L for the Price of One: On the Benefits of Using more than $t + 1$ Parties in Threshold Signing

Daniel Escudero[1], Yashvanth Kondi[2], Yifan Song[3] and Hernán Vanegas[4]

[1] TACEO**, daniel.escudero@protonmail.com
[2] Silence Laboratories, yash@ykondi.net
[3] Tsinghua University and Shanghai Qi Zhi Institute, yfsong@mail.tsinghua.edu.cn
[4] HashCloak Inc., hernan@hashcloak.com

Abstract. In threshold ECDSA a committee of N parties holds—say, Shamir—shares of degree t of a secret key, where typically $N \gg t$ for operational purposes (*e.g.* redundancy to prevent losing the key). At signing time, $t + 1$ parties can execute a protocol to produce a signature on a given message without leaking anything about the secret key. In this work we show that if we use $n = t + 2(\ell - 1) + 1$ parties for signing instead, we can compute ℓ signatures without increasing at all the communication costs per party, essentially getting $\ell\times$ more signatures almost for free in a dishonest majority.

Our result is achieved by making use of packed secret-sharing to distribute multiple secrets with no communication penalty. This introduces several challenges not present in the non-packed domain, which leads us to introduce two primitives that may be of independent interest: we show how to prove that a sharing contains small elements efficiently, and its use in distributing consistent sharings of the same secret modulo two different integers. We also show how to generate degree-2 preprocessing material with constant communication via an adaptation of the virtual parties idea by Bracha from 1987.

We compare the communication of our protocol to sign ℓ messages with respect to the state-of-the-art in $t + 1$-party ECDSA signing by (Doerner *et al*, S&P'24), which needs to be repeated ℓ times. Our results show that, for appropriate regimes of (t, n, ℓ), our protocol can achieve 5x less communication (and even a larger factor) than theirs while adding only a few extra parties for the computation.

Keywords: Threshold Signatures · ECDSA.

1 Introduction

A threshold signature scheme allows signing authority to be delegated to a threshold of parties, rather than a single signer. In general, a pool of N parties holds secret-shares of a signing key sk in such a way that no set of $\leq t$ parties can produce a signature, but any set of $\geq t + 1$ can do so, without leaking

** Work done while the author was at JP Morgan AI Research, USA.

anything about sk in the process. Threshold signatures have several applications, including enforcing organizational access policies, multi-factor authentication, and decentralized key management.

Due to its short keys and signatures and wide standardization, ECDSA—the *Elliptic Curve Digital Signature Algorithm*—is a popular choice of signature scheme for modern applications, including many cryptocurrencies. The usage of *threshold* ECDSA signing for decentralized key management is common practice today as a layer of defense in the context of cryptocurrency custody, where key theft can have immediate financial consequences. ECDSA is additionally used in other protocols such as DNSSEC where key theft could be a critical failure, and previous work has made the case that such settings would benefit from threshold signing as well [Dal+20].

Many of these settings may require a large volume of signatures to be issued on demand, such as signing DNSSEC requests at scale, or signing transactions at an institutional or higher level. Another application where a large amount of signatures is required is in bridges between blockchain ecosystems. Such bridges are platforms that allow the transfer of assets and information between blockchains. Decentralized bridges often require a committee of parties to sign a large number of transactions jointly. Operating at such scale amplifies considerations about the efficient use of available resources. While the parameter t of parties authorized to issue signatures is governed by security constraints, the number $n > t$ of parties used for the actual signing is determined by operational factors. Most deployments run $(t+1)$-party signing algorithms, which means only $n = t+1$ parties are required for signing. In particular, if $n > t+1$ parties are available for signing, a naive deployment might utilize only $t+1$ of the online parties to sign messages, leaving the remaining $n-t-1$ idle. A more resource-conscious approach might be to employ a load balancer to distribute the burden evenly across all n online parties. However, each of these cases involves running the same $(t+1)$-party threshold ECDSA protocol for each message to be signed. In this work, we develop a more sophisticated approach to deriving the most value out of the gap between n and t; our method allows for the production of ℓ signatures at effectively the cost of a single $t+1$-party signing instance (for the right parameters). Although the average communication complexity of our protocol improves as the gap between n and t increases, the protocol remains secure in the dishonest-majority setting, even in the worst case where $t = n-1$. This fact poses several technical challenges that were addressed throughout this research.

