannot be completed in a single round using existing techniques.

Challenge 2: one-round Step 2&3. For the multiplication in Step 2, employing existing two-round MtA constructions [37,23] to generate additive shares of $\gamma(x+e)$ necessitates an additional round in Step 3 to open the shares, making Step 2&3 three-round. Like DKLST23 and TX25, even though they have merged this process with their two-round Step 1, they still require three rounds in total. This motivates us to focus on threshold CL techniques for multiplications, inspired by [49,32]. In threshold CL, the secret decryption key is distributed among multiple parties, and any subset of more than t parties can decrypt. Based on this, computing $\gamma(x+e)$ within a single round is also not straightforward. This process, as in WMC24, typically requires two rounds as follows. First, it requires one round to produce the CL ciphertext of $\gamma(x+e)$ after generating (e, s). This is because the secret nonce γ is shared for this signing, and each party must broadcast the ciphertext of $\gamma_i(x+e)$ after performing a linear homomorphic operation to obtain $x+e$'s ciphertext[4]. Subsequently, the parties need another round to reveal the results of the distributed decryption. Thus, to complete Step 2&3 with only one round requires a more careful design.

In summary, the sequential nature and the necessity of interaction for these steps make achieving a two-round threshold signing technically challenging, even without considering robustness.

Our Solutions. We briefly describe our approach for achieving a one-round Step 1 and completing Steps 2–3 with only one other round, and we present the proof intuition for our scheme.

One-round robust generation of (e, s). To achieve the one-round goal, our intuition is that the shares e_i, s_i picked by each party should be verifiably unbiased. Intuitively, each party can invoke a verifiable random function (VRF) [40] to accomplish it within a single round. Given an input, the corresponding VRF output is a deterministic value generated using a secret key, accompanied by a zero-knowledge proof (ZKP) that ensures the output is valid (or unbiased). It is sufficient under security with abort, but it does not work for robust signing protocols. For instance, if a malicious party reveals an invalid output, a robust protocol does not abort; instead, it discards invalid outputs and aggregates only the valid ones to produce the final (e, s). Consequently, an adversary can still bias (e, s) by controlling whether the malicious parties provide valid outputs.

To address this issue, we leverage a t-out-of-n threshold version of the VRF, in which a single secret key is shared among the parties. Each party can broadcast its partial result, along with a correctness proof, via a one-round interaction. Then, any $t+1$ valid partial results can be combined into a fixed VRF output, which is encoded as (e, s). As the key is kept secret, the output cannot be known in advance by the adversary, making it simulatable in the security proof. Moreover, the threshold structure together with the correctness proof ensures the robustness of the (e, s) generation.

We utilize an efficient DDH-based construction of threshold VRF [22], whose proofs are both efficient and compact. The full VRF output for the input in (e.g., the message to be signed m) is $Z = \hat{x} \cdot \mathsf{H}_0(\mathsf{in})$ such that $(G, \hat{X}, \mathsf{H}_0(\mathsf{in}), Z) \in \mathbb{G}^4$ is a DDH tuple, where $(\hat{x}, \hat{X} = \hat{x} \cdot G)$ is the VRF key pair, $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{G}$ is random oracle. With $\hat{x}$ shared among all parties in a threshold structure, its distributed generation is efficient: each party holding $\hat{x}_i$ computes $Z_i = \hat{x}_i \cdot \mathsf{H}_0(\mathsf{in})$ and provides a proof that $(G, \hat{X}_i, \mathsf{H}_0(\mathsf{in}), Z_i) \in \mathbb{G}^4$ forms a DDH tuple. Finally, after assembling $t+1$ valid Z_i's into Z, we derive $(e, s) \leftarrow \mathsf{H}_1(Z)$ with random oracle $\mathsf{H}_1(\cdot)$.

Completing Step 2&3 with one additional round. As analyzed above, the main challenge we face is how to avoid the extra round required for generating the CL ciphertext of $\gamma(x+e)$. Our key insight is that this process can be initiated without waiting for the complete generation of (e, s). By separating γx and γe, the ciphertexts for γx and γ can be generated concurrently with the execution of threshold VRF (i.e., Step 1), then once e is generated, these ciphertexts can be combined non-interactively via local computation. More precisely, Round 1 further involves the broadcast of ciphertexts $\mathsf{CL.Enc}(\gamma_i x)$, $\mathsf{CL.Enc}(\gamma_i)$. In this manner, the full ciphertext of $\gamma(x+e)$

[4] The ciphertext of the signing key x has been assembled during the key generation phase.

can be generated after Round 1. Then, a single round of distributed decryption suffices to finalize the generation of $\gamma(x+e)$.

Meanwhile, we also need to handle the generation of $\gamma(G_1 + sH_0 + mH_1)$. Although this is a linear operation over the EC group, the robustness requirement prevents parties from simply revealing $\gamma_i(G_1 + sH_0 + mH_1)$ in the clear, since any malicious party providing an incorrect value would cause the signing to fail, given that γ has already been implicitly fixed at this point. To ensure robust generation, we employ the THE technique again. Unlike the above case, a more efficient threshold ElGamal is used here instead of CL, as the plaintext in this case is a group element.

In conclusion, our two-round scheme follows the high-level structure outlined below.

- **Round 1:** Each party P_i broadcasts (i) its VRF partial result for the generation of (e, s), (ii) CL ciphertexts of $\gamma_i x, \gamma_i$, and (iii) ElGamal ciphertexts of $\gamma_i(G_1 + mH_1), \gamma_i H_0$.
- **Round 2:** The parties combine (e, s), the CL ciphertext $\mathcal{C}_1$ of $\gamma(x+e)$, and the ElGamal ciphertext $\mathcal{C}_2$ of $\gamma(G_1 + sH_0 + mH_1)$. Each P_i then broadcasts its partial decryption results.

Lastly, we also design a ZKP system to ensure the consistency of γ across ciphertexts generated under both the CL and ElGamal schemes. The proof is instantiated via a Sigma protocol over class groups and EC groups; see Section 5 for further details.

Proof Intuition. For the simulation-based security proof to succeed, we must guarantee that the simulator has control over the generation of (A, e, s), which are provided by the ideal functionality. This is the primary concern in proving two-round schemes, akin to the challenge encountered in two-round threshold Schnorr [35,42]. However, their proof techniques are not suitable here, as our techniques diverge from those employed in the threshold Schnorr protocol.

In particular, our simulation combines the properties of DDH-based VRF, CL, and ElGamal encryptions, as sketched below. With the predefined (e, s), the simulator $\mathcal{S}$ must obtain the VRF output Z before the adversary and program the random oracle output $\mathsf{H}_1(Z) \to (e, s)$. To achieve this, the simulator first programs each query $\mathsf{H}_0(\mathsf{in}) \to b \cdot G$ with $b \leftarrow\!\$\ \mathbb{Z}_q$ and records the correspondence between b and in. When the signing query corresponding to in occurs, $\mathcal{S}$ sets Z as $b \cdot X$, which satisfies the DDH relation and is therefore identical to that in the real world.

Controlling the group element $A \in \mathbb{G}_1$ requires additional techniques. $\mathcal{S}$ (with no knowledge of x, γ) picks a random value $a \leftarrow\!\$\ \mathbb{Z}_q$ and regards it as $a := x\gamma$. It then simulates the honest parties' partial decryptions to make the decryption of the CL ciphertext $\mathcal{C}_1$ output a, and the decryption of the ElGamal ciphertext $\mathcal{C}_2$ output $a \cdot A$. Since a is randomly chosen by $\mathcal{S}$, it is clear that the original plaintexts of $\mathcal{C}_1, \mathcal{C}_2$ are not $a, a \cdot A$[5]. Intuitively, the adversary $\mathcal{A}$ cannot detect this inconsistency due to semantic security. However, the proof is not straightforward, since embedding the challenge ciphertexts into $\mathcal{C}_1$ and $\mathcal{C}_2$ is impossible, as they are not under $\mathcal{S}$'s control. We observe that the public key of the threshold encryption is under the simulator's control. Hence, we can embed the hard problem into the public key and leverage $\mathcal{A}$'s distinguishing capability to solve it. For instance, we embed the hard subgroup membership (HSM) of the class group into the threshold CL's public key. The detailed proof can be found in Lemmas 2 and 4.

1.3 Discussions

Threshold deterministic/randomized BBS+. Our design derives the nonces (e, s) as the output of TVRF, which are decided by a common input. We can simply set this input as the message to be signed, which makes our threshold signing output a *deterministic* BBS+ signature. Similarly, many works focus on thresholding or distributing the deterministic variants of other digital signatures, such as Schnorr and ECDSA [43,6,21].

The concrete security of the deterministic BBS+ scheme has been studied by Chairattana-Apirom and Tessaro in [17]. They demonstrate that it is comparable to its randomized counterpart under identical parameter choices. They define a deterministic variant of BBS+ by generating (e, s) via a pseudorandom function (PRF) $\mathsf{F}(\hat{x}, m)$, where $\hat{x}$ is the PRF secret key selected during

[5] This issue does not arise in protocols with more rounds [8,49,32], since their protocols allow $\mathcal{S}$ to obtain the adversary's plaintext in advance, thereby controlling that the plaintext of the assembled ciphertext is a predefined value (e.g., a or aA).

key generation. Our construction can be viewed as instantiating this PRF by setting $\mathsf{F}(\hat{x}, m) = \mathsf{H}_1(\hat{x} \cdot \mathsf{H}_0(m))$, with random oracles $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{G}$ and $\mathsf{H}_1 : \mathbb{G} \to \mathbb{Z}^2$. Notably, they analyze both the randomized and deterministic BBS+ schemes and report identical attack complexities.

Our threshold construction naturally extends to randomized BBS+. To generate randomized (e, s) values, we need to include another source of randomness as VRF's input. It can be the session ID *sid*, but doing so makes our signing protocol *stateful*, similar to threshold Raccoon [44]. Each signer must maintain a simple state regarding the used session IDs to ensure that the same *sid* is never reused.

We can remove the state by adding one more round, like [33,3]. The idea is very simple: in the first round, each one broadcasts a random string str_i. The parties then set $sid = \{\mathsf{str}_i\}_i$ and run the previous two rounds. Since *sid* must contain a uniform random string sampled by an honest party, it is guaranteed to be distinct from prior *sid*'s. That is, we get a three-round stateless threshold randomized BBS+. It still has advantages over existing three-round schemes [25,47], as it achieves both robustness and constant communication. In particular, since the broadcast of str_i is independent of the message, we can adopt the pipelining technique from [24] to evaluate the first round simultaneously with the third round of a previous signing, which allows signing to be completed with only two rounds of latency.

In summary, our work realizes a two-round stateless threshold deterministic BBS+, a two-round stateful threshold randomized BBS+, or a three-round stateless threshold randomized BBS+, which can still be reduced to two rounds via the pipelining technique. All of them provide robustness.

Extension. Our threshold techniques can be naturally extended to the BBS signature, which was implicitly proposed by Boneh, Boyen, and Shacham [5] as part of their group signature. Its security was recently proved formally by Tessaro and Zhu [48], and its deterministic version has been used in the RFC draft [38]. BBS eliminates the nonce s and has a shorter signature (A, e) with $A = \frac{G_1 + mH_1}{x+e}$. Thus, here we only derive e from VRF's output Z.

2 Preliminaries

Let $\mathbb{Z}_q$ be the finite field with q elements $\{0, 1, \cdots, q-1\}$, $[a]$ denote the set $\{1, \cdots, a\}$ for some positive a. By $s \leftarrow\!\!\$\ S$, we denote that s is chosen uniformly at random from a finite set S. For a number N, $|N|$ denotes the bit length of N; for a set S, $|S|$ denotes the size of S. Let $\lambda \in \mathbb{N}$ and $\lambda_s \in \mathbb{N}$ denote the security parameter and the statistical parameter, respectively. Through this paper, n always denotes the number of all parties $(P_1, \cdots, P_n)$, and $t < n$ denotes the threshold value as well as the maximum number of parties that are corrupted by the adversary. In the security proof, $\approx_c$ denotes computational indistinguishability, while $\approx_s$ denotes statistical indistinguishability.

2.1 Shamir's Secret Sharing

Secret Sharing over $\mathbb{Z}_q$. For t-out-of-n shares $\{x_i\}_{i\in[n]}$ of $x \in \mathbb{Z}_q$, we have $x = \sum_{i\in S} \lambda_{i,S} \cdot x_i \bmod q$ for any set $S \subseteq [n]$ with $|S| > t$, where $\lambda_{i,S} \coloneqq \prod_{j\in S\setminus\{i\}} \frac{-j}{i-j}$ is the Lagrange coefficient corresponding to $i \in S$.

Secret Sharing over $\mathbb{Z}$. For t-out-of-n shares $\{x_i\}_{i\in[n]}$ of $x \in \mathbb{Z}$, we have $\Delta^2 \cdot x = \sum_{i\in S} \Delta \cdot \lambda_{i,S} \cdot x_i$ for any set $S \subseteq [n]$ with $|S| > t$, where $\Delta = n!$ is necessary for security and eliminating division in the computation of $\lambda_{i,S}$.

2.2 Zero-Knowledge Proof

Let st be a statement of an NP language $\mathcal{L}$ and wi be its witness. A zero-knowledge proof (ZKP) allows a prover to convince the verifier that $(\mathsf{st}, \mathsf{wi}) \in \mathcal{R}$ without revealing wi. We always use a non-interactive zero-knowledge (NIZK) proof with $(\mathsf{Prove}, \mathsf{Verify})$ as a tool to design a larger protocol.

- $\mathsf{Prove}_{\mathcal{R}}(\mathsf{st}; \mathsf{wi}) \to \pi$: On input of the statement st and the secret witness wi, the probabilistic algorithm returns a proof π.

- $\mathsf{Verify}_{\mathcal{R}}(\mathsf{st}, \pi) \to 0/1$: The deterministic algorithm returns a bit, indicating the validity of π.

Our scheme needs NIZKs for some specific relations. For their definitions and NIZK instantiations, please refer to Sections 3 and 5.

2.3 BBS+ Signature Scheme

Bilinear Groups. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be a pairing group with the generators $G_1 \in \mathbb{G}_1, G_2 \in \mathbb{G}_2$ and an efficient pairing operation $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, such that for any $x, y \in \mathbb{Z}_q$, $\mathsf{e}(x \cdot G_1, y \cdot G_2) = xy \cdot \mathsf{e}(G_1, G_2)$. Let $\mathsf{BilinGen}$ be an efficient algorithm to generate $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, q) \leftarrow \mathsf{BilinGen}(1^\lambda)$, which includes the groups $\mathbb{G}_1, \mathbb{G}_2$, the generators $G_1 \in \mathbb{G}_1, G_2 \in \mathbb{G}_2$, and the order q with λ bits of security. There are three types of pairings [29]: type-1, $\mathbb{G}_1 = \mathbb{G}_2$; type-2, $\mathbb{G}_1 \neq \mathbb{G}_2$ and there exists an efficient isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$; type-3, $\mathbb{G}_1 \neq \mathbb{G}_2$ and there does not exist an efficient isomorphism ψ between $\mathbb{G}_1$ and $\mathbb{G}_2$.

This paper will use the type-3 pairing since we require that DDH is hard in $\mathbb{G}_1$. BBS+ is existentially unforgeable against chosen message attacks for type-3 pairing [10].

BBS+ Signatures. BBS+ can generate a signature for signing a vector of ℓ messages. Let pp be the public parameters containing $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, q)$. It proceeds as follows.

