2PC-MPC: Emulating Two Party ECDSA in Large-Scale MPC

Offir Friedman[1], Avichai Marmor[1], Dolev Mutzari[1], Omer Sadika[1], Yehonatan C. Scaly[1], Yuval Spiizer[1], and Avishay Yanai

dWallet Labs, `research@dwalletlabs.com`

Abstract. Motivated by the need for a massively decentralized network concurrently servicing many clients, we present novel low-overhead UC-secure, publicly verifiable, threshold ECDSA protocols with identifiable abort. For the first time, we show how to reduce the message complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ and the computational complexity from $\mathcal{O}(n)$ to practically $\mathcal{O}(1)$ (per party, where n is the number of parties). We require only a broadcast channel for communication. Therefore, we natively support use-cases like permissionless bridges and decentralized custody, where P2P channels between every pair of parties are infeasible. Consequently, the message complexity is reduced and the protocol is publicly verifiable. We enable all communication to be public (over a broadcast channel), by using a threshold additively homomorphic encryption scheme and novel zero-knowledge proofs. To further reduce the computation and communication overheads, our protocols employ novel batching and amortization techniques, which may be of independent interest.
Our second main contribution is the introduction of the notion of a *2PC-MPC* protocol – a two-party ECDSA protocol where the second party is *fully emulated* by a network of n parties. This notion assures that both the first party (the client) and (a threshold) of the network are required to participate in signing, while abstracting away the internal structure of the network. In particular, the communication and computation complexities of the client remain independent of the network properties (e.g. size). This allows ultimate decentralization in distributed custody use-cases, as recent growing interest in the industry demands.
We report that our implementation completes the signing phase in 1.23 and 12.703 seconds, for 256 and 1024 parties, respectively.

1 Introduction

The Elliptic Curve Digital Signature Algorithm (ECDSA) is one of NIST's standards for digital signing. Much effort has been made for thresholdizing ECDSA in the past years [GGN16, BGG17, LN18, GG19, DKLS19, CGG+20, DOK+20, DJN+20, GKSS20, CCL+20, ANO+22, BMP22, DKLS23, ZYP23, WMYC23], and threshold ECDSA schemes have been widely deployed. This is mainly due to its widespread use as an authentication scheme in Bitcoin, Ethereum and alike, in which the signature constitutes a proof of ownership of the funds. Threshold signature based solutions flourish in the cryptocurrency industry, and focus on two main types of use cases: *distributed custody* (e.g., Fireblocks, Zengo, and Qredo) and *decentralized bridges* (e.g., Thorchain, Keep Network, and the Internet Computer). Distributed custody leverages threshold signature protocols to mitigate the self custodial risks of loss and theft, by distributing the secret signing key of the client, who is the legal owner of the funds, between multiple signers. Typically, the key is shared between the client and a (potentially distributed) custodian in a way that requires the active consent of both in order to produce a signature. A decentralized bridge consists of a network of nodes who distributively manage multiple signing keys for different wallets, each holding a pool of funds on different blockchains used to form cross-chain liquidity pools. This enables a single entity (the network) to permissionlessly own, manage and transfer funds (and information) across different blockchains. When a client owning wallets on two different blockchains wishes to transfer funds between them it can send funds to one end of the bridge's liquidity pool and request a withdrawal of an equivalent amount to the client's wallet on the the other end. The bridge network then creates and distributedly signs a corresponding transaction that can then be broadcasted by the client to the other blockchain, to complete the exchange.

Due to inherent limitations of existing threshold signing protocols, all real world networks we are aware of are deployed with a rather small number of nodes – in the order of 10-20. This is not on par with the vision of decentralization, since when the number of nodes is small it is difficult to argue that these nodes are not colluding.

In this paper we address the question of whether we can achieve an efficient massively decentralized network for concurrently servicing numerous clients, and propose the first threshold signature protocol that can truly scale to practically any number of nodes. Our protocol enjoys properties that are very important in real-world deployments. This includes *UC-security* – which is necessary when multiple instances of the protocol should run concurrently, and *public-verifiability* – which is necessary in public (permissionless) networks with financial rewards and punishments, so that that parties who misbehave need to be *identifiable*.

To date, all practical threshold ECDSA schemes were designed to work over unicast channels, assuming secure point-to-point (P2P) channels between every pair of parties. While secure point-to-point channels can be constructed in theory (under an appropriate PKI setup), in practice they inherently incur a high network load (i.e., maintaining a high number of sessions concurrently) and a message complexity of $\Omega(n^2)$, as each party computes and sends a message in private to every other party. When aiming for a large scale protocol, it is not reasonable to rely on such pairwise channels. Therefore there is a need for a new design, relying on a broadcast channel instead.

Generic MPC protocols in the broadcast channel model are ubiquitous, and oftentimes enjoy extra desirable properties like *identifiable abort* (IA) and *public verifiability* (PV). The broadcast channel itself may either be implemented in-house by the protocol's participants or rely on an external implementation. In recent years the use of a blockchain as a broadcast channel has surged, as it offers immutability of messages (which enables easy recovery from failures) as well as incentive mechanisms for faithful participation. However, among the various multiparty threshold ECDSA protocols, [GGN16] is the only one designed to operate in the broadcast communication model. That protocol, however, relies on a trusted setup, which obviously hinders its applicability to the aforementioned use cases, which were incepted in order to avoid the trusted party in the first place. Emulating the setup phase of [GGN16] in a distributed setting is possible in theory (using e.g., [ACS02]) but is considered infeasible since it requires a distributed generation of safe primes from which the parameters to a zero-knowledge proof system are derived.

Designing a protocol over a broadcast channel (such as [GGN16]) would naturally improve the *message complexity* to $\mathcal{O}(n)$ (instead of the $\Omega(n^2)$ message complexity incurred by protocols over a point-to-point network), yet, each party intuitively reads and processes a message from every other party, and so the *computational complexity* remains $\Omega(n)$.

In this work we leverage the fact that all messages are public and present two ways to further reduce the computational complexity of processing these messages to practically $\mathcal{O}(1)$. See Section 1.2 for more details.

Considering concrete efficiency (i.e., wall-clock time to produce a signature) the DKLS protocol [DKLS23] is the state of the art, demonstrating that 250 parties can produce a signature in a few seconds. That protocol, however, relies on a P2P network which, as mentioned above, requires a large bandwidth. In addition, [DKLS23] lacks identifiable abort, which hinders its usefulness for permissionless use-cases such as the aforementioned distributed bridge, as it would be quite easy for every single party to mount a denial of service attack. In that context, the state-of-the-art protocol that enjoys identifiable abort is [CGG$^+$20], which offers two variants: the first requires each party to verify $\mathcal{O}(n^2)$ zero-knowledge proofs, and the second requires $\mathcal{O}(n)$ verifications at the cost of three additional communication rounds. Since verification is a heavyweight operation, both variants are not scalable with the number of parties, and become impractical already at about 20 parties.

2PC-MPC: A Novel Approach to Decentralized Protocols. Our improvements in communication and computation enable distributed signing solutions like bridges, where the signing key owner is the network itself, to scale to an extremely large number of participants. In contrast, when the owner of the signing key is an external client, as in the distributed custody use case, the signing protocol must be conducted between the custodian and the client. This was the driving force for designing practical two-party signature protocols (e.g., [Lin17]). In such protocols the custody plays as a service provider who cannot sign without the client's active consent. Such custody solutions increase the trust in the service provider but still suffer from centralization. For instance, the service provider may completely deny service from a client or censor specific transactions. In addition, it leaves open the following attack vector: Consider a client who is an organization with multiple authorised signers. Deploying a two-party signing protocol in that setting means duplicating the client's share of the signing key among all authorised signers, while the service provider possesses the other share.

Thus, it is sufficient for the service provider to collude with only a single member of the organization in order to steal all the client's funds.

These issues motivate the need for a decentralized service provider, meaning that the service provider's share of the signing key is now shared among a network of n participants that emulate the operation of the provider. This, however, requires a distributed signing protocol for an access structure that is different than the usual threshold access structure that is addressed in (almost all) previous works. The new access structure requires half of the signing key to be in posession of the client, whereas the network, collectively, possesses the other half. To produce a signature the client along with at least $t + 1$ honest parties must collaborate, keeping the client safe even in the extreme case that all network participants are compromised. Similar to the usual access structure discussed above (for decentralized bridges), a protocol for the new access structure must be proven secure under *universal composition* (UC) in order to concurrently serve many clients. The only work that addresses a similar access structure in the UC framework is [BMP22]; in that protocol, however, each party must separately verify zero-knowledge proofs from every other party and so it is not practical for networks as large as we envision.

Capturing that special access structure can take different approaches. For instance, one may model the client as n virtual parties (as done in [ZYP23] against a semi-honest adversary), so its power is equal to that of the service provider. This approach incurs a high computation and communication complexity on the client, that is dependent on the network size. An alternative would be to use an hierarchical threshold secret sharing scheme [Tas07] with two levels of hierarchy – the client is at the top and the network participants are at the bottom, such that reconstruction is possible only by a collaboration between the client and at least $t + 1$ network participants.

In this work we take a novel approach, named *2PC-MPC*, that is reminiscent to the hierarchical structure. Specifically, the client and each network participant possess only one share, however, in contrast to the hierarchical secret sharing scheme the client in our protocol is *agnostic* to the fact that it interacts with a distributed party. Rather, it follows the protocol as if it is interacting with a single party only (hence *2PC*), which is emulated by the network via an *MPC* protocol. We stress that being agnostic to the network structure does not imply that the structure is hidden from the user, and when deployed in a permissionless setting it is public. The benefit of our 2PC-MPC approach is that the client can always refer to the network as an abstract second-party. Thus, allowing the network to evolve, by adding new parties, removing old parties, or even changing its internal access structure, without involving the client at all.

1.1 Our Contribution

We propose the first *large-scale* UC-secure threshold ECDSA protocols and introduce the notion of *2PC-MPC*. Our protocol only uses a broadcast channel for communication, thus achieving public verifiability. It supports our novel 2PC-MPC approach, but can also be folded back to standard threshold access structure. We have a tradeoff between the round complexity and computational complexity of the participants: In 3 rounds we achieve linear computational complexity for each player, whereas if we go to 7 rounds (for the network participants only, not the client), the complexity per player goes down to constant (in practice). We guarantee UC-security with identifiable abort.

We implemented our protocol and report (in Section 7) on both the key-generation, presignature and signature times as a whole for sets of up to **1,024 parties**; for the signature part, we compare against what is reported in [DKLS19] and find that for $n = 256$ our protocol is $7 - 8\times$ faster, whereas their protocol becomes impractical for a larger number of parties. In addition, we provide a microbenchmark for proof and verification time of Schnorr proof batching and aggregation with up to 1,000 parties and batches of size 1,000. This is important in order to understand the impact of using an AHE scheme that requires range proofs (as in Paillier's scheme). Our protocols are natively designed to operate over a broadcast channel, which makes them inherently applicable to blockchain networks where direct point-to-point channels are not available. Our protocols are accompanied with a security proof via a UC simulation and hence are fit to real-world deployment, where a many instances of the protocol are run at the same time. In comparison to previous works, our protocols differ in two dimensions:

– *Access structure.* We address two access structures in one go. The first access structure is a two-level hierarchy, capturing the setting of $1 + n$ parties (may also be pronounced as 'one plus t out of n'). The first party is alone at the top level and the other n are at the second, such that a qualified set consists of the party at the first level together with any $t + 1$ parties from the second level. The second one is the usual t out of n ('threshold') access structure. Our protocol for the latter is a byproduct of the former. That is, by taking the protocol for the first structure and ignoring the first party we get the usual threshold signature protocol; consequently, and due to space limit, the exposition in the paper is focused to the former access structure.
– *Rounds vs. computational complexity.* Our protocols can be optimized to either round or computational complexity. When optimized to the number of rounds, we get a three (3) round protocol where presignature takes two rounds and the signature takes one, matching the state of the art protocol by Doerner et al. [DKLS23] (which is a concurrent and independent work). Our protocol is favorable to [DKLS23] in that it builds on a broadcast channel rather than P2P channels, which is leveraged to achieve identifiable abort (which is not possible in [DKLS23] as is). The downside of that variant of our protocol is that the number of expensive operations (exponentiations) to be done by each party is linear in the number of parties. One of our contribution is showing that the number of exponentiations can be reduced to practically constant, allowing *scaling to even thousands of participants*, but incurring seven (7) communication rounds (six of them for presignature).

Addressing the aforementioned hierarchical access structure, we introduce the concept of a *2PC-MPC threshold ECDSA*. In such a protocol the client is interacting with a network while being unaware of its topology, or even of the precise size of the network. From the client's point of view, it is interacting with a single party and hence its computation and communication overhead remains low. The network almost perfectly emulates the second party in a 2-out-of-2 ECDSA protocol (the client is the first party), where messages from the network's participants are aggregated into a single message which is effectively the only one received and processed by the client. This notion is important in case the network operates as a blockchain, where we cannot expect a client to directly interact with the blockchain's nodes. Instead, it broadcasts its messages to be processed by the blockchain and being notified whenever the blockchain has a response. This communication model reduces the attack surface of the system in a real deployment, since the user does not know the identities of the participants' endpoints over the network. We believe that our ideas may also find use in devising a *topology hiding protocol*, in which the network's properties (e.g., size and topology) are *hidden* from the user. Such a notion was recently addressed in theory for general MPC ([MOR14] and follow ups), but has never been tackled in the context of threshold ECDSA, and we leave it to future work. Following the discussion above, in Table 1 we provide a comprehensive comparison of threshold ECDSA protocols.

1.2 Our Techniques

We introduce a set of new techniques to achieve the aforementioned results.

Utilizing the Broadcast Model without Trusted Setup. As in prior works, we use additively homomorphic encryption with a single global public key, where the secret key is shared between all parties, together with a threshold decryption protocol. Whereas the above suffices in the semi-honest setting, it is insufficient when considering malicious security models. There, we need to make sure that we have a suitable zero-knowledge proof system in order to provide our UC identifiable abort guarantees. Whereas prior works either assumed a trusted setup [GGN16] or P2P channels [CMP20], we cannot rely on such solutions.

The main challenge is the reliance of prior works on range proofs that require trusted setup. Our approach is to rely on transparent-setup range proof systems, such as BulletProofs [BBB$^+$18]. However, solutions such as BulletProofs provide a range-guarantee only modulo the group order, which is insufficient. We divide the witness into "digits" and commit to each such digit individually. This by itself is does not solve the problem and we need to combine, in a nontrivial manner, a range proof (modulo the group order) together with a gap sigma protocol similar to those used in [GGN16].

Supporting 2PC-MPC Access. Recall that in such a protocol the client is interacting with a network while being unaware of its topology, or even of the precise size of the network.

To do this, we start with a 2PC protocol (see Appendix I), and distribute the functionality of one of the parties. To this end, we require *aggregatable* zero-knowledge proofs. That is, a set of parties, each holding a share of the witness, collaboratively generate the proof, in a small (constant) number of rounds. The protocol is aggregatable if, at every round, the messages of the individual parties

	Channels	IA	Threshold	Rounds	Messages	Computation	UC	Assumptions
[GGN16]	BC	No	$t < n$	6	$O(n)$	$O(n)$	No	sRSA, DCR
[BGG17]	BC	No	$t < n$	4	$O(n)$	$O(n)$	No	sRSA
[LN18]	P2P	No	$t < n$	7	$O(n^2)$	$O(n^2)$	No	DCR or OT, DDH
[GG19]	P2P & BC	No	$t < n$	8	$O(n^2)$	$O(n^2)$	No	sRSA, DCR, DDH
[DKLS19]	P2P & BC	No	$t < n$	$\lceil \log t \rceil + 6$	$O(n^2)$	$O(n^2)$	Yes	DH
[CGG+20]	P2P	Yes	$t < n$	4	$O(n^3)$	$O(n^3)$	Yes	sRSA, DCR, DDH
[CGG+20]	P2P	Yes	$t < n$	7	$O(n^2)$	$O(n^2)$	Yes	sRSA, DCR, DDH
[DJN+20]	P2P	No	$t < n/2$	4	$O(n^2)$	$O(n^2)$	Yes	-
[GKSS20]	P2P	Yes	$t < n$	13	$O(n^2)$	$O(n^2)$	Yes	DCR or OT, DDH
[CCL+20]	P2P	No	$t < n$	8	$O(n^2)$	$O(n^2)$	No	SR, LO, HSM, DDH
[ANO+22]	P2P & BC	No	$t = n - 1$	2	$O(n^2)$	$O(n^2)$	No	Ring-LPN
[DKLS23]	P2P	No	$t < n$	3	$O(n^2)$	$O(n^2)$	Yes	DH
[WMYC23]	P2P	Yes	$t < n$	5	$O(n^2)$	$O(n)$	No	SR, LO, HSM, DDH
Ours (Sec. 3)	BC	Yes	$t < n$	7	$O(n)$	$O(1)^*$	Yes	DCR**
Ours (Appx. M)	BC	Yes	$t < n$	3	$O(n)$	$O(n)$	Yes	DCR**

Table 1: Comparison of threshold ECDSA protocols. BC refers to a broadcast channel, P2P refers to pairwise point-to-point channels, IA refers to identifiable abort, and UC refers to universally composability. Cells with a green color refer to a favorable property. In the Threshold column, $t < n$ means that the protocol may deal with a dishonest majority, $t < n/2$ means that honest majority is assumed, and $t = n - 1$ means that the protocol requires all parties to participate. 'Rounds' refers to total number (presign + sign) of communication rounds (when applied, ignoring rounds used to implement the broadcast channel). 'Messages' refers to the message complexity of the protocol *in total for all parties* whereas 'Computation' refers to the computation complexity *per party*. The assumptions listed in the table are strong RSA (sRSA), decisional composite residuosity (DCR), {decisional} Diffie-Helman ({D}DH), oblivious transfer (OT) which may be constructed under various assumptions, hard subgroup membership (HSM) and ring-learning parity with noise (LPN), Strong Root (SR), Low Order (LO), Non-Colluding Semi-Honest Server (NC-SHS). * $\mathcal{O}(1)$ refers to the number of heavy operations like exponentiations and proof verification whereas the number of simple arithmetic operations like addition in a field/ring are linear in the number of parties. ** Our protocol relies on the semantic security of the additively homomorphic scheme, which in turn relies on DCR in case this is implemented via the Paillier encryption scheme. In the latter, we count only expensive operations like encryption/decryption and NIZK proof generation and verification.

are simply summed together, and this sum is the only information required for the next step of the protocol. Protocols of this form are particularly efficient since it requires minimal processing of the other parties' messages. We show how to achieve this for our 2PC protocol, which enjoys a convenient structure for this purpose.

Minimizing Computational Complexity. We leverage the fact that all messages are public and present two ways to reduce the computational complexity of processing these messages to practically $\mathcal{O}(1)$.

1. **Amortization.** Protocols over a broadcast channel often use an additively homomorphic encryption scheme, and invoke a threshold decryption sub-protocol at some stages (see, e.g., [GGN16]). Typically in that sub-protocol the parties broadcast their decryption shares for some ciphertext, which are

then combined locally by each party to get the final plaintext – incurring $\Omega(n)$ exponentiations per party.
Instead, we show that this can be done by a single designated party only, who can inform the others about the resulting plaintext in a verifiable manner. When many executions of threshold decryption are run in parallel, picking a different designated party for each ciphertext reduces the amortized complexity to $\mathcal{O}(1)$ exponentiations per party per ciphertext.

2. **Aggregation.** The security and correctness of the protocol are ensured by the fact that the messages broadcast by the parties are accompanied by a zero-knowledge proof. Then, the parties separately verify a proof broadcasted by any other party before proceeding to the next protocol step. Identifying the verification of these proofs as one of the computationally heaviest operations in many threshold ECDSA protocols, we again use the aggregateable nature of our zero-knowledge proof system as follows. We first aggregate the n proofs into a single aggregated proof, and then verify only one aggregated proof. While verification of the aggregated proof remains a heavyweight operation, the aggregation operation is orders of magnitude lighter. Thus, the overall computational complexity consists of a simple aggregation of n proofs (n multiplications in a group) followed by *only one proof verification* ($\mathcal{O}(1)$ exponentiations) and so, even considering thousands of parties, the computational complexity is reduced to practically $\mathcal{O}(1)$.

2 Preliminaries

We use $\kappa, s \in \mathbb{N}$ as a computational and statistical security parameters, resp. For $x, y \in \{0,1\}^*$ the expression $x||y$ is the concatenation of x and y. Uniformly sampling a random value x from a set X is denoted by $x \leftarrow X$. The result of a probabilistic algorithm A on inputs $x_1, x_2, \ldots$ is written by $x \leftarrow A(x_1, x_2, \ldots)$; in addition, when we want to explicitly mention the randomness used by the algorithm we write $x = A(x_1, x_2, \ldots\ ;\ r)$. $(\mathbb{G}, G, q)$ denote an elliptic curve group written in additive notation, a generator and the prime order of the group, respectively. We usually denote group elements by capital letters. For an element $R \in \mathbb{G}$, we write $R|_{x-axis}$ to denote its x-coordinate. We denote its bit-length by $||x||_2$, and let $\lceil x \rceil = 2^{||x||_2}$ be the closest power of two larger than x.

The ECDSA Signature. Given an elliptic curve group $(\mathbb{G}, G, q)$, a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ (modeled as a random oracle), secret key $x \in \mathbb{Z}_q$ and public key $X = x \cdot G$, respectively, a signature σ on a message $msg \in \{0,1\}^*$ is computed as follows: Let $m = H(msg)$, sample $k \leftarrow \mathbb{Z}_q$, compute $R = k^{-1} \cdot G$ and set $r = R|_{x-axis}$. Then, compute $s' = k(m + rx) \mod q$ and $s = \min(s', q - s')$ (to ensure uniqueness of the signature), and output the signature $\sigma = (r, s)$.

Verification is done as follows: Given a message msg and the signature $\sigma = (r, s)$, output `accept` if $(H(msg)s^{-1} \cdot G + rs^{-1} \cdot X)|_{x-axis} \mod q = r$ and $s < q - s$. Otherwise output `reject`. Indeed, if $\sigma = (r, s)$ is computed correctly on msg with $m = H(msg)$, then $ms^{-1} \cdot G + rs^{-1} \cdot X = ms^{-1} \cdot G + rxs^{-1} \cdot G =$

$(m+rx)s^{-1}\cdot G = (m+rx)k^{-1}(m+rx)^{-1}\cdot G = k^{-1}\cdot G = R$ and so, projection to the x coordinate results with $R|_{x-axis} = r$ as required.[1]

Additively Homomorphic Encryption. Our protocol requires an AHE (Additively Homomorphic Encryption). Unlike the approach of [BMP22, Lin17, GGN16], instead of explicitly instantiating our protocol with the Paillier crypto-system, we choose to work with a generic AHE scheme. This is done both to simplify proofs and make constructions more modular, as well as to support future implementations of the protocol using leveled or fully homomorphic encryption to save communication rounds. Following this approach enables us, for example, to abstract away the need of masking the plaintext, which resides in $\mathbb{Z}_q$, with a sufficiently large factor of q, to prevent information leakage upon decryption. Indeed, in Paillier, since the plaintext space is $\mathbb{Z}_N$ where inevitably $N \gg q$ ($||N|| \geq 2048$, whereas typically, $||q|| = 256$), the number of modular reductions that take place must be kept hidden. This would not be necessary if we chose an AHE with a plaintext space $\mathbb{Z}_q$ (or $\mathbb{Z}_q^n$) which can be achieved with class-groups based cryptography [BDO23]. Using a class-group based encryption is advantageous as it adjust the plaintext space to be exactly that needed by the ECDSA scheme, which enables *avoiding range proofs* in the protocol. When the AHE scheme's plaintext space does not match that needed by ECDSA, we need it to have *secure function evaluation* (see discussion below). As demonstrated in Fig. 2, the aforementioned range-proofs take a substantial portion of the pre- signature, which tips the scales to the use of Class-Group-based schemes. However, they are about $10\times$ slower than Paillier, and their security is less established (relying on stronger assumptions like Low-Order and Strong-Root). Should range proof become more efficient it would tip the scales back toward using Paillier's scheme.

The comprehensive formal definition of an AHE scheme is given in Appendix B. Below we only give our formal definition of secure function evaluation for AHE schemes, as it is not yet standardized in the literature. Informally, an *additively homomorphic encryption scheme* $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$ is parameterized with plaintext, randomness and ciphertext spaces $\mathcal{P}_{pk}$, $\mathcal{R}_{pk}$ and $\mathcal{C}_{pk}$, respectively, where $\mathcal{P}_{pk}$ is a $\mathbb{Z}$-module. Algorithm $\mathsf{Gen}(1^\kappa, \mathsf{aux}) \rightarrow (pk; sk)$ is used to generate a public-private key-pair; given a public key pk algorithm $\mathsf{Enc}(pk, \mathsf{pt}; \eta_{\mathsf{enc}}) \rightarrow \mathsf{ct}$ translates a plaintext pt and randomness η_{enc} to a ciphertext ct; given a private key sk algorithm $\mathsf{Dec}(sk, \mathsf{ct}) \rightarrow \mathsf{pt}$ translates a ciphertext to a plaintext; and the fact the the scheme is additively homomorphic means that there is an algorithm $\mathsf{Add}(pk, \mathsf{ct}_1, \mathsf{ct}_2) \rightarrow \mathsf{ct}_3$ that translates ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$ to a ciphertext ct_3 such that decryption of ct_3 is equal to the sum of the decryptions of ct_1 and ct_2.

We may write $\mathcal{P}$ and $\mathcal{C}$ instead of $\mathcal{P}_{pk}$ and $\mathcal{C}_{pk}$. In addition, we denote homomorphic addition of two ciphertexts ct_1 and ct_2 by $\mathsf{ct}_1 \oplus \mathsf{ct}_2$ and a multiplication of a ciphertext ct by a scalar α by $\alpha \odot \mathsf{ct}$. Every affine function can be efficiently computed homomorphically by an algorithm $\mathsf{Eval}(pk, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell; \eta_{\mathsf{eval}})$. Whenever

[1] Note that we use the non-ambiguous variant of ECDSA, which is the standard in blockchain applications. In addition, looking ahead, we exploit the property that for each (X, r, m) there exist only one valid signature.

we omit the randomness in $\mathsf{Eval}(pk, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$, we refer to the process of sampling a randomness $\eta_{\mathsf{eval}} \leftarrow \{0,1\}^{\mathsf{poly}(\kappa)}$ and running $\mathsf{Eval}(pk, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell; \eta_{\mathsf{eval}})$. Given a public key pk, an affine function $f(x_1, \ldots, x_\ell) = a_0 + \sum_{i=1}^{\ell} a_i x_i$ (with $a_i \in \mathbb{Z}$ and $\ell, \|a_i\|_2 \in \mathsf{poly}(\kappa)$ for all $0 \leq i \leq \ell$) and ℓ ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell \in \mathcal{C}$, Eval outputs a ciphertext ct.

The definitions for correctness and semantic security of an AHE schemes are standard in the literature and appear in Definitions B.2 and B.3. *Secure function evaluation* (Definition 2.1), states that evaluating some function f on ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell \in \mathcal{C}$ and giving the resulting ciphertext to an adversary who has the decryption key does not reveal anything about f, except what can be learned from the (decrypted) result alone. Note that *secure function evaluation* is stronger than *function privacy* as defined in [BPMW16], as here the adversary is in possession of the private decryption key sk and the randomness η_i used to encrypt each plaintext pt_i. Indeed, in our protocols, party B sends A an encryption of its ECDSA private key x, followed by party A homomorphically evaluating a private affine function over $\mathsf{ct}_x = \mathsf{Enc}(x)$. For instance, for a private affine function $f(x) = ax + b$ it sends $\mathsf{ct} = \mathsf{Eval}(pk, f, \mathsf{ct}_x)$. We must therefore ensure that A, learns nothing about a and b beyond $ax + b$ given x, pk, sk, η and ct, where $\mathsf{ct}_x = \mathsf{Enc}(pk, x; \eta)$. An alternative presentation of the definition can be formulated via an experiment in a standard way, see Definition B.5 (Appendix B) for a formal treatment.

Definition 2.1 *AHE has* secure function evaluation *if there exists a PPT algorithm $\mathcal{S}_{\mathsf{Eval}}$ such that for every PPT adversary $\mathcal{A}$, every ℓ-ary affine function f as above, every (pk, sk) and every plaintexts $\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell \in \mathcal{P}$ and randomness $\eta_1, \ldots, \eta_\ell$,*

$$\begin{aligned} &|\Pr[\mathcal{A}(1^\kappa, pk, sk, \{\mathsf{pt}_i, \eta_i\}_{i=1}^{\ell}, \mathsf{Eval}(pk, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))) = 1] \\ &- \Pr[\mathcal{A}(1^\kappa, pk, sk, \{\mathsf{pt}_i, \eta_i\}_{i=1}^{\ell}, \mathcal{S}_{\mathsf{Eval}}(pk, f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell))) = 1]| \leq \mathsf{neg}(\kappa) \end{aligned} \tag{1}$$

where $\mathsf{ct}_i \leftarrow \mathsf{Enc}(pk, \mathsf{pt}_i; \eta_i)$, and the probability is over the coins used by Eval and the random coins of $\mathcal{A}$ and $\mathcal{S}$.

Remark on the use of Paillier's AHE. When using the Paillier's AHE the 'actual' plaintext space ($\mathbb{Z}_N$) does not match the 'effective' plaintext space (in our case, this is $\mathbb{Z}_q$, in order to match the order of the ECDSA group), it is required to keep track on the upper bound on the effective plaintexts within the larger domain. To this end, the Enc function, in addition to the ciphertext, outputs the upper bound $\mathsf{PT} = q$ as well. Then, any ciphertext resulting from applying an ℓ-ary function on ℓ+1 ciphertexts $(\mathsf{ct}_0, \mathsf{PT}_0), \ldots, (\mathsf{ct}_\ell, \mathsf{PT}_\ell)$ is accompanied with the upper bound $\mathsf{PT} = \sum_{i=0}^{\ell} \mathsf{PT}_i$. This will come into play later, when proving that the Paillier's AHE has secure function evaluation, as well as when proving the security of the zero-knowledge proofs.

Threshold AHE. Let $B = \{B_1, \ldots, B_n\}$ be a set of parties and $U \subset B$, where $|U| \leq t < n$, be those parties under the control of the adversary. A threshold AHE $\mathcal{F}_{\mathsf{TAHE}}$ is defined in Functionality 1. The functionality is parameterized with an AHE scheme $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$ with semantic security and secure function evaluation. The parties in B can generate a key and prove its validity to the certifiers. In the context of our 2PC-MPC protocol, the certifiers are the users (represented by party A) who engage with B. Finally, any set of $t + 1$ honest parties can decrypt a ciphertext using a previously generated key, unless the adversary decides to abort.

FUNCTIONALITY 1 (*Threshold AHE:* $\mathcal{F}_{\mathsf{TAHE}}$)

The functionality interacts with $B = (B_1, \ldots, B_n)$, an adversary, and a set of certifiers. It is parameterized with $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$, κ and $t < n$.

- Upon receiving $(\mathbf{keygen}, \mathsf{sid})$ from all parties in B, for a fresh sid, run $(pk; sk) \leftarrow \mathsf{AHE.Gen}(1^\kappa)$, send pk to the adversary and receive the adversary's response $(\mathtt{continue}, U')$. If $\mathtt{continue} = 1$ or $U' \cap U = \emptyset$ then send $(\mathbf{pubkey}, \mathsf{sid}, pk)$ to all parties in B and store $(\mathsf{sid}, pk; sk)$, otherwise send $(\mathbf{pubkey}, \mathsf{sid}, \perp, U' \cap U)$.
- Upon receiving $(\mathbf{certify}, pk)$ from some party, if $(\cdot, pk; \cdot)$ is stored then return 1 to that party, otherwise return 0.
- Upon receiving $(\mathbf{decrypt}, pk, \mathsf{ct})$ from at least $t+1$ parties, if $(\mathsf{sid}, pk; sk)$ is stored for some sid and sk, compute $\mathsf{pt} = \mathsf{AHE.Dec}(sk, \mathsf{ct})$, send $(\mathbf{decrypted}, pk, \mathsf{ct}, \mathsf{pt})$ to the adversary and wait to the adversary's response $(\mathtt{continue}, U')$. If $\mathtt{continue} = 1$ or $U' \cap U = \emptyset$ then send $(\mathbf{decrypted}, pk, \mathsf{ct}, \mathsf{pt})$ to everyone, otherwise, send $(\mathbf{decrypted}, pk, \mathsf{ct}, \perp, U' \cap U)$ to everyone.

Homomorphic Commitments. Homomorphic commitments and their batched version are standard in the literature; for completeness, we provide a formal definition in Appendix C.1. For the notation, a commitment scheme consists of algorithms $(\mathsf{Setup}, \mathsf{Com})$. Algorithm Setup is given $(\mathbb{G}, G, q, T)$ for a batch size T. and outputs public parameters pp including the message and randomness spaces $\mathcal{M}_{\mathsf{pp}} = \mathbb{Z}_q^T$, $\mathcal{R}_{\mathsf{pp}} = \mathbb{Z}_q$. The commitment algorithm $\mathsf{Com}_{\mathsf{pp}}$ defines a function $\mathcal{M}_{\mathsf{pp}} \times \mathcal{R}_{\mathsf{pp}} \rightarrow \mathcal{C}_{\mathsf{pp}}$ that satisfies perfect hiding and computational binding. In addition, for homomorphic commitments Com is a group homomorphism: $\mathsf{Com}(m_1; \rho_1) \oplus \mathsf{Com}(m_2; \rho_2) = \mathsf{Com}(m_1 + m_2; \rho_1 + \rho_2)$.

Zero-Knowledge Proofs

We use the standard *zero-knowledge* and *committed zero-knowledge* functionalities defined by $((x, w), \cdot) \rightarrow (\cdot, (x, R(x, w))$ where the prover $\mathcal{P}$ has a statement x and a witness w, the verifier $\mathcal{V}$ obtains only the statement and its validity $R(x, w)$ (or alternatively $(x, w) \in L_R$ where L_R is the language associated with relation R. See Functionalities 9 ($\mathcal{F}_{\mathsf{zk}}^R$) for zero-knowledge and 11 ($\mathcal{F}_{\mathsf{com-zk}}^R$) for committed zero-knowledge (in Appendix C.2). These functionalities were formerly given in [Lin17]. In addition, we use a trivial extension to *batched* and *aggregateable* versions of zero-knowledge. Specifically, batched zero-knowledge is defined with $\mathcal{P}$ and $\mathcal{V}$ as above by $((\vec{x}, \vec{w}), \cdot) \rightarrow (\cdot, (\vec{x}, \forall_i R(x_i, w_i))$. When the relation R is associated with a homomorphism $(x; w) \in R \iff \phi(w) = x$, one can define an aggregateable zero-knowledge proof, as in Functionality 12. This is defined with provers $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and verifier $\mathcal{V}$ by $((x_1, w_1), \ldots, (x_n, w_n), \cdot) \rightarrow$

$(\cdot, (x, \forall_i R(x_i, w_i))$ where $x = \sum_i x_i$. In our protocols we choose to work with the ideal functionalities for (stand-alone, batched or aggregateable) zero-knowledge proofs as it makes it easier to work with. Our instantiation of the zero-knowledge proof is proven as a secure sigma protocol, given any range proof in the hybrid model, which in turn is shown to be sufficient to securely compute the aforementioned functionalities [HL10, Theorem 6.5.2].

Languages used in our protocols.

The notation we use to describe the relations follows: the language associated with relation R is the set $L_R = \{x \mid \exists w : (x, w) \in R\}$; we write $L[\texttt{params}] = \{S; s \mid f(\texttt{params}, S, s)\}$ to denote a language $\{x \mid \exists s : f(\texttt{params}, S, s) = 1\}$ where `params` typically refers to global parameters (e.g., a public-key of an encryption scheme, or a global verification key), S refers to a set of public values and s refers to a set of secret values. Note that in $\mathcal{F}^R_{\mathsf{zk}}$ (Functionality 9) we implicitly require 'knowledge' of the witness s from the prover P. Note that we may omit `params` from a language specification when it is clear from context. The languages in our protocols are listed below:

– *Knowledge of the discrete log of an elliptic-curve point.*

$$L_{\mathsf{DL}}[(\mathbb{G}, G, q)] = \{(P; x) \mid P = x \cdot G\} \tag{2}$$

– *Knowledge of decommitment.*

$$L_{\mathsf{DCom}}[\mathsf{pp}] = \{C; m, \rho \mid C = \mathsf{Com}_{\mathsf{pp}}(m; \rho) \wedge m \in \mathcal{M}_{\mathsf{pp}} \wedge \rho \in \mathcal{R}_{\mathsf{pp}} \wedge C \in \mathcal{C}_{\mathsf{pp}}\} \tag{3}$$

– *Commitment of discrete log.*

$$L_{\mathsf{DComDL}}[\mathsf{pp}, (\mathbb{G}, G, q)] = \{C, Y; m, \rho \mid C = \mathsf{Com}_{\mathsf{pp}}(m; \rho) \wedge Y = m \cdot G\} \tag{4}$$

– *Ratio between committed values is the discrete log.* (Recall that $\mathcal{M}_{\mathsf{pp}} = \mathbb{Z}_q$.)

$$\begin{aligned} L_{\mathsf{DComRatio}}[\mathsf{pp}, (\mathbb{G}, G, q)] = \{C_1, C_2, X; m, x, \rho_1, \rho_2 \mid & X = x \cdot G \\ \wedge C_1 =& \mathsf{Com}_{\mathsf{pp}}(m, \rho_1) \\ \wedge C_2 =& \mathsf{Com}_{\mathsf{pp}}(x \cdot m, \rho_2)\} \end{aligned} \tag{5}$$

– *Range of committed value.*

$$L_{\mathsf{Range}}[\mathsf{pp}, a, b] = \{C; m, \rho \mid (C; m, \rho) \in L_{\mathsf{DCom}}[\mathsf{pp}] \wedge a \leq m \leq b\} \tag{6}$$

Proving relation to this language is used as a building block for proving more complex relations that involve range claims, that is, Languages 7,8.

– *Committed affine evaluation.* This language is later on specified for the Paillier encryption scheme, see Appendix D.

$$\begin{aligned} L_{\mathsf{DComEval}}[\mathsf{pp}, pk, \vec{\mathsf{ct}}, q, \ell, s, \kappa] = \{ & \mathsf{ct}^{\mathsf{Eval}}, \boldsymbol{C}; \vec{a}, \rho, \omega, \eta \\ & \mid \mathsf{ct}^{\mathsf{Eval}} = \mathsf{AHE.Eval}(pk, f_{\vec{a}}(\vec{x}), \vec{\mathsf{ct}}; \eta) \\ & \wedge\ \boldsymbol{C} = \mathsf{Com}_{\mathsf{pp}}(\vec{a}; \rho) \\ & \wedge\ a_i \in [0, q)\ \wedge\ \rho \in \mathcal{R}_{\mathsf{pp}}\ \wedge\ \eta \in \mathcal{R}_{pk}\} \end{aligned} \tag{7}$$

– *AHE encryption of a discrete log.*

$$L_{\mathsf{EncDL}}[pk, (\mathbb{G}, G, q)] = \{(\mathsf{ct}, X; x, \eta) \mid \mathsf{ct} = \mathsf{AHE.Enc}(pk, x; \eta) \wedge X = x \cdot G \wedge x \in [0, q)\} \tag{8}$$

– *AHE analogue of a DH tuple; namely, ciphertexts of values x, y and xy.* This language uses the specific evaluation algorithm of the Paillier encryption scheme, see Appendix D.

$$\begin{aligned} L_{\mathsf{EncDH}}[pk, \mathsf{ct}_x] = \{(\mathsf{ct}_y, \mathsf{ct}_z; y, \eta_y, \eta_z) \mid \mathsf{ct}_y = \mathsf{AHE.Enc}(pk, y; \eta_y) \\ \wedge\, \mathsf{ct}_z = \mathsf{AHE.Eval}(pk, f, \mathsf{ct}_x; 0, \eta_z) \text{ s.t. } f(x) = y \cdot x \mod q \wedge y \in [0, q)\} \end{aligned} \tag{9}$$

– *Correct AHE key generation.* The specification of the language depends on the choice of the AHE scheme (Definition B.1) and so it is not mentioned here explicitly. The specification for the Paillier encryption scheme, denoted $L_{\mathsf{GenPal}}[\kappa]$, is given in 23.

$$L_{\mathsf{GenAHE}}[\kappa] = \{pk; \mathsf{aux} \mid (pk; sk) = \mathsf{AHE.Gen}(1^\kappa, \mathsf{aux})\} \tag{10}$$

Threshold Signature. Following the works by Canetti et al. [CMP20, CGG$^+$20] and follow ups [BMP22, Mak22], we use the threshold signature from Functionality 14 (Appendix F), which is not defined specifically for ECDSA; rather, it is a generic one, and the ECDSA part only comes to play whenever a signature has to be verified (in case it is not already stored in the functionality's repository). We refer the reader to [CMP20, CGG$^+$20] for a thorough discussion about the choices made in the functionality. An important observation in the aforementioned works is that if the environment cannot break the unforgeability of the signature scheme after the execution of (potentially multiple instances of) the protocol, then a 'trivial' simulation is perfect, where a 'trivial' simulation refers to the situation where a simulator samples all values for the honest parties as prescribed, and follows the instructions of the protocol. To complete the proof then, one has to show that indeed the environment cannot break unforgeability, which is done via a reduction, which in turn builds on a standard *stand-alone* simulation. This means that we can use some useful techniques like rewinding the adversary, and avoid expensive primitives like UC secure zero-knowledge proofs, when constructing the simulator. This simulation is formalized by the notion of $\mathcal{G}^*$-simulatability as defined below, $\mathcal{G}^*$ is an enhanced signing oracle that processes requests for key-generation, presigning and signing.

Signing Oracle. An enhanced signing oracle is modeled as a functionality with key generation, presign and sign commands. A secure implementation of the oracle in Functionality 2 is analogous to enhanced existential unforgeability security definition of the ECDSA signature (where the adversary has information about the randomness before seeing the signature, see [BMP22] for a discussion).

Definition 2.2 is for unforgeability in the context of an enhanced signing oracle $\mathcal{G}^*$, as given in [BMP22, Def. 3.2]. Note that the definition is described for ECDSA rather than a generic signature scheme, nevertheless, we note that the ECDSA scheme is proven (in [BMP22]) to be $\mathcal{G}^*$-existential unforgeable.

Definition 2.2 *We say that* $\mathsf{Sig} = (\mathit{Gen}, \mathsf{Sign}, \mathsf{Verfiy})$ *is $\mathcal{G}^*$-existentially unforgeable ECDSA scheme if for every adversary $\mathcal{A}$ there exists $\mu \in$ neg such that* $\Pr[\mathsf{Exp}_{\mathsf{EU}}^{\mathcal{G}^*}(\mathcal{A}, 1^\kappa) = 1] \leq \mu(\kappa)$*, where $\mathsf{Exp}_{\mathsf{EU}}^{\mathcal{G}^*}$ is defined in Experiment 3.*

FUNCTIONALITY 2 (*Enhanced Signing Oracle For ECDSA: $\mathcal{G}^*$*)

Parameters. A hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^*$.
Operation.

1. On input $(\textbf{keygen}, (\mathbb{G}, G, q))$, sample $x \leftarrow \mathbb{Z}_q$ and return $X = x \cdot G$. Store $(X; x)$ in memory and ignore future calls to **keygen**.
2. On input **pres**, sample $k \leftarrow \mathbb{Z}_q$ and return $R = k^{-1} \cdot G$. Store $(R; k)$ in memory and standby.
3. On input $(\textbf{sign}, msg, R)$ do:
   (a) Retrieve $(R; k)$ from memory. If no such R exists or R is undefined, sample $k \leftarrow \mathbb{Z}_q$ and (re)assign $R := k^{-1} \cdot G$.
   (b) Set $s = k(m + rx) \mod q$ where $m = \mathcal{H}(msg)$ and $r = R|_{x-axis}$.
   (c) Erase $(R; k)$ from memory and return $\sigma = (r, s)$.

EXPERIMENT 3 (*$\mathcal{G}^*$-Existential Unforgeability $\mathsf{Exp}^{\mathcal{G}^*}_{\mathsf{EU}}(\mathcal{A}, 1^\kappa)$*)

The experiment interact with an adversary $\mathcal{A}$ and is parameterized with 1^κ.

1. Call $\mathcal{G}^*$ with $(\textbf{keygen}, (\mathbb{G}, G, q))$ and hand X to $\mathcal{A}$.
2. The adversary $\mathcal{A}$ makes $\mathsf{poly}(\kappa)$ adaptive calls to $\mathcal{G}^*$ (on either of the interfaces **keygen**, **pres** or **sign**).
3. The adversary $\mathcal{A}$ returns (m, σ).

   The experiment's output is 1 iff $\mathsf{Verfiy}(X, m, \sigma) = 1$ and m was not queried to $\mathcal{G}^*$, otherwise output 0.

Definition 2.3 *A threshold signature protocol $\Pi = (\Pi_{\mathsf{kgen}}, \Pi_{\mathsf{pres}}, \Pi_{\mathsf{sign}})$ is $\mathcal{G}^*$-simulatable in the random oracle model if for every adversary $\mathcal{A}$ who corrupts at most t of the parties there exists a simulator $\mathcal{S}$ with a black box access to $\mathcal{A}$, to a random oracle $\mathcal{H}$ and to a signing oracle $\mathcal{G}^*$, such that:*

1. *The simulator $\mathcal{S}$ queries $\mathcal{G}^*$ only on messages given as input to the parties in $\Pi.\Pi_{\mathbf{sign}}$ (w.l.o.g. it is assumed that the parties agree on these messages),*
2. *The real and ideal distributions are computationally indistinguishable, namely:*

$$\{\textsc{Real}^{\mathcal{H}}_{\Pi,\mathcal{A}}(1^\kappa, z)\}_{\kappa\in\mathbb{N}, z\in\{0,1\}^*} \stackrel{c}{\equiv} \{\textsc{Ideal}^{\mathcal{H}}_{\mathcal{G}^*,\mathcal{S}}(1^\kappa, z)\}_{\kappa\in\mathbb{N}, z\in\{0,1\}^*}.$$

$\mathcal{G}^*$-Simulatability and UC Security. Let $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verfiy})$ be a signature scheme, and let $\Pi = (\Pi_{\mathsf{kgen}}, \Pi_{\mathsf{pres}}, \Pi_{\mathsf{sign}})$ be a threshold signature protocol for Sig. Following [CMP20], the idea is to show that if Sig and Π satisfy some limited security properties, like stand alone simulatability, then protocol Π UC-realizes the threshold signature functionality $\mathcal{F}_{\mathsf{tsig}}$ (Figure 14) in the strict global random oracle model: Let $\mathcal{A}$ be an adaptive adversary, $\textsc{Real}^{\mathcal{H}}_{\Pi,\mathcal{A}}(1^\kappa, z)$ be the adversary's view in the execution of Π in the presence of $\mathcal{A}$ and auxiliary input z, where $\mathcal{H}$ is a random oracle. Without loss of generality, we can assume that $\textsc{Real}^{\mathcal{H}}_{\Pi,\mathcal{A}}(1^\kappa, z) = (\mathsf{pk}^{\Pi}, \ldots)$ where pk is the public verification key output from the execution of Π_{kgen}. Next, for a PPT algorithm $\mathcal{S}$, with a black box access to $\mathcal{A}$, $\mathcal{H}$ and $\mathcal{G}^*$, we define $\textsc{Ideal}^{\mathcal{H}}_{\mathcal{G}^*,\mathcal{S}}(1^\kappa, z) = (\mathsf{pk}^{\mathcal{G}^*}, \textsc{Out}^{\mathcal{S}})$ that refers to the public verification key generated by $\mathcal{G}^*$ and the output of the simulator.

The following theorem [BMP22, Theorem 5.5] is in the heart of the proof technique that was introduced in [CMP20]. The theorem states that, for a signature scheme Sig that has enhanced unforgeability, if Π is $\mathcal{G}^*$-simulatable then it is UC-secure. Proving $\mathcal{G}^*$-simulatability is a much easier task as it can leverage non-UC primitives and in particular may involve rewinding of the adversary. A proof of the following theorem is given in Appendix G.

Theorem 2.4 *For a signature scheme* Sig *and a threshold signature protocol* Π *for* Sig*, let* $\mathcal{G}^*$ *be an enhanced signing oracle as defined above. Then* Π *UC realizes* $\mathcal{F}_{\mathsf{tsig}}$ *(Functionality 14) in the strict global random oracle model if*

1. Sig *is* $\mathcal{G}^*$*-existentially unforgeable (Definition 2.2), and*
2. Π *is* $\mathcal{G}^*$*-simulatable in the random oracle model (Definition 2.3).*
3. *The outputs of* $\Pi_{\mathbf{keygen}}$ *and* $\Pi_{\mathbf{sign}}$ *are all distinct (except with negligible probability).*

Discussion on $\mathcal{F}_{\mathsf{tsig}}$ (Functionality 14). While $\mathcal{F}_{\mathsf{tsig}}$ (Functionality F) is easy to UC-simulate, it comes with caveats. As introduced in [CMP20] (or [Can04] for non-threshold signatures), the functionality allows the adversary to set the public key, which is often used as an identifier in blockchain systems. In addition, it lets the adversary set independent (potentially different) signatures to the same value. Therefore, $\mathcal{F}_{\mathsf{tsig}}$ is extended to overcome those hurdles. In Step 1(c)i the functionality outputs an error if the same public key is certified in two different session id's, and similarly, in Step 2(c)ii the functionality 'rejects' repeating signatures (even repeating signatures for the same message). To prove that our protocol realizes that (extended) functionality we added condition 3 in Theorem 2.4. To that end, we utilize the fact that even when a single honest party participates in the protocol, the resulting public key is uniformly distributed over a large space and therefore never repeats, and the same holds for the signatures due to the random nonce.

3 The 2PC-MPC Protocol

For the 2PC-MPC protocol, party B is replaced with n independent parties $B_1, \ldots, B_n$ that together emulate party B interacting with A. When distributing the role of B we have to take a special care about the following building blocks:

– **Threshold AHE.** When B is distributed, all secret values of B are distributed as well. In particular, the AHE private key sk should also be shared amongst the parties $B_1, \ldots, B_n$. This leads to the definition of a threshold parameter $t \leq n$, and the assumption that the adversary may corrupt at most t parties out of $B_1, \ldots, B_n$. As an instantiation, one may use any UC-secure protocol for threshold AHE key generation and threshold decryption (e.g., [FMM+23] for a Paillier-based and [BDO23] for class-groups based encryption).
– **Range proof and proof aggregation.** In our protocol, the range statements are encapsulated within other languages. Proofs for those languages are composed

of two components: a Schnorr proof and a range proof (that share the same statement and witness). Previous works [GGN16, CGG+20, BMP22] use a *designated verifier* range proof. Namely, the proof parameter is comprised of a bi-prime that is a product of *safe primes* that are generated separately by each verifier, and so in order to prove a statement to the public one has to prove the statement to each verifier individually. Instead, in our zero-knowledge protocol we treat the range proof abstractly, which enables plugging any range proof implementation. This way, one can plug and play any range proof implementation, depending on the needs. In our case, to obtain the 2PC-MPC effect, it is important to use *publicly verifiable* aggregateable proofs. To this end, we use Bulletproof [BBB+18] for range statements as their proofs are aggregateable and have a trustless setup. Specifically, to prove a range statement to multiple verifiers in a single shot, we enhance the statement to include a commitment to the witness as well. The prover then sends (1) a Schnorr proof of knowledge for the enhanced statement, from which an extractor can extract a witness (but not necessarily in the adequate range), and (2) an independent range proof of knowledge on the committed value, from which an extractor can extract a witness in the specified range. Since the commitment used in both proofs is the same and the commitment scheme is computationally binding, we must have that the extractor extracted the same witness. The above, however, is not sufficient because it only states that the witness is in the right range $\mathcal{M}_{\mathsf{pp}}$, namely, it is in the right range after a reduction to some modulus. To solve this, a range check is added to the response of the range proof, which essentially ensures that the two extracted witnesses are in fact the same *over the integers*. This is true as long as the claimed range is much smaller than the commitment space $\mathcal{C}_{\mathsf{pp}}$. When this is not the case we decompose the witness $w = \sum_i w_i \Delta^i$ such that Δ is much smaller than $\mathcal{C}_{\mathsf{pp}}$, and then proving, using a batch proof, that all w_i's are smaller than Δ. The formal description of the proof appears in Section 5.2.

3.1 The protocol

Key generation. On the first execution of the key-generation in Protocol 4 the parties generate an AHE key: in Step 2c the parties $B_i, \ldots, B_n$ send $(\textbf{keygen}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{TAHE}}$ and wait to the response $(\textbf{pubkey}, \mathsf{sid}, pk, U' \cap U)$. If $pk = \perp$ then the parties abort and output the identified cheaters $U' \cap U$, otherwise, they continue with pk as their public encryption key. The other party A verifies the correctness of the public key by sending $(\textbf{certify}, pk)$ to that functionality.

Then, the secret signing key share of B is distributively sampled, so that each B_i samples its own share of x_i, provides $X_i = x_i \cdot G$ and $\mathsf{ct}_i = \mathsf{AHE.Enc}(pk, x_i; \rho_i)$ and proves that ct_i is an encryption of the discrete log of X_i. These proofs are being aggregated by functionality $\mathcal{F}^{L_{\mathsf{EncDL}}}_{\mathsf{agg-zk}}$ so A and every B_i receive the aggregated statement $X_B = \sum_i X_i$ and $\mathsf{ct}_{\mathsf{key}} = \bigoplus_i \mathsf{ct}_i$ (such that $\mathsf{ct}_{\mathsf{key}}$ is an encryption of the discrete log of X_B, denoted x_B).

Presigning and Signing. From A's perspective, the presign is almost the same as the two-party protocol in Appendix I, except that A also gets $\mathsf{Enc}(\gamma)$ and $\mathsf{Enc}(\gamma \cdot x_B)$, for some unknown γ. Consequentially, at the signing phase, A will homomorphically compute an encryption of its share of the signature multiplied by γ. Meanwhile, the parties $B_1, \ldots, B_n$ distributively generate their signature nonce share k_B as well as that masking value γ. These values are securely multiplied and only their product, $\gamma \cdot k_B \mod q$ is being exposed via threshold decryption, but this is deferred to the signing phase. By this, when the parties obtain A's encrypted signature share in the signing phase, the can eliminate γ by multiplication with $(\gamma k_B)^{-1}$, to derive the signature. See a formal description in Protocol 5 and 6.

PROTOCOL 4 (*Key-Generation Π_{keygen}:* $\mathsf{KeyGen}(\mathbb{G}, G, q)$)

The protocol is parameterized with the ECDSA group description $(\mathbb{G}, G, q)$. The protocol interacts with $n+1$ parties $A, B_1, \ldots, B_n$, where all parties hold pk as input. The parties agree on a fresh sid and do as follows:

1. ***A*'s first message**:
   (a) A samples a random $x_A \leftarrow \mathbb{Z}_q$ and computes $X_A = x_A \cdot G$.
   (b) A sends $(\textbf{com–prove}, \mathsf{pid}_A, X_A; x_A)$ to $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$.
2. **B_i's first message**:
   (a) Each B_i receives $(\textbf{receipt}, \mathsf{pid}_A)$ from $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$.
   (b) Each B_i samples a random $x_i \leftarrow \mathbb{Z}_q$ and computes $X_i = x_i \cdot G$.
   (c) *Only on first execution:* Each B_i sends $(\textbf{keygen}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{TAHE}}$ and wait for the functionality's response. If the functionality responds with $(\textbf{pubkey}, \mathsf{sid}, \perp, U' \cap U)$ then output $U' \cap U$ and abort; otherwise, if the functionality responds with $(\textbf{pubkey}, \mathsf{sid}, pk)$ then continue.
   (d) Each B_i computes $\mathsf{ct}_i = \mathsf{AHE.Enc}(pk, x_i; \rho_i)$ for a randomly chosen ρ_i.
   (e) Each B_i sends $(\textbf{prove}, \mathsf{sid}, \mathsf{pid}_i, X_i, \mathsf{ct}_i; x_i, \rho_i)$ to $\mathcal{F}^{L_{\mathsf{EncDL}}}_{\mathsf{agg-zk}}$.
   (f) Each B_i receives $(\textbf{proof}, \mathsf{sid}, X_B, \mathsf{ct}_{\mathsf{key}})$ from $\mathcal{F}^{L_{\mathsf{EncDL}}}_{\mathsf{agg-zk}}$, if not, it receives and outputs the set of corrupted parties and aborts.
3. ***A*'s second message**:
   (a) A receives $(\textbf{proof}, \mathsf{sid}, X_B, \mathsf{ct}_{\mathsf{key}})$ from $\mathcal{F}^{L_{\mathsf{EncDL}}}_{\mathsf{agg-zk}}$. Note that A implicitly receives pk as well, as it is part of the public parameters of language L_{EncDL}.
   (b) A sends $(\textbf{certify}, pk)$ to $\mathcal{F}_{\mathsf{TAHE}}$ and waits for its response. If the response is 1 then continue, otherwise abort.
   (c) A sends $(\textbf{decom–proof}, \mathsf{pid}_A)$ to $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$.
4. **B_i's verification.**
   (a) Each B_i receives $(\textbf{decom–proof}, \mathsf{pid}_A, X_A)$ from $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$, if not, it aborts.
5. **Output.**
   – A outputs $X = X_A + X_B$ and record $(\mathsf{keygen}, X_B, X, \mathsf{ct}_{\mathsf{key}}, pk)$.
   – Each B_i outputs $X = X_A + X_B$ and record $(\mathsf{keygen}, X_A, X, \mathsf{ct}_{\mathsf{key}}, pk)$.

Security. Theorem 2.4 states that it is sufficient to prove $\mathcal{G}^*$-simulatability in order to achieve UC security for a threshold signature $\mathcal{F}_{\mathsf{tsig}}$ as defined in

PROTOCOL 5 (*Presigning Π_{pres}:* Presign $((\mathbb{G}, G, q), sid)$)

The protocol interacts with $A, B_1, \ldots, B_n$ where everyone has pk and $\mathsf{ct}_{\mathsf{key}}$ as input. Before proceeding, all parties verify that sid has never been used before. Then they do as follows:

1. **A's message**:
   (a) A samples a random $k_A \leftarrow \mathbb{Z}_q$ and computes $K_A = \mathsf{Com}(k_A; \rho_1)$.
   (b) A sends $(\mathbf{prove}, \mathsf{sid}, \mathsf{pid}_A, K_A; k_A, \rho_1)$ to $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DCom}}}$.
2. **B_i's message**:
   (a) *First round:*
      i. B_i receives $(\mathbf{proof}, \mathsf{sid}, \mathsf{pid}_A, K_A)$ from $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DCom}}}$, if not, it aborts.
      ii. B_i samples $k_i \leftarrow \mathbb{Z}_q^*$ and computes $R_i = k_i \cdot G$. Denote $k_B = \sum_i k_i$.
      iii. B_i samples $\gamma_i \in [0, q), \eta^i_{\mathsf{mask}_1}, \eta^i_{\mathsf{mask}_2}, \eta^i_{\mathsf{mask}_3}, \eta^i_{\mathsf{mask}_4} \in \mathcal{R}_{pk}$, and computes
         A. $\mathsf{ct}^i_1 = \mathsf{AHE.Enc}(pk, \gamma_i; \eta^i_{\mathsf{mask}_1})$,
         B. $\mathsf{ct}^i_2 = \mathsf{AHE.Eval}(pk, f_i, \mathsf{ct}_{\mathsf{key}}; \eta^i_{\mathsf{mask}_2})$ where $f_i(x) = \gamma_i \cdot x$,
         C. $\mathsf{ct}^i_3 = \mathsf{AHE.Enc}(pk, k_i; \eta^i_{\mathsf{mask}_3})$.
      iv. B_i sends $(\mathbf{prove}, \mathsf{sid}\|\text{"}\gamma\text{"}, \mathsf{pid}_i, \mathsf{ct}^i_1, \mathsf{ct}^i_2; \gamma_i, \eta^i_{\mathsf{mask}_1}, \eta^i_{\mathsf{mask}_2})$ to $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_{\mathsf{key}}]}$.
      v. B_i sends $(\mathbf{prove}, \mathsf{sid}, \mathsf{pid}_i, R_i, \mathsf{ct}^i_3; k_i, \eta^i_{\mathsf{mask}_3})$ to $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDL}}}$.
   (b) *Second round:*
      i. B_i receives $(\mathbf{proof}, \mathsf{sid}\|\text{"}\gamma\text{"}, \mathsf{ct}_1, \mathsf{ct}_2)$ from $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_{\mathsf{key}}]}$ and $(\mathbf{proof}, \mathsf{sid}, R, \mathsf{ct}_3)$ from $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDL}}}$.
      ii. Otherwise, if B_i receives $(\mathbf{malicious}, \mathsf{sid}, U')$ from $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_{\mathsf{key}}]}$ and/or $(\mathbf{malicious}, \mathsf{sid}, U'')$ from $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDL}}}$, it records the malicious parties (M and/or M') and aborts.
      iii. B_i computes $\mathsf{ct}^i_4 = \mathsf{AHE.Eval}(pk, f'_i, \mathsf{ct}_1; \eta^i_{\mathsf{mask}_4})$ where $f'_i(x) = k_i \cdot x$.
      iv. B_i sends $(\mathbf{prove}, \mathsf{sid}\|\text{"k"}, \mathsf{pid}_i, \mathsf{ct}^i_3, \mathsf{ct}^i_4; k_i, \eta^i_{\mathsf{mask}_3}, \eta^i_{\mathsf{mask}_4})$ to $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_1]}$.
   (c) *Proof verification:*
      B_i receives $(\mathbf{proof}, \mathsf{sid}\|\text{"k"}, \mathsf{ct}_3, \mathsf{ct}_4)$ from $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_1]}$. Otherwise, if B_i receives $(\mathbf{malicious}, \mathsf{sid}, U')$ from $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_1]}$, it records the malicious parties and aborts.
3. **A's verification**
   (a) A receives $(\mathbf{proof}, \mathsf{sid}, R_B, \mathsf{ct}_3)$ from $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDL}}}$, if not, it aborts.
   (b) A receives $(\mathbf{proof}, \mathsf{sid}, \mathsf{ct}_1, \mathsf{ct}_2)$ from $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_{\mathsf{key}}]}$, if not, it aborts.
4. **Output**
   (a) A records $(\mathsf{presign}, \mathsf{sid}, R_B, \mathsf{ct}_1, \mathsf{ct}_2; k_A, \rho_1)$, where ct_1 and ct_2 are encryptions of γ and $\gamma \cdot x_B$.
   (b) B_i records $(\mathsf{presign}, \mathsf{sid}, R_B, K_A, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_4)$, where ct_4 encrypts $\gamma \cdot k_B \mod q$.

Functionality 14. This enables using tools like non-UC zero-knowledge proofs and the rewinding technique.

Theorem 3.1 *$\Pi_{\mathsf{ecdsa}} = (\Pi_{\mathsf{kgen}}, \Pi_{\mathsf{pres}}, \Pi_{\mathsf{sign}})$ (Protocols 4, 5 and 6) is $\mathcal{G}^*$-simulatable (according to Definition 2.3) in the $(\mathcal{F}_{\mathsf{TAHE}}, \mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{agg-zk}}, \mathcal{F}_{\mathsf{com-zk}})$-hybrid and random oracle model, assuming Com is computationally binding and AHE is a semantically secure encryption scheme with secure function evaluation.*

PROTOCOL 6 (*Signing* Π_{sign}: $\mathsf{Sign}\left((\mathbb{G},G,q),\mathit{sid},\mathit{msg}\right)$)

1. A**'s message**:
   (a) A computes $R=(k_A)^{-1}\cdot R_B(=k_A^{-1}k_B\cdot G)$ and $r=R|_{x-axis} \mod q$; denote $k=k_A^{-1}k_B$.
   (b) A samples $\rho_2\in\mathcal{R}_{\mathsf{pp}}$ and computes $U_A=\mathsf{Com}(k_A\cdot x_A;\rho_2)$.
   (c) A sets $a_1=r\cdot k_A\cdot x_A+m\cdot k_A$ and $a_2=r\cdot k_A$.
   (d) A homomorphically evaluates $\mathsf{ct}_1,\mathsf{ct}_2$, on its private function $f_A(x_1,x_2):=a_1x_1+a_2x_2$:
      – $\mathsf{ct}_A\leftarrow\mathsf{AHE.Eval}(pk,f_A,\mathsf{ct}_1,\mathsf{ct}_2;\eta_{\mathsf{eval}})$
   (e) A sends the following proofs:
      – A sends $(\mathbf{prove},\mathsf{sid},\mathsf{pid}_A,K_A,R_B;k_A,\rho_1)$ to $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComDL}}[\mathsf{pp},(\mathbb{G},R,q)]}$.
      – A sends $(\mathbf{prove},\mathsf{sid},\mathsf{pid}_A,K_A,U_A,X_A;k_A,x_A,\rho_1,\rho_2)$ to $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComRatio}}[\mathsf{pp},(\mathbb{G},G,q)]}$.
      – A computes $C_1=(r\odot U_A)\oplus(m\odot K_A)$ and $C_2=r\odot K_A$, and sends $(\mathbf{prove},\mathsf{sid},\mathsf{pid}_A,\mathsf{ct}_A,C_1,C_2;a_1,a_2,r\cdot\rho_2+m\cdot\rho_1,r\cdot\rho_1,\eta)$ to $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComEval}}[\mathsf{pp},pk,\mathsf{ct}_1,\mathsf{ct}_2]}$.
2. B_i**'s verification and output**:
   (a) B_i receives the following proofs, otherwise, it aborts.
      – $(\mathbf{proof},\mathsf{sid}||\mathsf{pid}_A,K_A,R_B)$ from $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComDL}}[\mathsf{pp},(\mathbb{G},R,q)]}$.
      – $(\mathbf{proof},\mathsf{sid}||\mathsf{pid}_A,K_A,U_A,X_A)$ from $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComRatio}}[\mathsf{pp},(\mathbb{G},G,q)]}$.
      – $(\mathbf{proof},\mathsf{sid}||\mathsf{pid}_A,\mathsf{ct}_A,C_1,C_2)$ from $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComEval}}[\mathsf{pp},pk,\mathsf{ct}_1,\mathsf{ct}_2]}$.
   (b) B_i verifies that the values used in the proofs are consistent with values obtained previously by B_i, specifically:
      – There are records $(\mathsf{keygen},X_A,X,\mathsf{ct}_{\mathsf{key}},pk)$ and $(\mathsf{presign},\mathsf{sid},R_B,K_A,\mathsf{pt}')$, and
      – $C_1=(r\odot U_A)\oplus(m\odot K_A)$ and $C_2=r\odot K_A$, where $r=R|_{x-axis}$.
   (c) B_i sends $(\mathbf{decrypt},pk,\mathsf{ct}_A)$ and $(\mathbf{decrypt},pk,\mathsf{ct}_4)$ to $\mathcal{F}_{\mathsf{TAHE}}$ and waits for its response. Let the responses be $(\mathbf{decrypted},pk,\mathsf{ct}_A,\mathsf{pt}_A,U_A)$ and $(\mathbf{decrypted},pk,\mathsf{ct}_4,\mathsf{pt}_4,U_4)$ respectively; if $\mathsf{pt}_A=\bot$ or $\mathsf{pt}_4=\bot$ then output $U_A\cup U_4$ and abort; otherwise compute $s'=\mathsf{pt}_4^{-1}\cdot\mathsf{pt}_A \mod q$ (which is equal to $(\gamma k_B)^{-1}\cdot((rk_Ax_A+mk_A)\gamma+rk_A\gamma x_B)=k^{-1}(rx+m) \mod q$ as required) and output $s=\min\{s',q-s'\}$ (to ensure uniqueness of the signature).
3. **Output:** B_i outputs $\sigma=(r,s)$.

Due to space limit, we defer the full proof to Appendix H and give here only the intuition. We separate the proof of $\mathcal{G}^*$-simulatability to two cases, where in the first case all parties in B are corrupted (and A is honest) and in the second A, together with $n-1$ parties in B are corrupted (and B_i for some i is honest). In the first case, the simulator can produce a signing key and presignature in exactly the same distribution as when produced by $\mathcal{G}^*$. The only difference is that the simulator encrypts the partial signature s_A directly (which is then multiplied by $(\gamma k_B)^{-1}$ to get the final signature), whereas in the real protocol party A encrypts some partial signature that is equal s_A only after a reduction modulo q. In addition, the simulator commits to zero, instead of the actual commitment U_A. However, the two cases are indistinguishable thanks to the fact that the AHE scheme has secure function evaluation and that the commitment scheme is perfectly hiding. In the second case, the simulator uses an encryption of zero instead of the actual ciphertexts $\mathsf{ct}_{\mathsf{key}}, \mathsf{ct}_1, \ldots, \mathsf{ct}_4$. The simulation is shown indistinguishable to the real execution (where these ciphertexts decrypt to values different than zero) using a series of hybrids, each changes one of the ciphertexts to be computed just like in the real execution. Then it is argued that consecutive hybrids are indistinguishable thanks to the semantic security of the AHE scheme.

On instantiation with the Paillier encryption scheme. The protocol described above impacts the parameters to be chosen by the underlying encryption scheme. Specifically, note that by the Definition of $\mathsf{Paillier.Eval}$ (in Appendix D) and by the constraint implied by the zero-knowledge protocol for language L_{DComEval} (in Section 5), the size of N, the bi-prime modulus of the encryption scheme's plaintext space (before reduction modulo q) must be treated carefully. Given that $n < \sigma$ (i.e., the number of participants is smaller than the statistical security parameter), we have that $N > q^7$ is sufficient to ensure the security of our protocol.

4 Enhanced Batch Schnorr Protocols

We build on the general Schnorr protocol from Appendix J and show how to extend it to an enhanced form (to treat a range claim as part of the statement).

Let $\phi : \mathbb{H} \to \mathbb{G}$ denote a group homomorphism from $(\mathbb{H}, +)$, the witness group, to $(\mathbb{G}, \cdot)$, the associated statement group, and $\boldsymbol{E} \subset \mathbb{Z}_{\geq 0}$ (zero and positive integers) denote the challenge space.

When the statement includes range statements, we use 'enhanced' witness and statement groups $\hat{\mathbb{H}}$ and $\hat{\mathbb{G}}$ that embed $\mathbb{H}$ and $\mathbb{G}$, respectively. Specifically, let $(\phi, \mathbb{H}, \mathbb{G}, \boldsymbol{E})$ and $(\hat{\phi}, \hat{\mathbb{H}}, \hat{\mathbb{G}}, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ be such that $\hat{\mathbb{H}} = \mathbb{H} \times \mathbb{K}$, $\hat{\mathbb{G}} = \mathbb{G} \times \mathbb{C}$, $\hat{\phi}(u, v) = (\phi(u), \theta(u; v))$ and $\boldsymbol{R}, \boldsymbol{S} \subseteq \mathbb{H}$ (where $\boldsymbol{R}$ is the domain from which the witnesses w_i are claimed to be and $\boldsymbol{S}$ is the domain from which the prover draws the mask for the witness). For our purposes we instantiate θ to be any homomorphic commitment scheme Com (see Definition C.1) with public parameters $\widehat{\mathsf{pp}}$, and so $\theta(u; v) \in \mathbb{C} = \mathcal{C}_{\widehat{\mathsf{pp}}}$ is a commitment to $u \in \mathbb{G}$ using randomness $v \in \mathbb{K} = \mathcal{R}_{\widehat{\mathsf{pp}}}$. When applied to a vector (or vectors) ϕ and θ are applied element-wise.

Definition 4.1 *An m-batch* **enhanced** *Schnorr protocol for the tuple $(\hat{\phi}, \hat{\mathbb{H}}, \hat{\mathbb{G}}, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ consists of the following process. For common input $\vec{X} \in \mathbb{G}^m$ and secret witness $\vec{w} \in \boldsymbol{R}^m$ to the prover, such that $\phi(w_j) = X_j$ for all $j \in [1, m]$:*

1. *$\mathcal{P}$ samples $\vec{v} \leftarrow \mathbb{K}^m, \rho \leftarrow \mathbb{K}$ and $\alpha \leftarrow \boldsymbol{S}$ and sends $\vec{Y} = (\theta(w_1; v_1), \ldots, \theta(w_m; v_m))$, $X^A = \phi(\alpha)$ and $Y^A = \theta(\alpha, \rho)$ to $\mathcal{V}$.*
2. *$\mathcal{P}$ sends $(\mathbf{prove}, \mathit{sid}, \mathit{pid}, \vec{Y}; \vec{w}, \vec{v})$ to $\mathcal{F}_{\mathsf{batch-zk}}^{L_{\mathsf{Range}}[\boldsymbol{R}]}$.*
3. *Upon receiving $\vec{Y}, X^A, Y^A$ from $\mathcal{P}$ and $(\mathbf{proof}, \mathit{sid}, \mathit{pid}, \vec{Y})$ from $\mathcal{F}_{\mathsf{batch-zk}}^{L_{\mathsf{Range}}[\boldsymbol{R}]}$ (same $\vec{Y}$ in both messages), $\mathcal{V}$ sends a random challenge $\vec{e} \leftarrow \boldsymbol{E}^m$ to $\mathcal{P}$.*
4. *$\mathcal{P}$ replies with $z = \alpha + \sum_{j=1}^{m} e_j \cdot w_j \in \mathbb{H}$ and $z' = \rho + \sum_{j=1}^{m} e_j \cdot v_j \in \mathbb{K}$.*
5. *$\mathcal{V}$ checks that $\hat{\phi}(z, z') = (X^A, Y^A) \prod_{j=1}^{m} (X_j, Y_j)^{e_j} \in \hat{\mathbb{G}}$, namely, $\phi(z) = X^A \prod_{j=1}^{m} X_j^{e_j} \in \mathbb{G}$ and $\theta(z, z') = Y^A \oplus (\bigoplus_{j=1}^{m} e_j \cdot Y_j) \in \mathbb{C}$ (recall that $\oplus$ denotes hom. addition in $\mathbb{C}$). In addition, $\mathcal{V}$ checks that $z \in \boldsymbol{S}$ and $z' \in \mathbb{K}$.*

We now present a security definition for an enhanced Schnorr protocol, which is analogous to Definition J.1. For $\mu, \varepsilon \in \mathsf{poly}(\kappa)$, we define the following properteis:

- *μ-HVZK. If $X_i = \phi(w_i)$ and $w_i \in \boldsymbol{R}$ for all $i \in [1, m]$, then the simulated transcript $(\vec{X}, \vec{A}, \vec{Y}, \vec{e}, z, z')$ and $(\mathbf{proof}, \mathit{sid}, \mathit{ssid}, \vec{Y})$ is statistically μ-close to the honest transcript, where $z \in \boldsymbol{S}$ and $z' \leftarrow \mathbb{K}$, $\vec{e} \leftarrow \boldsymbol{E}^m$, $Y_j \leftarrow \theta(0, \mathbb{K})$ for all $j \in [1, m]$, $A_1 = \phi(z) \prod_{j=1}^{m} X_j^{-e_j} \in \mathbb{G}$ and $A_2 = \theta(z, z') \oplus (\bigoplus_{j=1}^{m} -e_j \cdot Y_j) \in \mathbb{C}$.*
- *ε-Knowledge soundness. There exists a set V such that:*
  - *Sampling $E \leftarrow \boldsymbol{E}^{m \times (m+1)}$, it holds that $\Pr[E \in V] \geq 1 - \mathsf{neg}(\kappa)$.*
  - *There exists a PPT extractor $\mathcal{E}$ such that given $(1^\kappa, \vec{X}, \vec{Y}, X^A, Y^A, E, Z, Z')$ and $(\mathbf{proof}, \vec{w}, \vec{v})$ where $E = (\vec{e}_1, \ldots, \vec{e}_{m+1})$, $Z = (\vec{z}_1, \ldots, \vec{z}_{m+1})$ and $Z' = (\vec{z}'_1, \ldots, \vec{z}'_{m+1})$ such that $\mathcal{V}(\vec{X}, \vec{Y}, X^A, Y^A, (\mathbf{proof}, \theta(\vec{w}, \vec{v}), \vec{e}_j, \vec{z}_j, \vec{z}'_j) = 1$ for all $j \in [1, m+1]$, and $E \in V$, then,*

$$\Pr[\phi(\vec{w}) = \vec{X}, \theta(\vec{w}, \vec{v}) = \vec{Y}, \vec{w} \in \mathbb{H}^m, \vec{v} \in \mathbb{K}^m \\ \mid (\vec{w}, \vec{v}) \leftarrow \mathcal{E}(1^\kappa, \vec{X}, \vec{Y}, X^A, Y^A, E, Z, Z')] \geq 1 - \varepsilon.$$

  Importantly, assuming the extractor has $(\mathbf{proof}, \theta(\vec{w}, \vec{v}))$ is allowed as we assume $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{range}}}$ is instantiated with a proof of knowledge, and therefore these values can be extracted from it.