1. $\mathsf{BBS+.KGen}(pp, \ell) \to (\mathsf{sk}, \mathsf{pk})$:
   a. pick $\ell + 1$ group elements $\mathbf{H} \coloneqq \{H_k\}_{k \in [0,\ell]} \leftarrow\!\!\$\ \mathbb{G}_1^{\ell+1}$
   b. pick secret key $x \leftarrow\!\!\$\ \mathbb{Z}_q$, compute $X \coloneqq xG_2 \in \mathbb{G}_2$
   c. output $\mathsf{sk} \coloneqq (\mathbf{H}, x)$ and $\mathsf{pk} \coloneqq (\mathbf{H}, X)$.
2. $\mathsf{BBS+.Sign}(\mathsf{sk}, \mathbf{m}) \to \sigma$:
   a. parse sk as $(\{H_k\}_{k \in [0,\ell]}, x)$ and $\mathbf{m}$ as $\{m_k\}_{k \in [\ell]}$
   b. pick public nonces $e, s \leftarrow\!\!\$\ \mathbb{Z}_q$
   c. compute $A \coloneqq \dfrac{G_1 + sH_0 + \sum_{k \in [\ell]} m_k H_k}{x + e} \in \mathbb{G}_1$
   d. output the signature $\sigma \coloneqq (A, e, s)$.
3. $\mathsf{BBS+.Verify}(\mathsf{pk}, \mathbf{m}, \sigma) \to 0/1$:
   a. parse pk as $(\{H_k\}_{k \in [0,\ell]}, X)$ and σ as (A, e, s)
   b. output 1 if and only if

$$\mathsf{e}(A, X + eG_2) = \mathsf{e}(G_1 + sH_0 + \sum_{k \in [\ell]} m_k H_k, G_2).$$

Its deterministic variant, as defined in [17], replaces Step 2b with $e, s \leftarrow \mathsf{F}(\hat{x}, \mathbf{m})$ where $\mathsf{F}(\hat{x}, \cdot)$ is a pseudorandom function keyed by the signer's additional secret key $\hat{x}$.

2.4 (Threshold) CL Encryption

The class group framework [16] is defined by the public parameter $pp_{\mathsf{cl}} = (\tilde{s}, q, f, g_q, F, \mathcal{G}^q, \hat{\mathcal{G}})$ where $\hat{\mathcal{G}}$ is an unknown-order class group with order $q\hat{s}$, only the upper bound $\tilde{s}$ of $\hat{s}$ is given. It utilizes $\hat{\mathcal{G}}$'s three subgroups:

- $F \coloneqq \langle f \rangle$, a unique subgroup of order q with a *deterministic polynomial-time* algorithm $\mathsf{Dlog}(\cdot)$ that returns x on input $f^x \in F$;
- $\mathcal{G}^q \coloneqq \langle g_q \rangle$, a subgroup containing all q-th powers in $\mathcal{G}$ with an unknown order s s.t. $s | \hat{s}$;
- $\mathcal{G} \coloneqq F \times \mathcal{G}^q = \langle f g_q \rangle$, with order qs.

CL Encryption. The CL scheme and linear homomorphic operations are described below. Note that the secret key and randomness are sampled uniformly over the range $\mathcal{D}_q = [0, 2^{\lambda_s} q\tilde{s}]$ where λ_s is the statistical parameter.

- $\mathsf{CL.KGen}(1^\lambda, pp_{\mathsf{cl}}) \to (pk_{\mathsf{cl}}, k)$: output $k \leftarrow\!\!\$\ \mathcal{D}_q$ and $pk_{\mathsf{cl}} \coloneqq g_q^k$.
- $\mathsf{CL.Enc}(pk_{\mathsf{cl}}, m \in \mathbb{Z}_q) \to \mathcal{C}_m$: pick $\rho \leftarrow\!\!\$\ \mathcal{D}_q$ and output $\mathcal{C}_m \coloneqq (g_q^\rho, f^m pk_{\mathsf{cl}}^\rho)$.
- $\mathsf{CL.Dec}(k, \mathcal{C}_m) \to m$: parse $\mathcal{C}_m = (c_1, c_2)$ and output $m \coloneqq \mathsf{Dlog}(c_2 / c_1^k)$.

- Addition $\mathsf{CL.Add}(\{\mathcal{C}_{m_i}\}_i) \to \mathcal{C}_m$: parse $\mathcal{C}_{m_i} = (c_{1i}, c_{2i})$ and output $\mathcal{C}_m \coloneqq (\prod_i c_{1i}, \prod_i c_{2i})$ with $m \coloneqq \sum_i m_i$.
- Scalar Multiplication $\mathsf{CL.SMul}(a \in \mathbb{Z}_q, \mathcal{C}_m) \to \mathcal{C}_{am}$: parse $\mathcal{C}_m = (c_1, c_2)$, pick $\rho \leftarrow\!\$\ \mathcal{D}_q$, output $\mathcal{C}_{am} \coloneqq (c_1^a g_q^\rho, c_2^a pk_{\mathsf{cl}}^\rho)$.
- $\mathsf{CL.SMul}^*(a \in \mathbb{Z}_q, \mathcal{C}_m) \to \mathcal{C}_{am}$: deterministically output $\mathcal{C}_{am} \coloneqq (c_1^a, c_2^a)$.

Threshold CL Encryption [8]. In the threshold CL scheme, $\mathsf{CL.KGen}$ is replaced with a distributed key generation protocol $\mathsf{CL.DKG}$. The decryption process now consists of each party running partial decryption $\mathsf{CL.PartDec}$, followed by combining these results $\mathsf{CL.Combine}$. It will use Shamir's secret sharing over $\mathbb{Z}$.

- $\mathsf{CL.DKG}(1^\lambda, pp_{\mathsf{cl}}) \to (\{k_i\}_{i\in[n]}; (pk_{\mathsf{cl}}, \{pk_i\}_{i\in[n]}))$: it is a multi-party protocol that outputs secret k_i which is a t-out-of-n share of CL's decryption key k and public $\{pk_i = g_q^{\Delta \cdot k_i}\}_{i\in[n]}, pk_{\mathsf{cl}} = g_q^k$, to each party P_i.
- $\mathsf{CL.PartDec}(k_i, \mathcal{C}_m) \to (M_i, \tau_i)$: parse $\mathcal{C}_m = (c_1, c_2)$, each party P_i outputs $M_i \coloneqq c_1^{\Delta \cdot k_i}$ and $\tau_i \leftarrow \mathsf{Prove}_{\mathsf{clddh}}((g_q, pk_i, c_1, M_i); k_i)$.
- $\mathsf{CL.Combine}(\{M_i, \tau_i\}_{i\in[n]}, \mathcal{C}_m) \to m$: verify τ_i and let S be the set of i with valid τ_i, output $m \coloneqq \mathsf{Dlog}(\frac{c_2^{\Delta^3}}{\prod_{i\in S} M_i^{\Delta\lambda_{i,S}}}) \cdot \Delta^{-3} \mod q$.

Please refer to [8] for the multi-party protocol of CL's DKG $\mathsf{CL.DKG}(1^\lambda, pp_{\mathsf{cl}}) \to (\{k_i\}_{i\in[n]}; (pk_{\mathsf{cl}}, \{pk_i\}_{i\in[n]}))$.

2.5 (Threshold) ElGamal Encryption

ElGamal Encryption. In this context, it works over the group $\mathbb{G}_1 = \langle G_1 \rangle$ of prime order q and encrypts a group element, based on $\mathbb{G}_1$'s DDH assumption.

- $\mathsf{EG.KGen}(1^\lambda, (\mathbb{G}_1, G_1, q)) \to (\hat{X}, \hat{x})$: output $\hat{x} \leftarrow\!\$\ \mathbb{Z}_q$ and $\hat{X} \coloneqq \hat{x}G_1 \in \mathbb{G}_1$.
- $\mathsf{EG.Enc}(\hat{X}, M \in \mathbb{G}_1) \to \mathcal{C}_M$: pick $r \leftarrow\!\$\ \mathbb{Z}_q$ and output $\mathcal{C}_M \coloneqq (rG_1, M + r\hat{X})$.
- $\mathsf{EG.Dec}(\hat{x}, \mathcal{C}_M) \to M$: parse $\mathcal{C}_M = (A_1, A_2)$ and output $M \coloneqq A_2 - \hat{x}A_1$.
- $\mathsf{EG.Add}(\{\mathcal{C}_{M_i}\}_i) \to \mathcal{C}_M$: parse $\mathcal{C}_{M_i} = (A_{1i}, A_{2i})$ and output $\mathcal{C}_M \coloneqq (\sum_i A_{1i}, \sum_i A_{2i})$ with $M \coloneqq \sum_i M_i$.
- $\mathsf{EG.SMul}(a \in \mathbb{Z}_q, \mathcal{C}_M) \to \mathcal{C}_{aM}$: parse $\mathcal{C}_M = (A_1, A_2)$, pick $r \leftarrow\!\$\ \mathbb{Z}_q$, output $\mathcal{C}_{aM} \coloneqq (rG_1 + aA_1, r\hat{X} + aA_2)$.
- $\mathsf{EG.SMul}^*(a, \mathcal{C}_M) \to \mathcal{C}_{aM}$: deterministically output $\mathcal{C}_{aM} \coloneqq (aA_1, aA_2)$.

Threshold ElGamal. It applies Shamir's secret sharing over $\mathbb{Z}_q$.

- $\mathsf{EG.DKG}(1^\lambda, (\mathbb{G}_1, G_1, q)) \to (\{\hat{x}_i\}_{i\in[n]}; (\hat{X}, \{\hat{X}_i\}_{i\in[n]}))$: it is a multi-party protocol that outputs private $\hat{x}_i$ which is a t-out-of-n share of $\hat{x} \in \mathbb{Z}_q$ and public $\hat{X} = \hat{x}G_1, \{\hat{X}_i = \hat{x}_i G_1\}_{i\in[n]}$, to each party P_i.
- $\mathsf{EG.PartDec}(\hat{x}_i, \mathcal{C}_M) \to (M_i, \tau_i)$: parse $\mathcal{C}_M = (A_1, A_2)$, each party P_i outputs $M_i \coloneqq \hat{x}_i A_1$ and $\tau_i \leftarrow \mathsf{Prove}_{\mathsf{ddh}}((G_1, \hat{X}_i, A_1, M_i); \hat{x}_i)$.
- $\mathsf{EG.Combine}(\{M_i, \tau_i\}_{i\in[n]}, \mathcal{C}_M) \to M$: verify τ_i and let S be the set of i with valid τ_i, output $M \coloneqq A_2 - \sum_{i\in S} \lambda_{i,S} M_i$.

Please refer to [13] for the multi-party protocol of ElGamal's DKG $\mathsf{EG.DKG}(1^\lambda, (\mathbb{G}_1, G_1, q)) \to (\{\hat{x}_i\}_{i\in[n]}; (\hat{X}, \{\hat{X}_i\}_{i\in[n]}))$.

2.6 Threshold Verifiable Random Functions

A verifiable random function (VRF) [40] is a pseudorandom function that can be computed using a secret key, producing both an output and a proof that anyone can verify using the public key. In this context, we require a distributed notion of VRF, where each party can compute a partial result, and any $t+1$ valid partial results can be combined to get the full VRF output. Following [14], we present the syntax of threshold VRF here, and leave its construction to section 3.2.

- $\mathsf{VRF.DKG}(1^\lambda)$ outputs secret key shares tsk_i and public key shares tpk_i, $\forall i \in [n]$, and global public key tpk.
- $\mathsf{PartialEval}(r, tsk_i)$ where r is a global seed, is a deterministic algorithm which outputs Z_i, π_i where Z_i is a partial evaluation and π_i is a proof.
- $\mathsf{Combine}(\{tpk_i, Z_i, \pi_i\}_{i\in S})$ where $S \subseteq [n]$ with $|S| \geq t+1$, outputs either global evaluation Z corresponding to r, or $\perp$.

Note that the original verification algorithm of VRF is integrated into the $\mathsf{Combine}(\cdot)$ algorithm, where $t+1$ verified partial evaluations Z_i are assembled into the final VRF output.

3 Robust Threshold BBS+ Scheme

3.1 Distributed Key Generation

The key generation incorporates three distributed key generation (DKG) protocols: one for the key pair of CL encryption, one for the key pair of ElGamal encryption, and another for BBS+'s key pair. Since the key pair of BBS+ is very similar to that of ElGamal, except that the former is defined over $\mathbb{G}_2$ while the latter is over $\mathbb{G}_1$, we can invoke $\mathsf{EG.DKG}$ for both. At the same time, we require each party to get the BBS+ secret key x's CL ciphertext after this DKG process.

- DKG for CL's key pair (k, pk_{cl}): $\mathsf{CL.DKG}(1^\lambda, pp_{\mathsf{cl}}) \to (\{k_i\}_{i\in[n]}; (pk_{\mathsf{cl}}, \{pk_i\}_{i\in[n]}))$.
- DKG for ElGamal's key pair $(\hat{x}, \hat{X} = \hat{x}G_1 \in \mathbb{G}_1)$: $\mathsf{EG.DKG}(1^\lambda, (\mathbb{G}_1, G_1, q)) \to (\{\hat{x}_i\}_{i\in[n]}; (\hat{X}, \{\hat{X}_i\}_{i\in[n]}))$.
- DKG for BBS+'s key pair $(x, X = xG_2 \in \mathbb{G}_2)$: it can naturally invoke $\mathsf{EG.DKG}(1^\lambda, (\mathbb{G}_2, G_2, q)) \to (\{x_i\}_{i\in[n]}; (X, \{X_i\}_{i\in[n]}))$. Then publicly and deterministically derive $\ell+1$ random group elements $\mathbf{H}$ from X.
- Each party generates $\mathcal{C}_{x_i} \leftarrow \mathsf{CL.Enc}(pk_{\mathsf{cl}}, x_i)$ and the correctness proof $\pi_i \leftarrow \mathsf{Prove}_{\mathsf{cldl}}((pk_{\mathsf{cl}}, \mathcal{C}_{x_i}, G_2, X_i); x_i)$. Broadcast $(\mathcal{C}_{x_i}, \pi_i)$.
- After receiving $(\mathcal{C}_{x_j}, \pi_j)_{j\in[n]}$, verify π_j and let $\hat{\mathcal{P}}$ denote the set of j with valid π_j. Each party gets $\mathcal{C}_x \coloneqq \mathsf{CL.Add}(\{\mathsf{CL.SMul}^*(\lambda_{j,\hat{\mathcal{P}}}, \mathcal{C}_{x_j})\}_{j\in\hat{\mathcal{P}}})$ with $x \coloneqq \sum_{j\in\hat{\mathcal{P}}} \lambda_{j,\hat{\mathcal{P}}} \cdot x_j \mod q$.

It requires a zero-knowledge proof for the relation

$$\mathcal{R}_{\mathsf{cldl}} = \left\{ ((pk_{\mathsf{cl}}, \mathcal{C}_{x_i}, G_2, X_i); x_i) \middle| \begin{matrix} \mathcal{C}_{x_i} \leftarrow \mathsf{CL.Enc}(x_i, pk_{\mathsf{cl}}) \\ X_i = x_i G_2 \end{matrix} \right\}$$

Due to space constraints, the corresponding NIZK proof construction can be found in Appendix B.