4.1 Aggregation of Enhanced Schnorr Batch Proofs

Let Π denote a (μ, ε)-secure enhanced batch Schnorr protocol for $(\hat{\phi}, \hat{\mathbb{H}}, \hat{\mathbb{G}}, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$. We want to construct a proof aggregation among n provers $\mathcal{P}_1, \ldots, \mathcal{P}_n$, in possession of the witnesses w_j for each statement X_j, for $(\hat{\phi}, \hat{\mathbb{H}}, \hat{\mathbb{G}}, \boldsymbol{E}, n\boldsymbol{R}, n\boldsymbol{S})$, where $n\boldsymbol{R}$ and $n\boldsymbol{S}$ refer to the sets of n-sums from $\boldsymbol{R}$ and $\boldsymbol{S}$, respectively. This new tuple implies that $n\boldsymbol{R}, n\boldsymbol{S} \subseteq \mathbb{H}$. Namely, for every $j \in [m]$, party P_i has $w_{i,j} \in \boldsymbol{R}$ such that $w_j = \sum_i w_{i,j} \in n\boldsymbol{R}$ and $\phi(w_j) = X_j$. Denoting by $\vec{X}_i = (X_{i,j})_{j=1}^m = (\phi(w_{i,j}))_j$, we also have $X_j = \phi(w_j) = \phi(\sum_i w_{i,j}) = \prod_i \phi(w_{i,j}) = \prod_i X_{i,j}$. We show that under this aggregation process, HVZK and knowledge soundness are preserved. The protocol proceeds as follows, each prover $\mathcal{P}_i$:

1. Samples $\alpha_i \leftarrow \boldsymbol{S}, \rho_i \leftarrow \mathbb{K}, \vec{v}_i \leftarrow \mathbb{K}^m$, $u_i \leftarrow \{0,1\}^\kappa$ and computes $\vec{Y}_i = (\theta(w_{i,1}; v_{i,1}), \ldots, \theta(w_{i,m}; v_{i,m})))$ and $X_i^A = \phi(\alpha_i)$, $Y_i^A = \theta(\alpha_i, \rho_i)$. Then, it broadcasts $C_i = \mathcal{H}(\mathsf{sid}, \mathsf{pid}_i, \vec{X}_i, \vec{Y}_i, X_i^A, Y_i^A, u_i)$. Recall that $Y_{i,j} = \theta(w_{i,j}; v_{i,j}) = \mathsf{Com}_{\widehat{\mathsf{pp}}}(w_{j,i}; v_{j,i})$ is a homomorphic commitment.
2. Receives $C_{i'}$ from each party $1 \leq i' \leq n$.
3. Broadcasts $(\vec{X}_i, \vec{Y}_i, X_i^A, Y_i^A, u_i)$ and sends $(\mathbf{prove}, \mathsf{sid}, \mathsf{pid}_i, \vec{Y}_i; \vec{w}_i, \vec{v}_i)$ to $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{Range}}[\boldsymbol{R}]}$.
4. Upon receiving $(\vec{X}_{i'}, \vec{Y}_{i'}, X_{i'}^A, Y_{i'}^A, u_{i'})$ from $P_{i'}$ and $(\mathbf{proof}, \mathsf{sid}, \mathsf{pid}, \vec{Y}_{i'})$ from $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{Range}}[\boldsymbol{R}]}$ for all $i' \in [n]$, party P_i:
   - checks that $C_{i'} = \mathcal{H}(\mathsf{sid}, \mathsf{pid}_{i'}, \vec{X}_{i'}, \vec{Y}_{i'}, X_{i'}^A, Y_{i'}^A, u_{i'})$ and aborts otherwise;
   - computes $u = \sum_i u_i$, $X^A = \prod_{i' \in [n]} X_{i'}^A$, $Y^A = \prod_{i' \in [n]} Y_{i'}^A$, and $X_j = \prod_{i' \in [n]} X_{i',j}$, $Y_j = \prod_{i' \in [n]} Y_{i',j}$ for all $j \in [m]$;

– calculates $\vec{e} = \mathcal{H}(\mathsf{sid}, \{\mathsf{pid}_{i'}\}_{i'}, \vec{X}, \vec{Y}, X^A, Y^A, u)$;
– broadcasts $z_i = \alpha_i + \sum_{j=1}^{m} e_j \cdot w_{i,j}$ and $z'_i = \rho_i + \sum_{j=1}^{m} e_j \cdot v_{i,j} \in \mathbb{K}$.

Upon receiving $z_{i'}$ and $z'_{i'}$ from $P_{i'}$ for all $i' \in [n]$, the verifier $\mathcal{V}$:

1. computes $z = \sum_{i' \in [n]} z_{i'}$, $z' = \sum_{i' \in [n]} z'_{i'}$ and verifies that $z \in n\boldsymbol{S}$ and $z' \in \mathbb{K}$.
2. verifies that $\phi(z) = X^A \prod_{j=1}^{m} X_j^{e_j} \in \mathbb{G}$ and $\theta(z; z') = Y^A \oplus (\bigoplus_{j=1}^{m} e_j \cdot Y_j) \in \mathbb{C}$ (recall that $\oplus$ denotes homomorphic addition in $\mathbb{C}$).
3. If one of the above checks fail, validate them again per each j, to identify the set of malicious provers U'.
4. If no cheater detected then output `accept`.

Theorem 4.2 *The above protocol preserves completeness, zero-knowledge and knowledge soundness, that is:*

– Completeness. *If all $\mathcal{P}_i$'s are honest then $\mathcal{V}$ outputs* `accept`*.*
– zero-knowledge. *W.l.o.g suppose that $\mathcal{P}_1$ is honest and all $\mathcal{P}_2, \ldots, \mathcal{P}_n$ are corrupted. There exists a simulator $\mathcal{S}$ such that for every statement $\vec{X}$, the statistical distance between a real protocol execution and the output of the simulator $\mathcal{S}(\vec{X})$ is negligible (in the programmable random oracle model).*
– Knowledge Soundness. *This is the same as in Definition 4.1.*

Proof. Completeness of the protocol is trivial and omitted for lack of space.

The proof for zero-knowledge follows from the HVZK property of the single prover protocol, as follows:

1. $\mathcal{S}$ samples a random value $r \leftarrow \{0,1\}^{2\kappa}$ and broadcasts $(\mathsf{sid}, \mathsf{pid}_1, r)$.
2. $\mathcal{S}$ emulates $\mathcal{H}$ and receives from $\mathcal{A}$ the values it commits to.
3. $\mathcal{S}$ sets $\vec{X}_1 = \vec{X} / \prod_{i' \neq 1} \vec{X}_{i'}$, and samples $u_1 \leftarrow \{0,1\}^{\kappa}$, $\vec{v}_1 \in \mathbb{K}^m$ and computes $\vec{Y}_1 = \theta(\vec{0}, \vec{v}_1)$.
4. $\mathcal{S}$ defines $\mathcal{H}' \equiv \mathcal{H}$ except for $\mathcal{H}'(\mathsf{sid}, \mathsf{pid}_1, \vec{X}_1, \vec{Y}_1, u) = r$, and thereafter answers oracle queries according to $\mathcal{H}'$ (rather than $\mathcal{H}$).
5. $\mathcal{S}$ opens the commitment.
6. $\mathcal{S}$ runs the simulator for a single prover, on statement $(\mathsf{sid}, \mathsf{pid}_1, \vec{X}_1, \vec{Y}_1)$ and obtains $(X_1^A, Y_1^A, \vec{e}, \vec{z}, \vec{z'})$. It defines $\mathcal{H}'' \equiv \mathcal{H}'$ except for $\mathcal{H}''(\mathsf{sid}, \mathsf{pid}_1, \vec{X}_1, \vec{Y}_1, X_1^A, Y_1^A) = \vec{e}$, and thereafter answers oracle queries according to $\mathcal{H}''$ (rather than $\mathcal{H}'$).
7. $\mathcal{S}$ then broadcasts the simulated responses $(\mathsf{sid}, \mathsf{pid}_1, \vec{z}_1, \vec{z'}_1)$.

Due to the HVZK of the single prover protocol and union bound, the statistical distance between the simulated and real transcripts is $2^{-s} \cdot n$, where n is the number of participating provers. Therefore, we increase $\boldsymbol{S}$ further by n. □

5 Instantiating $\mathcal{F}_{\mathsf{zk}}$ for L_{DComEval}

We provide a full protocol and proof for L_{DComEval}. A standard way to batch zero-knowledge proofs is presented in Appendix J and the specific security proof for L_{DComEval} is given in Appendix K. In Appendix L we show how to instantiate $\mathcal{F}_{\mathsf{zk}}$ for the other languages; their security proofs are essentially the same.

5.1 The Language L_{DComEval}

Consider L_{DComEval} in Equation (7). We will instantiate AHE with the Paillier encryption scheme, in order to extract the randomness η in the final stage of the knowledge soundness proof. Recall that $\mathsf{Paillier.Eval}$ is defined in Appendix D. In particular, we stress that randomizers in $\mathcal{R}_{pk}$ are represented as pairs (ω, η). Furthermore, any homomorphic commitment scheme Com as in Definition C.1 with a sufficiently large message space suffices. Depending on the range proof implementation, Com may be a batched-Perdersen commitment, a list of Pedersen commitments per each a_i (as Bulletproof [BBB+18] requires), or a single Pedersen bit-commitment as in [DPLS19]. Our proof is indifferent to the particular instantiation. Furthermore, we set the following:

- $\phi(\vec{a}, \rho, \omega, \eta) = \big(\mathsf{Paillier.Eval}(pk, f_{\vec{a}}(\vec{x}), \vec{\mathsf{ct}}; \omega, \eta), \mathsf{Com}(\vec{a}, \rho)\big)$
- $\mathbb{H} = \mathbb{Z}^{\ell+1} \times \mathcal{R}_{\mathsf{pp}} \times \mathbb{Z} \times \mathcal{R}_{pk}$
- $\mathbb{G} = \mathcal{C}_{pk} \times \mathcal{C}_{\mathsf{pp}}$
- $\boldsymbol{E} = [0, 2^{\kappa})$
- $\boldsymbol{R} = [0, q)^{\ell+1} \times \mathcal{R}_{\mathsf{pp}} \times [0, 2^{s} \sum_{i=1}^{\ell} \mathsf{PT}_i) \times \mathcal{R}_{pk}$
- $\boldsymbol{S} = [0, q2^{\kappa+s})^{\ell+1} \times \mathcal{R}_{\mathsf{pp}} \times [0, 2^{2s+\kappa} \sum_{i=1}^{\ell} \mathsf{PT}_i) \times \mathcal{R}_{pk}$

Recall, from the description of the Paillier encryption scheme in Appendix D, that PT_i be the upper bound associated with ct_i. Furthermore, $\mathsf{PT}_{\mathsf{eval}}$ is defined to be the upper bound on the plaintext encrypted under the result of $\mathsf{Paillier.Eval}$, before the reduction modulo N takes place.

5.2 Schnorr Protocol for L_{DComEval}

In order to claim that $C = \mathsf{Com}_{\mathsf{pp}}(m; \rho)$ is a commitment over a message m that lies within a certain range $m \in [0, \Delta)$, we must have $[0, \Delta) \subset \mathcal{M}_{\mathsf{pp}}$. For example, in Pedersen commitments over an elliptic curve, where $\mathcal{M}_{\mathsf{pp}} = [0, q)$, this translates to $q > \Delta$. Indeed, $\mathcal{M}_{\mathsf{pp}} = [0, q)$ is only defined modulo q.

Instead of working with a high order group, we express every witness in base Δ. Specifically, denote $d := \log_{\Delta} \lceil q \rceil$; then we have $a_i = \sum_{j=0}^{d-1} a_{i,j} \Delta^j$; similarly, we define $\ell_\omega = \lceil \log_q 2^s \sum_{i=1}^{\ell} \mathsf{PT}_i \rceil$ and set $\omega = \sum_{i=0}^{\ell_\omega - 1} \omega_i \cdot q^i$, and $\omega_i = \sum_{j=0}^{d-1} \omega_{i,j} \Delta^j$;

Then, we treat the $a_{i,j}$'s and $\omega_{i,j}$'s as the witnesses, and $[0, \Delta)$ as the range to be claimed. This avoids arithmetic over high-order elliptic curves, as well as aligns all witnesses to lie in the same interval $[0, \Delta)$, hence simplifies the analysis. We require $q > \Delta \cdot (d(\ell+1+\ell_\omega) \cdot 2^{\kappa+s} + 2^{\kappa})$. Therefore, we can think of $\mathsf{Paillier.Eval}$ as follows:

$$\mathsf{ct}^{\mathsf{Eval}} = \mathsf{Paillier.Eval}(pk, f_{\vec{a}}(\vec{x}), \vec{\mathsf{ct}}; \omega, \eta) = \begin{bmatrix} a_0 \; a_1 \; \dots \; a_\ell \; \omega \end{bmatrix} \odot \begin{bmatrix} \mathsf{Enc}(1; 0) \\ \mathsf{ct}_1 \\ \vdots \\ \mathsf{ct}_\ell \\ \mathsf{Enc}(q; 0) \end{bmatrix} \oplus \mathsf{Paillier.Enc}(0; \eta) \quad (11)$$

Decomposing the witness in base Δ, we can write Eq. 11 as follows:

$$\mathsf{ct}^{\mathsf{Eval}} = \left[\Delta^0 \dots \Delta^{d-1}\right] \begin{bmatrix} a_{0,0} & a_{1,0} & \dots & a_{\ell,0} & \omega_{0,0} & \dots & \omega_{\ell_\omega-1,0} \\ \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{0,d-1} & a_{1,d-1} & \dots & a_{\ell,d-1} & \omega_{0,d-1} & \dots & \omega_{\ell_\omega-1,d-1} \end{bmatrix} \odot \begin{bmatrix} \mathsf{Enc}(1;0) \\ \mathsf{ct}_1 \\ \vdots \\ \mathsf{ct}_\ell \\ \mathsf{Enc}(q;0) \\ \vdots \\ \mathsf{Enc}(q^{\ell_\omega};0) \end{bmatrix} \oplus \mathsf{Paillier.Enc}(0;\eta) \tag{12}$$

Then, we can denote $a_{\ell+i,j} = \omega_{i-1,j}$ to simplify notation, and let A be the matrix in Eq. 12 whose $(i,j)^{\text{th}}$ entry is $a_{i,j}$ ($i \in [0,\ell_\omega], j \in [0,d-1]$). Furthermore, we denote by $\vec{v}_\Delta$ the row vector $\left[\Delta^0 \dots \Delta^{d-1}\right]$, and denote by $\vec{c}$ the column vector of ciphertexts $\left[\mathsf{Enc}(1;0)\ \mathsf{ct}_1 \dots \mathsf{ct}_\ell\ \mathsf{Enc}(1;0)\ \mathsf{Enc}(q;0) \dots \mathsf{Enc}(q^{\ell_\omega};0)\right]^T$. Therefore, equivalently, we prove the following language:

$$\begin{aligned} L_{\mathsf{DComEval}}[\mathsf{pp}, pk, \vec{\mathsf{ct}}] = \{\mathsf{ct}^{\mathsf{Eval}}, \boldsymbol{C}; A, \rho, \eta \mid & \mathsf{ct}^{\mathsf{Eval}} = \vec{v}_\Delta \cdot A \odot \vec{c} \oplus \mathsf{Enc}(0;\eta) \\ & \wedge\ \boldsymbol{C} = \mathsf{Com}_{\mathsf{pp}}(A;\rho) \\ & \wedge\ A \in [0,\Delta)^{(\ell+1+\ell_\omega)\times d} \\ & \wedge \rho \in \mathcal{R}_{\mathsf{pp}} \wedge \eta \in \mathcal{R}_{pk}\} \end{aligned}$$

Protocol 7 is now parameterized with

- $\phi(A,\rho,\eta) = (\vec{v}_\Delta \cdot A \odot \vec{c} \oplus \mathsf{Enc}(0;\eta), \mathsf{Com}(A;\rho))$
- $\mathbb{H} = \mathbb{Z}^{(\ell+1+\ell_\omega)\times d} \times \mathcal{R}_{\mathsf{pp}} \times \mathcal{R}_{pk}$
- $\mathbb{G} = \mathcal{C}_{pk} \times \mathcal{C}_{\mathsf{pp}}$
- $\boldsymbol{E} = [0, 2^\kappa)$
- $\boldsymbol{R} = [0,\Delta)^{(\ell+1+\ell_\omega)\times d} \times \mathcal{R}_{\mathsf{pp}} \times \mathcal{R}_{pk}$
- $\boldsymbol{S} = [0, \Delta \cdot d(\ell+1+\ell_\omega)\cdot 2^{\kappa+s})^{(\ell+1+\ell_\omega)\times d} \times \mathcal{R}_{\mathsf{pp}} \times \mathcal{R}_{pk}$

HVZK. We need to show that for every $(\mathsf{ct}^{\mathsf{Eval}}, \boldsymbol{C}; A, \rho, \eta) \in R_{\mathsf{DComEval}}[\mathsf{pp}, pk, \vec{\mathsf{ct}}]$ there exists a PPT simulator $\mathcal{S}$, such that

$$\begin{aligned} \mathcal{S}_{\mathsf{pp},pk,\vec{\mathsf{ct}}}(\mathsf{ct}^{\mathsf{Eval}}, \boldsymbol{C}) & \overset{s}{\equiv} \{\mathsf{View}(\mathcal{P}_{\mathsf{pp},pk,\vec{\mathsf{ct}}}(A,\rho,\eta) \leftrightarrow \mathcal{V}_{\mathsf{pp},pk,\vec{\mathsf{ct}}}(\mathsf{ct}^{\mathsf{Eval}}, \boldsymbol{C})\} \\ & \equiv \{\vec{v}_\Delta \cdot B \odot \vec{c} \oplus \mathsf{Enc}(0;\xi), \mathsf{Com}(B;\nu), \\ & \qquad (\mathbf{proof}, \mathsf{Com}(B;\nu)), e, B + e\cdot A, \xi + e\eta, \nu + e\rho\} \end{aligned}$$

over the random variables $B \in [0, \Delta \cdot d(\ell+1+\ell_\omega)\cdot 2^{\kappa+s})^{d\times(\ell+1+\ell_\omega)}$, $\xi \in \mathbb{Z}_N^*$ and $\nu \in \mathcal{R}_{\mathsf{pp}}$.

Consider the following simulator $\mathcal{S}$:

1. $\mathcal{S}$ uniformly samples $e \in [0,2^\kappa)$, $Z \in [0, \Delta \cdot d(\ell+1+\ell_\omega)2^{\kappa+s})^{d\times(\ell+1+\ell_\omega)}$, $z^{\mathsf{Eval}} \in \mathcal{R}_{pk}$, $z^{\mathsf{Com}} \in \mathcal{R}_{\mathsf{pp}}$,
2. $\mathcal{S}$ computes $\boldsymbol{C}_U = \mathsf{Com}(Z; z^{\mathsf{Com}}) \ominus (e \odot \boldsymbol{C})$.
3. $\mathcal{S}$ computes $\mathsf{ct}_U = \vec{v}_\Delta \cdot Z \odot \vec{c} \oplus \mathsf{Enc}(0; z^{\mathsf{Eval}}) \ominus (e \odot \mathsf{ct}^{\mathsf{Eval}})$.

PROTOCOL 7 (*ZK proof for* L_{DComEval})

1. $\mathcal{P}$ samples the matrix $B \in [0, \Delta \cdot d(\ell + 1 + \ell_\omega) \cdot 2^{\kappa+s})^{d\times(\ell+1+\ell_\omega)}$ as well as, $\nu \in \mathcal{R}_{\mathsf{pp}}$, $\xi \in \mathcal{R}^*_{pk}$, computes $\mathsf{ct}_U = \vec{v}_\Delta \cdot B \odot \vec{c} \oplus \mathsf{Enc}(0;\xi)$ and $\boldsymbol{C}_U = \mathsf{Com}_{\mathsf{pp}}(B;\nu)$, and sends $(\mathbf{prove}, \boldsymbol{C}; A, \rho)$ to $\mathcal{F}^{L_{\mathsf{Range}}[\mathsf{pp},0,\Delta]}_{\mathsf{zk-batch}}$, and $(\mathsf{ct}_U, \boldsymbol{C}_U)$ to $\mathcal{V}$.
2. Upon receiving $(\mathsf{ct}_U, \boldsymbol{C}_U)$ from $\mathcal{P}$ and $(\mathbf{proof}, \boldsymbol{C})$ from $\mathcal{F}^{L_{\mathsf{Range}}[\mathsf{pp},0,\lceil q\rceil]}_{\mathsf{zk-batch}}$ (with the same $\boldsymbol{C}$ in both messages), $\mathcal{V}$ sends a random challenge $e \leftarrow [0, 2^\kappa)$ to $\mathcal{P}$.
3. $\mathcal{P}$ responds with $(Z, z^{\mathsf{Eval}}, z^{\mathsf{Com}})$, where:
   – $Z = B + e \cdot A \in \mathbb{Z}^{d\times(\ell+1+\ell_\omega)}$,
   – $z^{\mathsf{Eval}} = \xi + e\eta \in \mathcal{R}_{pk}$, and
   – $z^{\mathsf{Com}} = \nu + e \cdot \rho \in \mathcal{R}_{\mathsf{pp}}$.
4. $\mathcal{V}$ verifies the following equations:

$$\vec{v}_\Delta \cdot Z \odot \vec{c} \oplus \mathsf{Enc}(0; z^{\mathsf{Eval}}) = \mathsf{ct}_U \oplus (e \odot \mathsf{ct}^{\mathsf{Eval}}) \tag{13}$$
$$\mathsf{Com}(Z; z^{\mathsf{Com}}) = \boldsymbol{C}_U \oplus (e \odot \boldsymbol{C}) \tag{14}$$

   In addition, $\mathcal{V}$ verifies that:
   – $z^{\mathsf{Com}} \in \mathcal{R}_{\mathsf{pp}}$, $z^{\mathsf{Eval}} \in \mathcal{R}_{pk}$, $\boldsymbol{C}, \boldsymbol{C}_U \in \mathcal{C}_{\mathsf{pp}}$, $\vec{\mathsf{ct}}, \mathsf{ct}^{\mathsf{Eval}}, \mathsf{ct}_U \in \mathcal{C}_{pk}$, and
   – $Z \in [0, \Delta \cdot (d(\ell + 1 + \ell_\omega) \cdot 2^{\kappa+s} + 2^\kappa))^{d\times(\ell+1+\ell_\omega)}$.

4. $\mathcal{S}$ outputs $(\mathsf{ct}_U, \boldsymbol{C}_U, (\mathbf{proof}, \boldsymbol{C}), e, Z, z^{\mathsf{Eval}}, z^{\mathsf{Com}}$.

First note that this transcript is accepting, as it is computed so that it passes verification. Z is sampled with a valid range for each entry, and ct_U and $\boldsymbol{C}_U$ are fixed such that the equations hold. We now argue that this transcript is indistinguishable from a transcript between a prover who knows the witness and a honest verifier. Indeed, e,z^{Eval} and z^{Com} are identically distributed, and Z is statistically indistinguishable. Indeed, in the simulation each of the entries $Z_{i,j}$ distributes uniformly (and independently) over $[0, \Delta \cdot d(\ell+1+\ell_\omega)2^{\kappa+s})$. In a real execution, each entry distributes uniformly over $[0, \Delta d(\ell+1+\ell_\omega)2^{\kappa+s}) + \varepsilon_{i,j}$ for some offset $\varepsilon_{i,j} \in [0, \Delta 2^\kappa)$, which encodes e times the corresponding witness in $[0, \Delta)$. Therefore, the statistical distance between each pair of entries is bounded by $\frac{1}{d(\ell+1+\ell_\omega})\cdot 2^s$. Therefore, the statistical distance is bounded by:

$$\begin{aligned}
\mathsf{SD}[\mathcal{S}, \mathcal{P} \leftrightarrow \mathcal{V}] &= \mathsf{SD}[\mathcal{S}|_{e,Z,z^{\mathsf{Eval}},z_{\mathsf{Com}}}, (\mathcal{P} \leftrightarrow \mathcal{V})|_{e,Z,z^{\mathsf{Eval}},z_{\mathsf{Com}}}] & (15)\\
&\le \mathsf{SD}[\mathcal{S}|_e, (\mathcal{P} \leftrightarrow \mathcal{V})|_e] + \mathsf{SD}[\mathcal{S}|_{z^{\mathsf{Eval}}}, (\mathcal{P} \leftrightarrow \mathcal{V})|_{z^{\mathsf{Eval}}}] + \mathsf{SD}[\mathcal{S}|_{z^{\mathsf{Com}}}, (\mathcal{P} \leftrightarrow \mathcal{V})|_{z^{\mathsf{Com}}}] + \\
&\quad \mathsf{SD}[\mathcal{S}|_Z, (\mathcal{P} \leftrightarrow \mathcal{V})|_Z] & (16)\\
&\le 0 + 0 + 0 + d(\ell + 1 + \ell_\omega) \cdot \frac{1}{d(\ell + 1 + \ell_\omega)2^s} = 2^{-s} & (17)
\end{aligned}$$

Where Eq. 15 follows since the rest of the transcript is deterministically determined due to the verification equations, and 16 follows because $e, z^{\mathsf{Eval}}, z_{\mathsf{Com}}$ and each entry of Z are mutually independent in both simulation and real execution (see Lemma C.4).

ε-Knowledge Soundness. Given two accepting transcripts $(\mathsf{ct}_U, \boldsymbol{C}_U, e, Z, z^{\mathsf{Eval}}, z^{\mathsf{Com}})$, $(\mathsf{ct}_U, \boldsymbol{C}_U, \hat{e}, \hat{Z}, \hat{z}^{\mathsf{Eval}}, \hat{z}^{\mathsf{Com}})$ where $e \ne \hat{e}$, and given $(\mathbf{proof}, A, \rho)$ where $\mathsf{Com}(A;\rho) =$

$\boldsymbol{C}$, we construct a PPT extractor $\mathcal{E}$ that outputs A, η, ρ such that $(\mathsf{ct}^{\mathsf{Eval}}, \boldsymbol{C}; A, \eta, \rho) \in L_{\mathsf{DComEval}}[pk, \mathsf{pp}, \vec{\mathsf{ct}}]$. The extractor is defined as follows:

- Using (e, Z, z^{Com}) and $(\hat{e}, \hat{Z}, \hat{z}^{\mathsf{Com}})$, apply Eq. 14 as follows. Compute:

$$\mathsf{Com}(Z - \hat{Z}; z^{\mathsf{Com}} - \hat{z}^{\mathsf{Com}}) = (e - e') \odot \boldsymbol{C} = \mathsf{Com}(\tilde{A}; \tilde{\rho})$$

  For $\tilde{A} = (e - \hat{e})^{-1}(Z - \hat{Z}) \mod q$ and $\tilde{\rho} = (e - \hat{e})^{-1}(z^{\mathsf{Com}} - \hat{z}^{\mathsf{Com}}) \mod q$. Note that $\tilde{A} = A \mod q$ because otherwise, the extractor finds two openings for the commitment Com, which breaks the binding of the commitment (for Pedersen, this translates to finding a discrete log). Therefore, ε must be negligible in κ, in contradiction. In particular, $\tilde{A}$ now has entries within $[0, \Delta)$, therefore $\tilde{A} = A$.
- Therefore, $(e - \hat{e})A = Z - \hat{Z} \mod q$. Thus, there exist $K \in \mathbb{Z}^{d \times (\ell + 1 + \ell_\omega)}$, such that:

$$qK = (Z - \hat{Z}) - (e - \hat{e})A \tag{18}$$

  Because the range checks validate, by triangular inequality, we can bound the ℓ_∞-norm (max absolute entry) of the RHS of Eq. 18 by $\Delta(d(\ell + 1 + \ell_\omega) \cdot 2^{\kappa + s} + 2^\kappa) + \Delta 2^\kappa < q$, and therefore we can deduce that $K = 0$, that is, the equation holds over $\mathbb{Z}$:

$$(e - \hat{e})A = Z - \hat{Z} \tag{19}$$

  In particular, the equation holds modulo $\mathcal{P}_{pk}$, that is, modulo N in case of the Paillier AHE cryptosystem.

Then, from Eq. 13 we have:

$$\vec{v}_\Delta \cdot (Z - \hat{Z}) \odot \vec{c} \oplus (\mathsf{Enc}(0; z^{\mathsf{Eval}}) \ominus \mathsf{Enc}(0; \hat{z}^{\mathsf{Eval}})) = (e - \hat{e}) \odot \mathsf{ct}^{\mathsf{Eval}}$$

Denote $\mathsf{ct}^0 = \vec{v}_\Delta \cdot A \odot \vec{c}$. Subtracting $(\ominus)$ $(e - \hat{e})\mathsf{ct}^0$ from both sides, by Eq. 19, we get

$$\mathsf{Enc}(0; z^{\mathsf{Eval}}) \ominus \mathsf{Enc}(0; \hat{z}^{\mathsf{Eval}}) = (e - \hat{e})(\mathsf{ct}^{\mathsf{Eval}} \ominus \mathsf{ct}^0) \tag{20}$$

To extract the randomizer η, we rewrite Eq. 20 in Paillier notation:

$$\Rightarrow \left(\frac{z^{\mathsf{Eval}}}{\hat{z}^{\mathsf{Eval}}}\right)^N = \left(\frac{\mathsf{ct}^{\mathsf{Eval}}}{\mathsf{ct}^0}\right)^{e - \hat{e}} \mod N^2$$

We use Lemma 5.1, by having $y = \frac{z^{\mathsf{Eval}}}{\hat{z}^{\mathsf{Eval}}}$, $x = \frac{\mathsf{ct}^{\mathsf{Eval}}}{\mathsf{ct}^0}$, $k = e - \hat{e}$, $M = N^2$ and observing that $N, e - e'$ are co-prime[2], to get $\tilde{\eta}$ such that $\tilde{\eta}^N = \frac{\mathsf{ct}^{\mathsf{Eval}}}{\mathsf{ct}^0} \mod N^2$ and so $\mathsf{ct}^{\mathsf{Eval}} \ominus \mathsf{ct}^0 = \mathsf{Enc}(N, 0; \tilde{\eta})$. By rearranging we get $\mathsf{ct}^{\mathsf{Eval}} = \vec{v}_\Delta \cdot A \odot \vec{c} \oplus \mathsf{Enc}(0; \tilde{\eta}) = \mathsf{Eval}(pk, f_{\vec{a}}(\vec{x}), \vec{\mathsf{ct}}; \omega, \tilde{\eta})$, where $\vec{a}$ and ω are in the required range.

[2] Since $N = p_1 p_2$ where $p_1, p_2 \gg 2^\kappa$, and $e \neq e'$.

Lemma 5.1 *Suppose that $y^N = x^k \mod M$, where k and N are co-prime and $x, y \in \mathbb{Z}_M^*$. Then, there exists $\eta \in \mathbb{Z}_M^*$, such that $\eta^N = x \mod M$. Furthermore, η can be computed efficiently given x, y, N, k.*

Proof. Since k and N are co-prime, there exists $u, v \in \mathbb{Z}$ such that $ku + Nv = 1$, and $u, v \in \mathsf{poly}(||M||_2)$. Thus: $x = x^{ku+Nv} = (y^u \cdot x^v)^N \mod M$ and therefore, $\eta := (y^u \cdot x^v)$ satisfies the equation $\eta^N = x \mod M$.

6 Optimizations

Presign batching. Instead of running multiple instances of the presign protocol in parallel, we can use batched Schnorr proofs, in which one proves statements that belong to multiple sessions at the same time. This is done by having each party sampling m nonces and following the same protocol as in Protocol 5, except that instead of sending a proof to $\mathcal{F}_{\mathsf{zk}}$, sending a proof of m statements in one shot to $\mathcal{F}_{\mathsf{batch-zk}}$. Then, in the signing Protocol 6, the parties use a fresh nonce on each invocation. A proof by simulation of this modified protocol works exactly the same as the one with a single-nonce presign. Instantiating $\mathcal{F}_{\mathsf{batch-zk}}$ is done via batched enhance Schnorr protocol as shown in Section 4 in general, and in Appendix K for the specific language L_{DComEval}.

We note that the proof of language L_{EncDH} in Step 2(b)iv (in the second round of Protocol 5) cannot be batched with our technique, as ct_1 is part of the public parameters and it is different in each presign instance. In this case $\mathcal{F}_{\mathsf{batch-zk}}$ is realized by invoking $\mathcal{F}_{\mathsf{zk}}$ multiple times in parallel. From A's perspective, however, the presign ends after the first round (i.e., in Step 3 A does not verifies the aforementioned proof), and so this does not impact its latency or communication overheads. We remark that since the computation and communication overheads incurred by the zero-knowledge proof is similar to that of computing the statement itself, protecting against a malicious adversary is about $2\times$ more expensive than protecting against a semi-honest adversary.

Optimized Threshold Decryption Intuitively, in order to realize $\mathcal{F}_{\mathsf{TAHE}}$'s decryption, each party will have to send its decryption share of a threshold AHE scheme, validate all decryption shares from each other party, then combine all decryption shares to get the plaintext and derive the signature. Although Tiresias [FMM+23] demonstrate that validating decryption shares can be scaled by batching multiple sid's, in our case, batching multiple signing requests from multiple users, combining the decryption shares is still computationally heavy, and scales at least linearly by the number of parties.

To resolve this issue, we observe that it is enough for a single party B_i to compute the plaintext and derive the signature. Therefore, when getting n signing requests from n different sid's, each party B_i could be responsible for decrypting a different ciphertext. Overall, each party will compute one decryption share per ciphertext, and will derive only one plaintext out of n, resulting with an amortized cost of only $\mathcal{O}(1)$ exponentiations.

We refer to Appendix E for the detailed proof and below explain the proof sketch. To simulate the new protocol, $\mathcal{S}$ first extracts the adversary's secret decryption shares sk_i for the TAHE scheme. Then, to simulate the honest party decryption shares for $\mathsf{ct}_4, \mathsf{ct}_A$, it uses these key shares and the corresponding plaintexts (which are simulated in the same way as in Simulation 19). Then, it calls the adversary which then outputs s', and there are two options:

1. (r, s') is a valid signature on m. Since there is only one correct value for s given r, m, this is the only message it can send to avoid the next round.
2. It sends $s' \neq s$, and therefore the signature doesn't verify. In this case, the parties fall back to the heavy protocol, and send their decryption shares with zk proofs. $\mathcal{S}$ can simulate those proofs for the honest parties.

$\mathcal{O}(1)$ User Complexity (Independent of n) As mentioned, Bulletproof's verification time scales linearly with the witness size. In particular, the verification time of the user A for the range proofs will depend on the number of parties n in B. To overcome this, we use the well-known range proof used back in [GGN16], that is based on a Pedersen commitment over the hidden-order group $\mathbb{Z}^*_{\hat{N}_A}$. We refer to [BMP22] for the details. The required changes in our protocol follow:

1. In key generation, the user A also has to generate the required public parameters for the above commitment, namely $\hat{N}$, and send it to $\mathcal{F}_{\mathsf{com-zk}}$. Therefore, when opening the commitment, A has to prove in zk that $\hat{N}_A$ is valid (in particular, composed of safe primes).
2. When realizing $\mathcal{F}_{\mathsf{agg-zk}}$, the parties in B will generate two proofs: one with Bulletproofs, and the other one respect to A's bi-prime $\hat{N}^A$.
3. Lastly, the above range proof is a *gap* sigma protocol. This means that while correctness and zk are ensured only for a prover that uses a witness in the correct range, knowledge soundness will extract a witness in a potentially larger domain, namely $\boldsymbol{S}$, rather than $\boldsymbol{R}$ as in Definition 4.1. Therefore, the user will treat ct_1 and ct_2 as encryptions of plaintexts in the broader domain $\boldsymbol{S}$, and will therefore use a mask ω in a proportionally larger domain $(\times 2^{\kappa+s})$ as well, to mask its secret linear evaluation function.

Three Rounds Protocol Below we show how to transform the 7-round 2PC-MPC Protocol described in Section 3 into a 3-round protocol for a typical threshold access structure. Intuitively, proof aggregation costs an additional round, as the parties run the protocol interactively. This aggregation is crucial for a 2PC-MPC access structure, as the parties must mimic a single party. However, for a threshold access structure, such aggregation is not strictly required and therefore a trade-off is presented between round and computation complexity.