3.2 Distributed Signing

Here, let $\mathcal{P}$ denote the index set of signers and $\mathbf{m}$ the message vector of length ℓ. Our robust scheme can satisfy that as long as $\mathcal{P}$ contains $t+1$ semi-honest parties, the signing protocol will always succeed. Recall the parties jointly generate a BBS+ signature as $(A = \frac{\gamma(G_1 + sH_0 + \sum_{k\in[\ell]} m_k H_k)}{\gamma(x+e)}, e, s)$, where e, s are public nonces and γ is used to mask x when opening $\gamma(x+e)$.

Zero-Knowledge Proofs. For security and robustness, we require ZKPs to ensure the correctness of sending messages. Specifically, we require the following relations.

$$\mathcal{R}_{\mathsf{ddh}} = \left\{ ((G_1, \hat{X}_i, Y, Z_i); \hat{x}_i) : \hat{X}_i = \hat{x}_i G_1, Z_i = \hat{x}_i Y \right\}.$$

The above relation is a DDH relation over the EC group $\mathbb{G}_1$. We also need the DDH relation over the class group, defined as

$$\mathcal{R}_{\mathsf{clddh}} = \left\{ ((g_q, pk_i, c_1, M_i); k_i) : pk_i = g_q^{\Delta\cdot k_i}, M_i = c_1^{\Delta\cdot k_i} \right\}.$$

Their ZKP constructions can be found in prior works, which are included in Appendix B.

Furthermore, we define a consistency relation between CL and ElGamal plaintexts as, with CL and ElGamal public keys $pk_{\mathsf{cl}}, \hat{X}$

$$\mathcal{R}_{\mathsf{con}} = \left\{ \left((\mathcal{C}_{\gamma_i}, \mathcal{C}_x, \mathcal{C}_{\gamma_i x}, H_0, \mathcal{C}_{D_{0i}}, D, \mathcal{C}_{D_{1i}}); \gamma_i \right) \middle| \begin{array}{c} \mathcal{C}_{\gamma_i} \leftarrow \mathsf{CL.Enc}(\gamma_i, pk_{\mathsf{cl}}) \\ \mathcal{C}_{\gamma_i x} \leftarrow \mathsf{CL.SMul}(\gamma_i, \mathcal{C}_x) \\ \mathcal{C}_{D_{0i}} \leftarrow \mathsf{EG.Enc}(\gamma_i H_0, \hat{X}) \\ \mathcal{C}_{D_{1i}} \leftarrow \mathsf{EG.Enc}(\gamma_i D, \hat{X}) \end{array} \right\}.$$

A complete description of its NIZK proof will be provided in Section 5.

Threshold VRF based on DDH. We use it to generate public nonces e, s with only one round of communication. Its DKG is also DL-type. Thus, here we use the same key pair as that of the threshold ElGamal scheme, over $\mathbb{G}_1$. It utilizes the random oracle $\mathsf{H}_0(\cdot) : \{0,1\}^* \to \mathbb{G}_1$.

- $\mathsf{PartialEval}(\mathbf{m}, sid, \hat{x}_i) \to (Z_i, \pi_i)$: it requires the message vector $\mathbf{m}$ and a session ID sid as a global seed, outputs $Z_i \coloneqq \hat{x}_i \mathsf{H}_0(\mathbf{m}, sid)$ and a DDH-relation proof π_i for $(G_1, \hat{X}_i, \mathsf{H}_0(\mathbf{m}, sid), Z_i)$.
- $\mathsf{Combine}(\{\hat{X}_i, Z_i, \pi_i\}_{i \in \mathcal{P}}) \to Z/\perp$: verify each π_i, set $\mathcal{P} = \mathcal{P} \setminus \{i\}$ if π_i is not valid; output $Z = \sum_i \lambda_{i,\mathcal{P}} Z_i$ if $|\mathcal{P}| \geq t+1$ and $\perp$ otherwise.

In this distributed way, Z is assembled correctly as long as $\mathcal{P}$ contains $t+1$ (semi-)honest parties.

Threshold Signing Construction. In the following, we present the concrete construction, which is also illustrated in Figure 1.

- **Round 1.** *Generating Nonces Phase 1.* With an agreed-on fresh session ID sid, the parties prepare VRF partial evaluations Z_i used to later generate the public nonces $e, s \in \mathbb{Z}_q$:
  (a) generate $Y \coloneqq \mathsf{H}_0(\mathbf{m}, sid)$ and $Z_i \coloneqq \hat{x}_i Y \in \mathbb{G}_1$
  (b) generate the DDH-relation proof

$$\pi_i \leftarrow \mathsf{Prove}_{\mathsf{ddh}}((G_1, \hat{X}_i, Y, Z_i); \hat{x}_i).$$

  Encrypting Phase. Based on the homomorphic property of CL and ElGamal encryptions, each party P_i generates the ciphertexts:
  (a) pick $\gamma_i \leftarrow\!\!\$\ \mathbb{Z}_q$, compute $D_{0i} \coloneqq \gamma_i H_0 \in \mathbb{G}_1$
  (b) $D \coloneqq G_1 + \sum_{k \in [\ell]} m_k H_k \in \mathbb{G}_1$ and $D_{1i} \coloneqq \gamma_i D$
  (c) generate the CL ciphertexts $\mathcal{C}_{\gamma_i} \leftarrow \mathsf{CL.Enc}(\gamma_i, pk_{\mathsf{cl}})$ and $\mathcal{C}_{\gamma_i x} \leftarrow \mathsf{CL.SMul}(\gamma_i, \mathcal{C}_x)$
  (d) generate the ElGamal ciphertexts $\mathcal{C}_{D_{0i}} \leftarrow \mathsf{EG.Enc}(D_{0i}, \hat{X})$ and $\mathcal{C}_{D_{1i}} \leftarrow \mathsf{EG.Enc}(D_{1i}, \hat{X})$.
  (e) generate the γ_i-consistency proof

$$\hat{\pi}_i \leftarrow \mathsf{Prove}_{\mathsf{con}}((\mathcal{C}_{\gamma_i}, \mathcal{C}_x, \mathcal{C}_{\gamma_i x}, H_0, \mathcal{C}_{D_{0i}}, D, \mathcal{C}_{D_{1i}}); \gamma_i).$$

  Broadcast $(Z_i, \pi_i; \mathcal{C}_{\gamma_i}, \mathcal{C}_{\gamma_i x}, \mathcal{C}_{D_{0i}}, \mathcal{C}_{D_{1i}}, \hat{\pi}_i)$.
- **Round 2.** *Generating Nonces Phase 2.* Each party first assembles (e, s) with a random oracle $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_q$, after receiving $\{Z_j, \pi_j\}_{j \in \mathcal{P}}$:
  (a) verify π_j to check whether Z_j are correct
  (b) if π_j is invalid, set $\mathcal{P} = \mathcal{P} \setminus \{j\}$, abort the protocol if $|\mathcal{P}| \leq t$
  (c) $Z \coloneqq \sum_{j \in \mathcal{P}} \lambda_{j,\mathcal{P}} Z_j \in \mathbb{G}_1$
  (d) $e \coloneqq \mathsf{H}_1(Z, 0), s \coloneqq \mathsf{H}_1(Z, 1) \in \mathbb{Z}_q$.

  Partially Decrypting Phase. The parties assemble the correct ciphertexts and partially decrypt them, after receiving $\{\mathcal{C}_{\gamma_j}, \mathcal{C}_{\gamma_j x}, \mathcal{C}_{D_{0j}}, \mathcal{C}_{D_{1j}}, \hat{\pi}_j\}_{j \in \mathcal{P}}$:
  (a) verify $\hat{\pi}_j$ to check whether $(\mathcal{C}_{\gamma_j}, \mathcal{C}_{\gamma_j x}, \mathcal{C}_{D_{0j}}, \mathcal{C}_{D_{1j}})$ are generated with the same γ_j
  (b) if $\hat{\pi}_j$ is not valid, set $\mathcal{P} = \mathcal{P} \setminus \{j\}$, abort the protocol if $|\mathcal{P}| \leq t$
  (c) sum all ciphertexts up

$$\mathcal{C}_{\gamma x} \coloneqq \mathsf{CL.Add}(\{\mathcal{C}_{\gamma_j x}\}_j),\ \mathcal{C}_{\gamma} \coloneqq \mathsf{CL.Add}(\{\mathcal{C}_{\gamma_j}\}_j)$$

$$\mathcal{C}_{D_0} \coloneqq \mathsf{EG.Add}(\{\mathcal{C}_{D_{0j}}\}_j), \mathcal{C}_{D_1} \coloneqq \mathsf{EG.Add}(\{\mathcal{C}_{D_{1j}}\}_j)$$

  (d) $\mathcal{C}_{\gamma x + \gamma e} \coloneqq \mathsf{CL.Add}(\mathcal{C}_{\gamma x}, \mathsf{CL.SMul}^*(e, \mathcal{C}_{\gamma}))$

$P_i((sid, \mathbf{m}, X, \hat{X}, pk_{\mathsf{cl}}, \mathcal{C}_x); (x_i, \hat{x}_i, k_i))$ $\{P_j\}_{j\neq i, j\in\mathcal{P}}$

Generating Nonces (e, s) Phase 1

$Y \coloneqq \mathsf{H}_0(\mathbf{m}, sid) \in \mathbb{G}_1, Z_i \coloneqq \hat{x}_i Y \in \mathbb{G}_1$

$\pi_i \leftarrow \mathsf{Prove}_{\mathsf{ddh}}((G_1, \hat{X}_i, Y, Z_i); \hat{x}_i)$

Encrypting Phase

Sample $\gamma_i \leftarrow\!\!\$\ \mathbb{Z}_q$, let $D \coloneqq G_1 + \sum_{k\in[\ell]} m_k H_k$

$\mathcal{C}_{\gamma_i} \leftarrow \mathsf{CL.Enc}(\gamma_i, pk_{\mathsf{cl}}), \mathcal{C}_{\gamma_i x} \leftarrow \mathsf{CL.SMul}(\gamma_i, \mathcal{C}_x)$

$\mathcal{C}_{D_{0i}} \leftarrow \mathsf{EG.Enc}(\gamma_i H_0, \hat{X}), \mathcal{C}_{D_{1i}} \leftarrow \mathsf{EG.Enc}(\gamma_i D, \hat{X})$

$\hat{\pi}_i \leftarrow \mathsf{Prove}_{\mathsf{con}}((\mathcal{C}_{\gamma_i}, \mathcal{C}_x, \mathcal{C}_{\gamma_i x}, H_0, \mathcal{C}_{D_{0i}}, D, \mathcal{C}_{D_{1i}}); \gamma_i)$ $\xrightarrow{(Z_i, \pi_i; \mathcal{C}_{\gamma_i}, \mathcal{C}_{\gamma_i x}, \mathcal{C}_{D_{0i}}, \mathcal{C}_{D_{1i}}, \hat{\pi}_i)}$ $\mathsf{Verify}_{\mathsf{ddh}}(\pi_i), \mathsf{Verify}_{\mathsf{con}}(\hat{\pi}_i)$

Generating Nonces (e, s) Phase 2

$Z \coloneqq \sum_{j\in\mathcal{P}} \lambda_{j,\mathcal{P}} Z_j \in \mathbb{G}_1$

$e \coloneqq \mathsf{H}_1(Z, 0), s \coloneqq \mathsf{H}_1(Z, 1) \in \mathbb{Z}_q$

Partially Decrypting Phase

$\mathcal{C}_{\gamma x} \coloneqq \mathsf{CL.Add}(\{\mathcal{C}_{\gamma_j x}\}_j), \mathcal{C}_{\gamma} \coloneqq \mathsf{CL.Add}(\{\mathcal{C}_{\gamma_j}\}_j)$

$\mathcal{C}_{D_0} \coloneqq \mathsf{EG.Add}(\{\mathcal{C}_{D_{0j}}\}_j), \mathcal{C}_{D_1} \coloneqq \mathsf{EG.Add}(\{\mathcal{C}_{D_{1j}}\}_j)$

$\mathcal{C}_{\gamma x+\gamma e} \coloneqq \mathsf{CL.Add}(\mathcal{C}_{\gamma x}, \mathsf{CL.SMul}^*(e, \mathcal{C}_{\gamma}))$

$\mathcal{C}_{sD_0+D_1} \coloneqq \mathsf{EG.Add}(\mathsf{EG.SMul}^*(s, \mathcal{C}_{D_0}), \mathcal{C}_{D_1})$

$(M_i, \tau_i) \leftarrow \mathsf{CL.PartDec}(\mathcal{C}_{\gamma x+\gamma e}, k_i)$

$(M_i', \tau_i') \leftarrow \mathsf{EG.PartDec}(\mathcal{C}_{sD_0+D_1}, \hat{x}_i)$ $\xrightarrow{(M_i, \tau_i, M_i', \tau_i')}$ $\mathsf{Verify}_{\mathsf{cliddh}}(\tau_i), \mathsf{Verify}_{\mathsf{ddh}}(\tau_i')$

Output of Protocol

$a \coloneqq \mathsf{CL.Combine}(\{M_j, \tau_j\}_{j\in\mathcal{P}}, \mathcal{C}_{\gamma x+\gamma e}) \in \mathbb{Z}_q$

$R \coloneqq \mathsf{EG.Combine}(\{M_j', \tau_j'\}_{j\in\mathcal{P}}, \mathcal{C}_{sD_0+D_1}) \in \mathbb{G}_1$

Output $(A \coloneqq \frac{R}{a}, e, s)$

Fig. 1: Robust Threshold BBS+ Signing Protocol

(e) $\mathcal{C}_{sD_0+D_1} \coloneqq \mathsf{EG.Add}(\mathsf{EG.SMul}^*(s, \mathcal{C}_{D_0}), \mathcal{C}_{D_1})$
(f) $(M_i, \tau_i) \leftarrow \mathsf{CL.PartDec}(\mathcal{C}_{\gamma x+\gamma e}, k_i)$
(g) $(M_i', \tau_i') \leftarrow \mathsf{EG.PartDec}(\mathcal{C}_{sD_0+D_1}, \hat{x}_i)$

Broadcast $(M_i, \tau_i, M_i', \tau_i')$.

- **Output.** If $\mathcal{P}$ contains at least $t+1$ (semi-)honest parties with valid τ_j, τ_j', then we can obtain:
  (a) $a \coloneqq \mathsf{CL.Combine}(\{M_j, \tau_j\}_{j\in\mathcal{P}}, \mathcal{C}_{\gamma x+\gamma e}) \in \mathbb{Z}_q$
  (b) $R \coloneqq \mathsf{EG.Combine}(\{M_j', \tau_j'\}_{j\in\mathcal{P}}, \mathcal{C}_{sD_0+D_1}) \in \mathbb{G}_1$
  (c) BBS+ signature $(A \coloneqq \frac{R}{a}, e, s)$.

Remark 1. As the second round only requires the decryption key shares, the protocol can be completed by a different set of parties across the two rounds. This flexibility enhances the protocol's robustness, as it remains functional even if some original parties become unavailable after the first round. Consequently, the system can maintain high availability in dynamic or unstable network environments.

4 Security Analysis

In this section, we prove the security of our scheme via the real-vs-ideal paradigm and the Universal Composability (UC) framework [11].

In the UC-framework proof, the real-world experiment involves: n parties running a protocol Π, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$ which is initialized with a random tape z. $\mathcal{Z}$ controls this experiment via activating the parties, providing their inputs, collecting their outputs, and interacting with $\mathcal{A}$[6]. The experiment ends when $\mathcal{Z}$ halts and outputs a decision bit, denoted as $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)$. The ideal-world experiment involves: n dummy parties, an ideal functionality $\mathcal{F}$, an ideal-process adversary $\mathcal{S}$, and an environment $\mathcal{Z}$. $\mathcal{S}$ can corrupt dummy parties and interact with $\mathcal{F}$ on their behalf; in addition, the simulator can communicate directly with $\mathcal{F}$ according to its specification. $\mathcal{S}$ can also interact with $\mathcal{Z}$ throughout the experiment. Its goal is to simulate the real-world execution so well that $\mathcal{Z}$ cannot tell the difference. The experiment ends when $\mathcal{Z}$ outputs a decision bit, denoted as $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z)$. A protocol Π UC-realizes a functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ s.t. for every environment $\mathcal{Z}$

$$\{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\substack{\lambda\in\mathbb{N},\\ z\in\{0,1\}^*}} \approx_s \{\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z)\}_{\substack{\lambda\in\mathbb{N},\\ z\in\{0,1\}^*}},$$

or $\approx_s$ is replaced with $\approx_c$ when the indistinguishability relies on some computational assumptions.

Functionality of Threshold BBS+. Our scheme realizes an idealized version of the threshold BBS+ task. It is a functionality of threshold BBS+ $\mathcal{F}_{\mathsf{BBS+}}$ in Figure 2, which is borrowed from [25] with added robustness considerations.

Specifically, robustness is shown in Figure 2, where the ideal functionality $\mathcal{F}_{\mathsf{BBS+}}$ does not abort even if the ideal adversary $\mathcal{S}$ issues an abort instruction—unless fewer than $t+1$ (semi-)honest parties remain. Under the honest majority assumption, this situation cannot arise. Moreover, each abort instruction from $\mathcal{S}$ in the ideal world corresponds to malicious behavior in the real protocol. In both worlds, this reveals the identities of corrupted parties, enabling the honest parties to exclude them. This property is commonly known as cheater identification [12,50,47].

4.1 Security Claims

To simplify the notation, we assume that the set of parties for both the key generation protocol and the signing protocol is $\mathcal{P}$ (in practice, the set of signing parties may be a subset of the set of key generation parties). To simplify the security analysis, we assume that $\mathcal{A}$ controls t parties indexed by $\mathcal{P}^*$, and the index set of honest parties is denoted by $\mathcal{H} := \mathcal{P} \setminus \mathcal{P}^*$.

We use the functionalities of CL and ElGamal's DKGs, presented in Figures 3 and 4. Our protocol description explicitly specifies which protocols can be used to realize the ideal functionalities. Therefore, the security proof in such a hybrid model with these ideal functionalities ensures the security of our protocol, according to the UC framework [11]. We provide the definitions of the underlying HSM and DDH hardness assumptions in Appendix A.

Theorem 1. *Based on the HSM and DDH assumptions, our scheme in Section 3 (denoted by Π) UC-realizes the robust threshold BBS+ functionality $\mathcal{F}_{\mathsf{BBS+}}$ in the $(\mathcal{F}_{\mathsf{cldkg}}, \mathcal{F}_{\mathsf{egdkg}})$-hybrid model and random oracle, against a malicious adversary that statically corrupts up to t fixed parties.*

Proof. This theorem is equivalent to the following statement. For any adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that uses $\mathcal{A}$ as a black box s.t. for every environment $\mathcal{Z}$,

$$\left\{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\right\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*} \approx_c \left\{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{BBS+}},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(\lambda, z)\right\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*}.$$

The simulator has oracle access to the adversary $\mathcal{A}$ and simulates for it the executions of our protocols. It forwards all messages from its own environment $\mathcal{Z}$ to $\mathcal{A}$, and vice versa. In the simulated execution, $\mathcal{S}^{\mathcal{A}}$ will interact with $\mathcal{A}$ on behalf of every honest party and on behalf of the ideal oracles $\mathcal{F}_{\mathsf{cldkg}}, \mathcal{F}_{\mathsf{egdkg}}$ and random oracles $\mathsf{H}_0, \mathsf{H}_1$. At the same time, the simulator (as an ideal

[6] We consider *static* adversaries, who announce which t parties to corrupt at the beginning of the experiment. This is the common assumption of existing works.

This functionality is parameterized by the public parameter pp, the threshold t, the party count n, and the message length ℓ. Let $\hat{\mathcal{P}} = [n]$ denote the set of indices for all initial parties. The ideal adversary $\mathcal{S}$ can corrupt up to t parties that are indexed by $\mathcal{P}^*$. This functionality ensures synchrony using Katz et al.'s framework [34], meaning it advances from one round to the next only when all honest parties indicate that it should.

Key Generation.

- On receiving (keygen, sid) from some party P_i for $i \in [n]$ where sid is fresh, send $(\mathsf{keygen}, sid, i)$ to $\mathcal{S}$.
- After receiving (keygen, sid) from all parties $P_1, \cdots, P_n$, if sid is agreed-upon, then
  1. run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{BBS+.KGen}(pp, \ell)$
  2. send pk as the public key to $\mathcal{S}$
  3. if receiving (abort, sid, j) from $\mathcal{S}$ s.t. $j \in \mathcal{P}^*$, remove j from the set $\hat{\mathcal{P}}$, $\hat{\mathcal{P}} = \hat{\mathcal{P}} \setminus \{j\}$
  4. send pk as the public key to all parties in $\hat{\mathcal{P}}$ and ignore any further call to keygen.

Signing.

- On receiving $(\mathsf{sign}, sid, \mathbf{m})$ from some party P_i for $i \in \hat{\mathcal{P}}$, if keygen was already called and sid was never used, send $(\mathsf{sign}, sid, \mathbf{m}, i)$ to $\mathcal{S}$.
- After receiving $(\mathsf{sign}, sid, \mathbf{m})$ from more than t parties, denote this signer set as $\mathcal{P}$, then
  1. run $\sigma \leftarrow \mathsf{BBS+.Sign}(\mathsf{sk}, \mathbf{m})$
  2. send the signature σ to $\mathcal{S}$
  3. if receiving (abort, sid, j) from $\mathcal{S}$ s.t. $j \in \mathcal{P}^* \wedge \mathcal{P}$, remove j from $\mathcal{P}$, $\mathcal{P} = \mathcal{P} \setminus \{j\}$.
  4. if $|\mathcal{P}| > t$ still holds, send σ to all parties in $\mathcal{P}$
  5. store (signed, sid) in memory.

Fig. 2: Robust Threshold BBS+ Functionality $\mathcal{F}_{\mathsf{BBS+}}$

adversary) interacts with the ideal functionality $\mathcal{F}_{\mathsf{BBS+}}$ on behalf of every corrupt party. In the following, we present the simulator $\mathcal{S}$'s interactions with $\mathcal{A}$ and $\mathcal{F}_{\mathsf{BBS+}}$.

Simulation of Key Generation. Let BBS+'s public key be X received from $\mathcal{F}_{\mathsf{BBS+}}$.

1. On receiving $(\mathsf{keygen}, sid, i)$ for some $i \in \mathcal{H}$ from $\mathcal{F}_{\mathsf{BBS+}}$, reply (keygen, sid) on behalf of $P_j, j \in \mathcal{P}^*$
2. On receiving $\mathsf{pk} = (\mathbf{H}, X)$ as the public key from $\mathcal{F}_{\mathsf{BBS+}}$, $\mathcal{S}$ simulates the key generation execution for $\mathcal{A}$.
   (a) As an IND-CPA adversary of CL encryption, $\mathcal{S}$ receives $pk_{\mathsf{cl}} \in \mathcal{G}^q$ as its public key. On receiving $(\mathsf{advshares}, sid, \{k_j\}_{j\in\mathcal{P}^*})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{cldkg}}$, construct a degree-$t$ polynomial $P_0(\cdot)$ over $\mathcal{G}^q$ using Lagrange interpolation in the exponent s.t. $P_0(0) = pk_{\mathsf{cl}}^{\Delta^2}$ and $P_0(j) = g_q^{\Delta \cdot k_j}$ for each $j \in \mathcal{P}^*$. Output $(sid, pk_{\mathsf{cl}}, \{P_0(i)\}_{i\in[n]})$ on behalf of $\mathcal{F}_{\mathsf{cldkg}}$.
   (b) As an IND-CPA adversary of ElGamal encryption, $\mathcal{S}$ receives $\hat{X} \leftarrow\!\!\$\ \mathbb{G}_1$ as its public key. On receiving $(\mathsf{advshares}, sid, \{\hat{x}_j\}_{j\in\mathcal{P}^*})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{egdkg}}$, construct a degree-$t$ polynomial $P_1(\cdot)$ over $\mathbb{G}_1$ using Lagrange interpolation in the exponent s.t. $P_1(0) = \hat{X}$ and $P_1(j) = \hat{x}_j G_1$ for each $j \in \mathcal{P}^*$. Output $(sid, \hat{X}, \{P_1(i)\}_{i\in[n]})$ on behalf of $\mathcal{F}_{\mathsf{egdkg}}$.
   (c) On receiving $(\mathsf{advshares}, sid, \{x_j\}_{j\in\mathcal{P}^*})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{egdkg}}$, construct a degree-$t$ polynomial $P_2(\cdot)$ over $\mathbb{G}_2$ using Lagrange interpolation in the exponent s.t. $P_2(0) = X$ and $P_2(j) = x_j G_2$ for each $j \in \mathcal{P}^*$. Output $(sid, X, \{P_2(i)\}_{i\in[n]})$ on behalf of $\mathcal{F}_{\mathsf{egdkg}}$. Let $\mathbf{H}$ be derived from X.
   (d) For each $i \in \mathcal{H}$, $\mathcal{S}$ generates $\mathcal{C}_{r_i} \leftarrow \mathsf{CL.Enc}(pk_{\mathsf{cl}}, r_i)$ with $r_i \leftarrow\!\!\$\ \mathbb{Z}_q$ and simulates the proof π_i. Broadcast $(\mathcal{C}_{r_i}, \pi_i)$ on behalf of honest party P_i.
   (e) Combine $\mathcal{C}_x$ honestly as prescribed.
3. If receiving P_j's invalid proof π_j, send (abort, sid, j) to $\mathcal{F}_{\mathsf{BBS+}}$ and remove P_j from the party set.

Simulation of Random Oracle $\mathsf{H}_0(\cdot)$. On receiving the query with input in from $\mathcal{A}$, pick $b \leftarrow\!\!\$\ \mathbb{Z}_q$, program $\mathsf{H}_0(\mathsf{in}) = bG_1$ and store (in, b) in a table $\mathcal{T}$.

Simulation of Signing. Let the generated signature be σ received from $\mathcal{F}_{\mathsf{BBS+}}$.

- Upon receiving (cldkg, sid) from some party P_i such that sid is fresh, send (cldkg, sid, i) to $\mathcal{S}$.
- After receiving (cldkg, sid) from all parties, if sid is agreed upon, then
  (a) sample $k \leftarrow\!\$\ \mathcal{D}_q$ and compute $pk_{\mathsf{cl}} \coloneqq g_q^k$,
  (b) receive $(\mathsf{advshares}, sid, \{k_j\}_{j\in\mathcal{P}^*})$ from $\mathcal{S}$,
  (c) sample a random polynomial $p(\cdot)$ of degree t with each coefficient in the interval $[0, B]$[a] s.t. $p(j) = k_j$ for each $j \in \mathcal{P}^*$ and $p(0) = \Delta \cdot k$,
  (d) for $i \in [n]$, compute $k_i \coloneqq p(i)$ and $pk_i \coloneqq g_q^{\Delta \cdot k_i}$,
  (e) send $(sid, pk_{\mathsf{cl}}, \{pk_i\}_{i\in[n]})$ to each P_i.

[a] According to [8, Theorem 16] and [49, Section III A], B could be set as $2^{2\lambda_s+1} \cdot t^2 n^n q\tilde{s}$.

Fig. 3: Functionality of CL's DKG $\mathcal{F}_{\mathsf{cldkg}}$

- Upon receiving (egdkg, sid) from some party P_i such that sid is fresh, send (egdkg, sid, i) to $\mathcal{S}$.
- After receiving (egdkg, sid) from all parties, if sid is agreed upon, then
  (a) sample $\hat{x} \leftarrow\!\$\ \mathbb{Z}_q$ and compute $\hat{X} \coloneqq \hat{x}G_1$,
  (b) receive $(\mathsf{advshares}, sid, \{\hat{x}_j\}_{j\in\mathcal{P}^*})$ from $\mathcal{S}$,
  (c) sample a random polynomial $p(\cdot)$ of degree t over $\mathbb{Z}_q$ s.t. $p(j) = \hat{x}_j$ for each $j \in \mathcal{P}^*$ and $p(0) = \hat{x}$,
  (d) for $i \in [n]$, compute $\hat{x}_i \coloneqq p(i) \bmod q$ and $\hat{X}_i \coloneqq \hat{x}_i G_1$,
  (e) send $(sid, \hat{X}, \{\hat{X}_i\}_{i\in[n]})$ to each P_i.

Fig. 4: Functionality of ElGamal's DKG $\mathcal{F}_{\mathsf{egdkg}}$

1. On receiving $(\mathsf{sign}, sid, \mathbf{m}, i)$ for some $i \in \mathcal{H}$ from $\mathcal{F}_{\mathsf{BBS+}}$, reply $(\mathsf{sign}, sid, \mathbf{m})$ on behalf of $P_j, j \in \mathcal{P}^*$.
2. On receiving $\sigma = (A, e, s)$ from $\mathcal{F}_{\mathsf{BBS+}}$, $\mathcal{S}$ simulates the signing execution for $\mathcal{A}$.
   (a) Retrieve $((\mathbf{m}, sid), b)$ from the table $\mathcal{T}$, program $\mathsf{H}_1(b\hat{X}, 0) = e$ and $\mathsf{H}_1(b\hat{X}, 1) = s$. If the programming fails, $\mathcal{S}$ aborts.
   (b) Construct a t-degree polynomial $P_3(\cdot)$ over $\mathbb{G}_1$ using Lagrange interpolation in the exponent s.t. $P_3(0) = b\hat{X}$ and $P_3(j) = \hat{x}_j bG_1$ for each $j \in \mathcal{P}^*$.
   (c) For each $i \in \mathcal{H}$, pick $\gamma_i \leftarrow\!\$\ \mathbb{Z}_q$ and generate $\mathcal{C}_{\gamma_i}, \mathcal{C}_{\gamma_i x}, \mathcal{C}_{D_{0i}}, \mathcal{C}_{D_{1i}}$ honestly as prescribed.
   (d) Simulate the zero-knowledge proofs $\pi_i, \hat{\pi}_i$. Broadcast $(P_3(i), \pi; \mathcal{C}_{\gamma_i}, \mathcal{C}_{\gamma_i x}, \mathcal{C}_{D_{0i}}, \mathcal{C}_{D_{1i}}, \hat{\pi}_i)$ on behalf of each honest party P_i.
   (e) For each $j \in \mathcal{P}^*$, receive the first-round messages $(Z_j, \pi_j; \mathcal{C}_{\gamma_j}, \mathcal{C}_{\gamma_j x}, \mathcal{C}_{D_{0j}}, \mathcal{C}_{D_{1j}}, \hat{\pi}_j)$ from $\mathcal{A}$, combine $\mathcal{C}_{\gamma(x+e)}, \mathcal{C}_{\gamma(D+sH_0)}$ honestly as prescribed.
   (f) Pick $a^* \leftarrow\!\$\ \mathbb{Z}_q$ and compute $R^* = a^* A$. Regard a^* as dummy $\gamma(x + e)$ and R^* as dummy $\gamma(D + sH_0)$.
   (g) Invoke $\mathsf{CL.Simulate}(\mathcal{C}_{\gamma(x+e)}, \{k_j\}_{j\in\mathcal{P}^*}, a^*) \to \{M_i, \tau_i\}_{i\in\mathcal{H}}$ where $M_i \in \mathcal{G}$ is P_i's partial result of CL decryption.
   (h) Invoke $\mathsf{EG.Simulate}(\mathcal{C}_{\gamma(D+sH_0)}, \{\hat{x}_j\}_{j\in\mathcal{P}^*}, R^*) \to \{\hat{M}_i, \hat{\tau}_i\}_{i\in\mathcal{H}}$ where $\hat{M}_i \in \mathbb{G}_1$ is P_i's partial result of ElGamal decryption.
   (i) Broadcast $(M_i, \tau_i, \hat{M}_i, \hat{\tau}_i)$ on behalf of each honest party P_i.
3. If receiving P_j's invalid zero-knowledge proof during the above process, send (abort, sid, j) to $\mathcal{F}_{\mathsf{BBS+}}$ and remove P_j from the party set.