Therefore, two changes are made. First, the user's secrets become public, e.g., $x_A = 0$ and $k_A = 1$, so that the protocol only involves the parties $\{B_i\}_{i=1}^n$. Second, in presign, we replace the $\mathcal{F}_{\mathsf{agg-zk}}$ functionalities with $\mathcal{F}_{\mathsf{com-zk}}$. At first glance, this results with a 5 round protocol. However, we may treat encrypted values as commitments, as observed in [CGG+20]. This saves commitment rounds and results with a 3-round protocol. In Appendix M, we specify the 2-round presign protocol, present its simulation in detail, and prove indistinguishability.

7 Instantiation & Evaluation

We instantiate the additively homomorphic encryption with Paillier encryption scheme [Pai99] using a 2048-bit bi-prime modulus N, and for the threshold Paillier protocol we use Tiresias implementation [FMM+23]. We use Pedersen commitment [Ped91] as our homomorphic commitment scheme Com over the same ECDSA group $\mathbb{G} = \mathtt{secp256k1}$. Random oracle calls are instantiated via SHA3-256. Zero-knowledge proofs for range statements, namely, instantiation of $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{range}}}$ as well as its batched and aggregateable versions, are implemented using Bulletproof [BBB+18]. Instantiating all other zero-knowledge proofs is done using our proof system as described in Sections 4 and 5.

Our protocol can be instantiated with any broadcast channel, and each instantiation may have a different impact on the full protocol's latency and throughput. The runtime of the whole signing phase is composed (at the parties in B) of computing the decryption share and aggregation of the shares from all parties. Notably, due to amortized decryption, we can decrypt n signatures and in parallel while keeping the latency fixed. The signature runtime for various n and t are given in Figure 1 for both party A (user), party B_i (server) and a party in [DKLS19]. This is the first time that a threshold ECDSA protocol is able to run for such a large number of parties. For [DKLS19] we draw a dashed line for the estimated time it would take (using a quadratic interpolation). Note that user runtime does not depend on the number of parties in the network. The runtime for key-generation and presignature protocols is given in Appendix A.

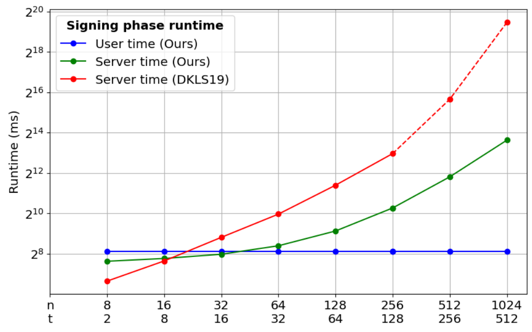

Fig. 1: Servers and user runtime for the signing phase.

Our implementation is written in pure Rust, and will be released as open source. In our implementation we use crypto-bigint[3] for computations over big integers and modular arithmetics over unknown order groups. k256[4] for the group operations. For the Bulletproofs we use dalek-cryptography[5] over the Ristretto

[3] https://github.com/RustCrypto/crypto-bigint
[4] https://github.com/RustCrypto/elliptic-curves/tree/master/k256
[5] https://github.com/dalek-cryptography/bulletproofs/tree/main

group[6]. All the above gives us a full constant-time implementation, which is especially important when implementing cryptography.

We evaluate the performance of our scheme with n (the number of parties) varying from 10 to 1000, and B (the batch size) varying from 1 (without batching) to 1000. For the threshold decryption protocol, we use $t = (2/3)n$ as the threshold. All experiments are conducted over MacBook Pro Apple M1 Max @ 3.22GHz utilizing a single core (single threaded). Parallelization can scale throughput perfectly with the number of cores as computation is completely independent.

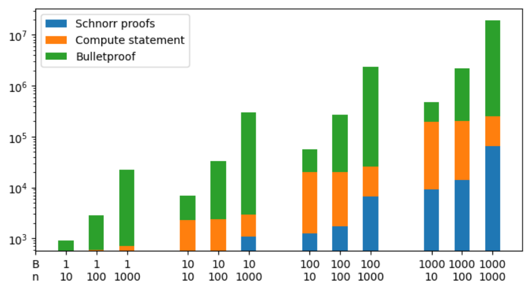

Fig. 2: Separation of the computational cost of the presign protocol (Protocol 5) to times, in milliseconds, for statement computation, Schnorr proofs and range proof generation + verification. Note the y-axis is in logarithmic scale, therefore the bottleneck is clearly the Bulletproofs. Therefore, when using Class Groups, the statement computation becomes the bottelneck, and malicious security is for essentially "free" (due to batching and aggregation).

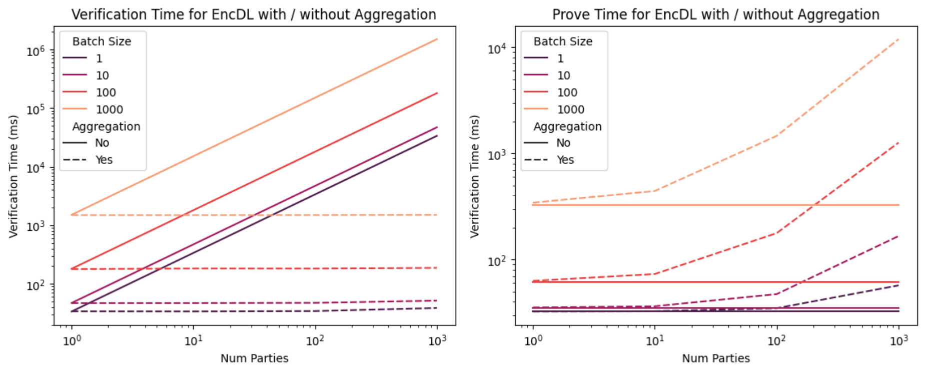

Fig. 3: Verification time of Schnorr proofs without aggregation scales linearly with n, whereas verification of an aggregated proof remains constant. Indeed, proof aggregation incurs a cost that asymptotically scales with the number of parties. However, the proof time is orders of magnitude cheaper than a single verification, therefore the computational overhead, dictated by proof verification, remains practically constant, even for thousand parties.

[6] https://ristretto.group/

References

ACS02. Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products. In *CRYPTO*, volume 2442, pages 417–432. Springer, 2002.

ANO$^+$22. Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *SP*, pages 2554–2572. IEEE, 2022.

Ban05. Endre Bangerter. *Efficient zero knowledge proofs of knowledge for homomorphisms*. PhD thesis, Citeseer, 2005.

BBB$^+$18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

BDO23. Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure multiparty computation from threshold encryption based on class groups. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO*, pages 613–645, 2023.

BGG17. Dan Boneh, Rosario Gennaro, and Steven Goldfeder. Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security. In *LATINCRYPT*, volume 11368 of *Lecture Notes in Computer Science*, pages 352–377. Springer, 2017.

BMP22. Constantin Blokh, Nikolaos Makriyannis, and Udi Peled. Efficient Asymmetric Threshold ECDSA for MPC-based Cold Storage. *Cryptology ePrint Archive*, 2022.

BPMW16. Florian Bourse, Rafaël Del Pino, Michele Minelli, and Hoeteck Wee. FHE Circuit Privacy Almost for Free. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 62–89. Springer, 2016.

Can04. Ran Canetti. Universally composable signature, certification, and authentication. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.*, pages 219–233. IEEE, 2004.

CCL$^+$20. Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In *PKC*, volume 12111 of *Lecture Notes in Computer Science*, pages 266–296. Springer, 2020.

CGG$^+$20. 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, 2020.

CHI$^+$21. Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, Abhi Shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In *(SP)*, pages 590–607. IEEE, 2021.

CMP20. Ran Canetti, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA. *IACR Cryptol. ePrint Arch.*, page 492, 2020.

DJN$^+$20. Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergård. Fast threshold ECDSA with honest majority. In *SCN*, volume 12238 of *LNCS*, pages 382–400. Springer, 2020.

DKLS19. Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *SP 2019*, pages 1051–1066. IEEE, 2019.

DKLS23. Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA in three rounds. *IACR Cryptol. ePrint Arch.*, page 765, 2023.

DOK$^+$20. Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In *ESORICS*, volume 12309, pages 654–673. Springer, 2020.

DPLS19. Rafaël Del Pino, Vadim Lyubashevsky, and Gregor Seiler. Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts. In *IACR International Workshop on Public Key Cryptography*, pages 344–373. Springer, 2019.

FMM$^+$23. Offir Friedman, Avichai Marmor, Dolev Mutzari, Yehonatan C. Scaly, Yuval Spiizer, and Avishay Yanai. Tiresias: Large scale, maliciously secure threshold paillier. *IACR Cryptol. ePrint Arch.*, page 998, 2023.

GG19. Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. *IACR Cryptol. ePrint Arch.*, page 114, 2019.

GGN16. Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *International Conference on Applied Cryptography and Network Security*, pages 156–174. Springer, 2016.

GKSS20. Adam Gagol, Jedrzej Kula, Damian Straszak, and Michal Swietek. Threshold ECDSA for decentralized asset custody. *IACR Cryptol. ePrint Arch.*, page 498, 2020.

HL10. Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010.

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

LN18. 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.

Mak22. Nikolaos Makriyannis. On the classic protocol for MPC schnorr signatures. *IACR Cryptol. ePrint Arch.*, page 1332, 2022.

Mau15. Ueli Maurer. Zero-knowledge proofs of knowledge for group homomorphisms. *Designs, Codes and Cryptography*, 77:663–676, 2015.

MOR14. Tal Moran, Ilan Orlov, and Silas Richelson. Topology-hiding computation. Cryptology ePrint Archive, Paper 2014/1022, 2014. `https://eprint.iacr.org/2014/1022`.

Pai99. Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In Jacques Stern, editor, *EUROCRYPT*, volume 1592, pages 223–238. Springer, 1999.

Ped91. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

Tas07. Tamir Tassa. Hierarchical threshold secret sharing. *J. Cryptol.*, 20(2):237–264, 2007.

WMYC23. 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.

ZYP23. Guy Zyskind, Avishay Yanai, and Alex 'Sandy' Pentland. Unstoppable wallets: Chain-assisted threshold ECDSA and its applications. *IACR Cryptol. ePrint Arch.*, page 832, 2023.

A Additional Runtime Results

Note that the runtime for the user does depend on the number of parties since as of now we do not know how to aggregate Bulletproofs. As mentioned in the paper, one can opt using an AHE scheme that support a plaintext space equivalent to the ECDSA group, thereby avoiding the use of range proofs, which will turn the user runtime independent of the number of parties.

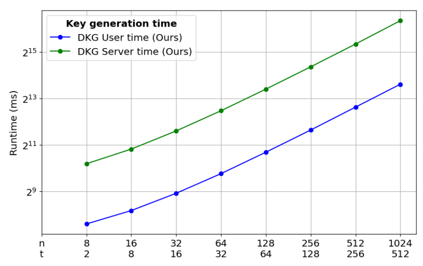

Fig. 4: (a)

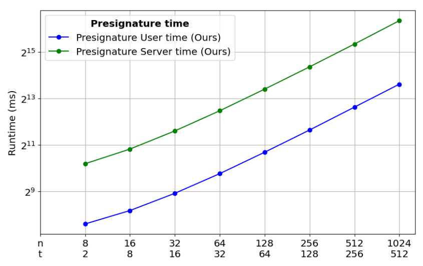

Fig. 5: (b)

Fig. 6: Runtime for our key-generation (a) and presignature (b) protocol for various number of parties, n and threshold t.

B Additively Homomorphic Encryption

The formal definition of AHE below are inspired from [CHI$^+$21] and [BPMW16]. We first describe the non-threshold version which is used in the two party protocol, and later show its extension to the threshold security model.

Definition B.1 (AHE) *An* additively homomorphic encryption scheme *is associated with an ensemble $\{\mathcal{K}_\kappa\}_\kappa$ and consists of four polynomial time algorithms: $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$ specified as follows:*

- $\mathsf{Gen}(1^\kappa, \mathsf{aux}) \to (pk; sk)$*. A probabilistic algorithm that is given a security parameter 1^κ and possibly some auxiliary information* aux *and samples a key-pair (pk, sk) from $\mathcal{K}_\kappa$. In the following, we assume that the resulting pk contains the description of the security parameter 1^κ, the auxiliary information* aux*, as well as the plaintext, randomness and ciphertext spaces $\mathcal{P}_{pk}$, $\mathcal{R}_{pk}$ and $\mathcal{C}_{pk}$, respectively, where $\mathcal{P}_{pk}$ is a $\mathbb{Z}$-module.*
  In addition, for the purpose in this paper we require an AHE scheme to be associated with a language $L_{\mathsf{GenAHE}}[\kappa] = \{pk; \mathsf{aux} \mid (pk; sk) = \mathsf{Gen}(1^\kappa, \mathsf{aux})\}$ (see Eq. 10) that expresses the fact that pk was generated correctly.
- $\mathsf{Enc}(pk, \mathsf{pt}; \eta_{\mathsf{enc}}) \to \mathsf{ct}$*. A deterministic algorithm that on input a public key pk, a plaintext $\mathsf{pt} \in \mathcal{P}_{pk}$ and randomness $\eta_{\mathsf{enc}} \in \mathcal{R}_{pk}$, outputs a ciphertext $\mathsf{ct} \in \mathcal{C}_{pk}$. We define $\mathsf{Enc}(pk, \mathsf{pt})$ as a probabilistic algorithm that first uniformly samples $\eta_{\mathsf{enc}} \in \mathcal{R}_{pk}$ and then outputs $\mathsf{ct} = \mathsf{Enc}(pk, \mathsf{pt}; \eta_{\mathsf{enc}}) \in \mathcal{C}_{pk}$.*

- $\mathsf{Dec}(sk, \mathsf{ct}) \to \mathsf{pt}$. *A deterministic algorithm that on input a secret key sk and a ciphertext $\mathsf{ct} \in \mathcal{C}_{pk}$ outputs a plaintext $\mathsf{pt} \in \mathcal{P}_{pk}$.*
- $\mathsf{Add}(pk, \mathsf{ct}_1, \mathsf{ct}_2) \to \mathsf{ct}_3$. *A deterministic algorithm that on input a public key pk and two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{C}_{pk}$ outputs a ciphertext $\mathsf{ct}_3 \in \mathcal{C}_{pk}$ such that if $\mathsf{ct}_1 = \mathsf{Enc}(pk, m_1; \eta_1)$ and $\mathsf{ct}_2 = \mathsf{Enc}(pk, m_2; \eta_2)$ then $\mathsf{ct}_3 = \mathsf{Enc}(pk, m_1 + m_1; \eta_1 + \eta_2)$, where $m_1 + m_2$ and $\eta_1 + \eta_2$ are over $\mathcal{P}_{pk}$ and $\mathcal{R}_{pk}$, respectively. Imposing correctness will ensure that Add is a homomorphic addition. Note that efficient homomorphic scalar multiplication is implied by the Add operation (e.g., via double-and-add).*

For brevity, we may write $\mathcal{P}$ and $\mathcal{C}$ instead of $\mathcal{P}_{pk}$ and $\mathcal{C}_{pk}$ when the public key pk is clear from the context. In addition, we denote homomorphic addition of two ciphertexts ct_1 and ct_2 by $\mathsf{ct}_1 \oplus \mathsf{ct}_2$ and a multiplication of a ciphertext ciphertext ct by a scalar α by $\alpha \odot \mathsf{ct}$.

Every affine function can be efficiently computed homomorphically by an algorithm $\mathsf{Eval}(pk, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell; \eta_{\mathsf{eval}})$. Whenever we omit the randomness in $\mathsf{Eval}(pk, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$, we refer to the process of sampling a randomness $\eta_{\mathsf{eval}} \leftarrow \{0,1\}^{\mathsf{poly}(\kappa)}$ and running $\mathsf{Eval}(pk, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell; \eta_{\mathsf{eval}})$. Given a public key pk, an affine function $f(x_1, \ldots, x_\ell) = a_0 + \sum_{i=1}^{\ell} a_i x_i$ (with $a_i \in \mathbb{Z}$ and $\ell, \|a_i\|_2 \in \mathsf{poly}(\kappa)$ for all $0 \leq i \leq \ell$) and ℓ ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell \in \mathcal{C}$, Eval outputs a ciphertext ct.

Definition B.2 (Correctness) *AHE is* correct *if for every κ, every $t \in \mathsf{poly}(\kappa)$, every ℓ-ary affine function f as above, and every plaintexts $\mathsf{pt}_1, \ldots, \mathsf{pt}_t \in \mathcal{P}$,*

$$\Pr[\mathsf{Dec}(sk, \mathsf{Eval}(pk, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))) = f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell)] \geq 1 - \mathsf{neg}(\kappa)$$

where $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ and $\mathsf{ct}_i \leftarrow \mathsf{Enc}(pk, \mathsf{pt}_i)$, and the probability is over the coins used by algorithms Gen, Enc and Eval.

Note that for correctness, algorithm Eval may simply output $\mathsf{ct} = \mathsf{Enc}(pk, a_0) \bigoplus_{i=1}^{\ell} a_i \odot \mathsf{ct}_i$, however, such an algorithm may not satisfy secure function evaluation (see below).

Definition B.3 (Semantic security) *AHE is* semantically secure *if for every PPT adversary $\mathcal{A}$ there exists a PPT algorithm $\mathcal{S}$ such that for every pair of PPTs $f, h : \{0,1\}^* \to \{0,1\}^*$, and every $\mathsf{pt} \in \mathcal{P}$:*

$$\begin{aligned} | \ \Pr[\mathcal{A}(1^\kappa, \mathsf{Enc}(pk, \mathsf{pt}), 1^{|\mathsf{pt}|}, h(1^\kappa, \mathsf{pt})) = f(1^\kappa, \mathsf{pt})] \\ - \Pr[\mathcal{S}(1^\kappa, 1^{|\mathsf{pt}|}, h(1^\kappa, \mathsf{pt})) = f(1^\kappa, \mathsf{pt})] \ | < \mathsf{neg}(\kappa) \end{aligned} \tag{21}$$

where $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ and h refers the auxiliary information function in possession of the adversary. The probability is over the random coins of Gen and Enc.

Definition B.4 (Secure Function Evaluation) *AHE has* secure function evaluation *if there exists a PPT algorithm $\mathcal{S}_{\mathsf{Eval}}$ such that for every PPT adversary $\mathcal{A}$, every ℓ-ary affine function f as above, every (pk, sk) and every plaintexts $\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell \in \mathcal{P}$ and randomness $\eta_1, \ldots, \eta_\ell$,*

$$\begin{aligned}&|\Pr[\mathcal{A}(1^\kappa, pk, sk, \{\mathsf{pt}_i, \eta_i\}_{i=1}^{\ell}, \mathit{Eval}(pk, f, (\mathit{ct}_1, \ldots, \mathit{ct}_\ell))) = 1] \\ &- \Pr[\mathcal{A}(1^\kappa, pk, sk, \{\mathsf{pt}_i, \eta_i\}_{i=1}^{\ell}, \mathcal{S}_{\mathit{Eval}}(pk, f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell))) = 1]| \leq \mathit{neg}(\kappa)\end{aligned} \tag{22}$$

where $\mathsf{ct}_i \leftarrow \mathit{Enc}(pk, \mathsf{pt}_i; \eta_i)$, and the probability is over the coins used by Eval and the random coins of $\mathcal{A}$ and $\mathcal{S}$.

An alternative formulation to the secure function evaluation property using Experiment 8 is given below. One can show equivalence between the two definitions.

Definition B.5 (Secure Function Evaluation (Experiment)) *AHE has* secure function evaluation *if there exists a PPT algorithm $\mathcal{S}_{\mathit{Eval}}$ such that for every PPT adversary $\mathcal{A}$, we have* $\Pr[\mathsf{Exp}_{\mathcal{A},\mathcal{S}_{\mathit{Eval}}}(1^\kappa) = 1] \leq \frac{1}{2} + \mathit{neg}(\kappa)$.

EXPERIMENT 8 (*Secure Function Evaluation* $\mathsf{Exp}_{\mathcal{A},\mathcal{S}_{\mathit{Eval}}}(1^\kappa)$)

1. $\mathcal{A}(1^\kappa)$ outputs $\left((pk, sk), \ell, \{\mathsf{pt}_i, \eta_i\}_{i=1}^{t}, f\right)$ where (pk, sk) is an AHE's key-pair, $\{\mathsf{pt}_i, \eta_i\}_{i=1}^{\ell}$ are plaintexts and randomness from which one can obtain ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$, and f is a ℓ-ary affine function as above (with coefficients chosen by $\mathcal{A}$).
2. The experiment picks $b \leftarrow \{0, 1\}$ uniformly at random, then
   - if $b = 0$: compute $\mathsf{ct}^{\mathsf{Eval}} = \mathsf{Eval}(pk, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))$, where $\mathsf{ct}_i = \mathsf{AHE.Enc}(pk, \mathsf{pt}_i; \eta_i)$ for all i,
   - if $b = 1$: compute $\mathsf{ct}^{\mathsf{Eval}} = \mathcal{S}_{\mathsf{Eval}}(pk, f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell))$.
3. Hand $\mathsf{ct}^{\mathsf{Eval}}$ to $\mathcal{A}$ and receive the bit b'.
4. Output 1 if $b = b'$ and 0 otherwise.

C Standard Definitions

Some definitions that are standard in the literature are provided in this section for completeness.

C.1 Commitments and homomorphic commitments

Definition C.1 (Batch Commitment) *A non-interactive commitment scheme consists of a pair of PPT algorithms ($\mathit{Setup}, \mathit{Com}$). The setup algorithm $\mathit{pp} \leftarrow \mathit{Setup}(\mathbb{G}, G, q, T)$, where $T \in \mathbb{N}$, outputs the spaces $\mathcal{M}_{\mathit{pp}} = \mathbb{Z}_q^T$, $\mathcal{R}_{\mathit{pp}} = \mathbb{Z}_q$ and $\mathcal{C}_{\mathit{pp}} = \mathbb{G}$ as well as a generators $H_1, \ldots, H_T$. The commitment algorithm $\mathit{Com}_{\mathit{pp}}$ defines a function $\mathcal{M}_{\mathit{pp}} \times \mathcal{R}_{\mathit{pp}} \rightarrow \mathcal{C}_{\mathit{pp}}$ for message space $\mathcal{M}_{\mathit{pp}}$, randomness space $\mathcal{R}_{\mathit{pp}}$ and commitment space $\mathcal{C}_{\mathit{pp}}$. For a message $\vec{m} \in \mathcal{M}_{\mathit{pp}}$, the algorithm draws $\rho \leftarrow R_{\mathit{pp}}$ uniformly at random, and computes commitment $C = \mathit{Com}_{\mathit{pp}}(\vec{m}; \rho)$. Whenever the public parameters are clear from the context we write Com instead of $\mathit{Com}_{\mathit{pp}}$.*

A commitment schemes has hiding *and* binding *properties as follows:*

- Perfect hiding. *For every adversary $\mathcal{A}$, every $pp \leftarrow \mathsf{Setup}(\cdot)$ and every $\vec{m}_0, \vec{m}_1 \in \mathcal{M}_{pp}$*

$$\Pr[\mathcal{A}(\vec{m}_0, \vec{m}_1, \mathsf{Com}(\vec{m}_b; \rho)) = b] = \frac{1}{2},$$

where the probability is under a uniform choice of $b \in \{0,1\}$ and $\rho \in \mathcal{R}_{pp}$.
- Computational binding. *For every PPT adversary $\mathcal{A}$ and every $pp \leftarrow \mathsf{Setup}(\cdot)$,*

$$\Pr[(\vec{m}_0, \vec{m}_1, \rho_0, \rho_1) \leftarrow \mathcal{A}(pp) : \vec{m}_0 \neq \vec{m}_1 \wedge \mathsf{Com}(\vec{m}_0; \rho_0) = \mathsf{Com}(\vec{m}_1; \rho_1)] \leq \mathsf{neg}(\kappa_q)$$

where κ_q is a computational security parameter associated with q, and the probability is under the random coins of $\mathcal{A}$. The above assumes $\vec{m}_0, \vec{m}_1 \in \mathcal{M}_{pp}$ and $\rho_0, \rho_1 \in \mathcal{R}_{pp}$.

Definition C.2 (Batch Homomorphic Commitments) *A homomorphic commitment scheme is a non-interactive commitment scheme such that $\mathcal{M}, \mathcal{R}$ and $\mathcal{C}$ are all abelian groups, and for all $\vec{m}_1, \vec{m}_2 \in \mathcal{M}$, $\rho_1, \rho_2 \in \mathcal{R}$ we have (in the following, '+' is defined differently for each group):*

$$\mathsf{Com}(\vec{m}_1; \rho_1) + \mathsf{Com}(\vec{m}_2; \rho_2) = \mathsf{Com}(\vec{m}_1 + \vec{m}_2; \rho_1 + \rho_2)$$

C.2 Zero Knowledge Proofs

The functionality is denoted by $\mathcal{F}_{\mathsf{zk}}^{R}$, and formally defined in Functionality 9 and its batched version in Functionality 10.

The notation we use to describe the relations follows: the language associated with relation R is the set $L_R = \{x \mid \exists w : (x, w) \in R\}$; we write $L[\texttt{params}] = \{S; s \mid f(\texttt{params}, S, s)\}$ to denote a language $\{x \mid \exists s : f(\texttt{params}, S, s) = 1\}$ where `params` typically refers to global parameters (e.g., a public-key of an encryption scheme, or a global verification key), S refers to a set of public values and s refers to a set of secret values. Note that in $\mathcal{F}_{\mathsf{zk}}^{R}$ (Functionality 9) we implicitly require 'knowledge' of the witness s from the prover P. Note that we may omit `params` from the description of the languages when they are clear from the context.

FUNCTIONALITY 9 (*Zero-Knowledge: $\mathcal{F}_{\mathsf{zk}}^{R}$*)

The functionality interacts with a prover $\mathcal{P}$ and a set of verifiers $\mathcal{V}_1, \mathcal{V}_2, \ldots$ (in the two-party case there is one verifier $\mathcal{V}$).

- Upon receiving $(\textbf{prove}, \mathsf{sid}, \mathsf{ssid}, x; w)$ from $\mathcal{P}$, if $(x, w) \notin R$ or $(\mathsf{sid}, \mathsf{ssid}, \cdot, \cdot)$ has been previously called then ignore the message. Otherwise, send $(\textbf{proof}, \mathsf{sid}, \mathsf{ssid}, x)$ to the verifiers.

As in [Lin17], in addition to the zero-knowledge functionality, we make use of a committed zero-knowledge proof, defined by Functionality 11. This functionality can be simply implemented by combining a commitment scheme with the ideal zero-knowledge functionality.

FUNCTIONALITY 10 (*Batched Zero-Knowledge:* $\mathcal{F}^{R}_{\mathsf{batch-zk}}$)

The functionality interacts with a prover $\mathcal{P}$ and a set of verifiers $\mathcal{V}_1, \mathcal{V}_2, \ldots$ (in the two-party case there is one verifier $\mathcal{V}$).

- Upon receiving $(\mathbf{prove}, \mathsf{sid}, \mathsf{ssid}, B, \vec{x}; \vec{w})$, where B is the batch size, from $\mathcal{P}$, if $(x_i, w_i) \notin R$ for some $i \in [B]$ or $(\mathsf{sid}, \mathsf{ssid}, \cdot, \cdot)$ has been previously called then ignore the message. Otherwise, send $(\mathbf{proof}, \mathsf{sid}, \mathsf{ssid}, B, \vec{x})$ to the verifiers.

FUNCTIONALITY 11 (*Committed Zero-Knowledge:* $\mathcal{F}^{R}_{\mathsf{com-zk}}$)

The functionality interacts with a prover $\mathcal{P}$ and a verifier $\mathcal{V}$.

- Upon receiving $(\mathbf{com\text{–}prove}, \mathsf{sid}, \mathsf{ssid}, x; w)$ from $\mathcal{P}$, if $(x, w) \notin R$ for some i or $(\mathsf{sid}, \mathsf{ssid}, \cdot, \cdot)$ has been previously called then ignore the message. Otherwise, store $(\mathsf{sid}, \mathsf{ssid}, x; w)$ and send $(\mathbf{receipt}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{V}$.
- Upon receiving $(\mathbf{decom\text{–}proof}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{P}$, if $(\mathsf{sid}, \mathsf{ssid}, x; w)$ has been stored then send $(\mathbf{decom\text{–}proof}, \mathsf{sid}, \mathsf{ssid}, x)$ to $\mathcal{V}$.

C.3 Proof Aggregation

When the relation R is associated with a homomorphism $(x; w) \in R \iff \phi(w) = x$, one can define an aggregated version of a zero-knowledge proof, as in Functionality 12. The aggregated functionality gets a witness w_i for each instance x_i, such that $\phi(w_i) = x_i$, and by the homomorphic property of ϕ, concludes that $\phi(\sum_i w_i) = \oplus_i x_i = x$. It then sends $(\mathbf{proof}, x)$ to the verifier. If any of the proofs fail the functionality alerts the corresponding set of malicious parties, to support identifiable abort.

FUNCTIONALITY 12 (*Aggregated Zero-Knowledge:* $\mathcal{F}^{R}_{\mathsf{agg-zk}}$)

The functionality interacts with a set of provers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, and a verifier $\mathcal{V}$.

- Upon receiving $(\mathbf{prove}, \mathsf{sid}, i, x_i; w_i)$ from $\mathcal{P}_i$ for each $1 \leq i \leq n$
  - If $(\mathsf{sid}, i, \cdot, \cdot)$ has been previously called for some i ignore the message.
  - Let U' be the set of all provers for which $(x_i, w_i) \notin R$. If $M \neq \emptyset$, send $(\mathbf{malicious}, \mathsf{sid}, U')$ to the verifier, otherwise, send $(\mathbf{proof}, \mathsf{sid}, \sum_i x_i)$ to the verifiers.

C.4 Statistical Distance

Definition C.3 (Statistical Distance) *Let $X, Y : \Omega \to [M]$ be two random variables. The* statistical distance *between X, Y, denoted $\mathsf{SD}(X, Y)$, is*

$$\mathsf{SD}(X, Y) := \sum_{w \in \Omega} |\Pr[X = w] - \Pr[Y = w]|$$

Lemma C.4 *Let $X = (X_1, \ldots, X_n)$ be mutually independent n random variables, and also $Y = (Y_1, \ldots, Y_n)$ be mutually independent random variables. Then $\mathsf{SD}[X, Y] \leq \mathsf{SD}[X_1, Y_1] + \ldots + \mathsf{SD}[X_n, Y_n]$.*

Proof. We prove by induction on n. Consider $n = 2$:

$$\begin{aligned}
\mathsf{SD}[(X_1, X_2), (Y_1, Y_2)] &:= \sum_{\alpha=(\alpha_1,\alpha_2)} |\Pr[X_1 = \alpha_1] \cdot \Pr[X_2 = \alpha_2] - \Pr[Y_1 = \alpha_1] \cdot \Pr[Y_2 = \alpha_2]| \\
&= \sum_{\alpha=(\alpha_1,\alpha_2)} |\Pr[X_1 = \alpha_1] \cdot \Pr[X_2 = \alpha_2] - \Pr[Y_1 = \alpha_1] \cdot \Pr[X_2 = \alpha_2] + \\
&\qquad \Pr[Y_1 = \alpha_1] \cdot \Pr[X_2 = \alpha_2] - \Pr[Y_1 = \alpha_1] \cdot \Pr[Y_2 = \alpha_2]| \\
&\leq \sum_{\alpha=(\alpha_1,\alpha_2)} \Pr[X_2 = \alpha_2] \cdot |\Pr[X_1 = \alpha_1] - \Pr[Y_1 = \alpha_1]| + \\
&\qquad \sum_{\alpha=(\alpha_1,\alpha_2)} \Pr[Y_1 = \alpha_1] \cdot |\Pr[X_2 = \alpha_2] - \Pr[Y_2 = \alpha_2]| \\
&= \mathsf{SD}[X_1, Y_1] + \mathsf{SD}[X_2, Y_2]
\end{aligned}$$

Then, by induction hypothesis:

$$\mathsf{SD}[X, Y] \leq \mathsf{SD}[X_1, Y_1] + \mathsf{SD}[(X_2, \ldots, X_n), (Y_2, \ldots, Y_n)] \leq \mathsf{SD}[X_1, Y_1] + \ldots + \mathsf{SD}[X_n, Y_n]$$

D Instantiating AHE with Paillier's Encryptioin Scheme

In this section we present a definition of the scheme $\mathsf{Paillier} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$. Note that we modify the original Paillier definition [Pai99]

- $\mathsf{Gen}(1^\kappa, q)$. Given a security parameter 1^κ and a prime q, sample $\mathsf{poly}(\kappa)$-bit primes p_1 and p_2 and output $(N; (p_1, p_2))$ where $N = p_1 \cdot p_2$ is the public encryption key and $sk = (p_1, p_2)$ is the secret key. Although we acknowledge that the actual plaintext space is $\mathbb{Z}_N$, we define a different, 'virtual', plaintext space as needed by the application; in our case it is $\mathbb{Z}_q$. Define $\mathcal{P} = (\mathbb{Z}_q, +)$, $\mathcal{R} = (\mathbb{Z}_N^*, \cdot)$ and $\mathcal{C} = (\mathbb{Z}_{N^2}^*, \cdot)$.
  The function Dec below expresses the scheme's homomorphism $\mathsf{Dec}_{sk} : \mathbb{Z}_{N^2}^* \to \mathbb{Z}_q$ where $\mathsf{Dec}_{sk}(\mathsf{ct}) = \mathsf{Dec}(sk, \mathsf{ct})$, such that $\mathsf{Dec}_{sk}(x_1 \cdot x_2) = \mathsf{Dec}_{sk}(x_1) + \mathsf{Dec}_{sk}(x_2) \mod q$.
  The scheme is associated with the the language L_{GenPal}, as defined below, that expresses the validity of a Paillier public key. As argued in [Lin17], the following language suffices for the purpose in this paper, although it does not completely expresses a honest execution of $\mathsf{Paillier.Gen}()$.
  $$L_{\mathsf{GenPal}}[\kappa] = \{N; \phi(N) \mid \gcd(N, \phi(N)) = 1\} \tag{23}$$
- $\mathsf{Enc}(pk, x; \eta)$. Given the public key N, a message $x \in \mathbb{Z}_q$ and randomness $\eta \in \mathbb{Z}_N^*$, output ct and PT, where
  $$\mathsf{ct} = \left[(1 + N)^x \cdot \eta^N \mod N^2\right] \quad \text{and} \quad \mathsf{PT} = q.$$

The algorithm outputs $\mathsf{PT} \in \mathbb{N}$ in order to keep track on the upper bound of the underlying plaintext, before applying a reduction modulo q (see the algorithm Dec below). This is necessary for security as we need that value to never exceed N, which is used in the underlying Paillier scheme.
Note that we may omit an explicit mentioning of the upper bound output of Enc when it does not impact the presentation.

– $\mathsf{Dec}(sk, \mathsf{ct})$. Given the secret key (p_1, p_2) and a ciphertext ct, compute $N = p_1 \cdot p_2$ and output

$$\mathsf{pt} = \left[\frac{[\mathsf{ct}^{\phi(N)} \mod N^2] - 1}{N} \cdot \phi(N)^{-1} \mod N \right] \mod q.$$

– $\mathsf{Add}(pk, \mathsf{ct}_1, \mathsf{ct}_2)$. Given the public key N and two ciphertexts ct_1 and ct_2, output $\mathsf{ct}_3 = \left[\mathsf{ct}_1 \cdot \mathsf{ct}_2 \mod N^2\right]$.
Indeed, if $\mathsf{ct}_1 = \mathsf{Enc}(x_1; \eta_1) = \left[(1+N)^{x_1} \cdot {\eta_1}^N \mod N^2\right]$ and $\mathsf{ct}_2 = \mathsf{Enc}(x_2; \eta_2) = \left[(1+N)^{x_2} \cdot {\eta_2}^N \mod N^2\right]$ then $\mathsf{ct}_3 = \left[(1+N)^{x_1+x_2} \cdot (\eta_1\eta_2)^N \mod N^2\right] = \mathsf{Enc}(x_1 + x_2 \mod N; \eta_1 \cdot \eta_2 \mod N)$. So if $\mathsf{Dec}(sk, \mathsf{ct}_1) = x_1' = x_1 \mod q$ and $\mathsf{Dec}(sk, \mathsf{ct}_2) = x_2' = x_2 \mod q$ then $\mathsf{Dec}(sk, \mathsf{ct}_3) = x_1 + x_2 \mod q$.

As a reminder, we need to specify the Eval algorithm, and show that it satisfies the secure function evaluation property in Eq. 1. The technique was implicitly introduced in [GGN16], yet we highlight it here explicitly. The idea is that, simply using $\oplus, \odot$, we get an encryption of the correct value as long as $f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell) < N$ *over the integers*. However, simply using $\oplus, \odot$ may reveal more information about f than desired. To this end, the Eval algorithm uses randomness, by both adding a sufficiently large random multiple of q and homomorphically adding a fresh encryption of a_0. Then, as long as $f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell) < N$ this does not change the value $f(\cdot) \mod q$, and statistically hides f. In our protocols, we make sure that $f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell) < N$ using zero-knowledge (range) proofs. Formally:

Definition D.1 (Eval) *On input a public key N, an affine function $f(x_1, \ldots, x_\ell) = a_0 + \sum_{i=1}^{\ell} a_i x_i$ where $a_i \in [0, q)$, and ℓ ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$, let PT_i be the upper bound associated with ct_i (that is, this is the maximal value one obtains from decrypting ct_i, but without reducing modulo q), then algorithm Eval outputs $(\mathsf{ct}, \mathsf{PT})$, where*

$$\mathsf{ct} = \mathsf{Enc}(pk, a_0 + \omega q; \eta) \bigoplus_{i=1}^{\ell} (a_i \odot \mathsf{ct}_i)$$

where ω is uniformly chosen from $[0, 2^s \sum_{i=1}^{\ell} \mathsf{PT}_i)$ and η is uniformly chosen from $\mathbb{Z}_N^*$. The upper bound associated with the resulting ct is

$$\mathsf{PT}_{\mathsf{eval}} = q + (2^s + 1) \cdot q \cdot \sum_{i=1}^{\ell} \mathsf{PT}_i$$

Theorem D.2 *If the decisional composite residuosity problem is hard relative to $\mathsf{Paillier}.\mathsf{Gen}$, then the Paillier encryption scheme is semantically-secure as per*

Definition B.3. In addition, algorithm Eval *in Definition D.1 is a secure function evaluation as per definition 2.1 for the Paillier encryption scheme.*

Proof.