Analysis. We use a sequence of hybrid experiments to show the indistinguishability between the ideal and real worlds. It begins with the real world, thus we have

$$\mathsf{H}_0 = \{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*}.$$

Hybrid H_1. In this hybrid, all honest parties and ideal oracles are replaced with $\mathcal{S}$, which plays their roles to interact with $\mathcal{A}$ and $\mathcal{Z}$. With only syntactical changes, it follows that $\mathsf{H}_1 = \mathsf{H}_0$.

Hybrid H_2. In this hybrid, $\mathcal{S}$ programs each output of random oracle H_0. On receiving the query input in, $\mathcal{S}$ picks $b \leftarrow\!\$\ \mathbb{Z}_q$ and responds bG_1 as the result of $\mathsf{H}_0(\mathsf{in})$. Since the distribution of the simulated output is still uniform over $\mathbb{G}_1$, $\mathsf{H}_2 = \mathsf{H}_1$.

Hybrid H_3. In this hybrid, $\mathcal{S}$ receives (e, s) from $\mathcal{F}_{\mathsf{BBS+}}$ and programs $\mathsf{H}_1(b\hat{X}, 0) = e, \mathsf{H}_1(b\hat{X}, 1) = s$ in the simulated signing execution, where b was retrieved from the records $\mathsf{H}_0(\mathbf{m}, sid) = bG_1$. If the programming is successful, the generated nonces would be e, s. Otherwise $\mathcal{S}$ would abort. Failed programming is always due to the situation that $\mathsf{H}_1(b\hat{X}, 0)$ or $\mathsf{H}_1(b\hat{X}, 1)$ has been programmed. Since b is selected uniformly in $\mathbb{Z}_q$, this happens with the probability of at most $2q_sq_h/q$ where q_s, q_h denote the maximum number of signing queries and random oracle queries, respectively. Thus, $\mathsf{H}_3 \approx_s \mathsf{H}_2$.

Hybrid H_4. In this hybrid, $\mathcal{S}$ simulates P_i's partial evaluation of VRF for each honest party $i \in \mathcal{H}$. Based on Lagrange interpolation, $\mathcal{S}$ can build t-degree polynomials $P_3(\cdot)$ over $\mathbb{G}_1$ and set $P_3(i)$ as party P_i's partial evaluation of the VRF with $P_3(0) = b\hat{X}$. At the same time, $\mathcal{S}$ simulates the zero-knowledge proof π_i.

Since $\mathcal{A}$'s partial evaluation $\hat{x}_j bG_1$ have been placed in $P_3(j)$, where $\hat{x}_j$ is received on behalf of $\mathcal{F}_{\mathsf{egdkg}}$, the simulated partial evaluations will combine $P_3(0) = b\hat{X}$ together with $\mathcal{A}$'s partial evaluations. Moreover, since $b\hat{X} = \hat{x}\mathsf{H}_0(\mathbf{m}, sid)$, the simulated shares are the same as those of the real execution. Based on the zero-knowledge property of proofs π_i, we have $\mathsf{H}_4 \approx_s \mathsf{H}_3$.

Hybrid H_5. In this hybrid, $\mathcal{S}$ can receive $\{x_j\}_{j\in\mathcal{P}^*}$ on behalf of $\mathcal{F}_{\mathsf{egdkg}}$. With only syntactical changes, it follows that $\mathsf{H}_5 = \mathsf{H}_4$.

Hybrid H_6. In this hybrid, $\mathcal{S}$ makes the following changes:

- As the adversary of CL's IND-CPA security, $\mathcal{S}$ receives pk_{cl} and sets it as CL's DKG output on behalf of $\mathcal{F}_{\mathsf{cldkg}}$.
- As the adversary of ElGamal's IND-CPA security, $\mathcal{S}$ receives $\hat{X}$ and sets it as ElGamal's DKG output on behalf of $\mathcal{F}_{\mathsf{egdkg}}$.
- $\mathcal{S}$ picks a random r_i and generates r_i's ciphertext $\mathcal{C}_{r_i}$ to simulate $\mathcal{C}_{x_i}$ (along with its zero-knowledge proof).
- $\mathcal{S}$ simulates P_i's partial decryption of CL and ElGamal ciphertexts $\mathcal{C}_{\gamma(x+e)}, \mathcal{C}_{\gamma(D+sH_0)}$ for each honest party $i \in \mathcal{H}$. With the received $A \in \mathbb{G}_1$ from $\mathcal{F}_{\mathsf{BBS+}}$, $\mathcal{S}$ picks a random value $a^* \in \mathbb{Z}_q$ and computes $R^* = a^*A$. Then, with the received $\{k_j\}_{j\in\mathcal{P}^*}$ on behalf of $\mathcal{F}_{\mathsf{cldkg}}$, $\mathcal{S}$ can invoke

$$\mathsf{CL.Simulate}(\mathcal{C}_{\gamma(x+e)}, \{k_j\}_{j\in\mathcal{P}^*}, a^*) \to \{M_i, \tau_i\}_{i\in\mathcal{H}}$$

$$\mathsf{EG.Simulate}(\mathcal{C}_{\gamma(D+sH_0)}, \{\hat{x}_j\}_{j\in\mathcal{P}^*}, R^*) \to \{\hat{M}_i, \hat{\tau}_i\}_{i\in\mathcal{H}}$$

  to let CL's and ElGamal's decryption results be m and M, respectively.

If the adversary $\mathcal{A}$ can distinguish $\mathcal{C}_{r_i}$ from the correct $\mathcal{C}_{x_i}$, then $\mathcal{S}$ can break CL's IND-CPA security by embedding the challenge ciphertext into $\mathcal{C}_{r_i}$. It is based on the HSM assumption. The indistinguishability between simulated partial decryptions in this hybrid and real partial decryptions in the above hybrids is a crucial step in our proof, and we prove this using Lemmas 2 and 4 based on the HSM and DDH assumptions, respectively. Thus, we have $\mathsf{H}_6 \approx_c \mathsf{H}_5$.

Hybrid H_7. In this hybrid, $\mathcal{S}$ receives X from $\mathcal{F}_{\mathsf{BBS+}}$ and sets X as BBS+'s DKG output on behalf of $\mathcal{F}_{\mathsf{egdkg}}$. $\mathcal{A}$ cannot notice this difference with only t shares of BBS+'s secret key. It follows that $\mathsf{H}_7 = \mathsf{H}_6$.

Since hybrid H_7 exactly demonstrates the ideal world, we have $\mathsf{H}_7 = \left\{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{BBS+}},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(\lambda, z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$, and by transitivity, we finally get Theorem 1 based on the HSM and DDH assumptions. □

Simulation of CL Partial Decryption. Given a honestly generated CL ciphertext $\mathcal{C} := (c_1, c_2)$, t secret key shares $\{k_j\}_{j\in\mathcal{P}^*}$ (hold by $\mathcal{A}$), the simulator can pick a random plaintext $m \leftarrow\!\!\$\ \mathbb{Z}_q$ and simulate honest parties' partial decryptions as below. With $\mathcal{P}_0 := \{0\} \cup \mathcal{P}^*$ and $\lambda^{(i)}_{j,\mathcal{P}_0} := \prod_{k\in\mathcal{P}_0\setminus\{j\}} \frac{i-k}{j-k}$ (specially, $\lambda_{j,\mathcal{P}_0} := \lambda^{(0)}_{j,\mathcal{P}_0}$), define $\mathsf{CL.Simulate}(\mathcal{C}, \{k_j\}_{j\in\mathcal{P}^*}, m) \to \{M_i, \tau_i\}_{i\in\mathcal{H}}$ as:

- let $\bar{M} := \frac{c_2^{\Delta}}{f^{\Delta\cdot m}} \in \mathcal{G}$, for each $i \in \mathcal{H}$

$$M_i := \bar{M}^{\Delta\cdot\lambda^{(i)}_{0,\mathcal{P}_0}} \cdot \prod_{j\in\mathcal{P}^*} c_1^{\Delta\cdot k_j\cdot\lambda^{(i)}_{j,\mathcal{P}_0}}, \tag{1}$$

- simulate the zero-knowledge proof τ_i for each $i \in \mathcal{H}$.

Lemma 1. *The simulated partial plaintexts* $\{M_i, \tau_i\}_{i\in\mathcal{H}}$ *satisfy* $\mathsf{CL.Combine}(\{M_i, \tau_i\}_{i\in\mathcal{P}}, \mathcal{C}) \to m$ *together with* $\mathcal{A}$*'s partial plaintexts* $\{M_j\}_{j\in\mathcal{P}^*}$ *with* $M_j \coloneqq c_1^{\Delta\cdot k_j}$.

Proof. Define a t-degree function over the class group $\mathcal{G}$, $\mathbb{P}_{\bar{M}}(\cdot)$ s.t. $\mathbb{P}_{\bar{M}}(0) = \bar{M}$, and $\mathbb{P}_{\bar{M}}(j) = c_1^{k_j}$ for $j \in \mathcal{P}^*$. According to Lagrange interpolation in the exponent, we have $M_i = (\mathbb{P}_{\bar{M}}(i))^{\Delta}$. After receiving $M_j = c_1^{\Delta k_j} = (\mathbb{P}_{\bar{M}}(j))^{\Delta}$ for $j \in \mathcal{P}^*$ from $\mathcal{A}$, they satisfy

$$\prod_{j\in\mathcal{P}} M_j^{\Delta\lambda_{j,\mathcal{P}}} = \prod_{j\in\mathcal{P}} (\mathbb{P}_{\bar{M}}(j))^{\Delta^2\lambda_{j,\mathcal{P}}} = \mathbb{P}_{\bar{M}}(0)^{\Delta^2} = \frac{c_2^{\Delta^3}}{f^{\Delta^3\cdot m}}.$$

This makes $\mathsf{CL.Combine}$ output m. □

Lemma 2. *Based on the HSM assumption, the simulated* $\{M_i, \tau_i\}_{i\in\mathcal{H}}$ *is indistinguishable from the real partial plaintexts generated via* $\mathsf{CL.PartDec}(k_i, \mathcal{C})$ *for any PPT adversary* $\mathcal{A}$.

Proof. Let m^* be the plaintext of $\mathcal{C}$ under public key $pk_{\mathsf{cl}} = g_q^k$. In the real $\mathsf{CL.PartDec}$ algorithm, P_i's partial plaintext is $M_i = c_1^{\Delta\cdot k_i} \in \mathcal{G}^q$, while the simulated one can be denoted as $M_i = c_1^{\Delta\cdot k_i} f^{\Delta\cdot(\delta_m)_i}$ which lies in $\mathcal{G}$ where $(\delta_m)_i$ is a share of $\delta_m = m^* - m$. The indistinguishability is based on the HSM assumption. Given a random class group element B, if the adversary $\mathcal{A}$ can distinguish the above two M_i's, then there exists another adversary $\mathcal{B}$ who can determine $B \in \mathcal{G}$ or $B \in \mathcal{G}^q$. We use a sequence of games to demonstrate the indistinguishability.

Game G_0. $\mathcal{S}$ generates $M_i = c_1^{\Delta\cdot k_i}$ and τ_i honestly for each honest party $i \in \mathcal{H}$, which aligns with the real world.

Game G_1. $\mathcal{S}$ computes $\bar{M} = \frac{c_2^{\Delta}}{f^{\Delta\cdot m^*}}$ and sets M_i as eq. (1). As m^* is the plaintext of $\bar{C}$, M_i generated in the above interpolation form is the same as that in Game 0. Thus, $G_1 = G_0$.

Game G_2. $\mathcal{S}$ simulates the zero-knowledge proof τ_i. Based on the zero-knowledge property, $G_2 \approx_s G_1$.

Game G_3. $\mathcal{S}$ changes the generation of CL's public key. In this game, $\mathcal{S}$ sets $pk_{\mathsf{cl}} \coloneqq g^{k'} \in \mathcal{G}$ with a random $k' \leftarrow\!\$\ \mathcal{D}_q$. This is realized by $\mathcal{S}$ simulating the functionality of the CL-DKG, where the public key is controlled by functionality.

To show the indistinguishability between G_2 and G_3, we construct the adversary $\mathcal{B}$ with the input of B. It sets $pk_{\mathsf{cl}} = B$. If $B \in \mathcal{G}^q$, it is the same as that of G_2. If $B \in \mathcal{G}$, it perfectly simulates that of G_3. Thus, $\mathcal{B}$ can solve the HSM problem if $\mathcal{A}$ can notice the difference between G_2 and G_3. Finally, we have $G_3 \approx_c G_2$ based on the HSM assumption.

Game G_4. $\mathcal{S}$ picks a random plaintext $m \leftarrow\!\$\ \mathbb{Z}_q$, computes $\bar{M} = \frac{c_2^{\Delta}}{f^{\Delta\cdot m}}$, and then sets M_i as eq. (1). This modification results in a difference: $\bar{M} = pk_{\mathsf{cl}}^{\Delta\cdot\rho}$ in the above games, while $\bar{M} = pk_{\mathsf{cl}}^{\Delta\cdot\rho} f^{\Delta\cdot\delta_m}$ in this game, where ρ is the randomness used in $\mathcal{C}$. As $\mathcal{S}$ has set $pk_{\mathsf{cl}} \in \mathcal{G}$ from the last game, the two $\bar{M}$'s are both uniformly random in $\mathcal{G}$. Thus, $\mathcal{A}$ cannot notice this difference, that is $G_4 = G_3$.

Game G_5. In this game, $\mathcal{S}$ receives a valid public key $pk_{\mathsf{cl}} \in \mathcal{G}^q$ and sets it as CL's DKG output on behalf of $\mathcal{F}_{\mathsf{cldkg}}$. The indistinguishability between G_5 and G_4 remains due to the HSM assumption, similarly to that between G_2 and G_3. Note that the setting of G_5 is consistent with that of Theorem 1's proof, where the simulator serves as the adversary for CL's IND-CPA security.