Correctness. For every κ, every ℓ, every ℓ-ary affine function f with coefficients in $[0, q)$, and every plaintexts $\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell \in \mathbb{Z}_q$ such that $\mathsf{pt}_i = \mathsf{Dec}(sk, \mathsf{ct}_i)$, let PT_i be the upper bound associated with ct_i (again, this is the maximal value one obtains from decrypting ct_i, but without reducing modulo q), as long as $\mathsf{PT}_{\mathsf{eval}} < N$ we have:

$$\begin{aligned}
\mathsf{Dec}(sk, \mathsf{Eval}(pk, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))) &= \mathsf{Dec}\left(sk, \mathsf{Enc}(pk, a_0 + \omega q; \eta) \bigoplus_{i=1}^{\ell} (a_i \odot \mathsf{ct}_i)\right) \\
&= \mathsf{Dec}\left(sk, \mathsf{Enc}(pk, a_0 + \omega q + \sum_{i=1}^{\ell} a_i \mathsf{pt}_i; \eta \cdot \prod_{i=1}^{\ell} (\eta_i)^{a_i})\right) \\
&= a_0 + \omega q + \sum_{i=1}^{\ell} a_i \mathsf{pt}_i \\
&= f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell) \mod q
\end{aligned}$$

where the first equality holds by our definition of Eval; the second equality holds because $a_0 + \omega q + \sum_{i=1}^{\ell} a_i \mathsf{pt}_i < q + q2^s \sum_{i=1}^{\ell} \mathsf{PT}_i + q \sum_{i=1}^{\ell} \mathsf{PT}_i = \mathsf{PT}_{\mathsf{eval}} < N$; the third equality follows from the correctness of decryption of the original Paillier encryption; and the forth equality holds by definition of f and the fact that $\omega q \bmod q = 0$.

Semantic security. Consider a trivial reduction from an adversary $\mathcal{A}$ breaking the CPA-security game for the modified Paillier scheme to an adversary $\mathcal{A}'$ breaking the CPA-security game for the original scheme. When $\mathcal{A}'$ gets a random public key $pk' = N$, it sets $pk = (N, q)$, and queries $\mathcal{A}$ on pk, receiving $m_0, m_1 \in \mathbb{Z}_q$, interpreting them as $m_0, m_1 \in [0, q)$. It then sends the two messages m_0, m_1, as elements of $\mathbb{Z}_N$, receiving $c' = \mathsf{AHE.Enc}(N', m_b; \eta')$. Then $\mathcal{A}'$ passes c' to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. Clearly, since the encryption algorithms coincide (only that our scheme considers a smaller plaintext space), the advantage of $\mathcal{A}'$ exactly equal the advantage of $\mathcal{A}$, which therefore cannot be small, assuming DCR is hard.

Secure function evaluation. We now show that Eval is *statistically* secure function evaluation. That is, for every ℓ-ary affine function f as above, every (pk, sk) and every ciphertexts $(\mathsf{ct}_1, \mathsf{PT}_1), \ldots, (\mathsf{ct}_\ell, \mathsf{PT}_\ell)$ such that $\mathsf{pt}_i = \mathsf{Dec}(sk, \mathsf{ct}_i)$, and every randomness $\eta_1, \ldots, \eta_\ell \in \mathcal{R}$, we have $SD(X, Y) < 2^{-s}$ for random variables X, Y as follows

$$\begin{aligned}
(X, \mathsf{PT}_{\mathsf{eval}}) &= \mathsf{Eval}(pk, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)) \quad \text{and} \\
Y &= \mathcal{S}(pk, f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell), \mathsf{PT}_{\mathsf{eval}}),
\end{aligned}$$

and $\mathcal{S}(pk, f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell), \mathsf{PT}_{\mathsf{eval}}) = \mathsf{Enc}(pk, f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell) + \omega q; \eta)$ for a uniformly chosen $\omega \leftarrow [0, 2^s \sum_{i=1}^{\ell} \mathsf{PT}_i)$ and $\eta \leftarrow \mathbb{Z}_N^*$. To see this, consider a_i, pt_i as values over the integers $\mathbb{Z}$ and let $a_0 + \sum_{i=1}^{\ell} a_i \mathsf{pt}_i = \tilde{\omega} \cdot q + f(\mathsf{pt}_1, \ldots, \mathsf{pt}_t)$, where $\tilde{\omega} \in [0, \sum_{i=1}^{\ell} \mathsf{PT}_i)$ and $0 \leq f(\mathsf{pt}_1, \ldots, \mathsf{pt}_t) < q$. Then,

$$\begin{aligned}
SD(X,Y) &= \frac{1}{2} \sum_{c \in \mathbb{Z}^*_{N^2}} |\Pr[X = c] - \Pr[Y = c]| \\
&= \frac{1}{2} \sum_{z \in \mathbb{Z}_N} \sum_{\eta_z \in \mathbb{Z}^*_N} |\Pr[X = \mathsf{Enc}(z; \eta_z)] - \Pr[Y = \mathsf{Enc}(z; \eta_z)]| \\
&= \frac{1}{2} \sum_{\omega \in [0, 2^s \sum_{i=1}^{\ell} \mathsf{PT}_i + \tilde{\omega})} \sum_{\eta_z \in \mathbb{Z}^*_N} \Bigg| \Pr[X = \mathsf{Enc}(f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell) + \omega q; \eta_z)] \\
&\quad - \Pr[Y = \mathsf{Enc}(f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell) + \omega q; \eta_z)] \Bigg|
\end{aligned}$$

In the above, notice that we use Enc on input $z \in \mathbb{Z}_N$; this is valid as we consider z as the plaintext result of some evaluation operation, before reducing modulo q, rather than a fresh ciphertext.

For every $f, \{\mathsf{pt}_i, \eta_i\}_{i=1}^{\ell}$ and ω, it holds that η_x, η_y are independent, uniformly distributed random variables (over $\mathbb{Z}^*_N$), thus:

$$\begin{aligned}
SD(X,Y) &= \frac{1}{2} \sum_{\omega \in [0, 2^s \sum_{i=1}^{\ell} \mathsf{PT}_i + \tilde{\omega})} \Bigg| \Pr[X = \mathsf{Enc}(f(\mathsf{pt}_1, \ldots, \mathsf{pt}_t) + \omega q] \\
&\quad - \Pr[Y = \mathsf{Enc}(f(\mathsf{pt}_1, \ldots, \mathsf{pt}_t) + \omega q)] \Bigg|.
\end{aligned}$$

Now, we divide the range $[0, 2^s \sum_{i=1}^{\ell} \mathsf{PT}_i + \tilde{\omega})$ to three non-overlapping parts: $I_1 = [0, \tilde{\omega})$, $I_2 = [\tilde{\omega}, 2^s \sum_{i=1}^{\ell} \mathsf{PT}_i)$ and $I_3 = [2^s \sum_{i=1}^{\ell} \mathsf{PT}_i, 2^s \sum_{i=1}^{\ell} \mathsf{PT}_i + \tilde{\omega})$ and separately calculate the statistical distance for each. By the definition of Eval and $\mathcal{S}$ we get:

$$\begin{aligned}
\forall \omega \in I_1 : &\quad \Pr[X = \mathsf{Enc}(f(\mathsf{pt}_1, \ldots, \mathsf{pt}_t) + \omega q] = 0 \\
\text{and} &\quad \Pr[Y = \mathsf{Enc}(f(\mathsf{pt}_1, \ldots, \mathsf{pt}_t) + \omega q)] = \tfrac{1}{2^s \sum_{i=1}^{\ell} \mathsf{PT}_i}\ , \\
\forall \omega \in I_2 : &\quad \Pr[X = \mathsf{Enc}(f(\mathsf{pt}_1, \ldots, \mathsf{pt}_t) + \omega q] = \tfrac{1}{2^s \sum_{i=1}^{\ell} \mathsf{PT}_i} \\
\text{and} &\quad \Pr[Y = \mathsf{Enc}(f(\mathsf{pt}_1, \ldots, \mathsf{pt}_t) + \omega q)] = \tfrac{1}{2^s \sum_{i=1}^{\ell} \mathsf{PT}_i}\ , \\
\forall \omega \in I_3 : &\quad \Pr[X = \mathsf{Enc}(f(\mathsf{pt}_1, \ldots, \mathsf{pt}_t) + \omega q] = \tfrac{1}{2^s \sum_{i=1}^{\ell} \mathsf{PT}_i} \\
\text{and} &\quad \Pr[Y = \mathsf{Enc}(f(\mathsf{pt}_1, \ldots, \mathsf{pt}_t) + \omega q)] = 0.
\end{aligned}$$

Therefore, we get the following, which implies secure function evaluation as per Equation 1

$$SD(X,Y) = \frac{\tilde{\omega}}{2^s \sum_{i=1}^{\ell} \mathsf{PT}_i} < \frac{\sum_{i=1}^{\ell} \mathsf{PT}_i}{2^s \sum_{i=1}^{\ell} \mathsf{PT}_i} = \frac{1}{2^s}$$

□

E Optimized Threshold Decryption

The realization of $\mathcal{F}_{\mathsf{TAHE}}$ is done using a tuple of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{TDec}, \mathsf{Rec})$. $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Add}$ works similarly to the definition of AHE while TDec is an operation that each party runs locally on the chiphertext and generate a decryption share. Collecting $t + 1$ such decryption shares as inputs to Rec outputs a correct decryption. Since the TDec operation is local typically one must add a ZK proof that this was done correctly and verify all proofs before applying Rec to them. Verifying these proofs, especially when the number of parties (hence t) is large, can dominate the protocol's run time. In addition, the combination in Rec itself may be also very expensive, especially when the plaintext space is of unknown order and so the secret decryption key is shared over the integers (as in Paillier). One possible optimization is for one party to locally compute Rec and provide a proof of correct recombination. Unfortunately, generation and verification of such proofs is not efficient in general. But in the case when the encryption is of a *signature* one may verify the correctness by verifying the signature. In this section we formalize this intuition and show that in our signing protocol this is in fact possible. Furthermore we show that in the context of this protocol it is possible to forgo the ZK proofs when the signature validates and only use them when it fails.

Let us first formalize the TAHE algorithms and its correctness and security defintions.

Definition E.1 (TAHE) *Threshold additively homomorphic encryption scheme is associated with an ensemble $\{\mathcal{K}_\kappa\}_\kappa$ and consists of five polynomial time algorithms $\mathsf{TAHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{TDec}, \mathsf{Rec})$ specified as follows:*

- $\mathsf{Gen}(1^\kappa, i, t, n, \mathsf{aux}) \to (\mathsf{pk}, \mathsf{sk_i})$. *A probabilistic (possibly interactive) protocol that is given a security parameter 1^κ, the number of parties n, a threshold $1 < t < n$ and possibly some auxiliary information aux and samples a key-pair $(pk, \{sk\}_{i\in[n]}) \in \mathcal{K}_\kappa$. In the following, we assume that the resulting pk contains the description of the security parameter 1^κ, the auxiliary information aux, as well as the plaintext, randomness and ciphertext spaces $\mathcal{P}_{pk}, \mathcal{R}_{pk}, \mathcal{C}_{pk}$, respectively, where $\mathcal{P}_{pk}$ is a $\mathbb{Z} -$ module.*
- $\mathsf{Enc}(pk, \mathsf{pt}; \eta_{\mathsf{enc}}) \to \mathsf{ct}$. *A deterministic algorithm that on input a public key pk, a plaintext $\mathsf{pt} \in \mathcal{P}_{pk}$ and randomness $\eta_{\mathsf{enc}} \in \mathcal{R}_{pk}$, outputs a ciphertext $\mathsf{ct} \in \mathcal{C}_{pk}$. We define $\mathsf{Enc}(pk, \mathsf{pt})$ as a probabilistic algorithm that first uniformly samples $\eta_{\mathsf{enc}} \leftarrow \mathcal{R}_{pk}$ and then outputs $\mathsf{ct} = \mathsf{Enc}(pk, \mathsf{pt}; \eta_{\mathsf{enc}}) \in \mathcal{C}_{pk}$*
- $\mathsf{TDec}(sk_i, \mathsf{ct}) \to \mathsf{ct}_i$. *A deterministic algorithm that on input a secret key sk and a ciphertext $\mathsf{ct} \in \mathcal{C}_{pk}$ outputs a plaintext $\mathsf{pt} \in \mathcal{P}_{pk}$.*
- $\mathsf{Rec}(S, \{\mathsf{ct}_i\}_{i\in S}) \to \mathsf{pt}$. *A deterministic algorithm that on input of a set $S \in \binom{[n]}{t+1}$ and a tuple of elements $\{\mathsf{ct}_i\}_{i\in S} \in \mathcal{C}^S_{pk}$ outputs a elements $\mathsf{pt} \in \mathcal{P}_{pk}$.*
- $\mathsf{Add}(pk, \mathsf{ct}_1, \mathsf{ct}_2) \to \mathsf{ct}_3$. *A deterministic algorithm that on input a public key pk and two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{C}_{pk}$ outputs a ciphertext $ct_3 \in \mathcal{C}_{pk}$.*

Correctness. TAHE is *correct* if for every κ, every $t \in \mathsf{poly}(\kappa)$, every ℓ-ary affine function f, any subset $S \in \binom{[n]}{t+1}$ and every plaintexts $\mathsf{pt}_1, \ldots, \mathsf{pt}_t \in \mathcal{P}$,

$$\Pr[\mathsf{Rec}(S, \{\mathsf{TDec}(sk_i, \mathsf{Eval}(pk, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)))\}_{i\in S}) = f(\mathsf{pt}_1, \ldots, \mathsf{pt}_\ell)] \geq 1-\mathsf{neg}(\kappa)$$

where $(pk, \{sk_i\}_{i\in[n]}) \leftarrow \mathsf{Gen}(1^\kappa)$ and $\mathsf{ct}_i \leftarrow \mathsf{Enc}(pk, \mathsf{pt}_i)$, and the probability is over the coins used by algorithms Gen, Enc and Eval.

Semantic Security. TAHE is semantically secure if for every PPT adversary $\mathcal{A}$ there exists a PPT algorithm $\mathcal{S}$ such that for every pair of PPTs $f, h : \{0,1\}^* \to \{0,1\}^*$, and every $\mathsf{pt} \in \mathcal{P}$.

$$\begin{aligned}|\ \Pr[\mathcal{A}(1^\kappa, &\mathsf{Enc}(pk, \mathsf{pt}), 1^{|\mathsf{pt}|}, h(1^\kappa, \mathsf{pt})) = f(1^\kappa, \mathsf{pt})] \\ &- \Pr[\mathcal{S}(1^\kappa, 1^{|\mathsf{pt}|}, h(1^\kappa, \mathsf{pt})) = f(1^\kappa, \mathsf{pt})]\ | < \mathsf{neg}(\kappa)\end{aligned}$$

where $(pk, \{sk\}_{i\in[n]}) \leftarrow \mathsf{Gen}(1^\kappa)$ and h refers to the auxiliary information function in possession of the adversary. The probability is over the random coins of Gen and Enc.

Share Security. TAHE is said to be share secure if for every PPT adversary $\mathcal{A}$ there exist a PPT algorithm $\mathcal{S}$ such that

$$\begin{aligned}|\ \Pr[\mathcal{A}(1^\kappa, &\{\mathsf{TDec}(sk_i, \mathsf{ct})\}_{i\notin U}) = 1] \\ &- \Pr[\mathcal{A}(1^\kappa, \mathcal{S}(\mathsf{pt}, \{\mathsf{TDec}(sk_i, \mathsf{ct})\}_{i\in U})) = 1]\ | < \mathsf{neg}(\kappa)\end{aligned}$$

Where $(pk, \{sk_i\}_{i\in[n]}) \leftarrow \mathsf{Gen}(1^\kappa)$ and $U \in \binom{[n]}{t}$. This notation formalizes the idea that the decryption shares do not admit any information on the shares sk_i of the secret key.

Simulatable Key Generation. TAHE scheme is said to have a simulatable key generation if the protocol Gen realizes the functinlaity $\mathcal{F}_{\mathbf{keygen}}$. This functionality is essentially the key generaiton command from the functionality $\mathcal{F}_{\mathsf{TAHE}}$.

Language for correct formation of shares.

$$L_{\mathsf{TDec}} = \{C, \mathsf{ct}, \mathsf{ct}_i; \rho, sk_i \mid C = \mathsf{Com}(sk_i, \rho) \wedge \mathsf{ct}_i = \mathsf{TDec}(\mathsf{ct}, sk_i) \wedge sk_i \leftarrow \mathsf{Gen}(1^\kappa, i)\}$$

We assume that there exist a realization of $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{TDec}}}$.

E.1 The Protocol

Assume we have a TAHE scheme that is correct, semantically secure, share secure, admits secure function evaluation and that Gen realizes the key generation functionality $\mathcal{F}_{\mathbf{keygen}}$. We define a new protocol $\Pi'_{\mathsf{ecdsa}} = (\Pi'_{\mathbf{keygen}}, \Pi'_{\mathbf{pres}}, \Pi'_{\mathbf{sign}})$ derived from protocols 4, 5 and 6, respectively, where $\Pi'_{\mathbf{pres}} = \Pi_{\mathbf{pres}}$, with the following modifications:

FUNCTIONALITY 13 (*TAHE Key Generation:* $\mathcal{F}_{\mathsf{TAHE-\mathbf{keygen}}}$)

The functionality interacts with parties $B = (B_1, \dots, B_n)$, an adversary $\mathcal{A}$ corrupting a set $U \in \binom{[n]}{t}$ of the parties, and a set of certifiers. It is parameterized by a TAHE scheme $\mathsf{TAHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{TDec}, \mathsf{Rec}, \mathsf{Add})$, and a commitment scheme Com, with a computational security parameter κ and with threshold parameter $t \in [1, n-1]$.

- Upon receiving $(\mathbf{keygen}, \mathsf{sid})$ from all parties in B, for a fresh sid, samples $(pk, \{sk_i\}_{i\in[n]}) \leftarrow \mathcal{K}_\kappa$, calculate $C_i = \mathsf{Com}(sk_i, \rho_i)_{i\in[n]}$ send $(pk, \{sk_i\}_{i\in U}, \{\rho_i\}_{i\in U})$ to the adversary and receive the adversary's response $(\mathtt{continue}, U')$. If $\mathtt{continue} = 1$ or $U' \cap U = \emptyset$ then send $(\mathbf{pubkey}, \mathsf{sid}, pk, \{C_i\}_{i\in[n]})$ to all parties in B.
- Upon receiving $(\mathbf{certify}, pk)$ from some certifier, if $(\cdot, pk; \cdot)$ is stored then return 1 to that certifier, otherwise return 0.

1. **Key Generation.** $\mathcal{F}_{\mathsf{TAHE}}$ is replaced with a call to $\mathcal{F}_{\mathsf{TAHE-\mathbf{keygen}}}$.
2. **Sign.** The two calls to TAHE with the **decrypt** command in Step 2c of Protocol 6 are replaced as follows:
   (a) B_i calculates decryption shares $\mathsf{ct}^i_A = \mathsf{TDec}(\mathsf{ct}_A, sk_i), \mathsf{ct}^i_4 = \mathsf{TDec}(\mathsf{ct}_4, sk_i)$ and broadcasts ct^i_A and ct^i_4.
   (b) B_i calculates $i^* = \mathcal{H}(\mathsf{sid}) \mod (n)$. Then i^* is the designated party who will reconstruct the decrypted value out of the decryption shares. If $i = i^*$, B_i calculates $\mathsf{pt}_A = \mathsf{Rec}(\{\mathsf{ct}^i_A\}_{i\in S})$ and $\mathsf{pt}_4 = \mathsf{Rec}(\{\mathsf{ct}^i_4\}_{i\in S'})$ for $S, S' \in \binom{[n]}{t+1}$. In addition, it calculate $s' = \mathsf{pt}_4^{-1} \cdot \mathsf{pt}_A \mod q$ and $s = \min(s', q - s')$ as done in Protocol 6.
   (c) B_i verifies that (r, s) is a valid signature on msg and $r = R|_{x-axis} \mod q$ (where R previously obtained in the protocol), if so – broadcast s, upon which every party applies the same verification. Otherwise continue to the Fault Detection step below.
   (d) **Fault Detection.**
      i. Upon receiving s from i^* if $s \neq \perp$ and (r, s) is not valid consider i^* corrupted and abort. Otherwise ($s = \perp$) continue.
      ii. Send $(C_i, \mathsf{ct}_A, \mathsf{ct}^i_A; sk_i, \rho)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TDec}}$. Consider a party malicious if no message is received from $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TDec}}$.

E.2 Proof

Again following Theorem 2.4, it is sufficient to prove $\mathcal{G}^*$-simulatability, where $\mathcal{G}^*$ is a signing oracle, and so we prove the following:

Theorem E.2 *:*

$\Pi'_{\mathsf{ecdsa}} = (\Pi'_{\mathsf{kgen}}, \Pi'_{\mathsf{pres}}, \Pi'_{\mathsf{sign}})$ is $\mathcal{G}^*$-simulatable (according to Definition 2.3) in the $(\mathcal{F}_{\mathbf{keygen}}, \mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{agg-zk}}, \mathcal{F}_{\mathsf{com-zk}})$-hybrid and random oracle model.

Proof. Simulating the messages A sends B_i is the same since no changes occurred between Π_{ecdsa} and Π'_{ecdsa} regarding these messages. We thus focus on the case were A is corrupted along with a subset $U \in \binom{[n]}{t}$ of the parties. We will describe the changes in the simulations. Simulations for Key Generation, Presign and sign that is, 17,18 and 19 respectively.

1. **General.** Replace any calls to AHE with calls to TAHE.
2. **Key Generation.** We replace command 3.(b) in the original by the following: Upon receiving $(\mathbf{keygen}, \mathsf{sid})$ from $\mathcal{A}$ on behalf of all B_i for $i \in U$ emulate $\mathcal{F}_{\mathbf{keygen}}$. Generate $(pk, \{sk_i\}_{i\in U}, \{\rho_i\}_{i\in U})$, send them to the adversary and await it's response $(\mathbf{continue}, U')$. If $U' \cap U = \emptyset$ send $(\mathbf{newkey}, \mathsf{sid}, pk, \{C_i\}_{i\in[n]})$ to all parties, otherwise abort.
3. **Pre-sign** This simulator is unchanged.
4. **Sign** We replace command 4.(c) with the following:
   (a) In step 3.(c).(i), calculate $\mathsf{ct}^i_A = \mathsf{TDec}(\mathsf{ct}_A, sk_i), \mathsf{ct}^i_4 = \mathsf{TDec}(\mathsf{ct}_4, sk_i)$ for $i \in U$. Call upon $\mathcal{S}_{shares}$ from the definition of Share Security to generate $\mathsf{ct}^i_A, \mathsf{ct}^i_4$ from $\mathsf{pt}_A, \{\mathsf{ct}^i_A\}_{i\in U}$ and $\mathsf{pt}_4, \{\mathsf{ct}^i_4\}_{i\in U}$ respectively. Send to the adversary $(\mathsf{ct}^i_A, \mathsf{ct}^i_4)_{i\notin U}$.
   (b) In step 3.(c).ii, upon receiving $\{\bar{\mathsf{ct}}^i_A, \bar{\mathsf{ct}}^i_A\}_{i\in U}$ if $i^* \in U$ the adversary should also send s or $(\mathbf{abort})$. If $i^* \notin U$ run Rec on some random subset $S \leftarrow \binom{[n]}{t+1}$ of the decryption shares to get s. If $\mathcal{S}$ has s verify (r, s) is valid, if it is not or **abort** was sent go to Fault Detection.
   (c) **Fault Detection.** Emulate $\mathcal{F}^{\mathsf{TDec}}_{\mathsf{zk}}$ to send the adversary proofs of correct generation of the shares for the honest parties. Further more consider parties in U as cheaters if they sent $\bar{\mathsf{ct}}^i_4 \neq \mathsf{ct}^i_4$ or $\bar{\mathsf{ct}}^i_A \neq \mathsf{ct}^i_A$.

First notice that from Semantic Security and Secure function evaluation of the TAHE any messages generated by the original simulator by calls to AHE are still indistinguishable in the new simulation. Further more notice that by Simulatable Key Generation the messages sent during the key generation simulation are indistinguishable. The only new messages except for fault detection are the shares which are indistinguishable by the Share Security property of the TAHE. All is left to show $\mathcal{G}^*$ simulatability is to prove that aborts happens in the real world and the ideal world. Let us look at different cases:

1. Valid signature: If the signature is valid both the real and ideal worlds will not abort. Furthermore since there is only one valid signature given that r is constant there is no choice of decryption shares for the adversary that can result in a different signature that is valid.
2. Invalid signature: If the signature is not valid $i^* \in U$, in this case both the real world and the ideal world will consider them cheaters and abort. If $i^* \notin U$ we know by the simulation so far that a correct decryption would have resulted in a correct signature. Thus the only way for a signature to be invalid is for one of the decryption shares of the adversary to be invalid. This will be caught in both the real and ideal world by the ideal functionality $\mathcal{F}^{\mathsf{TDec}}_{\mathsf{zk}}$.

F Ideal Functionality for Threshold Signature

Functionality 14 formalizes the ideal threshold signature scheme. The functionality is parameterized with a set of signatories $P = \{P_1, \ldots, P_n\}$ and an efficiently representable access structure Γ. In practice Γ will be either the threshold access structure (t, n) or the hierarchical one $1 + (t, n)$. Both structures are parameterized with $t < n$. In the former it is sufficient that $t + 1$ parties to produce a signature and in the latter $t + 1$ as well as another designated parties must collaborate.

G Proof of Theorem 2.4

We follow the same proof structure from [CMP20, BMP22]. Following [CMP20, Lemma 7.3], we show that if protocol $\Pi = (\Pi_{\mathsf{keygen}}, \Pi_{\mathsf{pres}}, \Pi_{\mathsf{sign}})$ (in Protocols 4-6) does not UC-realize Functionality 14 ($\mathcal{F}_{\mathsf{tsig}}$), then there exists an environment $\mathcal{Z}$ that can forge signatures for previously unsigned messages in an execution of Π. As described in [CMP20, Section 7.4.1] and [BMP22], the simulator simply runs the code of the honest parties as prescribed by Π, i.e. the simulator samples all the secrets as prescribed and plays against the adversary in exactly the same way as in the real execution. To interact with the functionality the simulator proceeds as follows:

1. At the end of an execution of Protocol Π_{keygen}, the simulator sends the obtained public key X, together with the ECDSA verification algorithm,$\mathcal{V}$, to the functionality. Then, the simulator sends to the functionality the expected key confirmation on behalf of the malicious parties. If Π_{keygen} is halted for any reason, then the simulator is instructed to halt as well.
2. At the end of an execution of Protocol Π_{sign} the simulator sends the computed signature $\sigma = (r, s)$ to the functionality. (The simulator does not interact with the functionality if it fails to calculate the signature, or during the simulation of Π_{pres}).

The above simulation is perfect unless $\mathcal{Z}$ can cause the ideal functionality to output an error (errors in the functionality appear in red color). There are three potential errors in our extended functionality: (i) Identifier – the same public key X was certified for two different sid's 1(c)i; (ii) Uniqueness of signatures – the same signature σ is provided for two different sid's 2(c)ii; and (iii) Unforgeability – a valid signature that was never recorded before, is sent for verification. In the functionality, a verification of an unrecorded signature always returns $\beta = 0$ (Step 3b) exactly for that reason, to reflect the fact that the signature scheme is unforgeable.

In the following we explain why any of the above error states is reached with only a negligible probability.

1. Reaching the Identifier state is possible only in case the environment causes the parties in the real world to reach to the same public key in two different

FUNCTIONALITY 14 (*Ideal Threshold Signature Functionality:* $\mathcal{F}^{\Gamma}_{\mathsf{tsig}}$)

Parameters: $P = P_1, \ldots, P_n$ is the set of signatories and $\Gamma \subseteq 2^P$ is an efficient representation of a monotone access structure over P.

1. **Key-generation:**
   (a) Upon receiving $(\mathbf{keygen}, \mathsf{ssid})$ from some party P_i, interpret $\mathsf{ssid} = (\ldots, P)$, where $P = (P_1, \ldots, P_n)$.
   - If $P_i \in P$, send to $\mathcal{S}$ and record $(\mathbf{keygen}, \mathsf{ssid}, P_i)$.
   - Otherwise ignore the message.

   (b) Once $(\mathbf{keygen}, \mathsf{ssid}, P_j)$ is recorded for all $P_j \in P$, send $(\mathbf{newkey}, \mathsf{ssid})$ to the adversary $\mathcal{S}$ and do:
   i. Upon receiving $(\mathbf{newkey}, \mathsf{ssid}, X, \mathcal{V})$ from $\mathcal{S}$, record $(\mathsf{ssid}, X, \mathcal{V})$.
   ii. Upon receiving $(\mathbf{newkey}, \mathsf{ssid})$ from $P_i \in P$, output $(\mathbf{newkey}, \mathsf{ssid}, X)$ if it is recorded.

   (c) Once $(\mathsf{ssid}, X, \mathcal{V})$ is recorded, upon receiving $(\mathbf{confirmkey}, \mathsf{ssid})$ from each $P_i \in P$:
   i. If there is no set $P' \in \Gamma$ for which all $P_i \in P$ are corrupted, and there is a record of $(\mathbf{confirmkey}, \mathsf{ssid}', X, \ldots)$ with $\mathsf{ssid}' \neq \mathsf{ssid}$, output an error. (Identifier)
   ii. Record $(\mathbf{confirmkey}, \mathsf{ssid}, X, \mathcal{V})$.
2. **Signing:**
   (a) Upon receiving $(\mathbf{sign}, \mathsf{sid} = (\mathsf{ssid}, \ldots), m)$ from some party P_i, send to $\mathcal{S}$ and record $(\mathbf{sign}, \mathsf{sid}, m, i)$

   (b) Upon receiving $(\mathbf{sign}, \mathsf{sid} = (\mathsf{ssid}, \ldots), m, j)$ from $\mathcal{S}$, if P_j is corrupted record $(\mathbf{sign}, \mathsf{sid}, m, j)$.

   (c) Once $(\mathbf{sign}, \mathsf{sid}, m, i)$ is recorded for all $P_i \in P'$ for some $P' \in \Gamma$, send $(\mathbf{sign}, \mathsf{sid}, m)$ to the adversary $\mathcal{S}$ and, upon receiving $(\mathbf{signature}, \mathsf{sid}, m, \sigma)$ from $\mathcal{S}$,
   i. If the tuple $(\mathsf{sid}, m, \sigma, 0)$ is recorded, output an error.
   ii. If a tuple $(\mathsf{sid}', \cdot, \sigma, 1)$ is recorded, and there is no set $P' \in \Gamma$ for which all $P_i \in P'$ are corrupted, then record $(\mathsf{sid}, m, \sigma, 0)$. (Uniqueness of signatures)
   iii. Else, record $(\mathsf{sid}, m, \sigma, 1)$.

   (d) Upon receiving $(\mathbf{signature}, \mathsf{sid}, m)$ from $P_i \in P$:
   i. If $(\mathsf{sid}, m, \sigma, 1)$ is recorded, output $(\mathbf{signature}, \mathsf{sid}, m, \sigma)$ to P_i.
   ii. Else ignore the message.
3. **Verification:**
   Upon receiving $(\mathbf{sig\text{-}verify}, \mathsf{sid}, m, \sigma, X)$ from a party $\mathcal{X}$, send $(\mathbf{sig\text{-}verify}, \mathsf{sid}, m, \sigma, X)$ to $\mathcal{S}$ and do:
   (a) If a tuple $(\mathsf{sid}, m, \sigma, \beta')$ is recorded, then set $\beta = \beta'$.
   (b) Else (m was never signed), if there is no set $P' \in \Gamma$ for which all $P_i \in P'$ are corrupted, set $\beta = 0$. (Unforgeability)
   (c) Else (m was never signed and there is a corrupted set $P' \in \Gamma$), set $\beta = \mathcal{V}(m, \sigma, X)$.
   Record (m, σ, β) and output $(\mathbf{istrue}, \mathsf{sid}, m, \sigma, \beta)$ to $\mathcal{X}$.

runs of Π_{keygen} that engage at least one honest party (not necessarily the same honest party in both of them). We note that this protocol is input-less, and depends only on the random tape of the parties, which, by examining the protocol description. Specifically, each honest party P_i picks an independent uniform share x_i on every protocol run. Consider the list of all x_i's that were ever generated by all honest parties P_i's, then with overwhelming probability this list has no repeating value. Now, the protocol guarantees an independence choice of the x_i's between *all parties* (even the corrupted ones) for a specific run, by the invocation of $\mathcal{F}_{\mathsf{com-zk}}$ in the first step. Thus, at the end of the first step the future resulting secret key is guaranteed to be uniformly distributed, or the protocol aborts, in which case no key is generated. Since the public key space is isomorphic to the secret key space, and the generated secret key at each run of Π_{keygen} involves at least one independent uniformly chosen share x_i then the resulting secret key (and hence public key) is unique.

2. Reaching the Uniqueness of signatures state is possible only in case the environment causes the parties in the real world to reach to the same signature, on different runs of $(\Pi_{\mathsf{pres}}, \Pi_{\mathsf{sign}})$ (potentially, on different session id's). Following the same argument above (but now referring to values k_i's in Protocol Π_{pres}), we get that this is only happening with negligible probability.
3. We argue that reaching the Unforgeability state is possible only in case the adversary managed to forge a signature for a message that was never signed. Suppose that an adversary $\mathcal{A}$ outputs a forgery in the real world (where executing Π, potentially concurrently many times, under the 'orchestration' of $\mathcal{Z}$), then, we construct another adversary $\mathcal{A}'$ that is given a public key X and works as follows: $\mathcal{A}'$ runs $\mathcal{A}$ and $\mathcal{Z}$ internally and do as follows. Whenever $\mathcal{Z}$ invokes Π_{keygen}, except one special time, our adversary $\mathcal{A}'$ simulates the honest parties and together with $\mathcal{A}$ they generate a public key. At the special time, $\mathcal{A}'$ uses the simulatability of Π to produce a key identical to that of its input, namely X. From that point and on, $\mathcal{A}'$ simulates the honest parties at all protocol instances (either key generation or sign). If $\mathcal{A}$ at any point outputs a valid signature under the X key for a message that was never signed, then $\mathcal{A}'$ uses it as its forgery. If $\mathcal{A}$ forges a signature with a non-negligible probability then, since $\mathcal{A}$ picks the instance of Π_{keygen} for which it injects X at random, and since the number of instances is polynomial in κ, and since the adversary cannot distinguish between the simulation of these instances, the probability that $\mathcal{A}$ forges a signature under X (and hence $\mathcal{A}'$ forges a signature under X as well) is non-negligible.

H Security 2PC-MPC

Following Theorem 2.4, it is sufficient to prove $\mathcal{G}^*$-simulatability for the $1+n$ protocol, where $\mathcal{G}^*$ is a signing oracle, and so we prove the following:

Theorem H.1 *$\Pi_{\mathsf{ecdsa}} = (\Pi_{\mathsf{kgen}}, \Pi_{\mathsf{pres}}, \Pi_{\mathsf{sign}})$ (Protocols 4, 5 and 6) is $\mathcal{G}^*$-simulatable (according to Definition 2.3) in the $(\mathcal{F}_{\mathsf{TAHE}}, \mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{agg-zk}}, \mathcal{F}_{\mathsf{com-zk}})$-hybrid and random oracle model, assuming Com is computationally binding and AHE is a semantically secure encryption scheme with secure function evaluation.*

We separate the proof into two cases: in the first case all B_i's are corrupted and A is honest, and in the second case A as well as $n-1$ of the B_i's are corrupted. We note that a similar simulation will go through for a weaker adversary, who corrupts less n or $n-1$ parties of B, respectively.

Case 1: A is Honest The formal simulation is presented in Simulations 15-16.

To show that the simulation is indistinguishable from the real protocol, we define two hybrids where the first hybrid coincides with the simulation (Figures 15-16) and the second hybrid coincides with the real execution. Namely:

1. **Hybrid1** is similar to the simulation in Figures 15-16, except that $\mathcal{S}$ emulates $\mathcal{G}^*$ locally (instead of using $\mathcal{G}^*$ as an oracle).
2. **Hybrid2** is similar to **Hybrid1** except that, in Step 3 in Simulation 16 the simulator $\mathcal{S}$ computes $\mathsf{ct}_A \leftarrow \mathsf{AHE.Eval}(pk, f_A, \mathsf{ct}_{\mathsf{key}})$ using the correct f_A and $\mathsf{ct}_{\mathsf{key}}$, where f_A's coefficients a_0, a_1 are determined by the correct values x_A and k_A. $\mathcal{S}$ can do this because now (when emulating $\mathcal{G}^*$ locally) it has access to these secrets. In that case, $\mathcal{S}$ knows x_A since it generates the secret signing key x and receives B's share when B calls $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{EncDL}}}$ in Step 3 in Simulation 15, then, it computes $x_A = x - x_B \mod q$; in addition $\mathcal{S}$ knows k_A since it generates the secret nonce k and receives B's share when B calls $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{DL}}}$ in Step 4(a)ii in Simulation 15, then $\mathcal{S}$ computes $k_A = k^{-1} \cdot k_B$.

Hybrid1 is identically-distributed to the simulation (in Figures 15-16) since the only difference is that in the simulation $\mathcal{G}^*$'s random tape is used to sample the secret key (which determines the public key) and the nonce, whereas in the hybrid this is done by the simulator. Nevertheless, in both cases these values are sampled from the exact same distribution. Note that $\mathcal{G}^*$ also produces signatures on messages of the adversary's choice, however, a signature is fully determined by the message, the secret key and the nonce.

Hybrid1 and **Hybrid2** are computationally indistinguishable. We reduce indistinguishability between the two hybrids to the secure function evaluation property (Definition B.5) of the AHE scheme: Let $\mathcal{A}$ be an adversary that distinguishes between **Hybrid1** and **Hybrid2** with probability ϵ. We construct an adversary $\mathcal{A}'$ that breaks the secure function evaluation property with the same probability as follows: Run **Hybrid1** above except of the following modification. In Step 3 under command Π_{sign} in Simulation 16, engage in Experiment 8 by handing $((pk, sk), \ell = 1, \{\mathsf{pt} = x_B, \eta\}, f(x) = f_A)$ to the experiment, where x_B, η and f_A's coefficients are extracted as explained above. Receive ciphertext $\mathsf{ct}_{\mathsf{Eval}}$ from the experiment and use $\mathsf{ct}_A = \mathsf{ct}_{\mathsf{Eval}}$ for the rest of the simulation. Run $\mathcal{A}$ with the resulting simulation and respond to the experiment with whatever bit that $\mathcal{A}$ outputs. Observe that if the experiment's bit is 0 then the resulting simulation is distributed identically to **Hybrid1** and if the bit is 1 then the resulting simulation is distributed identically to distributed **Hybrid2**. Therefore, $\mathcal{A}'$ breaks AHE's secure function evaluation with the same probability as $\mathcal{A}$ distinguishes between **Hybrid1** and **Hybrid2**, namely, with probability ϵ. Since AHE has secure function evaluation, this means that $\epsilon \in \mathsf{neg}(\kappa)$.

Finally, it is easy to see that **Hybrid2** is identical to the real execution, where $\mathcal{S}$ takes the role of the uncorrupted party A and runs $\mathcal{A}$ internally. Note that the real-execution is run in the $(\mathcal{F}_{\mathsf{TAHE}}, \mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{agg-zk}}, \mathcal{F}_{\mathsf{com-zk}})$-hybrid model, which are perfectly simulated. We stress that this hybrid is identically distributed to the real execution even though in Step 2 under command Π_{pres} in Simulation 15 the simulator commits to zero instead of the correct value, as Com is a perfectly hiding commitment scheme. □

H.1 Case 2: B_i ($i \in [n]$) is Honest

In this case we consider an adversary $\mathcal{A}$ who corrupts party A as well as $n-1$ parties in B. The simulator is presented in Simulations 17,18 and 19.

We define the following hybrids where **Hybrid1** coincides with the simulation (Figures 17,18 and 19) and **Hybrid7** coincides with the real execution. Note that in **Hybrid1** all the ciphertexts $\mathsf{ct}_{\mathsf{key}}, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{ct}_4$ are decrypted to zero. These values are gradually replaced with correct values as we transit from **Hybrid$\boldsymbol{i}$** to **Hybrid$\boldsymbol{i+1}$**. The correct values are known to the simulators in the hybrids as the simulator takes the role of $\mathcal{G}^*$ instead of using it as an oracle. This does not change the distribution of the values as the simulator samples the key and the presignature as well as computes the signature exactly the same as $\mathcal{G}^*$ would. In more detail:

1. **Hybrid1** is similar to the simulation in Figures 17,18 and 19, except that $\mathcal{S}$ emulates $\mathcal{G}^*$ locally (instead of using $\mathcal{G}^*$ as an oracle).
2. **Hybrid2** is similar to **Hybrid1** except that, $\mathcal{S}$ encrypts the correct value x_B instead of zero in Step 3c in Simulation 17. $\mathcal{S}$ can do this because it now that it emulates $\mathcal{G}^*$ locally, it has access to these secrets.
3. **Hybrid3** is similar to **Hybrid2** except that in Step 4(a)iii in Simulation 18, $\mathcal{S}$ picks its own $\gamma_i \in [0, q)$, computes $\mathsf{ct}_1^i = \mathsf{AHE.Enc}(\gamma_i; \eta^j_{\mathsf{mask}_1})$ and then outputs $\mathsf{ct}_1 = \bigoplus_{j\in[n]} \mathsf{ct}_1^j$ instead of an encryption of zero (recall that $\oplus$ is the homomorphic addition operation of the AHE scheme).
4. **Hybrid4** is similar to **Hybrid3** except that in Step 4(a)iii in Simulation 18, $\mathcal{S}$ computes $\mathsf{ct}_2^i = \mathsf{AHE.Eval}(pk, f_i, \mathsf{ct}_{\mathsf{key}}; \eta^i_{\mathsf{mask}_2})$ where $f_i(x) = \gamma_i \cdot x$, and then outputs $\mathsf{ct}_2 = \bigoplus_{j\in[n]} \mathsf{ct}_2^j$ instead of an encryption of zero.
5. **Hybrid5** is similar to **Hybrid4** except that in Step 4(a)iv in Simulation 18, $\mathcal{S}$ computes $\mathsf{ct}_2^i = \mathsf{AHE.Eval}(pk, f_i, \mathsf{ct}_{\mathsf{key}}; \eta^i_{\mathsf{mask}_2})$ where $f_i(x) = \gamma_i \cdot x$, and then outputs $\mathsf{ct}_2 = \bigoplus_{j\in[n]} \mathsf{ct}_2^j$ instead of an encryption of zero.
6. **Hybrid6** is similar to **Hybrid5** except that in Step 4(a)iv in Simulation 18, $\mathcal{S}$ computes $\mathsf{ct}_4^i = \mathsf{AHE.Eval}(pk, f_i', \mathsf{ct}_1; \eta^i_{\mathsf{mask}_4})$ where $f_i'(x) = k_i \cdot x$, and then outputs $\mathsf{ct}_4 = \bigoplus_{j\in[n]} \mathsf{ct}_4^j$ instead of an encryption of zero.
7. **Hybrid7** is similar to **Hybrid6** except that in Step 4b in Simulation 19, $\mathcal{S}$ computes $\mathsf{pt}_4 = \gamma k_B \mod q$ and $\mathsf{pt}_A = k_A \cdot \gamma(r \cdot x + m) \mod q$.

Hybrid1 and **Hybrid2** are computationally indistinguishable – distinguishing the two reduces to breaking the CPA-security of the encryption scheme AHE. Indeed, consider an adversary $\mathcal{A}'$ engaged in the CPA-security game of AHE. Adversary $\mathcal{A}'$ gets a random public key pk', sends two messages $m_0 = 0$ and

$m_1 \leftarrow \mathbb{Z}_q$ to the experiment, and receives back a ciphertext $\mathsf{ct}' \leftarrow \mathsf{AHE.Enc}(pk, m_b)$. Then, $\mathcal{A}'$ uses $\mathcal{S}$ from **Hybrid1** with the following changes:

1. Instead of generating new AHE key pair, it uses $pk = pk'$ and $\mathsf{ct}_{\mathsf{key}} = \mathsf{ct}'$ from the CPA-security game.
2. It uses $X_B = m_1 \cdot G$.

Then, if $b = 0$, the view of $\mathcal{A}'$ is identically distributed to **Hybrid1**, whereas if $b = 1$, the view of $\mathcal{A}'$ is identically distributed to **Hybrid2**. Therefore, any advantage of $\mathcal{A}'$ in distinguishing the to hybrids directly translates to an advantage in the CPA-security game. Since AHE is semantically secure (which is equivalent to CPA-security), we conclude that the distributions of the two hybrids are computationally indistinguishable.

Then for all $i = 2, \ldots, 6$ we have that **Hybrid*i*** is indistinguishable from **Hybrid*i* + 1** as distinguishing the two is reduced to the security of secure function evaluation property of the AHE scheme, similar to the reduction presented in the case of honest A above. Finally, it is easy to see that **Hybrid7** is identically distributed to the real execution of the protocol.

I The Two Party Protocol

We consider two parties, Alice and Bob, denoted by A and B respectively. Looking ahead, party B can be simulated by a set of n parties. Alice and Bob hold a common reference string $\mathsf{sid} \in \{0,1\}^*$ called the session identifier. Moreover, they have public identities $\mathsf{pid}_A, \mathsf{pid}_B$, respectively.

I.1 Key Generation

The key-generation protocol is formally described in Protocol 20. The parties run a protocol to obtain a random group element $X \in \mathbb{G}$ as the public key, for which they will hold an additive sharing $x_A + x_B = x$ where $X = x \cdot G$. This works by A choosing a random integer $x_A \leftarrow \mathbb{Z}_q$ and computing the group element $X_A = x_A \cdot G$. Party A then commits to X_A and the fact that it knows its discrete log x, using the $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$ functionality. Party B then samples its own share $x_B \leftarrow \mathbb{Z}_q$, computes $X_B = x_B \cdot G$ and sends X_B to A, along with a proof of knowledge of x_B. Next, party A verifies the proof (and aborts if verification fails), and then decommits to X_A and its proof of knowldege, which is verified by B. At this stage, both parties can compute and store the ECDSA verification key $X = X_A + X_B = (x_A + x_B) \cdot G = x \cdot G$.

In addition to the above steps, B generates its own AHE key-pair (pk, sk) and hand pk to A along with a proof of validity, which is done through the $\mathcal{F}^{L_{\mathsf{GenAHE}}}_{\mathsf{zk}}$ functionality. Using pk, party B encrypts its own secret key share, and sends the ciphertext $\mathsf{ct}_{\mathsf{key}}$ to A along with a proof that the ciphertext indeed hides its secret key share (which is the discrete log of X_B). If all proofs go through, the parties output their secret key shares, x_A, x_B and the public values X_A, X_B, pk and $\mathsf{ct}_{\mathsf{key}}$ (party B outputs its decryption key sk as well).

I.2 Presigning

The presigning protocol is formally described in Protocol 21. The goal of the protocol is to randomly choose the signature nonce k and have the parties hold their shares k_A and k_B. In the first message A commit to its random secret k_A and prove knowledge of it. In the second message B picks its random secret k_B and send $R_B = (k_B)^{-1} \cdot G$ to A, in addition to proving knowledge of the discrete log of R_B. Given R_B, party A can compute the value $R = R_B^{(k_A)^{-1}}$, which is used in the signature protocol.

I.3 Signing

The first message of A consists of a ciphertext ct_A that is computed by a secure evaluation of an affine function $f_A(x) = a_0 + a_1 \cdot x$ on $\mathsf{ct}_{\mathsf{key}}$, where $a_0 = rk_Ax_A + mk_A$ and $a_1 = rk_A$ are two coefficients known to A. The result is an encryption of $rk_Ax_A + mk_A + rk_Ax_B = k_A(rx + m)$. Party A sends B the values K_A and ct_A computed earlier, along with a proof that ct_A was evaluated correctly with respect to k_A, x_A, r, m and ct_key. That proof is separated to multiple statements, which are proven individually. The first statement is that K_A is a commitment of the discrete log of R_B with respect to the basis element R; namely, that $k_A \cdot R = R_B$, or alternatively that $k_A^{-1} \cdot R_B = R$, which means that $k^{-1} \cdot G = R$ as required. The second statement is that the value committed in U_A equals the value committed in K_A times the discrete log of X_A, namely $(k_A \cdot x_A)$. Finally, using the commitments K_A and U_A (to values k_A and k_Ax_A, resp.) and given $r = R|_{x-axis} \mod q$ and m, both parties can homomorphically compute the commitments C_0 and C_1 to the coefficients a_0 and a_1, respectively. The third statement is that ct_A is an encryption of $f_A(\mathsf{ct}_{\mathsf{key}})$ where C_0 and C_1 are the commitments to f_A's coefficients.

Apart from verifying the above proofs sent from party A, party B decrypt ct_A and obtains $\mathsf{pt} = k_A(m + rx)$. Party B can obtain the signature by multiplying pt with k_B modulo q.

I.4 Security of the Two-Party Protocol

Theorem 2.4 states that in order to prove UC security of the protocol one only has to prove that it is $\mathcal{G}^*$-simulatable, as that property guarantees that the adversary (the environment) cannot break the signature scheme's unforgeability even after an execution of (potentially multiple instances of) the protocol. In that case, there is a 'trivial' simulator in the UC model: The simulator $\mathcal{S}$ participates in the ideal world (with $\mathcal{F}_{\mathsf{tsig}}$), and simulates the real-world protocol $\Pi_{\mathsf{2ecdsa}} = (\Pi_{\mathsf{2kgen}}, \Pi_{\mathsf{2pres}}, \Pi_{\mathsf{2sign}})$ by running $\mathcal{A}$ internally, taking the role of all honest parties (and the $(\mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{com}})$ functionalities). Specifically:

- Upon receiving $(\mathbf{keygen}, \mathsf{ssid}, P_i)$ from all parties, $\mathcal{S}$ initiates Π_{2kgen} between the simulated honest parties and the real adversary $\mathcal{A}$, resulting with a key X. Then, upon receiving $(\mathbf{newkey}, \mathsf{ssid})$, $\mathcal{S}$ sends $(\mathbf{newkey}, \mathsf{ssid}, X, \mathcal{V})$ where

$\mathcal{V}$ is the verification algorithm for the signature in the real world (which is the ECDSA verification algorithm in our case). Then once all (not simulated) honest parties query $(\mathbf{newkey}, \mathsf{ssid})$, the functionality will transfer them $(\mathbf{newkey}, \mathsf{ssid}, X)$, and therefore the honest parties will output the same public key X. Therefore, the key generation simulation is perfect by the definition of $\mathcal{F}_{\mathsf{tsig}}$.

– Upon receiving $(\mathbf{sign}, \mathsf{sid}, m, P_i)$ from all parties, $\mathcal{S}$ initiates Π_{2pres} and then Π_{2sign}, again by simulating the honest parties and oracle access to $\mathcal{A}$, resulting with a signature σ. Upon receiving $(\mathbf{sign}, \mathsf{sid}, m)$, $\mathcal{S}$ responds with $(\mathbf{signature}, \mathsf{sid}, m, \sigma)$. Then, by definition, whenever an honest party queries $\mathcal{F}_{\mathsf{tsig}}$ for the signature, it gets $(\mathbf{signature}, \mathsf{sid}, m, \sigma)$. Therefore, again, the simulation is perfect, unless there was a record of $(\mathsf{sid}, m, \sigma, 0)$, in which case the functionality outputs an error. However, there are only two ways in which $(m, \sigma, 0)$ is recorded. In both cases, some party $\mathcal{X}$, potentially $\notin \mathbf{P}$, queried $(\mathbf{sig\text{-}verify}, \mathsf{sid}, m, \sigma, X)$ on a message m (note that only $(m, \sigma, 1)$ is recorded in the signing phase). In the first case, m was never signed, while not all parties in $\mathbf{P}$ are corrupted/quarantined. In this case, the functionality records $(m, \sigma, 0)$. The reason being that, if no signing query on m is recorded, and not all parties are corrupted, if the signature scheme is unforgeable, the signature must be invalid, and there is no reason to validate it. Nevertheless, if an adversary can forge a signature, this will result with a distinction between the real and ideal world. For the second case, all parties in $\mathbf{P}$ are corrupted, in which case the adversary knows the private key and is able to sign messages, therefore, they are validated with $\mathcal{V}$ and there will be no difference from the real world in this scenario.

Thus, the view of the environment $\mathcal{Z}$ in the ideal world (which is comprised of the view of $\mathcal{A}$ when run internally, and the outputs of honest parties in the ideal world, where $\mathcal{S}$ interacts with $\mathcal{F}_{\mathsf{tsig}}$) is identical to $\mathcal{Z}$'s view in the real world (which is comprised of $\mathcal{A}$'s view and the outputs of honest parties in the real world), unless $\mathcal{A}$ is able to forge signatures.

Therefore, theorem 2.4 states that it is only required to prove the following:

Theorem I.1 *$\Pi_{\mathsf{2ecdsa}} = (\Pi_{\mathsf{2kgen}}, \Pi_{\mathsf{2pres}}, \Pi_{\mathsf{2sign}})$ is $\mathcal{G}^*$-simulatable (Def. 2.3) in the $(\mathcal{F}_{TAHE}, \mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{com-zk}})$-hybrid and random oracle model.*

Proof. We will first simulate the two-party protocol in the $(\mathcal{F}_{\mathsf{TAHE}}, \mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{com-zk}}, \mathcal{G}^*, \mathcal{H})$-hybrid model. The formal description of the simulator $\mathcal{S}$ is given in Figures 23,24, simulating protocol against a corrupt A and B respectively. To show that the above simulation is indistinguishable from the real protocol, we define two hybrids where the first hybrid coincides with the simulation (Figures 23,24) and the second hybrid coincides with the real execution. Namely:

1. **Hybrid1** is similar to the simulation in Figures 23,24, except that $\mathcal{S}$ emulates $\mathcal{G}^*$ locally (instead of using $\mathcal{G}^*$ as an oracle).
2. **Hybrid2** is similar to **Hybrid1** except that, when A is corrupted, $\mathcal{S}$ encrypts the correct value x_B instead of zero in Step 4 of command Π_{2kgen}, in Simulation 23. $\mathcal{S}$ can do this because it previously got x_A and now (when emulating $\mathcal{G}^*$ locally) it has access to x and thus $x_B = x - x_A \mod q$.

3. **Hybrid3** is similar to **Hybrid2** except that, when B is corrupted, in Step 3 in Simulation 24 $\mathcal{S}$ computes $\mathsf{ct}_A \leftarrow \mathsf{AHE.Eval}(pk, f_A, \mathsf{ct}_{\mathsf{key}})$ using the correct f_A and $\mathsf{ct}_{\mathsf{key}}$, where f_A's coefficients a_0, a_1 are determined by the correct values x_A and k_A. $\mathcal{S}$ can do this because now (when emulating $\mathcal{G}^*$ locally) it has access to these secrets. In that case, $\mathcal{S}$ knows x_A since it generates the secret signing key x and receives B's share when B calls $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{EncDL}}}$ in Step 2 in Simulation 24, then, it computes $x_A = x - x_B \mod q$; in addition $\mathcal{S}$ knows k_A since it generates the secret nonce k and receives B's share when B calls $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DL}}[\mathbb{G}, R_B, q]}$ in Step 4 in Simulation 24, then $\mathcal{S}$ computes $k_A = k \cdot k_B^{-1}$.

Hybrid1 is identically-distributed to the simulation (in Figures 23,24) since the only difference is that in the simulation $\mathcal{G}^*$'s random tape is used to sample the secret key (which determines the public key) and the nonce, whereas in the hybrid this is done by the simulator. Nevertheless, in both cases these values are sampled from the exact same distribution. Note that $\mathcal{G}^*$ also produces signatures on messages of the adversary's choice, however, a signature is fully determined by the message, the secret key and the nonce.

Hybrid1 and **Hybrid2** are computationally indistinguishable in the case of a corrupted A and identically distributed in the case of a corrupted B. As for the former, distinguishing the two reduces to breaking the CPA-security of the encryption scheme AHE. Indeed, consider an adversary $\mathcal{A}'$ engaged in the CPA-security game of AHE. Adversary $\mathcal{A}'$ gets a random public key pk', sends two messages $m_0 = 0$ and $m_1 \leftarrow \mathbb{Z}_q$ to the experiment, and receives back a ciphertext $\mathsf{ct}' \leftarrow \mathsf{AHE.Enc}(pk, m_b)$. Then, $\mathcal{A}'$ uses $\mathcal{S}$ from Figure 23 (when A is corrupted) with the following changes:

1. In line 4 of Π_{2kgen}, it uses $pk = pk'$ and $\mathsf{ct}_{\mathsf{key}} = \mathsf{ct}'$ from the CPA-security game.
2. In line 5 of Π_{2kgen} it uses $X_B = m_1 \cdot G$.
3. It responds to all $\mathcal{S}$'s calls to $\mathcal{G}^*$ by perfectly simulating $\mathcal{G}^*$.

Then, if $b = 0$, the view of $\mathcal{A}'$ is identically distributed to **Hybrid1**, whereas if $b = 1$, the view of $\mathcal{A}'$ is identically distributed to **Hybrid2**. Therefore, any advantage of $\mathcal{A}'$ in distinguishing the to hybrids directly translates to an advantage in the CPA-security game. Since AHE is semantically secure (which is equivalent to CPA-security), we conclude that the distributions of the two hybrids are computationally indistinguishable.

Hybrid2 and **Hybrid3** are computationally indistinguishable in the case of a corrupted B and identically distributed in the case of a corrupted A. We can reduce indistinguishability of the two hybrids to the secure function evaluation property (Definition B.5) of the AHE scheme: Consider an $\mathcal{A}$ be an adversary that distinguishes between **Hybrid2** and **Hybrid3** with probability ϵ. We construct $\mathcal{A}'$ that breaks the secure function evaluation property with the same probability as follows: Run **Hybrid2** as the simulator $\mathcal{S}$, except that in Step 3 under command Π_{2sign} in Simulation 24, engage in Experiment 8 by handing $((pk, sk), \ell = 1, \{\mathsf{pt} = x_B, \eta\}, f(x) = f_A)$ to the experiment, where x_B and η are extracted by simulating $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{DL}}}$ and f_A's coefficients are extracted as explained above. Receive ciphertext $\mathsf{ct}_{\mathsf{Eval}}$ from the experiment and use $\mathsf{ct}_A = \mathsf{ct}_{\mathsf{Eval}}$. Run $\mathcal{A}$ with the resulting simulation and respond to the experiment with whatever bit that $\mathcal{A}$ outputs. Observe that if the experiment's bit is 0 then the resulting simulation is distributed identically to **Hybrid2** and if the bit is 1 then the resulting simulation is distributed identically to distributed **Hybrid3**. Therefore, $\mathcal{A}'$ breaks AHE's secure function evaluation with the same probability as $\mathcal{A}$ distinguishes between **Hybrid2** and **Hybrid3**, namely, with probability ϵ. Since AHE has secure function evaluation, this means that $\epsilon \in \mathsf{neg}(\kappa)$.

Finally, it is easy to see that **Hybrid3** is identical to the real execution, where $\mathcal{S}$ takes the role of the uncorrupted party and runs $\mathcal{A}$ internally. Note that the real-execution is run in the $(\mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{com}})$-hybrid model, hence, all zero-knowledge proofs and commitments are perfectly simulated. We stress that this hybrid is identically distributed to the real execution even though in Step 2 under command Π_{2pres} in Simulation 24 the simulator commits to zero instead of the correct value. This holds since we use a perfectly hiding commitment scheme Com.

J Schnorr Protocols

We adapt and modify the definition from [BMP22] and describe the general form of Schnorr protocols. We describe the protocol in a batched form, the non-batched form is derived by simply setting the batch size to $m = 1$. Hereafter, let $\phi : \mathbb{H} \to \mathbb{G}$ denote a group homomorphism from $(\mathbb{H}, +)$, the witness group, to $(\mathbb{G}, \cdot)$, the associated statement group, and $\boldsymbol{E} \subset \mathbb{Z}_{\geq 0}$ (zero and positive integers) denote the challenge space. It is assumed that (the description of) the tuple $(\phi, \mathbb{H}, \mathbb{G}, \boldsymbol{E})$ is efficiently generated by a probabilistic polynomial time algorithm given 1^κ, and that ϕ is efficiently computable as a function of 1^κ.

Definition J.1 *An m-batch Schnorr protocol Π for tuple $(\phi, \mathbb{H}, \mathbb{G}, \boldsymbol{E})$ consists of the following interactive process. For a common input $\vec{X} = (X_j)_{j=1}^m \in \mathbb{G}^m$ to both $\mathcal{P}$ and $\mathcal{V}$ and secret input $\vec{w} = (w_j)_{j=1}^m \in \mathbb{H}^m$ to $\mathcal{P}$:*

1. *$\mathcal{P}$ samples $\alpha \leftarrow \mathbb{H}$ and sends $A = \phi(\alpha)$ to the verifier.*
2. *$\mathcal{V}$ replies with $\vec{e} \leftarrow \boldsymbol{E}^m$.*
3. *$\mathcal{P}$ replies with $z = \alpha + \sum_{j=1}^m e_j \cdot w_j \in \mathbb{H}$, s.t. for $e \in \mathbb{Z}$ and $w \in \mathbb{H}$ we have $e \cdot w = \underbrace{w + \ldots + w}_{e \text{ times}}$.*
4. *$\mathcal{V}$ checks that $\phi(z) = A \prod_{j=1}^m X_j^{e_j} \in \mathbb{G}$ and $z \in \mathbb{H}$, and outputs 1 or 0 accordingly.*

For $\mu, \varepsilon \in \mathsf{poly}(\kappa)$, we may associate the following properties to a protocol:

- *μ-HVZK. If there exists $\{w_i\}_i^m$ such that $w_i \in \mathbb{H}$ and $X_i = \phi(w_i)$ for all $i \in [1, m]$, then the simulated transcript $(\vec{X}, A, \vec{e}, z)$, where $z \leftarrow \mathbb{H}$, $\vec{e} \leftarrow \boldsymbol{E}^m$, and $A = \phi(z) \prod_{j=1}^m X_j^{-e_j}$, is statistically μ-close to the honest transcript.*

– *ε-Knowledge soundness. There exists a set V such that:*
  - *Sampling $E \leftarrow \boldsymbol{E}^{m\times(m+1)}$, it holds that $\Pr[E \in V] \geq 1 - \mathsf{neg}(\kappa)$.*
  - *There exists a PPT extractor $\mathcal{E}$ such that given $(1^\kappa, \vec{X}, A, E, Z)$ where $E = (\vec{e}_1, \dots, \vec{e}_{m+1})$ and $Z = (\vec{z}_1, \dots, \vec{z}_{m+1})$ such that $\mathcal{V}(\vec{X}, A, \vec{e}_j, \vec{z}_j) = 1$ for all $j \in [1, m+1]$, and $E \in V$, then,*

$$\Pr\left[\phi(\vec{w}) = \vec{X}, \vec{w} \in \mathbb{H} \mid \vec{w} \leftarrow \mathcal{E}(1^\kappa, \vec{X}, A, E, Z)\right] \geq 1 - \varepsilon.$$

It is easy to see that completeness and honest-verifier zero-knowledge for the batched version are guaranteed as long as they are guaranteed for the non-batched version.

K Batched Zero-Knowledge Proof for L_{DComEval}

We apply the batched Schnorr protocol from Definition 4.1. The batched statement consists of m statements as in Eq. 7, namely, $(\mathsf{ct}_k^{\mathsf{Eval}}, \boldsymbol{C}_k; A_k, \rho_k, \eta_k) \in L_{\mathsf{DComEval}}[pk, \mathsf{pp}, \vec{\mathsf{ct}}]$ such that $\mathsf{ct}_k^{\mathsf{Eval}} = \vec{v}_\Delta \cdot A_k \odot \vec{c} \oplus \mathsf{AHE.Enc}(0; \eta_k)$, $\boldsymbol{C}_k = \mathsf{Com}_{\mathsf{pp}}(A_k; \rho_k)$, and $A_k \in [0, \Delta)^{d\times(\ell+1+\ell_\omega)}$, for each $1 \leq k \leq m$.

When batching m statements, the challenge space $\boldsymbol{E}$ is adjusted to be $[0, m \cdot 2^{\kappa+3})$. When $m = 1$, the knowledge extraction required two accepting transcripts, where $e \neq \hat{e}$, which happens with probability $2^{-\kappa}$. The analogous requirement for the batched protocol, would require an $(m+1) \times (m)$ matrix, where each row corresponds to a uniformly sampled challenge $\vec{e}_k \leftarrow \boldsymbol{E}^m$, augmented with a column of ones, to be invertible. This is the definition of the set V for ε-extractability. In Corollary K.2, we show that the probability that the determinant is zero is bounded by $\frac{2}{|\boldsymbol{E}|}$.

In addition, the statistical zero-knowledge parameter is also adjusted, and instead of 2^s we use $m \cdot 2^s$. Therefore, the set $\boldsymbol{S}$ is adjusted to be $[0, 2\Delta m^2 \cdot d(\ell + 1 + \ell_\omega)2^{\kappa+s})^{d\times(\ell+1+\ell_\omega)} \times \mathcal{R}_{\mathsf{pp}} \times \mathcal{R}_{pk}$. This comes from the fact that the batched protocol is essentially a sum of m statements, therefore the statistical distance increases by a factor of m which we cancel out by adjusting s. Note however that the statistical distance of an m-tuple of non-batched protocols from an m-tuple of simulated protocols is also bounded by $m \cdot 2^s$.

As a result, we now require $q > \Delta m \cdot (d(\ell + 1 + \ell_\omega) \cdot 2^{\kappa+s} + 2^\kappa)$. The protocol proceeds as follows:

1. $\mathcal{P}$ samples

$$B \in [0, 4\Delta \cdot m \cdot d(\ell + 1 + \ell_\omega) \cdot 2^{\kappa+s})^{d\times(\ell+1+\ell_\omega)},\ \nu \in \mathcal{R}_{\mathsf{pp}},\ \xi \in \mathcal{R}^*_{pk},$$

   computes
   - $\mathsf{ct}_U = \vec{v}_\Delta \cdot B \odot \vec{c} \oplus \mathsf{Enc}(0; \xi)$,
   - $\boldsymbol{C}_U = \mathsf{Com}_{\mathsf{pp}}(B; \nu)$,

   and sends
   - $(\mathbf{prove}, \boldsymbol{C}; A, \rho)$ to $\mathcal{F}_{\mathsf{zk-batch}}^{L_{\mathsf{Range}}[\mathsf{pp}, 0, \Delta]}$, and
   - $(\mathsf{ct}_U, \boldsymbol{C}_U)$ to $\mathcal{V}$.

2. Upon receiving $(\mathsf{ct}_U, \boldsymbol{C}_U)$ from $\mathcal{P}$ and $(\mathbf{proof}, \boldsymbol{C})$ from $\mathcal{F}_{\mathsf{zk-batch}}^{L_{\mathsf{Range}}[\mathsf{pp},0,\lceil q\rceil]}$ (with the same $\boldsymbol{C}$ in both messages), $\mathcal{V}$ sends random challenges $e_1, \ldots, e_m \leftarrow [0, 2^\kappa)$ to $\mathcal{P}$.
3. $\mathcal{P}$ responds with $(Z, z^{\mathsf{Eval}}, z^{\mathsf{Com}})$, where:
   - $Z = B + \sum_k e_k \cdot A_k \in \mathbb{Z}^{d\times(\ell+1+\ell_\omega)}$,
   - $z^{\mathsf{Eval}} = \xi + \sum_k e_k \eta_k \in \mathcal{R}_{pk}$, and
   - $z^{\mathsf{Com}} = \nu + \sum_k e_k \rho_k \in \mathcal{R}_{\mathsf{pp}}$.
4. $\mathcal{V}$ verifies the following equations:

$$\vec{v}_\Delta \cdot Z \odot \vec{c} \oplus \mathsf{Enc}(0; z^{\mathsf{Eval}}) = \mathsf{ct}_U \oplus \left(\bigoplus_k e_k \odot \mathsf{ct}_k^{\mathsf{Eval}} \right) \tag{24}$$

$$\mathsf{Com}(Z; z^{\mathsf{Com}}) = \boldsymbol{C}_U \oplus \left(\bigoplus_k e_k \odot \boldsymbol{C}_k \right) \tag{25}$$

   In addition, $\mathcal{V}$ verifies that:
   - $z^{\mathsf{Com}} \in \mathcal{R}_{\mathsf{pp}}$, $z^{\mathsf{Eval}} \in \mathcal{R}_{pk}$, $\boldsymbol{C}, \boldsymbol{C}_U \in \mathcal{C}_{\mathsf{pp}}$, $\vec{\mathsf{ct}}, \mathsf{ct}^{\mathsf{Eval}}, \mathsf{ct}_U \in \mathcal{C}_{pk}$, and
   - $Z \in [0, \Delta m \cdot (d(\ell+1+\ell_\omega) \cdot 2^{\kappa+s} + 2^\kappa))^{d\times(\ell+1+\ell_\omega)}$.

Completeness. This follows directly from the completeness of the underlying protocol with $m = 1$, the homomorphic properties of $\mathsf{AHE.Enc}$ and Com, and the adjusted range check for Z.

HVZK. Denote $\sigma_{\bar{k}} = B + \sum_{k=1}^{\bar{k}} e_k \cdot A_k$, where B is sampled from $[0, \Delta \cdot m \cdot d(\ell+1+\ell_\omega) \cdot 2^{\kappa+s})^{d\times(\ell+1+\ell_\omega)}$, and e_k is sampled uniformly from $[0, 2^\kappa)$, as in the real execution of the batched protocol. Then σ_0 is the simulated transcript, σ_m is the real execution, and σ_1 is the real execution for a single statement. Then by the HVZK property of the underlying protocol, $\mathsf{SD}[\sigma_0, \sigma_1] \leq \frac{1}{m \cdot 2^s}$. Therefore, by induction:

$$\begin{aligned}\mathsf{SD}[\sigma_0, \sigma_m] &= \mathsf{SD}[\sigma_0, \sigma_{m-1} + e_m \cdot A_m] \leq \mathsf{SD}[\sigma_0, \sigma_{m-1}] + \mathsf{SD}[\sigma_0, \sigma_0 + e_m \cdot A_m] \\ &\leq (m-1)\frac{1}{m \cdot 2^s} + \frac{1}{m \cdot 2^s} = 2^{-s}\end{aligned}$$

In the following we demonstrate that knowledge soundness is implied by the knowledge soundness of the non-batched proof system.

Knowledge soundness. Define V such that $E^T = \left[\vec{e}_1 \ldots \vec{e}_{m+1}\right]^T \in V$ if and only if $\vec{e}_k \in \boldsymbol{E}^m$ and

$$\det\left(\begin{bmatrix} \vec{e}_1 \| 1 \\ \vdots \\ \vec{e}_{m+1} \| 1 \end{bmatrix}\right) \neq 0 \mod q, p_1, p_2$$

where $N = p_1 p_2$ is the Paillier modulus. By Corollary K.2, $\Pr[E \in V | E \leftarrow \boldsymbol{E}^{(m+1)\times m}] \leq 2^{-\kappa+1} \cdot 3$ (by union bound on the primes q, p_1, p_2).

Next, we construct a PPT extractor $\mathcal{E}$ such that given $m+1$ accepting transcripts $(\mathsf{ct}_U, \boldsymbol{C}_U, \vec{e}_k, Z_k, z_k^{\mathsf{Eval}}, z_k^{\mathsf{Com}})$ for $1 \leq k \leq m$ such that $\left[\vec{e}_1 \ldots \vec{e}_{m+1}\right]^T \in V$, and $(\textbf{proof}, A, \rho)$ such that $\mathsf{Com}(A; \rho) = \boldsymbol{C}$, outputs $(A_k, \rho_k, \eta_k)_{k=1}^m$ such that $(\mathsf{ct}_k^{\mathsf{Eval}}, \boldsymbol{C}_k; A_k, \rho_k, \eta_k) \in L_{\mathsf{DComEval}}[pk, \mathsf{pp}, \vec{\mathsf{ct}}]$ for all $1 \leq k \leq m$.

$\mathcal{E}$ works as follows. It sets the matrix $E = \begin{bmatrix} \vec{e}_1 \| 1 \\ \vdots \\ \vec{e}_{m+1} \| 1 \end{bmatrix} \in [0, 2^\kappa)^{(m+1)\times(m+1)}$. Then for each session $1 \leq k \leq m+1$ it holds that:

$$\mathsf{Com}_{\mathsf{pp}}(Z_k; z_k^{\mathsf{Com}}) = \left(\bigoplus_{k'=1}^{m} e_{k,k'} \odot \boldsymbol{C}_{k'} \right) \oplus \boldsymbol{C}_U \tag{26}$$

Let $\partial E = \begin{bmatrix} \vec{e}_1 - \vec{e}_{m+1} \\ \vdots \\ \vec{e}_m - \vec{e}_{m+1} \end{bmatrix} \in \boldsymbol{E}^{m \times m}$, $\partial Z_k = Z_k - Z_{m+1}$ and $\partial z_k^{\mathsf{Com}} = z_k^{\mathsf{Com}} - z_{m+1}^{\mathsf{Com}}$ for all $1 \leq k \leq m$. Also, denote by $\partial e_{k,k'} := e_{k,k'} - e_{m+1,k'}$ the (k,k') entry of ∂E. By subtracting Eq. 26 where $k = m+1$ from Eq. 26 for any other $1 \leq k \leq m$, we get

$$\mathsf{Com}_{\mathsf{pp}}(\partial Z_k; \partial z_k^{\mathsf{Com}}) = \bigoplus_{k'=1}^{m} \partial e_{k,k'} \odot \boldsymbol{C}_{k'} \tag{27}$$

We can write this system of equations in a matrix-form as follows:

$$\begin{bmatrix} \mathsf{Com}_{\mathsf{pp}}(\partial Z_1; \partial z_1^{\mathsf{Com}}) \\ \vdots \\ \mathsf{Com}_{\mathsf{pp}}(\partial Z_m; \partial z_m^{\mathsf{Com}}) \end{bmatrix} = \partial E \odot \begin{bmatrix} \boldsymbol{C}_1 \\ \vdots \\ \boldsymbol{C}_m \end{bmatrix} \tag{28}$$

where the RHS is a matrix-vector product. Since $E \in V$ and $\det(\partial E) = \det(E)$, we have that $\det(E) \neq 0$ modulo q and N. Thus, we can compute $(\partial E)^{-1}$ in $\mathbb{Z}_q$ such that $(\partial E)^{-1} \cdot \partial E = \mathsf{id}$. By multiplying $(\partial E)^{-1}$ on both sides of Eq. 28 we get

$$\text{LHS} = (\partial E)^{-1} \odot \begin{bmatrix} \mathsf{Com}_{\mathsf{pp}}(\partial Z_1; \partial z_1^{\mathsf{Com}}) \\ \vdots \\ \mathsf{Com}_{\mathsf{pp}}(\partial Z_m; \partial z_m^{\mathsf{Com}}) \end{bmatrix} := \begin{bmatrix} \mathsf{Com}_{\mathsf{pp}}(A'_1; \rho'_1) \\ \vdots \\ \mathsf{Com}_{\mathsf{pp}}(A'_m; \rho'_m) \end{bmatrix}$$

$$\text{RHS} = (\partial E)^{-1} \cdot \partial E \odot \begin{bmatrix} \boldsymbol{C}_1 \\ \vdots \\ \boldsymbol{C}_m \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_1 \\ \vdots \\ \boldsymbol{C}_m \end{bmatrix}$$

and so we get that

$$\begin{bmatrix} A'_1 \\ \vdots \\ A'_m \end{bmatrix} := (\partial E)^{-1} \cdot \begin{bmatrix} \partial Z_1 \\ \vdots \\ \partial Z_m \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \rho'_1 \\ \vdots \\ \rho'_m \end{bmatrix} := (\partial E)^{-1} \cdot \begin{bmatrix} \partial z_1^{\mathsf{Com}} \\ \vdots \\ \partial z_m^{\mathsf{Com}} \end{bmatrix}$$

are vectors, whose k^{th} entries correspond to a valid decommitment for $\boldsymbol{C}_k$, for all $1 \le k \le m$. Therefore, it must hold that $A'_k = A_k$ for each $1 \le k \le m$ (for A_k extracted from the range proof), otherwise one could use $\mathcal{E}$ to break the binding property of Com.