Finally, we get $G_5 \approx_c G_0$ based on the HSM assumption. Game G_5 demonstrates our simulated partial decryption, which is indistinguishable from the real output of $\mathsf{CL.PartDec}$ for any PPT adversary $\mathcal{A}$. □

Simulation of ElGamal Partial Decryption. Given a honestly generated ElGamal ciphertext $\mathcal{C} \coloneqq (A_1, A_2) \in \mathbb{G}_1^2$ (generated in a distributed manner), t secret key shares $\{\hat{x}_j\}_{j\in\mathcal{P}^*}$ (hold by $\mathcal{A}$), $\mathcal{S}$ can pick $M \leftarrow\!\$\ \mathbb{G}_1$ and simulate honest parties' partial decryptions as below. Define $\mathsf{EG.Simulate}(\mathcal{C}, \{\hat{x}_j\}_{j\in\mathcal{P}^*}, M) \to \{M_i, \tau_i\}_{i\in\mathcal{H}}$ as:

- let $\bar{M} \coloneqq A_2 - M$, for each $i \in \mathcal{H}$

$$M_i \coloneqq \lambda_{0,\mathcal{P}_0}^{(i)} \cdot \bar{M} + \sum_{j\in\mathcal{P}^*} \lambda_{j,\mathcal{P}_0}^{(i)} \cdot \hat{x}_j A_1, \tag{2}$$

- simulate the zero-knowledge proof τ_i for each $i \in \mathcal{H}$.

Lemma 3. *The simulated* $\{M_i, \tau_i\}_{i\in\mathcal{H}}$ *satisfy* $\mathsf{EG.Combine}(\{M_i,\tau_i\}_{i\in\mathcal{P}}, \mathcal{C}) \to M$ *where* $M_j := \hat{x}_j A_1$ *for* $j \in \mathcal{P}^*$.

Proof. According to Lagrange interpolation, the simulated M_i can satisfy $\sum_{i\in\mathcal{H}} \lambda_{i,\mathcal{P}} M_i + \sum_{j\in\mathcal{P}^*} \lambda_{j,\mathcal{P}} M_j = \bar{M}$ together with $\mathcal{A}$'s $\{M_j\}_j$. Thus, $\mathsf{EG.Combine}$ outputs M with these inputs. □

Lemma 4. *Based on the DDH assumption over* $\mathbb{G}_1$*, the simulated* $\{M_i, \tau_i\}_{i\in\mathcal{H}}$ *is indistinguishable from the real partial plaintexts generated via* $\mathsf{EG.PartDec}(\hat{x}_i, \mathcal{C})$ *for any PPT adversary* $\mathcal{A}$.

Proof. Let M^* be the correct plaintext of $\mathcal{C} = (A_1, A_2)$ under the public key $\hat{X}$. In the real execution, M_i is defined as $\hat{x}_i A_1$, while in our simulation, the simulated M_i can be denoted as $\hat{x}_i A_1 + (\delta_M)_i$ where $(\delta_M)_i$ is a share of $\delta_M = M^* - M$. If $\mathcal{A}$ can notice this difference, then there exists another adversary $\mathcal{B}$ who can solve the DDH problem of $(G_1, X^*, Y^*, Z^*) \in \mathbb{G}_1^4$ with $X^* = x^* G_1, Y^* = y^* G_1$, distinguishing $Z^* = x^* y^* G_1$ or $Z^* = z^* G_1$ where $x^*, y^*, z^* \leftarrow\!\$\ \mathbb{Z}_q$.

Game G_0. $\mathcal{S}$ generates $M_i = \hat{x}_i A_1$ and τ_i honestly, which is the real world.

Game G_1. $\mathcal{S}$ computes $\bar{M} = A_2 - M^*$ and sets M_i as eq. (2). M_i generated in this way is equal to $\hat{x}_i A_1$, thus $G_1 = G_0$.

Game G_2. $\mathcal{S}$ simulates τ_i. It is $G_2 \approx_s G_1$ due to the zero-knowledge property.

Game G_3. $\mathcal{S}$ picks a index i^* of honest party, and changes the generation way of P_{i^*}'s ElGamal ciphertext (A_{1i^*}, A_{2i^*}) as its contribution to $\mathcal{C} = (\sum_i A_{1i}, \sum_i A_{2i})$. Concretely, it samples $A_{1i^*}, Z \leftarrow\!\$\ \mathbb{G}_1$, and generates $A_{2i^*} = Z + M_{i^*}^*$ where $M_{i^*}^*$ is P_{i^*}'s contribution to the plaintext M^*.

To show the indistinguishability between G_2 and G_3, we construct a DDH distinguisher $\mathcal{B}$. Given a DDH problem of $(G_1, X^*, Y^*, Z^*) \in \mathbb{G}_1^4$, $\mathcal{B}$ sets $(\hat{X}, A_{1i^*}, A_{2i^*}) = (X^*, Y^*, Z^* + M_{i^*}^*)$. If $(G_1, X^*, Y^*, Z^*) \in \mathbb{G}_1^4$ is a DDH tuple, then it is the same as that of Game G_2. If (G_1, X^*, Y^*, Z^*) is not a DDH tuple, it is distributed identically to that of Game G_3. Thus, if $\mathcal{A}$ can notice the modification between Game G_2 and Game G_3, then $\mathcal{B}$ can solve the DDH problem.

Game G_4. $\mathcal{S}$ picks a random $M \leftarrow\!\$\ \mathbb{G}_1$, computes $\bar{M} = A_2 - M$ and sets M_i as eq. (2). The difference now is: in G_3 $\bar{M} = Z + \sum_{i\neq i^*} r_i \hat{X}$ while $\bar{M} = Z + \sum_{i\neq i^*} r_i \hat{X} + M^* - M$ in this game. Since $Z \leftarrow\!\$\ \mathbb{G}_1$ is independent of the public key $\hat{X}$ from Game G_3, the two $\bar{M}$'s are both uniformly random in $\mathbb{G}_1$. That is $G_4 = G_3$. This game is exactly our simulated partial decryption.

Finally, we get that the simulated partial plaintexts are indistinguishable from the real ones based on the DDH assumption. □

Remark 2 (Security under dishonest-majority settings). As specified in the ideal functionality $\mathcal{F}_{\mathsf{BBS+}}$, if the condition $n \geq 2t + 1$ is not met (i.e., under dishonest-majority settings), Step 4 of the signing process may result in $|\mathcal{P}| \leq t$ after excluding malicious parties, leading to protocol abort. However, this does not compromise the underlying security. It follows from the definition of $\mathcal{F}_{\mathsf{BBS+}}$, which outputs a signature only via prescribed interfaces for each signing request initiated by an authorized set of parties. Our security proof establishes that the simulator succeeds, demonstrating that $\mathcal{A}$ learns nothing about honest parties' key shares beyond the signature itself. Thus, it proves that $\mathcal{A}$ cannot produce a signature on a new message without the honest parties' participation.

Furthermore, our scheme achieves identifiable abort in dishonest-majority settings, allowing honest parties to identify the specific malicious parties who caused the protocol to abort. This property is guaranteed by the ZKPs integrated into our construction. More generally, the scheme that satisfies robustness always supports cheater identification, as robustness requires honest parties to consistently and timely exclude malicious transcripts from the protocol execution.

5 Zero-Knowledge Proof for $\mathcal{R}_{\mathsf{con}}$

In this section, we construct the non-interactive zero-knowledge (NIZK) proof for the relation $\mathcal{R}_{\mathsf{con}}$. It can be first designed as a Σ-protocol and converted to a non-interactive proof system via the Fiat-Shamir transform [27] in the random oracle model [4]. Following the analysis in [12], since we

do not require this proof to be a proof of knowledge in our UC security proof, the Fiat-Shamir transform is sufficient instead of the more expensive Fischlin transform [28].

A Σ-protocol for a statement-witness pair $(\mathsf{st}, \mathsf{wi})$ in a binary relation $\mathcal{R}$ is a three-round protocol between the prover $\mathcal{P}$ and the verifier $\mathcal{V}$.

1. $\mathcal{P}$ takes a random tape τ as input and sends the first-round message com to $\mathcal{V}$.
2. $\mathcal{V}$ sends a random challenge ch to $\mathcal{P}$.
3. $\mathcal{P}$ takes $(\mathsf{st}, \mathsf{wi}, \tau, \mathsf{ch})$ as input and sends a response resp to $\mathcal{V}$.
4. $\mathcal{V}$ verifies whether $(\mathsf{com}, \mathsf{ch}, \mathsf{resp})$ is valid, and outputs a bit to indicate it.

Please refer to Appendix B.1 for the security definition of the Σ-protocol.

The relation $\mathcal{R}_{\mathsf{con}}$ is a consistency relation between CL and ElGamal plaintexts. Given the CL and ElGamal public keys $pk_{\mathsf{cl}} \in \mathcal{G}^q, \hat{X} \in \mathbb{G}_1$, the statement $\mathsf{st} = (\mathcal{C}_{\gamma_i}, \mathcal{C}_x, \mathcal{C}_{\gamma_i x}, H_0, \mathcal{C}_{D_{0i}}, D, \mathcal{C}_{D_{1i}})$ satisfies:

- γ_i is the plaintext of the CL ciphertext $\mathcal{C}_{\gamma_i} = (C_{\gamma_i 1}, C_{\gamma_i 2}) = (g_q^{\beta_1}, f^{\gamma_i} pk_{\mathsf{cl}}^{\beta_1})$;
- γ_i is the multiplicative factor between the plaintexts of CL ciphertexts $\mathcal{C}_{\gamma_i x}$ and $\mathcal{C}_x = (C_{x_1}, C_{x_2})$, with $\mathcal{C}_{\gamma_i x} = (C_{M_1}, C_{M_2}) = (C_{x_1}^{\gamma_i} g_q^{\beta_2}, C_{x_2}^{\gamma_i} pk_{\mathsf{cl}}^{\beta_2})$;
- $\gamma_i H_0$ is the plaintext of the ElGamal ciphertext $\mathcal{C}_{D_{0i}} = (C_{01}, C_{02}) = (\alpha_1 G_1, \gamma_i H_0 + \alpha_1 \hat{X})$;
- $\gamma_i D$ is the plaintext of the ElGamal ciphertext $\mathcal{C}_{D_{1i}} = (C_{11}, C_{12}) = (\alpha_2 G_1, \gamma_i D + \alpha_2 \hat{X})$.

Besides γ_i being the primary witness, the randomness $\alpha_1, \alpha_2, \beta_1, \beta_2$ used in the ciphertexts can be viewed as implicit witnesses.

Σ-Protocol of $\mathcal{R}_{\mathsf{con}}$. In its construction, a noteworthy point is that although the witness γ_i is a plaintext value in $\mathbb{Z}_q$, we cannot mask it using another randomness $\gamma_i' \in \mathbb{Z}_q$ in the form of $z_i \coloneqq \gamma_i' + \mathsf{ch} \cdot \gamma_i \mod q$ where ch is the challenge value. This is because in the ciphertext $\mathcal{C}_{\gamma_i x}$, γ_i appears as the exponent of the ciphertext elements C_{x_1} and C_{x_2}, which belong to the class group of unknown order. If we compute the corresponding response z_i in $\mathbb{Z}_q$ in the form above, the verification procedure will run into an issue. Taking the base C_{x_1} as an example, $C_{x_1}^{z_i \bmod q} \neq C_{x_1}^{\gamma_i'} C_{x_1}^{\mathsf{ch} \cdot \gamma_i}$ will cause the verification to fail. Instead, we choose γ_i' from a larger set $[0, 2^{\lambda+\lambda_s} q]$ for masking γ_i where λ is the security parameter and bit-length of ch, λ_s is the statistical security parameter. Then, the corresponding response $z_i \coloneqq \gamma_i' + \mathsf{ch} \cdot \gamma_i$ is generated over the integers. In this way, $C_{x_1}^{z_i} = C_{x_1}^{\gamma_i'} C_{x_1}^{\mathsf{ch} \cdot \gamma_i}$ holds.

A complete description of its Σ-protocol is provided in the following:

1. With the witness $\mathsf{wi} \coloneqq (\gamma_i, \alpha_1, \alpha_2, \beta_1, \beta_2)$ as input, $\mathcal{P}$ picks $\gamma_i' \leftarrow\!\$\ [0, 2^{\lambda+\lambda_s} q]$, $\alpha_1', \alpha_2' \leftarrow\!\$\ \mathbb{Z}_q$ and $\beta_1', \beta_2' \leftarrow\!\$\ [0, 2^{\lambda+\lambda_s} \mathcal{D}_q]$. Then, $\mathcal{P}$ performs the EC group operations: $T_{01} \coloneqq \alpha_1' G_1$, $T_{02} \coloneqq \gamma_i' H_0 + \alpha_1' \hat{X}$, $T_{11} \coloneqq \alpha_2' G_1$, and $T_{12} \coloneqq \gamma_i' D + \alpha_2' \hat{X}$. And, it computes $T_{\gamma_i 1} \coloneqq g_q^{\beta_1'}$, $T_{\gamma_i 2} \coloneqq f^{\gamma_i'} pk_{\mathsf{cl}}^{\beta_1'}$, $T_{M_1} \coloneqq C_{x_1}^{\gamma_i'} g_q^{\beta_2'}$, and $T_{M_2} \coloneqq C_{x_2}^{\gamma_i'} pk_{\mathsf{cl}}^{\beta_2'}$ over the class group. Finally, $\mathcal{P}$ sends the commitment message $\mathsf{com} \coloneqq (T_{01}, T_{02}, T_{11}, T_{12}, T_{\gamma_i 1}, T_{\gamma_i 2}, T_{M_1}, T_{M_2})$ to $\mathcal{V}$.
2. $\mathcal{V}$ picks the challenge value $\mathsf{ch} \leftarrow\!\$\ [0, 2^\lambda]$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $z_i \coloneqq \gamma_i' + \mathsf{ch} \cdot \gamma_i \in \mathbb{Z}$, $a_1 \coloneqq \alpha_1' + \mathsf{ch} \cdot \alpha_1 \in \mathbb{Z}_q$, $a_2 \coloneqq \alpha_2' + \mathsf{ch} \cdot \alpha_2 \in \mathbb{Z}_q$, $b_1 \coloneqq \beta_1' + \mathsf{ch} \cdot \beta_1 \in \mathbb{Z}$, and $b_2 \coloneqq \beta_2' + \mathsf{ch} \cdot \beta_2 \in \mathbb{Z}$. Finally, $\mathcal{P}$ sends the response $\mathsf{resp} \coloneqq (z_i, a_1, a_2, b_1, b_2)$ to $\mathcal{V}$.
4. $\mathcal{V}$ outputs 1 if and only if $z_i \in [0, 2^{\lambda+\lambda_s} q]$, $b_1, b_2 \in [0, 2^{\lambda+\lambda_s} \mathcal{D}_q]$, $a_1 G_1 = T_{01} + \mathsf{ch} C_{01}$, $a_2 G_1 = T_{11} + \mathsf{ch} C_{11}$, $z_i H_0 + a_1 \hat{X} = T_{02} + \mathsf{ch} C_{02}$, $z_i D + a_2 \hat{X} = T_{12} + \mathsf{ch} C_{12}$, $g_q^{b_1} = T_{\gamma_i 1} C_{\gamma_i 1}^{\mathsf{ch}}$, $f^{z_i} pk_{\mathsf{cl}}^{b_1} = T_{\gamma_i 2} C_{\gamma_i 2}^{\mathsf{ch}}$, $C_{x_1}^{z_i} g_q^{b_2} = T_{M_1} C_{M_1}^{\mathsf{ch}}$, and $C_{x_2}^{z_i} pk_{\mathsf{cl}}^{b_2} = T_{M_2} C_{M_2}^{\mathsf{ch}}$.

Due to space limitations, we leave the security proof of this Σ-protocol to Appendix B.2.