Multiplying by ∂E, we get $\partial E \cdot A_k = \partial Z_k \mod q$. Therefore, there exist some $I_k \in \mathbb{Z}^{d\times(\ell+1+\ell_\omega)}$ such that

$$q \cdot I_k = \partial E \cdot A_k - \partial Z_k$$

holds over the integers. Next, take the $\|\cdot\|_\infty$ norm to bound the the absolute value of entries on the RHS, and use the triangular inequality to get:

$$\|\partial E{\cdot}A_k{-}\partial Z_k\|_\infty \le \Delta m{\cdot}2^\kappa{+}\Delta{\cdot}m{\cdot}(d(\ell{+}1{+}\ell_\omega)2^{\kappa+s}) = \Delta m{\cdot}(d(\ell{+}1{+}\ell_\omega){\cdot}2^{\kappa+s}{+}2^\kappa) < q$$

By picking q to be that large, we conclude that $\partial E \cdot A_k = \partial Z_k$ holds over the integers, for all $1 \le k \le m$. It therefore also holds modulo N, which enables us to extract the η_k's in what follows, for the Paillier encryption.

Since all $m{+}1$ transcript are accepting, we know that for every $1 \le k \le m{+}1$:

$$\vec{v}_\Delta \cdot Z_k \odot \vec{c} \oplus \mathsf{Enc}(0; z_k^{\mathsf{Eval}}) = \left(\bigoplus_{k'=1}^{m} e_{k,k'} \odot \mathsf{ct}_{k'}\right) \oplus \mathsf{ct}_U$$

Subtracting ($\ominus$) Eq. 24 for $k = m+1$ from all other equations in 24, we get:

$$\vec{v}_\Delta \cdot \partial Z_k \odot \vec{c} \oplus \mathsf{Enc}(0; \partial z_k^{\mathsf{Eval}}) = \bigoplus_{k'=1}^{m} \partial e_{k,k'} \odot \mathsf{ct}_{k'}^{\mathsf{Eval}} \tag{29}$$

where $\partial z_k^{\mathsf{Eval}} = z_k^{\mathsf{Eval}} - z_{m+1}^{\mathsf{Eval}}$.

Now denote $\mathsf{ct}_k^0 := \vec{v}_\Delta \cdot A_k \odot \vec{c}$, and subtract the following:

$$\begin{aligned}\bigoplus_{k'=1}^{m} \partial e_{k,k'} \odot \mathsf{ct}_{k'}^0 &= \bigoplus_{k'=1}^{m} \partial e_{k,k'} \odot (\vec{v}_\Delta \cdot A_k \odot \vec{c}) = \bigoplus_{k'=1}^{m} \vec{v}_\Delta \cdot (\partial e_{k,k'} \cdot A_k) \odot \vec{c} \\ &= \vec{v}_\Delta \cdot \left(\bigoplus_{k'=1}^{m} \partial e_{k,k'} \cdot A_k\right) \odot \vec{c} = \vec{v}_\Delta \cdot \partial Z_k \odot \vec{c}\end{aligned}$$

(since $\partial E \cdot \begin{bmatrix} A_1 \\ \vdots \\ A_m \end{bmatrix} = \begin{bmatrix} \partial Z_1 \\ \vdots \\ \partial Z_m \end{bmatrix}$ is established above over the integers) from both sides of Eq. 29 and get:

$$\mathsf{Enc}(pk, 0; \partial z_k^{\mathsf{Eval}}) = \bigoplus_{k'=1}^{m} \partial e_{k,k'} \odot (\mathsf{ct}_{k'}^{\mathsf{Eval}} - \mathsf{ct}_{k'}^0)$$

In matrix notation, we get:

$$\begin{bmatrix}\mathsf{Enc}(pk,0;\partial z_1^{\mathsf{Eval}})\\ \vdots \\ \mathsf{Enc}(pk,0;\partial z_m^{\mathsf{Eval}})\end{bmatrix} = \partial E \odot \begin{bmatrix}\mathsf{ct}_1^{\mathsf{Eval}} - \mathsf{ct}_1^0 \\ \vdots \\ \mathsf{ct}_m^{\mathsf{Eval}} - \mathsf{ct}_m^0\end{bmatrix}$$

Now, each equation involves multiple ciphertexts, so we need another step in order to use Lemma 5.1. We cannot compute an inverse for ∂E over the ciphertext domain since it is of unknown order, and computing such value is equivalent to factorization. However, to use the lemma, it is sufficient to compute a matrix F such that $F \cdot \partial E = \delta \cdot \mathsf{id}$, meaning, it is enough to get a scalar matrix (even a diagonal matrix actually). This can be done using Gaussian elimination over the integers without division, or alternatively, we can use $(\partial E)^{-1} = \frac{\mathsf{adj}(\partial E)}{\det(\partial E)}$ (over $\mathbb{Q}$), and set $F = \mathsf{adj}(\partial E)$ and $\delta = \det(\partial E)$. Note that $\delta \neq 0$ in $\mathbb{Z}_{p_1}$ nor in $\mathbb{Z}_{p_2}$ (where $N = p_1 p_2$), since $E \in V$. Also note that $\mathsf{adj}(\partial E)$ and $\det(\partial E)$ can be computed in polynomial time over the integers. Therefore,

$$\begin{bmatrix}\mathsf{Enc}(pk,0;\mathsf{adj}(\partial E)\cdot\partial z_1^{\mathsf{Eval}})\\ \vdots \\ \mathsf{Enc}(pk,0;\mathsf{adj}(\partial E)\cdot\partial z_m^{\mathsf{Eval}})\end{bmatrix} = \det(\partial E) \odot \begin{bmatrix}\mathsf{ct}_1^{\mathsf{Eval}} - \mathsf{ct}_1^0 \\ \vdots \\ \mathsf{ct}_m^{\mathsf{Eval}} - \mathsf{ct}_m^0\end{bmatrix}$$

We may now use Lemma 5.1 to extract the all randomizers η_k, and get $\mathsf{ct}_k^{\mathsf{Eval}} = \vec{v}_\Delta \cdot A_k \odot \vec{c} + \mathsf{Enc}(0;\eta_k)$.

Lemma K.1 *Let $k_0, m \in \mathbb{N}$ s.t. $k_0 < m$, and let $\vec{e}_1,\ldots,\vec{e}_{k_0} \in [0,2^\kappa)^{m+1}$ be linearly independent over $\mathbb{Z}_q$, then the probability that $\vec{e}_1,\ldots,\vec{e}_{k_0+1}$ are linearly independent over $\mathbb{Z}_{\hat{q}}$ where $\vec{e}_{k_0+1}$ is uniformly chosen from $[0,2^\kappa)^{m+1}$ with $\vec{e}_{k_0+1,m+1} = 1$ is at least $1 - \frac{1}{2^{\kappa(m+1-k_0)}}$.*

Proof. We bound the probability that $\vec{e}_{k_0+1} \in \text{span}(\vec{e}_1,\ldots,\vec{e}_{k_0})$. Consider $\langle \vec{e}_1,\ldots,\vec{e}_{k_0}\rangle^\perp$, the linear subspace of all vectors perpendicular to $\vec{e}_i$ for all $i \in \{1,\ldots,k_0\}$. Let $\{\vec{f'}_{k_0+1},\ldots,\vec{f'}_{m+1}\}$ be a Gaussian-eliminated basis of $\langle \vec{e}_1,\ldots,\vec{e}_{k_0}\rangle^\perp$. A vector is in $\text{span}(\vec{e}_1,\ldots,\vec{e}_{k_0})$ if and only if it is perpendicular to $\vec{f'}_i$ for each $i \in \{k_0+1,\ldots,m+1\}$. The challenge is that the distribution of $\vec{e}_{k_0+1}$ is over $[0,2^\kappa)^{m+1}$, and not the whole space $\mathbb{Z}_q^{m+1}$ (in which case the probability is exactly $\frac{1}{q^{m+1-k_0}}$). Let pv_i be the first non-zero entry of $\vec{f'})_i$, so $\mathsf{pv}_{k_0+1} > \ldots > \mathsf{pv}_{m+1}$. W.l.o.g, we will assume that $\mathsf{pv}_i = i$, for convenience.

Let $\vec{u}$ be a vector from $[0,2^\kappa)^{m+1}$ such that $\vec{u} \perp \vec{f'}_i$ for each $i \in \{k_0+1,\ldots,m+1\}$. Let

$$S_{\vec{u}} = \{\vec{u}' = (u'_0,\ldots,u'_{m+1}) \in [0,2^\kappa)^{m+1} \mid \forall\, 1 \le i \le k_0 : u'_i = u_i \wedge \\ \exists\, k_0+1 \le i \le m+1 : u'_i \neq u_i\}$$

Then observe that $|S_{\vec{u}}| = 2^{\kappa(m+1-k_0)} - 1$, and that for any $\vec{u'} \in S_{\vec{u}}$, $\vec{u'} \notin \mathsf{span}(\vec{e}_1,\ldots,\vec{e}_{k_0})$.Indeed, there exist some pivot entry $i > k_0$ such that $u'_i \neq u_i$.

Therefore, $\langle \vec{u'}, \vec{f'}_i \rangle = \langle \vec{u}, \vec{f'}_i \rangle + (u'_i - u_i) \cdot 1 \neq 0$. Indeed, $\vec{u} \perp \vec{f'}_i$, and $\vec{u}, \vec{u'}$ only differ in the last $m + 1 - k_0$ entries. However $f'_j = 0$ for $i \neq j > k_0$ because $\vec{f'}$ is after Gaussian elimination. Finally, since q is prime, $u'_i - u_i \neq 0$ and $f'_i \neq 0$, the product is also non-zero, as desired.

Therefore, for every choice of non-pivot coordinates ($i \leq k_0$) in $[0, 2^\kappa)$, the probability over the choice of pivot coordinates ($i > k_0$) that $\vec{u} \perp \vec{f'}_i$ for all $i \in \{k_0 + 1, \dots, m + 1\}$ is bounded by $\frac{1}{2^{\kappa(m+1-k_0)}}$. Therefore:

$$\begin{aligned}
&\Pr\left[\vec{e}_{k_0+1} \in \mathsf{span}(\vec{e}_1, \dots, \vec{e}_{k_0})\right] = \Pr[\forall_{i=k_0+1}^{m+1} \vec{e}_{k_0+1} \perp \vec{f'}_i] \\
&= \sum_{\{v_1,\dots,v_{k_0}\} \in [0,2^\kappa)} \Pr[\forall i \leq k_0 : e_{k_0+1,i} = u_i] \cdot \Pr[\forall i > k_0 : \vec{e}_{k_0+1} \perp \vec{f'}_i \mid \forall i \leq k_0 : e_{k_0+1,i} = u_i] \\
&\leq \sum_{\{v_1,\dots,v_{k_0}\} \in [0,2^\kappa)} \Pr[\forall i \leq k_0 : e_{k_0+1,i} = u_i] \cdot \frac{1}{2^{\kappa(m+1-k_0)}} = \frac{1}{2^{\kappa(m+1-k_0)}}
\end{aligned}$$

Corollary K.2 *For uniformly chosen matrix $E \in [0, 2^\kappa)^{(m+1)\times(m+1)}$, such that the first column is all ones,* $\Pr[\det(E) \neq 0] =$*.*

Proof. Denote the columns of E by $\vec{e}_1, \dots, \vec{e}_{m+1}$, then $\det(E) \neq 0$ iff $\vec{e}_1, \dots, \vec{e}_{m+1}$ are linearly independent. Let $\mathsf{L}_{\mathsf{k}_0}$ for $1 \leq k_0 \leq m + 1$ be the event that $\vec{e}_{k_0} \notin \mathrm{span}(\vec{e}_1, \dots, \vec{e}_{k_0-1})$, then by Lemma K.1:

$$\begin{aligned}
\Pr[\det(E) \neq 0] &= \Pr[L_1] \cdot \Pr[L_2 \mid L_1] \cdot \Pr[L_3 \mid L_1 \wedge L_2] \cdot \ldots \cdot \Pr[L_m \mid L_1 \wedge \ldots \wedge L_{m+1}] \\
&\geq 1 \cdot \left(1 - \frac{1}{2^{\kappa m}}\right) \cdot \ldots \cdot \left(1 - \frac{1}{2^\kappa}\right) \geq 1 - \frac{1}{2^\kappa} \cdot \frac{1}{1 - 2^{-\kappa}} \geq 1 - \frac{1}{2^{\kappa-1}}
\end{aligned}$$

L Instantiating $\mathcal{F}_{\mathsf{zk}}$ for the Remaining Languages

Below we specify the parameters for the other Schnorr based languages:

– *Knowledge of the discrete log of an elliptic-curve point.*

$$L_{\mathsf{DL}}[(\mathbb{G}, G, q)] = \{(P; x) \mid P = x \cdot G\}$$

The homomorphism is $\phi : \mathbb{H} = \mathbb{Z}_q \to \mathbb{G}, x \mapsto x \cdot G$.

– *Knowledge of decommitment.*

$$L_{\mathsf{DCom}}[\mathsf{pp}] = \{C; m, \rho \mid C = \mathsf{Com}(m; \rho) \wedge m \in \mathcal{M}_{\mathsf{pp}} \wedge \rho \in \mathcal{R}_{\mathsf{pp}} \wedge C \in \mathcal{C}_{\mathsf{pp}}\}$$

The homomorphism is $\phi : \mathbb{H} = \mathcal{M}_{\mathsf{pp}} \times \mathcal{R}_{\mathsf{pp}} \to \mathcal{C}_{\mathsf{pp}}, (m, \rho) \mapsto \mathsf{Com}(m; \rho)$.

– *AHE encryption of a discrete log.*

$$L_{\mathsf{EncDL}}[pk, (\mathbb{G}, G, q)] = \{(ct, X; x, \eta) \mid ct = \mathsf{AHE.Enc}(pk, x; \eta) \wedge X = x \cdot G \wedge x \in [0, \lceil q \rceil)\}$$

To prove this language, we enhanced it with a homomorphic commitment on x. The homomorphism is $\phi : \mathbb{H} = \mathbb{Z} \times \mathcal{R}_{pk} \times \mathcal{R}_{\mathsf{pp}} \to \mathcal{C}_{pk} \times \mathbb{G} \times \mathcal{C}_{\mathsf{pp}}, (x, \eta, \rho) \mapsto (\mathsf{AHE.Enc}(x; \eta), x \cdot G, \mathsf{Com}(x; \rho))$. In addition, $\boldsymbol{R} = [0, q) \times \mathcal{R}_{pk}$, and $\boldsymbol{S} = [0, q2^s) \times \mathcal{R}_{pk}$.

– *AHE encryption of a tuple.*

$$L_{\mathsf{EncDH}}[pk, \mathsf{ct}_x] = \{(\mathsf{ct}_y, \mathsf{ct}_z; y, \eta_y, \eta_z) \mid \mathsf{ct}_y = \mathsf{AHE.Enc}(pk, y; \eta_y) \wedge \mathsf{ct}_z = \mathsf{AHE.Eval}(pk, f, \mathsf{ct}_x; \eta_z) \text{ s.t. } f(x) = y \cdot x \mod q\} \tag{30}$$

The homomorphism is $\phi : \mathbb{H} = \mathbb{Z} \times \mathcal{R}_{pk}^2 \times \mathcal{R}_{\mathsf{pp}} \to \mathcal{C}_{pk}^2 \times \mathcal{C}_{\mathsf{pp}}, (x, \eta_y, \eta_z, \rho) \mapsto (\mathsf{AHE.Enc}(y; \eta_y), \mathsf{AHE.Eval}(pk, f(x) := y \cdot x, \mathsf{ct}_x; 0, \eta_z), \mathsf{Com}(y; \rho))$. In addition, $\boldsymbol{R} = [0, q) \times \mathcal{R}_{pk}^2 \times \mathcal{R}_{\mathsf{pp}}$ and $\boldsymbol{S} = [0, q2^s) \times \mathcal{R}_{pk}^2 \times \mathcal{R}_{\mathsf{pp}}$.

The languages L_{DComDL} and $L_{\mathsf{DComRatio}}$ cannot be directly derived, since they articulate relations between the coefficients $\vec{a}$ of f. Nevertheless, these are typical Schnorr arguments for which you can find proofs in e.g. [Ban05, Mau15]:

– *Commitment of discrete log.*

$$L_{\mathsf{DComDL}}[\mathsf{pp}, (\mathbb{G}, G, q)] = \{C, Y; m, \rho \mid C = \mathsf{Com}_{\mathsf{pp}}(m; \rho) \wedge Y = m \cdot G\}$$

The homomorphism is $\phi : \mathbb{H} = \mathcal{M}_{\mathsf{pp}} \times \mathcal{R}_{\mathsf{pp}} \to \mathcal{C}_{\mathsf{pp}} \times \mathbb{G}, (m, \rho) \mapsto (\mathsf{Com}_{\mathsf{pp}}(m; \rho), m \cdot G)$.

– *Ratio between committed values is the discrete log.* (Recall that $\mathcal{M}_{\mathsf{pp}} = \mathbb{Z}_q$.)

$$L_{\mathsf{DComRatio}}[\mathsf{pp}, (\mathbb{G}, G, q)] = \{C_1, C_2, X; m, x, \rho_1, \rho_2 \mid X = x \cdot G \wedge C_1 = \mathsf{Com}_{\mathsf{pp}}(m, \rho_1) \wedge C_2 = \mathsf{Com}_{\mathsf{pp}}(x \cdot m, \rho_2)\}$$

To prove this language, we do a little trick. We use Pedersen commitments:

$$\mathsf{Com}_{\mathsf{pp}}(m; \rho) := \mathsf{Ped.Com}_{\mathbb{G}, G, H, q}(m, \rho) = m \cdot G + \rho \cdot H$$

Therefore:

$$C_2 = \mathsf{Ped.Com}_{\mathbb{G}, G, H, q}(x \cdot m, \rho_2) = \mathsf{Ped.Com}_{\mathbb{G}, X, H, q}(m, \rho_2)$$

Hence, equivalently, we prove relation to the following language:

$$L_{\mathsf{DComRatio}}[\mathsf{pp}, X] = \{C_1, C_2; m, \rho_1, \rho_2 \mid C_1 = \mathsf{Com}_{G,H}(m, \rho_1) \wedge C_2 = \mathsf{Com}_{X,H}(m, \rho_2)\}$$

The homomorphism is $\phi : \mathbb{H} = \mathbb{Z}_q^3 \to \mathbb{G}^2, (m, \rho_1, \rho_2) \mapsto (\mathsf{Com}_{G,H}(m; \rho_1), \mathsf{Com}_{X,H}(m; \rho_2))$. Note that knowledge of x can be proven separately.

We also note that in L_{DComEval} used in our protocol, we build on Com to specifically be a Pedersen commitment over the ECDSA group $\mathbb{G}$, with the generator G being also the message base for the commitment. However, as all other languages, we can enhance L_{DComEval} with an additional homomorphic commitment, on which the range proofs are proven. To simplify the presentation of the proof technique, we chose not to do it in Section 5.2.

M Three Rounds Threshold ECDSA

As mentioned, the key generation and signing protocol remain intact. In what follows, we present the 2-round presign protocol, its simulation, and prove indistinguishability.

M.1 2-Round Presign

As mentioned earlier, we don't aggregate the proofs. Instead, each party sends its own proof and validates the proofs from all other parties. Moreover, in the first round, each party commits to $k_i \cdot G, \mathsf{Enc}(k_i)$, but already starts to work on encrypted values, and sends $\mathsf{Enc}(\gamma_i)$ and $\mathsf{Enc}(\gamma_i \cdot x_B)$. In the second round, since $\mathsf{Enc}(\gamma)$ is already known, parties can immediately send $\mathsf{Enc}(k_i\gamma)$. At the same time, they can decommit to $k_i \cdot G$ and $\mathsf{Enc}(k_i)$, and proof validity with a zk proof that ties $k_i \cdot G, \mathsf{Enc}(k_i)$ together via L_{EncDL}, and $\mathsf{Enc}(k_i), \mathsf{Enc}(k_i\gamma)$ via $L_{\mathsf{EncDH}}[\mathsf{Enc}(\gamma)]$.

M.2 Simulation

We define the following hybrids where **Hybrid1** coincides with the simulation (Figure 26) and **Hybrid7** coincides with the real execution. Note that in **Hybrid1** all the ciphertexts $\mathsf{ct}_{\mathsf{key}}, \mathsf{ct}_1^i, \mathsf{ct}_2^i, \mathsf{ct}_3^i, \mathsf{ct}_4^i$ are decrypted to zero. These values are gradually replaced with correct values as we transit from **Hybrid*i*** to **Hybrid*i*+1**. The correct values are known to the simulators in the hybrids as the simulator takes the role of $\mathcal{G}^*$ instead of using it as an oracle. This does not change the distribution of the values as the simulator samples the key and the presignature as well as computes the signature exactly the same as $\mathcal{G}^*$ would. In more detail:

1. **Hybrid1** is similar to the simulation in Figure 26, except that $\mathcal{S}$ emulates $\mathcal{G}^*$ locally (instead of using $\mathcal{G}^*$ as an oracle).
2. **Hybrid2** is similar to **Hybrid1** except that, $\mathcal{S}$ encrypts the correct value x_B instead of zero in Step 3c in Simulation 17.
3. **Hybrid3** is similar to **Hybrid2** except that in Step 2(a)ii in Simulation 26, $\mathcal{S}$ picks its own $\gamma_i \in [0, q)$, computes and uses $\mathsf{ct}_1^i = \mathsf{AHE.Enc}(\gamma_i; \eta^j_{\mathsf{mask}_1})$ instead of an encryption of zero.
4. **Hybrid4** is similar to **Hybrid3** except that in Step 2(a)ii in Simulation 26, $\mathcal{S}$ computes and uses $\mathsf{ct}_2^i = \mathsf{AHE.Eval}(pk, f_i, \mathsf{ct}_{\mathsf{key}}; \eta^i_{\mathsf{mask}_2})$ where $f_i(x) = \gamma_i \cdot x$, instead of an encryption of zero.
5. **Hybrid5** is similar to **Hybrid4** except that in Step **??** in Simulation 26, $\mathcal{S}$ computes and uses $\mathsf{ct}_3^i = \mathsf{AHE.Enc}(k; \eta^j_{\mathsf{mask}_3})$, instead of an encryption of zero.
6. **Hybrid6** is similar to **Hybrid5** except that in Step 2(b)v in Simulation 26, $\mathcal{S}$ computes and uses $\mathsf{ct}_4^i = \mathsf{AHE.Eval}(pk, f'_i, \mathsf{ct}_1; \eta^i_{\mathsf{mask}_4})$ where $f'_i(x) = k_i \cdot x$, instead of an encryption of zero.
7. **Hybrid7** is similar to **Hybrid6** except that in Step 4b in Simulation 19, $\mathcal{S}$ computes $\mathsf{pt}_4 = \gamma k_B \mod q$ and $\mathsf{pt}_A = k_A \cdot \gamma(r \cdot x + m) \mod q$.

For all $i = 1, \ldots, 7$ we have that **Hybrid*i*** is indistinguishable from **Hybrid*i*+1** as distinguishing the two reduces to breaking the CPA-security of the encryption scheme, either directly or by breaking the secure function evaluation. Since AHE is semantically secure (which is equivalent to CPA-security), we conclude that the distributions of the two hybrids are computationally indistinguishable.

SIMULATION 15 (*$\mathcal{G}^*$-Simulation for Π_{kgen} and Π_{pres} - All parties B_i are corrupted*)

All parties B_i are corrupted by the adversary $\mathcal{A}$:
$\mathcal{S}$ runs $\mathcal{A}$ internally, and responds to the following invocations

– $\Pi_{\mathsf{kgen}}\left((\mathbb{G}, G, q)\right)$:
  1. Simulate $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$ by sending $(\textbf{receipt}, \mathsf{pid}_A)$ to $\mathcal{A}$.
  2. *Only on first execution:* Simulate $\mathcal{F}_{\mathsf{TAHE}}$ by receiving $(\textbf{keygen}, \mathsf{sid})$ from $\mathcal{A}$ on behalf of all B_i's. Run $(pk; sk) \leftarrow \mathsf{AHE.Gen}(1^\kappa)$, send pk to the adversary and receive the adversary's response $(\texttt{continue}, U')$. If $U' = U' \cap [n]$ is empty then send $(\textbf{pubkey}, \mathsf{sid}, pk)$ to $\mathcal{A}$, otherwise, send $(\textbf{pubkey}, \mathsf{sid}, \perp, U')$ to $\mathcal{A}$ and abort (simulating a later abort by party A).
  3. Simulate $\mathcal{F}^{L_{\mathsf{EncDL}}}_{\mathsf{agg-zk}}$ by receiving $(\textbf{prove}, \mathsf{sid}, \mathsf{pid}_i, X_i, \mathsf{ct}_i; x_i, \rho_i)$ from $\mathcal{A}$ on behalf of every B_i. Let $M \subseteq [n]$ be the set for which $\mathsf{ct}_i \neq \mathsf{AHE.Enc}(pk, x_i; \rho_i)$ or $X_i \neq x_i \cdot G$. If $M = \emptyset$ then send $(\textbf{proof}, \mathsf{sid}, X_B, \mathsf{ct}_{\mathsf{key}})$ to $\mathcal{A}$ where $\mathsf{ct}_{\mathsf{key}} = \bigoplus_{i\in[n]} \mathsf{ct}_i$, otherwise send $(\textbf{malicious}, \mathsf{sid}, M)$ to $\mathcal{A}$ and abort (simulating a later abort by party A).
  4. Query $\mathcal{G}^*$ on input **keygen** and obtain X; then set $X_A = X - X_B$.
  5. Simulate $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$ by sending $(\textbf{decom–proof}, \mathsf{pid}_A, X_A)$ to $\mathcal{A}$.
  6. Record $(\mathsf{keygen}, X_B, X, \mathsf{ct}_{\mathsf{key}}, pk)$ and output whatever $\mathcal{A}$ outputs.

– $\Pi_{\mathsf{pres}}\left((\mathbb{G}, G, q), \mathsf{sid}\right)$
  1. Query $\mathcal{G}^*$ on input **pres** and obtain R.
  2. Compute $K_A = \mathsf{Com}(0, \rho_1)$ for some $\rho_1 \leftarrow \mathcal{R}_{pp}$.
  3. *First round:*
     (a) Simulate $\mathcal{F}^{L_{\mathsf{DCom}}}_{\mathsf{zk}}$ by sending $(\textbf{proof}, \mathsf{sid}||\mathsf{pid}_A, K_A)$ to $\mathcal{A}$.
     (b) Simulate $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{agg-zk}}$ by receiving $(\textbf{prove}, \mathsf{sid}, \mathsf{pid}_i, R_i; k_i)$ from $\mathcal{A}$ on behalf of every B_i. Let $M \subseteq [n]$ be the set for which $k_i \cdot G \neq R_i$. If $M = \emptyset$ then send $(\textbf{proof}, \mathsf{sid}, R_B)$ to $\mathcal{A}$, otherwise send $(\textbf{malicious}, \mathsf{sid}, M)$ to $\mathcal{A}$ and abort (simulating a later abort by party A).
     (c) Simulate $\mathcal{F}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_{\mathsf{key}}]}_{\mathsf{agg-zk}}$ by receiving $(\textbf{prove}, \mathsf{sid}||\gamma, \mathsf{pid}_i, \mathsf{ct}^i_1, \mathsf{ct}^i_2; \gamma_i, \eta^i_{\mathsf{mask}_1}, \eta^i_{\mathsf{mask}_2})$ from $\mathcal{A}$ on behalf of every B_i. Let $M \subseteq [n]$ be the set for which $\mathsf{ct}^i_1 \neq \mathsf{AHE.Enc}(\gamma_i; \eta^i_{\mathsf{mask}_1})$ or $\mathsf{ct}^i_2 \neq \mathsf{AHE.Eval}(pk, f_i, \mathsf{ct}_{\mathsf{key}}; \eta^i_{\mathsf{mask}_2})$ where $f_i(x) = \gamma_i \cdot x$. If $M = \emptyset$ then send $(\textbf{proof}, \mathsf{sid}||\gamma, \mathsf{ct}_1, \mathsf{ct}_2)$ to $\mathcal{A}$, otherwise send $(\textbf{malicious}, \mathsf{sid}, M)$ to $\mathcal{A}$ and abort (simulating a later abort by party A).
     (d) Simulate $\mathcal{F}^{L_{\mathsf{EncDL}}}_{\mathsf{agg-zk}}$ by receiving $(\textbf{prove}, \mathsf{sid}, \mathsf{pid}_i, R_i, \mathsf{ct}^i_3; k_i, \eta^i_{\mathsf{mask}_3})$ from $\mathcal{A}$ on behalf of every B_i. Let $M \subseteq [n]$ be the set for which $\mathsf{ct}^i_3 \neq \mathsf{AHE.Enc}(pk, k_i; \eta^i_{\mathsf{mask}_3})$ or $R_i \neq k_i \cdot G$. If $M = \emptyset$ then send $(\textbf{proof}, \mathsf{sid}, R, \mathsf{ct}_3)$ to $\mathcal{A}$, otherwise send $(\textbf{malicious}, \mathsf{sid}, M)$ to $\mathcal{A}$ and abort (simulating a later abort by party A).
  4. *Second round:*
     (a) Simulate $\mathcal{F}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_1]}_{\mathsf{agg-zk}}$ by receiving $(\textbf{prove}, \mathsf{sid}||k, \mathsf{pid}_i, \mathsf{ct}^i_3, \mathsf{ct}^i_4; k_i, \eta^i_{\mathsf{mask}_3}, \eta^i_{\mathsf{mask}_4})$ from $\mathcal{A}$ on behalf of every B_i. Let $M \subseteq [n]$ be the set for which $\mathsf{ct}^i_3 \neq \mathsf{AHE.Enc}(pk, k_i; \eta^i_{\mathsf{mask}_3})$ or $\mathsf{ct}^i_4 \neq \mathsf{AHE.Eval}(pk, f'_i, \mathsf{ct}_1; \eta^i_{\mathsf{mask}_4})$ where $f'_i(x) = k_i \cdot x$. If $M = \emptyset$ then send $(\textbf{proof}, \mathsf{sid}||k, \mathsf{ct}_3, \mathsf{ct}_4)$ to $\mathcal{A}$, otherwise send $(\textbf{malicious}, \mathsf{sid}||k, M)$ to $\mathcal{A}$ and abort (simulating a later abort by party A).
  5. Record $(\mathsf{presign}, \mathsf{sid}, R_B, R, \mathsf{ct}_1, \mathsf{ct}_2; \gamma, k_B, \rho_1)$, where ct_1 and ct_2 are encryptions of γ and $\gamma \cdot x_B$, and output whatever $\mathcal{A}$ outputs.

SIMULATION 16 (*$\mathcal{G}^*$-Simulation for Π_{sign}) - All parties B_i are corrupted*)

- $\Pi_{\mathsf{sign}}((\mathbb{G}, G, q), \mathsf{sid}, msg)$:
  1. Retrieve R from the record $(\mathsf{presign}, \mathsf{sid}, R_B, R, \mathsf{ct}_1, \mathsf{ct}_2; \gamma, k_B, \rho_1)$, if such a record does not exist then abort.
  2. Query $\mathcal{G}^*$ on input $(\mathbf{sign}, msg, R)$, obtain $\sigma = (r, s)$ and set $m = \mathcal{H}(msg)$.
  3. Set $\mathsf{ct}_A \leftarrow \mathcal{S}_{\mathsf{Eval}}(pk, s_A)$ where $s_A = (\gamma k_B) \cdot s \mod q$ and $\mathcal{S}_{\mathsf{Eval}}$ is the simulator associated with the AHE scheme (Def. 2.1).
  4. Compute $U_A = \mathsf{Com}(0, \rho_2)$ for a random ρ_2, then compute $C_1 = (r \odot U_A) \oplus (m \odot K_A)$ and $C_2 = r \odot K_A$.
  5. Simulate zero-knowledge proofs:
     - Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComDL}}[\mathsf{pp}, \mathbb{G}, q]}$ by sending $(\mathbf{proof}, \mathsf{sid}||\mathsf{pid}_A, K_A, R, R_B)$ to $\mathcal{A}$.
     - Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComRatio}}[\mathsf{pp}, \mathbb{G}, G, q]}$ by sending $(\mathbf{proof}, \mathsf{sid}||\mathsf{pid}_A, K_A, U_A, X_A)$ to $\mathcal{A}$.
     - Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComEval}}[\mathsf{pp}, pk, \mathsf{ct}_1, \mathsf{ct}_2]}$ by sending $(\mathbf{proof}, \mathsf{sid}||\mathsf{pid}_A, \mathsf{ct}_A, C_1, C_2)$ to $\mathcal{A}$.
  6. Output whatever $\mathcal{A}$ outputs.

SIMULATION 17 (*$\mathcal{G}^*$-Simulation for Π_{kgen} - Parties $A \cup \{B_j\}_{j \in [n] \setminus \{i\}}$ are corrupted*)

Parties $A \cup \{B_j\}_{j \in [n] \setminus \{i\}}$ are corrupted (controlled by adversary $\mathcal{A}$):
$\mathcal{S}$ runs adversary $\mathcal{A}$ internally, and responds to the following invocations

- $\Pi_{\mathsf{2kgen}}((\mathbb{G}, G, q))$:
  1. ***A*'s first message:**
     (a) Simulate $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{DL}}}$ by receiving $(\mathbf{com\text{-}prove}, \mathsf{pid}_A, X_A; x_A)$.
  2. Query $\mathcal{G}^*$ on input **keygen** and obtain X and set $X_B = X - X_A$.
  3. ***B*'s first message:**
     (a) $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{DL}}}$ by sending $(\mathbf{receipt}, \mathsf{pid}_A)$ to $\mathcal{A}$.
     (b) *Only on first execution:* Simulate $\mathcal{F}_{\mathsf{TAHE}}$ by receiving $(\mathbf{keygen}, \mathsf{sid})$ from $\mathcal{A}$ on behalf of all B_j's. Run $(pk; sk) \leftarrow \mathsf{AHE.Gen}(1^\kappa)$, send pk to the adversary and receive the adversary's response $(\mathtt{continue}, U')$. If $U' = U' \cap [n]$ is empty then send $(\mathbf{pubkey}, \mathsf{sid}, pk)$ to $\mathcal{A}$, otherwise, send $(\mathbf{pubkey}, \mathsf{sid}, \perp, U')$ to $\mathcal{A}$ and abort (simulating a later abort by party A).
     (c) Simulate $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDL}}}$ by receiving $(\mathbf{prove}, \mathsf{sid}, \mathsf{pid}_j, X_j, \mathsf{ct}_j; x_j, \rho_j)$ from $\mathcal{A}$ on behalf of every B_j. Let $M \subseteq [n]$ be the set for which $\mathsf{ct}_i \neq \mathsf{AHE.Enc}(pk, x_i; \rho_i)$ or $X_i \neq x_i \cdot G$. If $M = \emptyset$ then send $(\mathbf{proof}, \mathsf{sid}, X_B, \mathsf{ct}_{\mathsf{key}})$ to $\mathcal{A}$ where $\mathsf{ct}_{\mathsf{key}} = \mathsf{AHE.Enc}(pk, 0; \eta)$ for a randomly chosen η; otherwise send $(\mathbf{malicious}, \mathsf{sid}, M)$ to $\mathcal{A}$ and abort (simulating a later abort by party A).
  4. ***A*'s second message:**
     (a) Simulate $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDL}}}$ by sending $(\mathbf{prove}, \mathsf{pid}_B, \mathsf{ct}_{\mathsf{key}}, pk, X_B)$ to $\mathcal{A}$ (directed to party A; recall that $\mathsf{ct}_{\mathsf{key}}$ is an encryption of 0).
     (b) Simulate $\mathcal{F}_{\mathsf{TAHE}}$ by receiving $(\mathbf{certify}, pk')$ from $\mathcal{A}$. If $pk' = pk$ from above then respond with 1, otherwise respond with 0.
     (c) Simulate $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{DL}}}$ by receiving $(\mathbf{decom\text{-}proof}, \mathsf{pid}_A, X_A)$ from $\mathcal{A}$ (party A), and abort if $X_A \neq x_A \cdot G$. Otherwise, send $(\mathbf{decom\text{-}proof}, \mathsf{pid}_A, X_A)$ to $\mathcal{A}$ (parties B_j's).
  5. Record $(\mathsf{keygen}, X_A, X_B, X, \mathsf{ct}_{\mathsf{key}}, pk; sk)$ and output whatever $\mathcal{A}$ outputs.