NIZK proof for $\mathcal{R}_{\mathsf{con}}$. By using a random oracle $\mathsf{H}_{\mathsf{zk}} : \{0,1\}^* \to [0, 2^\lambda]$, the generation of the challenge $\mathsf{ch} \leftarrow \mathsf{H}_{\mathsf{zk}}(\mathsf{st}, \mathsf{com})$ no longer requires $\mathcal{V}$'s involvement. In the above verification, $\mathcal{V}$ requires $(\mathsf{com}, \mathsf{ch}, \mathsf{resp})$ as input, which means that the NIZK proof must include all three components. To achieve a smaller proof size, we modify the verification process in 4. With only $(\mathsf{ch}, \mathsf{resp})$ as input, $\mathcal{V}$ computes $T_{01} \coloneqq a_1 G_1 - \mathsf{ch} C_{01}$ and T_{11}, T_{02}, T_{12} in the same way. Also, it computes $T_{\gamma_i 1} \coloneqq g_q^{b_1} / C_{\gamma_i 1}^{\mathsf{ch}}$ [7]

[7] This inversion operation can be implemented by the class group library `BICYCL`.

and $T_{\gamma_i 2}, T_{M_1}, T_{M_2}$ in the same way. After combining them into com, $\mathcal{V}$ checks $\mathsf{H}_{\mathsf{zk}}(\mathsf{st}, \mathsf{com}) \stackrel{?}{=} \mathsf{ch}$. In this way, the NIZK proof is $(\mathsf{ch}, \mathsf{resp})$.

The $\mathcal{R}_{\mathsf{con}}$ proof has a size of 413.25 bytes, with proving and verification times of about 30 and 50 milliseconds (ms), respectively. The concrete parameters and experimental environment will be given in the next section.

6 Performance

We compare our scheme with three existing ones: DKLST23, WMC24, and TX25 in terms of per-party sending communication cost and local computation time. When considering network latency in either LAN or WAN settings, our advantage becomes even more pronounced, since our scheme requires the fewest communication rounds.

n^*	Tools	2	5	10	15	20	25	30
DKLST23 [25] non-robust	OT	75.9	302.4	679.9	1057.4	1434.9	1812.4	2189.9
TX25 [47]	CL	**1.56**	4.5	9.4	14.3	19.2	24.1	29
WMC24 [49]	CL	3.23	3.23	3.23	3.23	3.23	3.23	3.23
Ours	CL	2.02	**2.02**	**2.02**	**2.02**	**2.02**	**2.02**	**2.02**

Table 2: **The per-party upload communication costs (in KB) of (robust) threshold BBS+ signing.** n^* is the number of signing participants. Except for DKLST23, all other schemes are robust.

n^*	Tools	2	5	10	15	20	25	30
DKLST23 [25] non-robust	OT	4	9.5	20	29	38	46	55
TX25 [47]	CL	**201**	713	2248	4513	7521	11299	15827
WMC24 [49]	CL	478	976	1769	2573	3405	4232	5079
Ours	CL	260	**503**	**924**	**1346**	**1788**	**2206**	**2626**

Table 3: **The per-party local computation times (in ms) of (robust) threshold BBS+ signing.**

Our implementation utilized `BICYCL`[8] [7], an open-source C++ library for class-group arithmetic. We also used `mcl`[9] library to implement a pairing-based bilinear group. Our implementation can be found in the link `https://github.com/Jiangjiang-jiang/Multiparty-BBS-Plus-Implementation`.

The computational security level is set to be $\lambda = 128$ bits, and the statistical security parameter is $\lambda_s = 40$ bits. We use SHA256 to instantiate hash functions. BBS+ was taken over the pairing curve BLS12_381, of which the group order has 256 bits. For CL encryption, we use the class group $\mathsf{CL\text{-}HSM}_q$ with 256-bit plaintext space $\mathbb{Z}_q$ and 1827-bit Δ_K. Experiments were run on a MacBook Pro with macOS Sonoma 14.7, 8GB RAM, and an Apple M2 Pro chip.

Table 2 shows the comparison results of communication costs. Compared with MtA-based schemes (DKLST23 and TX25), our scheme maintains a constant upload communication overhead by using a threshold CL, similar to WMC24. However, our improvements in (e, s) generation and the use of threshold CL further reduce this constant, lowering the communication cost from 3.23

[8] https://gite.lirmm.fr/crypto/bicycl
[9] https://github.com/herumi/mcl

KB to 2.02 KB. Our scheme is robust and requires a broadcast channel, in which a sender transmits the same message to all recipients simultaneously, similar to other robust schemes [20]. In contrast, DKLST23 does not require a broadcast channel but lacks robustness. A broadcast channel is generally considered more expensive than a point-to-point channel. However, if robustness is not taken into account and our scheme is executed over point-to-point communication channels, it remains more communication-efficient than DKLST23.

As shown in Table 3, our scheme consistently achieves better computational efficiency than WMC24, considering the number of signers n^* increasing from 2 to 30. Compared with TX25, their scheme only shows an advantage when $n^* < 5$. This is because their per-party computational complexity grows quadratically, whereas ours grows linearly, as shown in Table 1. In particular, DKLST23 utilizes OT protocols to implement the MtA functionality, resulting in extremely low computational overhead. The trade-off, however, is higher communication costs and a lack of robustness.

References

1. Amit Agarwal, Alexander Bienstock, Ivan Damgård, and Daniel Escudero. Honest majority GOD MPC with o(depth(c)) rounds and low online communication. *IACR Cryptol. ePrint Arch.*, page 1479, 2024. `https://eprint.iacr.org/2024/1479`.
2. Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-taa. In *International conference on security and cryptography for networks*, pages 111–125. Springer, 2006.
3. Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. Glacius: Threshold schnorr signatures from DDH with full adaptive security. In *EUROCRYPT*, volume 15602, pages 304–334. Springer, 2025.
4. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73. ACM, 1993.
5. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55. Springer, 2004.
6. Dan Boneh, Iftach Haitner, Yehuda Lindell, and Gil Segev. Exponent-vrfs and their applications. In *EUROCRYPT*, volume 15607, pages 195–224. Springer, 2025.
7. Cyril Bouvier, Guilhem Castagnos, Laurent Imbert, and Fabien Laguillaumie. I want to ride my BICYCL : BICYCL implements cryptography in class groups. *J. Cryptol.*, 36(3):17, 2023.
8. Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure multiparty computation from threshold encryption based on class groups. In *CRYPTO*, volume 14081, pages 613–645. Springer, 2023.
9. Ernie Brickell and Jiangtao Li. Enhanced privacy id from bilinear pairing. *Cryptology ePrint Archive*, 2009.
10. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong diffie hellman assumption revisited. In *International Conference on Trust and Trustworthy Computing*, pages 1–20. Springer, 2016.
11. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.
12. Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *CCS*, pages 1769–1787. ACM, 2020.
13. Ignacio Cascudo and Bernardo David. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In *EUROCRYPT*, volume 14655, pages 216–248. Springer, 2024.
14. Ignacio Cascudo, Bernardo David, Omer Shlomovits, and Denis Varlakov. Mt. random: Multi-tiered randomness beacons. In *ACNS*, pages 645–674. Springer, 2023.
15. Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In *Public-Key Cryptography - PKC*, volume 12111 of *Lecture Notes in Computer Science*, pages 266–296. Springer, 2020.
16. Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In *CT-RSA*, volume 9048, pages 487–505. Springer, 2015.
17. Rutchathon Chairattana-Apirom and Stefano Tessaro. On the concrete security of BBS/BBS+ signatures. *ASIACRYPT*, 2025. `https://eprint.iacr.org/2025/1093`.
18. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.
19. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369. ACM, 1986.
20. Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4):1157–1186, 2017.

21. Hila Dahari-Garbian, Ariel Nof, and Luke Parker. Trout: Two-round threshold ECDSA from class groups. *IACR Cryptol. ePrint Arch.*, page 1666, 2025. https://eprint.iacr.org/2025/1666.
22. Poulami Das, Andreas Erwig, Sebastian Faust, Philipp-Florens Lehwalder, Julian Loss, Ziyan Qu, and Siavash Riahi. Bip32-compatible threshold wallets. In *AsiaCCS*, pages 856–872. ACM, 2025.
23. Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *S&P*, pages 980–997. IEEE Computer Society, 2018.
24. Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA in three rounds. In *S&P*. IEEE Computer Society, 2024. https://eprint.iacr.org/2023/765.
25. Jack Doerner, Yashvanth Kondi, Eysa Lee, Abhi Shelat, and LaKyah Tyner. Threshold BBS+ signatures for distributed anonymous credential issuance. In *S&P*, pages 773–789. IEEE, 2023.
26. Sebastian Faust, Carmit Hazay, David Kretzler, Leandro Rometsch, and Benjamin Schlosser. Non-interactive threshold BBS+ from pseudorandom correlations. In *CT-RSA*, volume 15598, pages 198–222. Springer, 2025.
27. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, volume 263, pages 186–194. Springer, 1986.
28. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*, pages 152–168. Springer, 2005.
29. Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discret. Appl. Math.*, 156(16):3113–3121, 2008.
30. Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *CCS*, pages 1179–1194. ACM, 2018.
31. Manu Sporny Greg Bernstein. BBS Data Integrity Cryptosuite . https://github.com/w3c/vc-di-bbs.
32. Bowen Jiang, Guofeng Tang, and Haiyang Xue. Three-round (robust) threshold ECDSA from threshold CL encryption. In *ACISP*, volume 15658, pages 224–244. Springer, 2025.
33. Shuichi Katsumata, Michael Reichle, and Kaoru Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. In *CRYPTO*, volume 14926, pages 459–491. Springer, 2024.
34. Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC*, volume 7785, pages 477–498. Springer, 2013.
35. Chelsea Komlo and Ian Goldberg. FROST: flexible round-optimized schnorr threshold signatures. In *SAC*, volume 12804, pages 34–65. Springer, 2020.
36. Yehuda Lindell. Fast secure two-party ECDSA signing. In *CRYPTO*, volume 10402, pages 613–644. Springer, 2017.
37. Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *CCS*, pages 1837–1854. ACM, 2018.
38. Tobias Looker, Vasilis Kalos, Andrew Whitehead, and Mike Lodder. The BBS Signature Scheme. https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-signatures/09/.
39. MATTR. mattrglobal/bbs-signatures: An implementation of bbs+ signatures for node and browser environments. https://github.com/mattrglobal/bbs-signatures.
40. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130. IEEE Computer Society, 1999.
41. Microsoft. microsoft/bbs-node-reference: Typescript/node reference implementation of bbs signature. https://github.com/microsoft/bbs-node-reference.
42. Jonas Nick, Tim Ruffing, and Yannick Seurin. Musig2: Simple two-round schnorr multi-signatures. In *CRYPTO*, volume 12825, pages 189–221. Springer, 2021.
43. Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces. In *CCS*, pages 1717–1731. ACM, 2020.
44. Rafaël Del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarinen. Threshold raccoon: Practical threshold signatures from standard lattice assumptions. In *EUROCRYPT*, volume 14652, pages 219–248. Springer, 2024.
45. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, volume 435, pages 239–252. Springer, 1989.
46. Guofeng Tang, Shuai Han, Li Lin, Changzheng Wei, and Ying Yan. Batch range proof: How to make threshold ECDSA more efficient. In *CCS*. ACM, 2024. https://eprint.iacr.org/2024/1677.
47. Guofeng Tang and Haiyang Xue. Robust threshold ECDSA with online-friendly design in three rounds. In *S&P*, pages 203–221. IEEE, 2025.
48. Stefano Tessaro and Chenzhi Zhu. Revisiting BBS signatures. In *EUROCRYPT*, volume 14008, pages 691–721. Springer, 2023.
49. Harry W. H. Wong, Jack P. K. Ma, and Sherman S. M. Chow. Secure multiparty computation of threshold signatures made more efficient. In *NDSS*. The Internet Society, 2024.
50. Harry W. H. Wong, Jack P. K. Ma, Hoover H. F. Yin, and Sherman S. M. Chow. Real threshold ECDSA. In *NDSS*. The Internet Society, 2023.

51. Xiaofei Wu, Tian Qiu, Guofeng Tang, Yuqing Niu, Bowen Jiang, Jun Zhou, Haiyang Xue, and Guomin Yang. Two-Party BBS+ Signature in Two Passes. In *ESORICS*, 2026. `https://eprint.iacr.org/2026/573`.
52. Haiyang Xue, Man Ho Au, Mengling Liu, Kwan Yin Chan, Handong Cui, Xiang Xie, Tsz Hon Yuen, and Chengru Zhang. Efficient multiplicative-to-additive function from joye-libert cryptosystem and its application to threshold ECDSA. In *CCS*, pages 2974–2988. ACM, 2023.
53. Haiyang Xue, Man Ho Au, Xiang Xie, Tsz Hon Yuen, and Handong Cui. Efficient online-friendly two-party ECDSA signature. In *CCS*, pages 558–573. ACM, 2021.

A Hardness Assumptions.

Definition 1 (Decisional Deffie-Hellman, DDH). *Let $\mathbb{G}$ be a cyclic group of prime order q with generator G. For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, with $a, b, c \leftarrow\$ \mathbb{Z}_q$,*

$$|\Pr[\mathcal{A}(aG, bG, abG) \to 1] - \Pr[\mathcal{A}(aG, bG, cG) \to 1]| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\cdot)$ is a negligible function and λ is the security parameter.

Definition 2 (Hard Subgroup Membership, HSM). *For the class group $\mathcal{G} = F \times \mathcal{G}^q$ with $\mathcal{F} = \langle f \rangle, \mathcal{G}^q = \langle g_q \rangle$ (Section 2.4) and any PPT adversary $\mathcal{A}$,*

$$\left| 2 \cdot \Pr\left[\begin{array}{c} b = b^* : x \leftarrow\$ \mathcal{D}_q, u \leftarrow\$ \mathbb{Z}_q, \\ b \leftarrow\$ \{0,1\}, Z_0 = f^u g_q^x, Z_1 = g_q^x, \\ b^* \leftarrow \mathcal{A}(\mathcal{G}, Z_b, \mathsf{Dlog}(\cdot)) \end{array} \right] - 1 \right| \leq \mathsf{negl}(\lambda).$$

Definition 3 (Rough Order Assumption [8], RO_C). *Let $C \in \mathbb{N}$ and $\mathcal{D}_C^{\mathsf{rough}}$ be the uniform distribution over the set*

$$\left\{ \rho \in \{0,1\}^\lambda | \mathcal{G} \leftarrow \mathsf{CLGen}(1^\lambda, q; \rho) \wedge \forall \mathsf{prime}\ p < C : p \nmid \mathsf{ord}(\mathcal{G}) \right\},$$

where CLGen is the algorithm for generating the class group. For any PPT adversary $\mathcal{A}$,

$$\begin{aligned} |\Pr[\mathcal{A}(1^\lambda, \rho_0) \to 1 | \rho_0 \leftarrow\$ \{0,1\}^\lambda] - \\ \Pr[\mathcal{A}(1^\lambda, \rho_1) \to 1 | \rho_1 \leftarrow\$ \mathcal{D}_C^{\mathsf{rough}}]| \leq \mathsf{negl}(\lambda). \end{aligned}$$

B Supplementary Material about ZKPs

B.1 Security Definition of Σ-Protocols

The Σ-protocol is called a zero-knowledge argument if it satisfies the following properties.

Definition 4 (Completeness). *If $(\mathsf{st}, \mathsf{wi}) \in \mathcal{R}$, then*

$$\Pr[\langle \mathcal{P}(\mathsf{st}, \mathsf{wi}), \mathcal{V}(\mathsf{st}) \rangle \to 1] \geq 1 - \mathsf{negl}(\lambda).$$

Definition 5 (Soundness). *If $\mathsf{st} \notin \mathcal{R}$, for any probabilistic polynomial time (PPT) prover $\mathcal{P}^*$, it satisfies*

$$\Pr[\langle \mathcal{P}^*(\mathsf{st}), \mathcal{V}(\mathsf{st}) \rangle \to 1] \leq \mathsf{negl}(\lambda).$$

Definition 6 (Honest-Verifier Zero-Knowledge, HVZK). *There exists a PPT simulator $\mathcal{S}$ that takes $(\mathsf{st}, \mathsf{ch})$ as input, and outputs an accepting transcript $(\mathsf{st}, \mathsf{com}, \mathsf{ch}, \mathsf{resp})$ which is indistinguishable from a real one generated by $\mathcal{P}$.*

The proof-of-knowledge property is a stronger property than soundness. It is not mandatory; for example, it is unnecessary in our application.

Definition 7 (Proof of Knowledge). *There exists a PPT extractor $\mathcal{E}$ that can extract a witness wi' for $(\mathsf{st}, \mathsf{wi}') \in \mathcal{R}$ from two accepting transcripts $(\mathsf{st}, \mathsf{com}, \mathsf{ch}, \mathsf{resp})$, $(\mathsf{st}, \mathsf{com}, \mathsf{ch}', \mathsf{resp}')$ with $\mathsf{ch} \neq \mathsf{ch}'$.*

B.2 Security Proof of $\mathcal{R}_{\mathsf{con}}$ Σ-Protocol

The completeness and soundness are both straightforward. The protocol may reject a valid statement only if the randomly chosen $\gamma_i' > 2^{\lambda+\lambda_s}q - 2^{\lambda}q$, $\beta_1' > 2^{\lambda+\lambda_s}\mathcal{D}_q - 2^{\lambda}\mathcal{D}_q$, or $\beta_2' > 2^{\lambda+\lambda_s}\mathcal{D}_q - 2^{\lambda}\mathcal{D}_q$. It happens with probability of $3 \cdot 2^{-\lambda_s}$. Since our challenge space has λ bits, the soundness error is $2^{-\lambda}$.

Fact 1. *Let $a, b \leftarrow\$ [0, R]$ and $\delta \leftarrow\$ [0, K]$ be independently chosen. Then the distributions of a and $b + \delta$ are K/R statistically close.*

Lemma 5. *Our Σ-protocol for $\mathcal{R}_{\mathsf{con}}$ satisfies the HVZK property.*

Proof. We construct a PPT simulator $\mathcal{S}$, with the input $(\mathsf{st}, \mathsf{ch})$. $\mathcal{S}$ picks $z_i \leftarrow\$ [0, 2^{\lambda+\lambda_s}q]$, $b_1, b_2 \leftarrow\$ [0, 2^{\lambda+\lambda_s}\mathcal{D}_q]$, and $a_1, a_2 \leftarrow\$ \mathbb{Z}_q$. Then, it computes

$$T_{01} = a_1 G_1 - \mathsf{ch} C_{01}, T_{02} = z_i H_0 + a_1 X - \mathsf{ch} C_{02},$$

$$T_{11} = a_2 G_1 - \mathsf{ch} C_{11}, T_{12} = z_i D + a_2 \hat{X} - \mathsf{ch} C_{12},$$

$$T_{\gamma_i 1} = g_q^{b_1} / C_{\gamma_i 1}^{\mathsf{ch}}, T_{\gamma_i 2} = f^{z_i} pk_{\mathsf{cl}}^{b_1} / C_{\gamma_i 2}^{\mathsf{ch}},$$

$$T_{M_1} = C_{x_1}^{z_i} g_q^{b_2} / C_{M_1}^{\mathsf{ch}}, T_{M_2} = C_{x_2}^{z_i} pk_{\mathsf{cl}}^{b_2} / C_{M_2}^{\mathsf{ch}}.$$

$\mathcal{S}$ sets $\mathsf{com} = (T_{01}, T_{02}, T_{11}, T_{12}, T_{\gamma_i 1}, T_{\gamma_i 2}, T_{M_1}, T_{M_2})$ and $\mathsf{resp} = (z_i, a_1, a_2, b_1, b_2)$.

The simulated z_i is chosen uniformly from $[0, 2^{\lambda+\lambda_s}q]$, while the real one is $z_i = \gamma_i' + \mathsf{ch} \cdot \gamma_i$ with $\gamma_i' \leftarrow\$ [0, 2^{\lambda+\lambda_s}q], \gamma_i \leftarrow\$ \mathbb{Z}_q$. From Fact 1, the distribution of the simulated z_i is $2^{-\lambda_s}$ close to the real distribution. A similar conclusion also holds for b_1, b_2. At the same time, the distribution of the simulated a_1, a_2 is identical to the real one. Thus, $\mathcal{S}$ can output the accepting transcripts $(\mathsf{com}, \mathsf{ch}, \mathsf{resp})$ which are statistically indistinguishable from a real one generated by $\mathcal{P}$. □

Although our application does not require the proof-of-knowledge property, we still provide it for potential future applications. There exists an extractor $\mathcal{E}$ that can only extract the plaintext component γ_i, but not the randomness used in the CL ciphertexts. This is typical of ZKPs over the class group [8,13].

Lemma 6. *Based on the rough order RO_C assumption there exists a PPT extractor $\mathcal{E}$ that can extract γ_i from two accepting transcripts with $\mathsf{ch} \neq \mathsf{ch}'$:*

$$\mathcal{E}(\mathsf{ct}, (\mathsf{com}, \mathsf{ch}, \mathsf{resp}), (\mathsf{com}, \mathsf{ch}', \mathsf{resp}')) \to \gamma_i.$$

Proof. Recall that $\mathsf{ct} = (\mathcal{C}_{\gamma_i}, \mathcal{C}_{\gamma_i x}, \mathcal{C}_{D_{0i}}, \mathcal{C}_{D_{1i}})$, $\mathsf{com} = (T_{01}, T_{02}, T_{11}, T_{12}, T_{\gamma_i 1}, T_{\gamma_i 2}, T_{M_1}, T_{M_2})$, $\mathsf{resp} = (z_i, a_1, a_2, b_1, b_2)$, and $\mathsf{resp}' = (z_i', a_1', a_2', b_1', b_2')$. Since they are accepting transcripts, the following equations hold. We have

$$\frac{a_1 - a_1'}{\mathsf{ch} - \mathsf{ch}'} G_1 = C_{01}, \quad \frac{a_2 - a_2'}{\mathsf{ch} - \mathsf{ch}'} G_1 = C_{11} \tag{3}$$

$$\frac{z_i - z_i'}{\mathsf{ch} - \mathsf{ch}'} H_0 + \frac{a_1 - a_1'}{\mathsf{ch} - \mathsf{ch}'} \hat{X} = C_{02} \tag{4}$$

$$\frac{z_i - z_i'}{\mathsf{ch} - \mathsf{ch}'} D + \frac{a_2 - a_2'}{\mathsf{ch} - \mathsf{ch}'} \hat{X} = C_{12}, \tag{5}$$

where the inversion of $(\mathsf{ch} - \mathsf{ch}')$ is taken modulo q. In the class group, we also have

$$g_q^{b_1 - b_1'} = C_{\gamma_i 1}^{\mathsf{ch} - \mathsf{ch}'}, \quad f^{z_i - z_i'} pk_{\mathsf{cl}}^{b_1 - b_1'} = C_{\gamma_i 2}^{\mathsf{ch} - \mathsf{ch}'} \tag{6}$$

$$C_{x_1}^{z_i - z_i'} g_q^{b_2 - b_2'} = C_{M_1}^{\mathsf{ch} - \mathsf{ch}'}, \quad C_{x_2}^{z_i - z_i'} pk_{\mathsf{cl}}^{b_2 - b_2'} = C_{M_2}^{\mathsf{ch} - \mathsf{ch}'} \tag{7}$$

Suppose $qs = \mathsf{ord}(\mathcal{G})$ is C-rough. Since $|\mathsf{ch} - \mathsf{ch}'| < C$, it must be invertible modulo qs. Then we can write:

$$g_q^{\frac{b_1 - b_1'}{\mathsf{ch} - \mathsf{ch}'}} = C_{\gamma_i 1}, \quad f^{\frac{z_i - z_i'}{\mathsf{ch} - \mathsf{ch}'}} pk_{\mathsf{cl}}^{\frac{b_1 - b_1'}{\mathsf{ch} - \mathsf{ch}'}} = C_{\gamma_i 2} \tag{8}$$

$$C_{x_1}^{\frac{z_i-z_i'}{\mathsf{ch}-\mathsf{ch}'}} g_q^{\frac{b_2-b_2'}{\mathsf{ch}-\mathsf{ch}'}} = C_{M_1}, \quad C_{x_2}^{\frac{z_i-z_i'}{\mathsf{ch}-\mathsf{ch}'}} pk_{\mathsf{cl}}^{\frac{b_2-b_2'}{\mathsf{ch}-\mathsf{ch}'}} = C_{M_2}. \tag{9}$$

Since $(C_{x_1}, C_{x_2}) = (g_q^\rho, f^x pk_{\mathsf{cl}}^\rho)$ is a valid CL ciphertext with unknown ρ, x, eq. (9) can be rewritten as

$$g_q^{\rho\cdot\frac{z_i-z_i'}{\mathsf{ch}-\mathsf{ch}'}+\frac{b_2-b_2'}{\mathsf{ch}-\mathsf{ch}'}} = C_{M_1}, \quad f^{x\cdot\frac{z_i-z_i'}{\mathsf{ch}-\mathsf{ch}'}} pk_{\mathsf{cl}}^{\rho\cdot\frac{z_i-z_i'}{\mathsf{ch}-\mathsf{ch}'}+\frac{b_2-b_2'}{\mathsf{ch}-\mathsf{ch}'}} = C_{M_2}. \tag{10}$$

Denote $\gamma_i = \frac{z_i-z_i'}{\mathsf{ch}-\mathsf{ch}'} \mod q$. Clearly, γ_i is the plaintext of $\mathcal{C}_{\gamma_i}$ from eq. (8) and the multiplicative factor of $\mathcal{C}_{\gamma_i x}$ from eq. (10). At the same time, $\gamma_i H_0, \gamma_i D$ is the plaintext of $\mathcal{C}_{D_{0i}}, \mathcal{C}_{D_{1i}}$, respectively from eqs. (3) to (5). □

B.3 Σ-Protocol for $\mathcal{R}_{\mathsf{cldl}}$

The corresponding Σ-protocol of the relation

$$\mathcal{R}_{\mathsf{cldl}} = \left\{ ((pk_{\mathsf{cl}}, \mathcal{C}_{x_i}, G_2, X_i); (x_i, \rho)) \middle| \begin{matrix} \mathcal{C}_{x_i} = (g_q^\rho, f^{x_i} pk_{\mathsf{cl}}^\rho) \\ X_i = x_i G_2 \end{matrix} \right\}$$

can refer to [15].

1. $\mathcal{P}$ picks $x_i' \leftarrow\$ \mathbb{Z}_q, \rho' \leftarrow\$ [0, 2^{\lambda+\lambda_s}\mathcal{D}_q]$. Then $\mathcal{P}$ computes $X_i' = x_i' G_2$ and $\mathcal{C}_{x_i'} = (g_q^{\rho'}, f^{x_i'} pk_{\mathsf{cl}}^{\rho'})$. It sends $\mathsf{com} = (X_i', \mathcal{C}_{x_i'})$ to $\mathcal{V}$.
2. $\mathcal{V}$ selects $\mathsf{ch} \leftarrow\$ [0, 2^\lambda]$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $z_i = x_i' + \mathsf{ch} \cdot x_i$ (over $\mathbb{Z}_q$), $\bar{\rho} = \rho' + \mathsf{ch} \cdot \rho$ (over $\mathbb{Z}$), then sends $\mathsf{resp} = (z_i, \bar{\rho})$ to $\mathcal{V}$.
4. $\mathcal{V}$ accepts the proof if all conditions hold:
   i) $\bar{\rho} \in [0, 2^{\lambda+\lambda_s}\mathcal{D}_q]$
   ii) $z_i G_2 = X_i' + \mathsf{ch} X_i$
   iii) $(g_q^{\bar{\rho}}, f^{z_i} pk_{\mathsf{cl}}^{\bar{\rho}}) = \mathcal{C}_{x_i'} \mathcal{C}_{x_i}^{\mathsf{ch}}$.

B.4 Σ-Protocol for $\mathcal{R}_{\mathsf{clddh}}$

We also need the DDH relation over the class group, defined as

$$\mathcal{R}_{\mathsf{clddh}} = \left\{ ((g_q, pk_i, c_1, M_i); k_i) : pk_i = g_q^{\Delta \cdot k_i}, M_i = c_1^{\Delta \cdot k_i} \right\}.$$

Since k_i is a share of secretly sharing $k \in \mathcal{D}_q$ over $\mathbb{Z}$, the bound of k_i is $B = \mathcal{D}_q 2^{\lambda_s+1} n^{t+n+2}$ [8,49]. Its Σ-protocol can refer to [49].

1. $\mathcal{P}$ picks $k_i' \leftarrow\$ [0, 2^{\lambda+\lambda_s} B]$ and computes $pk_i' = g_q^{\Delta \cdot k_i'}, M_i' = c_1^{\Delta \cdot k_i'}$. It sends $\mathsf{com} = (pk_i', M_i')$ to $\mathcal{V}$.
2. $\mathcal{V}$ selects $\mathsf{ch} \leftarrow\$ [0, 2^\lambda]$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $z_i = k_i' + \mathsf{ch} \cdot k_i$ (over $\mathbb{Z}$), then sends $\mathsf{resp} = z_i$ to $\mathcal{V}$.
4. $\mathcal{V}$ accepts the proof if all conditions hold:
   i) $z_i \in [0, 2^{\lambda+\lambda_s} B]$
   ii) $g_q^{\Delta \cdot z_i} = pk_i' \cdot pk_i^{\mathsf{ch}}$
   iii) $c_1^{\Delta \cdot z_i} = M_i' \cdot M_i^{\mathsf{ch}}$.

B.5 Σ-Protocol for $\mathcal{R}_{\mathsf{ddh}}$

The proof for the DDH relation over the EC group $\mathbb{G}_1$ is much more efficient than the above one. Although the design idea is similar, the required operation is significantly cheaper. The relation is defined as

$$\mathcal{R}_{\mathsf{ddh}} = \left\{ ((G_1, \hat{X}_i, Y, Z_i); \hat{x}_i) : \hat{X}_i = \hat{x}_i G_1, Z_i = \hat{x}_i Y \right\}.$$

The corresponding proof has been well studied for a long time [45].

1. $\mathcal{P}$ picks $x_i' \leftarrow\$ \mathbb{Z}_q$ and computes $X_i' = x_i' G_1, Z_i' = x_i' Y$. It sends $\mathsf{com} = (X_i', Z_i')$ to $\mathcal{V}$.
2. $\mathcal{V}$ selects $\mathsf{ch} \leftarrow\$ [0, 2^\lambda]$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $z_i = x_i' + \mathsf{ch} \cdot \hat{x}_i$ (over $\mathbb{Z}_q$).
4. $\mathcal{V}$ accepts the proof if all conditions hold:
   i) $z_i G_1 = X_i' + \mathsf{ch} \hat{X}_i$
   ii) $z_i Y = Z_i' + \mathsf{ch} Z_i$.