SIMULATION 18 (*$\mathcal{G}^*$-Simulation for Π_{pres} - Parties $A \cup \{B_j\}_{j\in[n]\setminus\{i\}}$ are corrupted*)

- $\Pi_{\mathsf{2pres}}\left((\mathbb{G},G,q),\mathsf{sid}\right)$
  1. **A's message:**
     (a) Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DCom}}}$ by receiving $(\textbf{prove},\mathsf{sid},\mathsf{pid}_A,K_A;k_A,\rho_1)$ from $\mathcal{A}$. Abort if $K_A \neq \mathsf{Com}(k_A,\rho_1)$.
  2. Query $\mathcal{G}^*$ on input **pres** and obtain R.
  3. Set $R_B = k_A \cdot R$ (implying $R = (k_A)^{-1} \cdot R_B$ as required).
  4. **B's message:**
     (a) *First round:*
        i. Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DCom}}}$ by sending $(\textbf{proof},\mathsf{sid}||\mathsf{pid}_A,K_A)$ to $\mathcal{A}$.
        ii. Simulate $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{DL}}}$ by receiving $(\textbf{prove},\mathsf{sid},\mathsf{pid}_j,R_j;k_j)$ from $\mathcal{A}$ on behalf of every B_j. Let $M \subseteq [n]\setminus\{i\}$ be the set for which $k_j \cdot G \neq R_j$. If $M=\emptyset$ then send $(\textbf{proof},\mathsf{sid},R_B)$ to $\mathcal{A}$, otherwise send $(\textbf{malicious},\mathsf{sid},M)$ to $\mathcal{A}$ and abort (simulating an abort by party B_i).
        iii. Simulate $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDH}}[pk,\mathsf{ct}_{\mathsf{key}}]}$ by receiving $(\textbf{prove},\mathsf{sid}\|\gamma,\mathsf{pid}_j,\mathsf{ct}_1^j,\mathsf{ct}_2^j;\gamma_j,\eta^j_{\mathsf{mask}_1},\eta^j_{\mathsf{mask}_2})$ from $\mathcal{A}$ on behalf of every B_j. Let $M \subseteq [n]\setminus\{i\}$ be the set for which $\mathsf{ct}_1^j \neq \mathsf{AHE.Enc}(\gamma_j;\eta^j_{\mathsf{mask}_1})$ or $\mathsf{ct}_2^j \neq \mathsf{AHE.Eval}(pk,f_j,\mathsf{ct}_{\mathsf{key}};\eta^j_{\mathsf{mask}_2})$ where $f_j(x)=\gamma_j\cdot x$. If $M=\emptyset$, pick $\eta^i_{\mathsf{mask}_3},\eta^i_{\mathsf{mask}_4} \leftarrow \mathcal{R}_{pk}$, compute $\mathsf{ct}_1 = \mathsf{AHE.Enc}(pk,0;\eta^i_{\mathsf{mask}_1})$ and $\mathsf{ct}_2 = \mathsf{AHE.Enc}(pk,0;\eta^i_{\mathsf{mask}_2})$, and send $(\textbf{proof},\mathsf{sid}\|\gamma,\mathsf{ct}_1,\mathsf{ct}_2)$ to $\mathcal{A}$. Otherwise send $(\textbf{malicious},\mathsf{sid},M)$ to $\mathcal{A}$ and abort (simulating an abort by party B_i).
        iv. Simulate $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDL}}}$ by receiving $(\textbf{prove},\mathsf{sid},\mathsf{pid}_j,R_j,\mathsf{ct}_3^j;k_j,\eta^j_{\mathsf{mask}_3})$ from $\mathcal{A}$ on behalf of every B_j. Let $M \subseteq [n]\setminus\{i\}$ be the set for which $\mathsf{ct}_3^j \neq \mathsf{AHE.Enc}(pk,k_j;\eta^j_{\mathsf{mask}_3})$ or $R_j \neq k_j \cdot G$. If $M=\emptyset$ then pick random $\eta^i_{\mathsf{mask}_3},\eta^i_{\mathsf{mask}_4} \leftarrow \mathcal{R}_{pk}$, compute $\mathsf{ct}_3 = \mathsf{AHE.Enc}(pk,0;\eta^i_{\mathsf{mask}_3})$ and $\mathsf{ct}_4 = \mathsf{AHE.Enc}(pk,0;\eta^i_{\mathsf{mask}_4})$ and send $(\textbf{proof},\mathsf{sid},R,\mathsf{ct}_3)$ to $\mathcal{A}$, otherwise send $(\textbf{malicious},\mathsf{sid},M)$ to $\mathcal{A}$ and abort (simulating an abort by party B_i).
     (b) *Second round:*
        i. Simulate $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDH}}[pk,\mathsf{ct}_1]}$ by receiving $(\textbf{prove},\mathsf{sid}\|k,\mathsf{pid}_j,\mathsf{ct}_3^j,\mathsf{ct}_4^j;k_j,\eta^j_{\mathsf{mask}_3},\eta^j_{\mathsf{mask}_4})$ from $\mathcal{A}$ on behalf of every B_j. Let $M \subseteq [n]$ be the set for which $\mathsf{ct}_3^j \neq \mathsf{AHE.Enc}(pk,k_j;\eta^j_{\mathsf{mask}_3})$ or $\mathsf{ct}_4^j \neq \mathsf{AHE.Eval}(pk,f'_j,\mathsf{ct}_1;\eta^j_{\mathsf{mask}_4})$ where $f'_j(x)=k_j\cdot x$. If $M=\emptyset$ then send $(\textbf{proof},\mathsf{sid}\|k,\mathsf{ct}_3,\mathsf{ct}_4)$ to $\mathcal{A}$, otherwise send $(\textbf{malicious},\mathsf{sid}\|k,M)$ to $\mathcal{A}$ and abort (simulating an abort by party B_i).
  5. **A's verification**
     (a) Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DL}}}$ by sending $(\textbf{proof},\mathsf{sid}||\mathsf{pid}_B,R_B)$ to $\mathcal{A}$.
     (b) Simulate $\mathcal{F}_{\mathsf{agg-zk}}^{L_{\mathsf{EncDH}}[pk,\mathsf{ct}_{\mathsf{key}}]}$ by sending $(\textbf{proof},\mathsf{sid},\mathsf{ct}_1,\mathsf{ct}_2)$ to $\mathcal{A}$.
  6. Record $(\mathsf{presign},\mathsf{sid},R_B,\{\gamma_j\}_{j\in[n]\setminus\{i\}},R,K_A,\mathsf{ct}_1,\mathsf{ct}_2,\mathsf{ct}_4;k_A,\rho_1)$ and output whatever $\mathcal{A}$ outputs.

SIMULATION 19 (*$\mathcal{G}^*$-Simulation for Π_{sign} - Parties $A \cup \{B_j\}_{j\in[n]\setminus\{i\}}$ are corrupted*)

- $\Pi_{\mathsf{2sign}}\left((\mathbb{G}, G, q), \mathsf{sid}, msg\right)$:
  1. Set $m = H(msg)$.
  2. **A's message:**
     (a) Receive from $\mathcal{A}$ the following proofs (abort if any of these is not received):
       - $(\mathbf{prove}, \mathsf{sid}||\mathsf{pid}_A, K_A, R, R_B; k_A, \rho_1)$ by simulating $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComDL}}[\mathsf{pp},\mathbb{G},q]}$.
       - $(\mathbf{prove}, \mathsf{sid}||\mathsf{pid}_A, K_A, U_A, X_A; k_A, x_A, \rho_1, \rho_2)$ by simulating $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComRatio}}[\mathsf{pp},\mathbb{G},G,q]}$.
       - $(\mathbf{prove}, \mathsf{sid}||\mathsf{pid}_A, \mathsf{ct}_A, C_0, C_1; a_0, a_1, \alpha_1, \alpha_2, \eta)$ by simulating $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComEval}}[\mathsf{pp},pk,\mathsf{ct}_1,\mathsf{ct}_2]}$.

     (b) Abort if any of the following is incorrect:
       - $(K_A, R, R_B; k_A, \rho_1) \in L_{\mathsf{DComDL}}[\mathsf{pp}, \mathbb{G}, q]$.
       - $(K_A, U_A, X_A; k_A, x_A, \rho_1, \rho_2) \in L_{\mathsf{DComRatio}}[\mathsf{pp}, \mathbb{G}, G, q]$.
       - $(\mathsf{ct}_A, C_0, C_1; a_0, a_1, \alpha_1, \alpha_2, \eta) \in L_{\mathsf{DComEval}}[\mathsf{pp}, pk, \mathsf{ct}_1, \mathsf{ct}_2]$ where $\alpha_1 = r \cdot \rho_2 + m \cdot \rho_1$ and $\alpha_2 = r \cdot \rho_1$.
       - There are records $(\mathsf{presign}, \mathsf{sid}, R_B, \{\gamma_j\}_{j\in[n]\setminus\{i\}}, R, K_A, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_4; k_A, \rho_1)$ and $(\mathsf{keygen}, X_A, X_B, X, \mathsf{ct}_{\mathsf{key}}, pk; sk)$.
       - $C_0 = (r \odot U_A) \oplus (m \odot K_A)$ and $C_1 = r \odot K_A$, where $r = R|_{x-axis}$.
  3. Query $\mathcal{G}^*$ on input $(\mathbf{sign}, msg, R)$, and obtain $\sigma = (r, s)$.
  4. **B_i's verification and output**:
     (a) Simulate zero-knowledge proofs:
       - Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComDL}}[\mathsf{pp},\mathbb{G},q]}$ by sending $(\mathbf{proof}, \mathsf{sid}||\mathsf{pid}_A, K_A, R, R_B)$ to $\mathcal{A}$.
       - Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComRatio}}[\mathsf{pp},\mathbb{G},G,q]}$ by sending $(\mathbf{proof}, \mathsf{sid}||\mathsf{pid}_A, K_A, U_A, X_A)$ to $\mathcal{A}$.
       - Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComEval}}[\mathsf{pp},pk,\mathsf{ct}_1,\mathsf{ct}_2]}$ by sending $(\mathbf{proof}, \mathsf{sid}||\mathsf{pid}_A, \mathsf{ct}_A, C_1, C_2)$ to $\mathcal{A}$.

     (b) Pick a random $\tilde{\gamma}, \tilde{k}_B \in [0, q)$ and compute $\mathsf{pt}_4 = \tilde{\gamma} \cdot \tilde{k}_B \mod q$ and $\mathsf{pt}_A = s \cdot \mathsf{pt}_4 \mod q$.

     (c) Simulate $\mathcal{F}_{\mathsf{TAHE}}$ by receiving $(\mathbf{decrypt}, pk, \mathsf{ct}_A)$ and $(\mathbf{decrypt}, pk, \mathsf{ct}_4)$ from $\mathcal{A}$ on behalf of all B_j's. Send $(\mathbf{decrypted}, pk, \mathsf{ct}_4, \mathsf{pt}_4)$ and $(\mathbf{decrypted}, pk, \mathsf{ct}_A, \mathsf{pt}_A)$ to the adversary. If the adversary responds with `continue` $= 1$ or $U' \cap U = \emptyset$ on both of them then output $(\mathbf{decrypted}, pk, \mathsf{ct}_4, \mathsf{pt}_4)$ and $(\mathbf{decrypted}, pk, \mathsf{ct}_A, \mathsf{pt}_A)$ again to the adversary (this time directed to parties B_j's).

PROTOCOL 20 (*Two-Party Key-Generation* Π_{2kgen}: $\mathsf{KeyGen}(\mathbb{G}, G, q)$)

The protocol is parameterized with a group description $(\mathbb{G}, G, q)$, computational security parameter 1^κ, and with a session id sid. The parties do as follows:

1. A**'s first messages**:
   (a) A samples a random $x_A \leftarrow \mathbb{Z}_q$ and computes $X_A = x_A \cdot G$.
   (b) A sends $(\textbf{com–prove}, \mathsf{pid}_A, X_A; x_A)$ to $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$.
2. B**'s first message**:
   (a) B receives $(\textbf{receipt}, \mathsf{pid}_A)$ from $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$.
   (b) B samples a random $x_B \leftarrow \mathbb{Z}_q$ and computes $X_B = x_B \cdot G$.
   (c) B runs $(pk, sk) \leftarrow \mathsf{AHE.Gen}(1^\kappa, \mathsf{aux})$ where aux its randomness, and $\mathsf{ct}_{\mathsf{key}} = \mathsf{AHE.Enc}(pk, x_B; \eta)$ where $\eta \leftarrow \mathcal{R}_{pk}$.
   (d) B sends $(\textbf{prove}, \mathsf{pid}_B, pk; \mathsf{aux})$ to $\mathcal{F}^{L_{\mathsf{GenAHE}}}_{\mathsf{zk}}$.
   (e) B sends $(\textbf{prove}, \mathsf{pid}_B, X_B, \mathsf{ct}_{\mathsf{key}}; x_B, \eta)$ to $\mathcal{F}^{L_{\mathsf{EncDL}}}_{\mathsf{zk}}$.
3. A**'s second message**:
   (a) A receives $(\textbf{proof}, \mathsf{pid}_B, X_B, \mathsf{ct}_{\mathsf{key}})$ from $\mathcal{F}^{L_{\mathsf{EncDL}}}_{\mathsf{zk}}$, if not, it aborts.
   (b) A receives $(\textbf{proof}, \mathsf{pid}_B, pk)$ from $\mathcal{F}^{L_{\mathsf{GenAHE}}}_{\mathsf{zk}}$, if not, it aborts.
   (c) A sends $(\textbf{decom–proof}, \mathsf{pid}_A)$ to $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$.
4. B**'s verification.**
   (a) B receives $(\textbf{decom–proof}, \mathsf{pid}_A, X_A)$ from $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$, if not, it aborts.
5. **Output.**
   – A outputs $X = X_A + X_B$ and record $(\mathsf{keygen}, X_B, X, \mathsf{ct}_{\mathsf{key}}, pk)$.
   – B outputs $X = X_A + X_B$ and record $(\mathsf{keygen}, X_A, X, \mathsf{ct}_{\mathsf{key}}, pk; sk)$.

PROTOCOL 21 (*Two-Party Presigning* Π_{2pres}: $\mathsf{Presign}\,((\mathbb{G}, G, q), \mathit{sid})$)

Let ssid be a fresh sub-session identifier agreed upon by A and B. The parties proceed as follows:

1. A**'s message**:
   (a) A samples a random $k_A \leftarrow \mathbb{Z}_q$ and computes $K_A = \mathsf{Com}(k_A; \rho_1)$.
   (b) A sends $(\textbf{prove}, \mathsf{sid}, \mathsf{pid}_A, K_A; k_A, \rho_1)$ to $\mathcal{F}^{L_{\mathsf{DCom}}}_{\mathsf{zk}}$.
2. B**'s message**:
   (a) B receives $(\textbf{proof}, \mathsf{sid}, \mathsf{pid}_A, K_A)$ from $\mathcal{F}^{L_{\mathsf{DCom}}}_{\mathsf{zk}}$, if not, it aborts.
   (b) B samples a random $k_B \leftarrow \mathbb{Z}_q^*$ and computes $R_B = (k_B)^{-1} \cdot G$.
   (c) B sends $(\textbf{prove}, \mathsf{sid}, \mathsf{pid}_B, G; k_B)$ to $\mathcal{F}^{L_{\mathsf{DL}}[\mathbb{G}, R_B, q]}_{\mathsf{zk}}$.
3. A**'s verification**
   (a) A receives $(\textbf{proof}, \mathsf{sid}, \mathsf{pid}_B, G)$ from $\mathcal{F}^{L_{\mathsf{DL}}[\mathbb{G}, R_B, q]}_{\mathsf{zk}}$, if not, it aborts.
4. **Output**
   (a) A records $(\mathsf{presign}, \mathsf{sid}, R_B, R; k_A)$, where $R = (k_A)^{-1} \cdot R_B$.
   (b) B records $(\mathsf{presign}, \mathsf{sid}, R_B, K_A; k_B)$.

PROTOCOL 22 (*Two-Party Signing Π_{2sign}:* $\mathsf{Sign}\left((\mathbb{G}, G, q), \mathit{sid}, msg\right)$)

Inputs:

1. A has $(X, \mathsf{ct}_{\mathsf{key}})$ from the key-generation protocol and $(\mathsf{sid}, R_B, R; k_A)$ from the presign protocol.
2. B has $(X, \mathsf{ct}_{\mathsf{key}}; sk)$ from the key-generation protocol and $(\mathsf{sid}, R_B, K_A; k_B)$ from the presign protocol.
3. A and B locally compute $m = H(msg)$ and verify there is a record of sid being used in a presign protocol which was not used before in a sign protocol.

The protocol:

1. **A's message**:
   (a) A computes $r = R|_{x-axis} \mod q$.
   (b) A samples $\rho_2 \in \mathcal{R}_{\mathsf{pp}}$ and computes $U_A = \mathsf{Com}(k_A \cdot x_A; \rho_2)$.
   (c) A sets $a_0 = r \cdot k_A \cdot x_A + m \cdot k_A$ and $a_1 = r \cdot k_A$.
   (d) A homomorphically evaluates $\mathsf{ct}_{\mathsf{key}}$, on its private function $f_A(x) := a_0 + a_1 \cdot x$:
      – $\mathsf{ct}_A \leftarrow \mathsf{AHE.Eval}(pk, f_A, \mathsf{ct}_{\mathsf{key}}; \eta)$
   (e) A sends the following proofs:
      – A sends $(\textbf{prove}, \mathsf{sid}, \mathsf{pid}_A, K_A, R, R_B; k_A, \rho_1)$ to $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComDL}}[\mathsf{pp}, \mathbb{G}, q]}$.
      – A sends $(\textbf{prove}, \mathsf{sid}, \mathsf{pid}_A, K_A, U_A, X_A; k_A, x_A, \rho_1, \rho_2)$ to $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComRatio}}[\mathsf{pp}, \mathbb{G}, G, q]}$.
      – A computes $C_0 = (r \odot U_A) \oplus (m \odot K_A)$ and $C_1 = r \odot K_A$, and sends $(\textbf{prove}, \mathsf{sid}, \mathsf{pid}_A, \mathsf{ct}_A, C_0, C_1; a_0, a_1, r \cdot \rho_2 + m \cdot \rho_1, r \cdot \rho_1, \eta)$ to $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComEval}}[\mathsf{pp}, pk, \mathsf{ct}_{\mathsf{key}}]}$.
2. **B's verification and output**:
   (a) B receives the following proofs, otherwise, it aborts.
      – $(\textbf{proof}, \mathsf{sid}, \mathsf{pid}_A, K_A, R, R_B)$ from $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComDL}}[\mathsf{pp}, \mathbb{G}, q]}$.
      – $(\textbf{proof}, \mathsf{sid}, \mathsf{pid}_A, K_A, U_A, X_A)$ from $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComRatio}}[\mathsf{pp}, \mathbb{G}, G, q]}$.
      – $(\textbf{proof}, \mathsf{sid}, \mathsf{pid}_A, \mathsf{ct}_A, C_0, C_1)$ from $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComEval}}[\mathsf{pp}, pk, \mathsf{ct}_{\mathsf{key}}]}$.
   (b) B aborts if any of the following is incorrect:
      – There are records $(\mathsf{presign}, \mathsf{sid}, R_B, K_A; k_B)$ and $(\mathsf{keygen}, X_A, X, \mathsf{ct}_{\mathsf{key}}, pk; sk)$ for some X and sk; that is, the values used in the proofs are consistent with values obtained in the key generation and presign sub-protocols;
      – $C_0 = (r \odot U_A) \oplus (m \odot K_A)$ and $C_1 = r \odot K_A$, where $r = R|_{x-axis}$.
   (c) B computes $\mathsf{pt} = \mathsf{AHE.Dec}(sk, \mathsf{ct}_A)$.
   (d) B computes $s' = k_B \cdot \mathsf{pt} \mod q$ and $s = \min\{s', q - s'\}$ (to ensure uniqueness of the signature).
   (e) B outputs $\sigma = (r, s)$.

SIMULATION 23 (*$\mathcal{G}^*$-Simulation for $\Pi_{\mathsf{2ecdsa}} = (\Pi_{\mathsf{2kgen}}, \Pi_{\mathsf{2pres}}, \Pi_{\mathsf{2sign}})$ - Party A is corrupted*)

Party A is corrupted (controlled by $\mathcal{A}$):
$\mathcal{S}$ runs $\mathcal{A}$ internally, and responds to the following invocations

- $\Pi_{\mathsf{2kgen}}((\mathbb{G}, G, q))$:
  1. Simulate $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{DL}}}$ by receiving $(\mathbf{com\text{–}prove}, \mathsf{pid}_A, X_A; x_A)$.
  2. Query $\mathcal{G}^*$ on input **keygen** and obtain X.
  3. Set $X_B = X - X_A$.
  4. Run $(pk, sk) \leftarrow \mathsf{AHE.Gen}(1^\kappa)$ and $\mathsf{ct}_{\mathsf{key}} = \mathsf{AHE.Enc}(pk, 0; \eta)$ for a randomly chosen η.
  5. Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{EncDL}}}$ by sending $(\mathbf{prove}, \mathsf{pid}_B, \mathsf{ct}_{\mathsf{key}}, pk, X_B)$ to $\mathcal{A}$.
  6. Simulate $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{DL}}}$ by receiving $(\mathbf{decom\text{–}proof}, \mathsf{pid}_A, X_A)$.
  7. Verify that $X_A = x_A \cdot G$, if not then abort.
  8. Record $(\mathsf{keygen}, X_A, X, \mathsf{ct}_{\mathsf{key}}, pk; sk)$ and output whatever $\mathcal{A}$ outputs.
- $\Pi_{\mathsf{2pres}}((\mathbb{G}, G, q), \mathsf{sid})$
  1. Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DCom}}}$ by receiving $(\mathbf{prove}, \mathsf{sid}, \mathsf{pid}_A, K_A; k_A, \rho_1)$ from $\mathcal{A}$.
  2. Query $\mathcal{G}^*$ on input **pres** and obtain R.
  3. Set $R_B = k_A \cdot R$ (implying $R = (k_A)^{-1} \cdot R_B$ as required).
  4. Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DL}}}$ by sending $(\mathbf{proof}, \mathsf{sid}||\mathsf{pid}_B, R_B)$ to $\mathcal{A}$.
  5. Record $(\mathsf{sid}, K_A; \cdot)$ and output whatever $\mathcal{A}$ outputs.
- $\Pi_{\mathsf{2sign}}((\mathbb{G}, G, q), \mathsf{sid}, M)$:
  1. Receive from $\mathcal{A}$ the following proofs (abort if any of these is not received):
     - $(\mathbf{prove}, \mathsf{sid}||\mathsf{pid}_A, K_A, R, R_B; k_A, \rho_1)$ by simulating $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComDL}}[pp,\mathbb{G},q]}$.
     - $(\mathbf{prove}, \mathsf{sid}||\mathsf{pid}_A, K_A, U_A, X_A; k_A, x_A, \rho_1, \rho_2)$ by simulating $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComRatio}}[pp,\mathbb{G},G,q]}$.
     - $(\mathbf{prove}, \mathsf{sid}||\mathsf{pid}_A, \mathsf{ct}_A, C_0, C_1; a_0, a_1, \alpha_1, \alpha_2, \eta)$ by simulating $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{DComEval}}[pp,pk,\mathsf{ct}_{\mathsf{key}}]}$.
  2. Aborts if any of the following is incorrect:
     - The values $\mathcal{A}$ provided to $\mathcal{F}_{\mathsf{zk}}$ satisfy languages $L_{\mathsf{DComDL}}[pp, \mathbb{G}, q]$, $L_{\mathsf{DComRatio}}[pp, \mathbb{G}, G, q]$ and $L_{\mathsf{DComEval}}[pp, pk, \mathsf{ct}_{\mathsf{key}}]$; for the latter, in particular verify that $\alpha_1 = r \cdot \rho_2 + m \cdot \rho_1$ and $\alpha_2 = r \cdot \rho_1$.
     - There are records $(\mathsf{sid}, R_B, K_A; \cdot)$ and $(\mathsf{keygen}, X_A, \cdot, \mathsf{ct}_{\mathsf{key}}, pk; \cdot)$;
     - $C_0 = (r \odot U_A) \oplus (m \odot K_A)$ and $C_1 = r \odot K_A$, where $r = R|_{x-axis}$.
  3. Query $\mathcal{G}^*$ on input $(\mathbf{sign}, M, R)$, obtain and output $\sigma = (r, s)$.

SIMULATION 24 (*$\mathcal{G}^*$-Simulation for $\Pi_{\mathsf{2ecdsa}} = (\Pi_{\mathsf{2kgen}}, \Pi_{\mathsf{2pres}}, \Pi_{\mathsf{2sign}})$ - Party B is corrupted*)

Party B is corrupted (controlled by $\mathcal{A}$):
$\mathcal{S}$ runs $\mathcal{A}$ internally, and responds to the following invocations

- $\Pi_{\mathsf{2kgen}}\,((\mathbb{G}, G, q))$:
  1. Simulate $\mathcal{F}^{L_{\mathsf{DL}}}_{\mathsf{com-zk}}$ by sending $(\textbf{receipt}, \mathsf{pid}_A)$ to $\mathcal{A}$.
  2. Simulate $\mathcal{F}^{L_{\mathsf{EncDL}}}_{\mathsf{zk}}$ by receiving $(\textbf{prove}, \mathsf{pid}_B, X_B, \mathsf{ct}_{\mathsf{key}}, pk; x_B, \eta)$ from $\mathcal{A}$. If $\mathsf{ct}_{\mathsf{key}} \neq \mathsf{AHE.Enc}(pk, x_B; \eta)$ or $X_B \neq x_B \cdot G$ then abort. Otherwise:
     (a) Query $\mathcal{G}^*$ on input **keygen** and obtain X.
     (b) Set $X_A = X - X_B$.
     (c) Simulate $\mathcal{F}_{\mathsf{com-zk}}$ by sending $(\textbf{decom–proof}, \mathsf{pid}_A, X_A)$ to $\mathcal{A}$.
     (d) Record $(\mathsf{keygen}, X_B, X, \mathsf{ct}_{\mathsf{key}}, pk)$ and output whatever $\mathcal{A}$ outputs.
- $\Pi_{\mathsf{2pres}}\,((\mathbb{G}, G, q), \mathsf{sid})$
  1. Query $\mathcal{G}^*$ on input **pres** and obtain R.
  2. Compute $K_A = \mathsf{Com}(0, \rho)$ for some $\rho \leftarrow \mathcal{R}_{pp}$.
  3. Simulate $\mathcal{F}^{L_{\mathsf{DCom}}}_{\mathsf{com-zk}}$ by sending $(\textbf{proof}, \mathsf{sid}||\mathsf{pid}_A, K_A)$ to $\mathcal{A}$.
  4. Simulate $\mathcal{F}^{L_{\mathsf{DL}}[\mathbb{G}, R_B, q]}_{\mathsf{zk}}$ by receiving $(\textbf{prove}, \mathsf{sid}||\mathsf{pid}_B, G; k_B)$ from $\mathcal{A}$. If $k_B \cdot R_B \neq G$ then abort.
  5. Record (sid, R_B, R) and output whatever $\mathcal{A}$ outputs.
- $\Pi_{\mathsf{2sign}}\,((\mathbb{G}, G, q), \mathsf{sid}, M)$:
  1. Retrieve R from the record $(\mathsf{presign}, \mathsf{sid}, \cdot, R; \cdot)$, if such a record does not exist then abort.
  2. Query $\mathcal{G}^*$ on input $(\textbf{sign}, M, R)$, obtain $\sigma = (r, s)$ and set $m = \mathcal{H}(M)$.
  3. Set $\mathsf{ct}_A \leftarrow \mathcal{S}_{\mathsf{Eval}}(pk, s')$ where $s' = s \cdot (k_B)^{-1} \mod q$ and $\mathcal{S}_{\mathsf{Eval}}$ is the simulator associated with the AHE scheme (Def. 2.1).
  4. Simulate zero-knowldege proofs:
     - Simulate $\mathcal{F}^{L_{\mathsf{DComDL}}[\mathsf{pp}, \mathbb{G}, q]}_{\mathsf{zk}}$ by sending $(\textbf{proof}, \mathsf{sid}||\mathsf{pid}_A, K_A, R, R_B)$ to $\mathcal{A}$.
     - Simulate $\mathcal{F}^{L_{\mathsf{DComRatio}}[\mathsf{pp}, \mathbb{G}, G, q]}_{\mathsf{zk}}$ by sending $(\textbf{proof}, \mathsf{sid}||\mathsf{pid}_A, K_A, U_A, X_A)$ to $\mathcal{A}$.
     - Simulate $\mathcal{F}^{L_{\mathsf{DComEval}}[\mathsf{pp}, pk, \mathsf{ct}_{\mathsf{key}}]}_{\mathsf{zk}}$ by sending $(\textbf{proof}, \mathsf{sid}||\mathsf{pid}_A, \mathsf{ct}_A, C_0, C_1)$ to $\mathcal{A}$.
  5. Output whatever $\mathcal{A}$ outputs.

PROTOCOL 25 (*2-Round Presigning* $\Pi_{\mathsf{2r-pres}}$: $\mathsf{Presign}_{\mathsf{2r}}\left((\mathbb{G}, G, q), \mathit{sid}\right)$)

The protocol interacts with $B_1, \ldots, B_n$ where everyone has pk and $\mathsf{ct}_{\mathsf{key}}$ as input. Before proceeding, all parties verify that sid has never been used before. Then each party B_i:

1. First Round:
   (a) Samples $\gamma_i \in [0, q), \eta^i_{\mathsf{mask}_1}, \eta^i_{\mathsf{mask}_2}$, and computes
      i. $\mathsf{ct}^i_1 = \mathsf{AHE.Enc}(pk, \gamma_i; \eta^i_{\mathsf{mask}_1})$,
      ii. $\mathsf{ct}^i_2 = \mathsf{AHE.Eval}(pk, f_i, \mathsf{ct}_{\mathsf{key}}; \eta^i_{\mathsf{mask}_2})$ where $f_i(x) = \gamma_i \cdot x$,

      and sends $(\mathbf{prove}, \mathsf{sid}, \mathsf{pid}_i, \mathsf{ct}^i_1, \mathsf{ct}^i_2; \gamma_i, \eta^i_{\mathsf{mask}_1}, \eta^i_{\mathsf{mask}_2})$ to $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_{\mathsf{key}}]}$.
   (b) Samples $k_i \leftarrow \mathbb{Z}^*_q$, and $\eta^i_{\mathsf{mask}_3}$.
   (c) Computes $R_i = k_i \cdot G$ and $\mathsf{ct}^i_3 = \mathsf{AHE.Enc}(pk, k_i; \eta^i_{\mathsf{mask}_3})$ and sends $(\mathbf{com\text{–}prove}, \mathsf{sid}, \mathsf{pid}_i, R_i, \mathsf{ct}^i_3; k_i, \eta^i_{\mathsf{mask}_3})$ to $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{EncDL}}}$.
2. *Second round:*
   (a) Receives for each party index j:
      i. $(\mathbf{proof}, \mathsf{sid}, \mathsf{pid}_j, \mathsf{ct}^j_1, \mathsf{ct}^j_2)$ from $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_{\mathsf{key}}]}$,
      ii. $(\mathbf{receipt}, \mathsf{sid}, \mathsf{pid}_j)$ from $\mathcal{F}_{\mathsf{com-zk}}$,
   (b) Computes $\mathsf{ct}_1 := \oplus_j \mathsf{ct}^j_1$, $\mathsf{ct}_2 := \oplus_j \mathsf{ct}^j_2$.
   (c) Sends $(\mathbf{decom\text{–}proof}, \mathsf{sid}, \mathsf{pid}_i)$ to $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{EncDL}}}$.
   (d) Samples $\eta^i_{\mathsf{mask}_4} \in \mathcal{R}_{pk}$,
   (e) Computes $\mathsf{ct}^i_4 = \mathsf{AHE.Eval}(pk, f'_i, \mathsf{ct}_3; \eta^i_{\mathsf{mask}_4})$ where $f'_i(x) = k_i \cdot x$.
   (f) Sends $(\mathbf{prove}, \mathsf{sid}, \mathsf{pid}_i, \mathsf{ct}^i_3, \mathsf{ct}^i_4; k_i, \eta^i_{\mathsf{mask}_3}, \eta^i_{\mathsf{mask}_4})$ to $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_1]}$.
3. *Proof verification:*
   B_i aborts if one of the following is not received:
   (a) $(\mathbf{decom\text{–}proof}, \mathsf{sid}, \mathsf{pid}_j, R_j, \mathsf{ct}^j_3)$ from $\mathcal{F}_{\mathsf{com-zk}}$,
   (b) $(\mathbf{proof}, \mathsf{sid}, \mathsf{ct}^j_3, \mathsf{ct}^j_4)$ from $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_1]}$
4. **Output:**
   B_i records $(\mathsf{presign}, \mathsf{sid}, R, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_4)$, where:
   (a) $R = R_1 + \ldots + R_n$.
   (b) ct_1 is an encryption of $\gamma \mod q$.
   (c) ct_2 is an encryption of $\gamma \cdot x_B \mod q$.
   (d) $\mathsf{ct}_4 := \oplus_i \mathsf{ct}^i_4$ is an encryption of $\gamma \cdot k_B \mod q$.

SIMULATION 26 (*$\mathcal{G}^*$-Simulation for $\Pi_{\mathsf{2r-pres}}$ - Parties $\{B_j\}_{j\in[n]\setminus\{i\}}$ are corrupted*)

- $\Pi_{\mathsf{2r-pres}}((\mathbb{G}, G, q), \mathsf{sid})$
  1. Query $\mathcal{G}^*$ on input **pres** and obtain R.
  2. B**'s message:**
     (a) *First round:*
        i. Sample $\eta^i_{\mathsf{mask}_1}, \eta^i_{\mathsf{mask}_2}, \eta^i_{\mathsf{mask}_3}$.
        ii. Compute $\mathsf{ct}^i_1 = \mathsf{AHE.Enc}(pk, 0; \eta^i_{\mathsf{mask}_1}), \mathsf{ct}^i_2 = \mathsf{AHE.Enc}(pk, 0; \eta^i_{\mathsf{mask}_2}), \mathsf{ct}^i_3 = \mathsf{AHE.Enc}(pk, 0; \eta^i_{\mathsf{mask}_3})$.
        iii. Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_{\mathsf{key}}]}$ and send $(\mathbf{proof}, \mathsf{sid}, \mathsf{pid}_i, \mathsf{ct}^i_1, \mathsf{ct}^i_2)$ to $\mathcal{A}$.
        iv. Simulate $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{EncDL}}}$ and send $(\mathbf{receipt}, \mathsf{sid}, \mathsf{pid}_i)$ to $\mathcal{A}$.
     (b) *Second round:*
        i. For each party index $j \neq i$, receive from $\mathcal{A}$ (on behalf of B_j):
           A. $(\mathbf{prove}, \mathsf{sid}\|\gamma, \mathsf{pid}_j, \mathsf{ct}^j_1, \mathsf{ct}^j_2; \gamma_j, \eta^j_{\mathsf{mask}_1}, \eta^j_{\mathsf{mask}_2})$ by simulating $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_{\mathsf{key}}]}$.
           B. $(\mathbf{com\text{-}prove}, \mathsf{sid}, \mathsf{pid}_j, R_j, \mathsf{ct}^j_3; k_j, \eta^j_{\mathsf{mask}_3})$ by simulating $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{EncDL}}}$.
        ii. Let $M \subseteq [n] \setminus \{i\}$ be the set for which $(\mathsf{ct}^j_1, \mathsf{ct}^j_2; \gamma_j, \eta^j_{\mathsf{mask}_1}, \eta^j_{\mathsf{mask}_2}) \notin L_{\mathsf{EncDH}}[pk, \mathsf{ct}_{\mathsf{key}}]$. If $M \neq \emptyset$ send $(\mathbf{malicious}, \mathsf{sid}, M)$ to $\mathcal{A}$ and abort (simulating an abort by party B_i).
        iii. Compute $\mathsf{ct}_1 := \oplus_i \mathsf{ct}^i_1$, $\mathsf{ct}_2 := \oplus_i \mathsf{ct}^i_1$.
        iv. Simulate $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{EncDL}}}$ and send $(\mathbf{decom\text{-}proof}, \mathsf{sid}, \mathsf{pid}_i, R_i, \mathsf{ct}^i_3)$ to $\mathcal{A}$.
        v. Sample $\eta^i_{\mathsf{mask}_4}$ and compute $\mathsf{ct}^i_4 = \mathsf{AHE.Enc}(pk, 0; \eta^i_{\mathsf{mask}_4})$.
        vi. Simulate $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_1]}$ and send $(\mathbf{proof}, \mathsf{sid}, \mathsf{ct}^i_3, \mathsf{ct}^i_4)$ to $\mathcal{A}$.
        vii. For each party index $j \neq i$, receive from $\mathcal{A}$ (on behalf of B_j):
           A. $(\mathbf{decom\text{-}proof}, \mathsf{sid}, \mathsf{pid}_j)$ by simulating $\mathcal{F}_{\mathsf{com-zk}}^{L_{\mathsf{EncDL}}}$.
           B. $(\mathbf{prove}, \mathsf{sid}, \mathsf{pid}_j, \mathsf{ct}^j_3, \mathsf{ct}^j_4; k_j, \eta^j_{\mathsf{mask}_3}, \eta^j_{\mathsf{mask}_4})$ by simulating $\mathcal{F}_{\mathsf{zk}}^{L_{\mathsf{EncDH}}[pk, \mathsf{ct}_1]}$
        viii. Let $M \subseteq [n] \setminus \{i\}$ be the set for which $(R_j, \mathsf{ct}^j_3; k_j, \eta^j_{\mathsf{mask}_3}) \notin L_{\mathsf{EncDL}}$ or $(\mathsf{ct}^j_3, \mathsf{ct}^j_4; k_j, \eta^j_{\mathsf{mask}_3}, \eta^j_{\mathsf{mask}_4}) \notin L_{\mathsf{EncDH}}[pk, \mathsf{ct}_1]$. If $M \neq \emptyset$, send $(\mathbf{malicious}, \mathsf{sid}, M)$ to $\mathcal{A}$ and abort (simulating an abort by party B_i).
  3. Record $(\mathsf{presign}, \mathsf{sid}, R, \{\gamma_j\}_{j\in[n]\setminus\{i\}}, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_4)$ and output whatever $\mathcal{A}$ outputs.