1.1 Our Contribution

Our main contribution is a novel threshold ECDSA protocol tailored for the setting where (1) several signatures are produced in parallel, and (2) more than $t+1$ parties are available for signing. Previous works would produce ℓ signatures by using a committee of size $t+1$ (even if more parties were available for signing),[5]

[5] There are other works (discussed in Section 1.2) that consider *honest majority* or *two-thirds honest majority* regimes, where $2t+1$ or $3t+1$ parties sign, respectively,

involving roughly $\ell\times$ the communication costs of generating only one signature. In our work, we show that we can do better by adding only a small amount of extra parties to the committee. More precisely, we show a protocol Π_{Sign} for threshold ECDSA signatures between $n = t + 2(\ell - 1) + 1$ parties with the following features:

- In the online phase, each party communicates only 2 EC elements to sign ℓ different messages. This communication is *independent of* ℓ, and this works *for any* $\ell \geq 1$. The online protocol is *non-interactive*, as it consists of only *one round* where the parties send messages to the entity interested in learning the signatures.
- In the offline phase each party communicates less than $171.25 + \kappa \cdot 140.37$ EC elements for ℓ signatures, where κ is a statistical security parameter.[6] The offline protocol takes around 24 rounds.

Another interpretation of our result is the following: consider a set of $t + 1$ parties, and suppose they used our protocol for signing *one* message. If $2(\ell - 1)$ additional parties join, then the total set of parties can now sign ℓ messages, without increasing the amount of data that each party in the initial set of $t + 1$ parties had to communicate if they were to sign one message. Therefore, from the perspective of these initial $t + 1$ parties, they have signed ℓ messages at the same cost as one![7]

Our protocol has potential practical impact. In the online phase of the state-of-the-art in $(t + 1)$-party signing [Doe+24], each party must send two messages to the receiver of the signature, leading to $2(t+1)$ messages, *per signature*. This is $2\ell(t + 1)$ for ℓ signatures. Our protocol uses $n = t + 2(\ell - 1) + 1$ parties and each party also sends two messages to the receiver, leading to $2n$ messages. However, in contrast to [Doe+24], *this is already the communication for* ℓ *signatures*! Note that our online communication is $\approx 2/t + 1/\ell$ times the communication in [Doe+24], which leads to drastic benefits as t and/or ℓ grows. Finally, although our benefits are most notable in the online phase, our offline phase also provides communication gains compared to [Doe+24]. We refer the reader to Section 8 for a detailed discussion on communication, including estimates for various choices of n, t and ℓ, and comparisons to [Doe+24].

While our protocol in Section 7 uses the same signing key for all input messages, it is easy to extend our technique to handle ℓ different signing keys for the same batch. We give this extension in Section 7.2. This feature is important in applications where the same set of parties hold shares of different signing keys, such as the commonly used delegated custody model for cryptocurrencies.

---

and these works aim for robustness or asynchronous networks and are not relevant for this discussion.

[6] The actual communication is smaller than this, we use this bound for ease of presentation. Furthermore, most of this offline interaction is not only independent of the message, but independent of the *key*. See Section 8 for details.

[7] Of course, there *is* a cost: now the protocol is run among $n = t + 2(\ell - 1) + 1$ parties instead of $t + 1$. However, the bandwidth requirements for each party remains constant.

Finally, we point out that our protocol only requires the global pool of N parties to have degree-t Shamir sharings of the signing key sk, as in standard threshold ECDSA. In fact, as we point out in Remark 2 in page 48, our protocol can easily accommodate for signing ℓ different messages *using different keys*, which expands the range of use-cases. We also remark that *we do not need to asume an honest majority in our committee*, that is, it may be the case that $t \geq n/2$.[8] We provide a detailed overview of our techniques in Section 2.

We believe that, on top of our concrete protocol and the ideas of our construction, an implicit contribution of our work is the observation that adding more parties in threshold signing *without being necessarily in the honest majority regime* has the potential of adding several practical benefits, when coupled with carefully constructed protocols such as ours.

Other Contributions of Potential Independent Interest. In the course of developing our threshold ECDSA protocol, we stumbled upon several challenges that required novel solutions. Some of these can be abstracted in such a way that they may find applications outside the context of our work. These are the following:

- *Range proofs for Shamir-shared values.* In this work we show how to distributed (packed) Shamir-shared sharings $[\boldsymbol{x}]$ such that $\|\boldsymbol{x}\|_\infty < B$, for some bound B. A naive solution for this problem requires the dealer to secret-share the bit decomposition of each entry of $\boldsymbol{x}$. We develop a novel technique that enables such checks without the extra overhead. Crucially, our "range proofs" do not assume a *slack* between what an honest dealer and what can be proven for a potentially corrupt one. This makes the problem much more challenging and the corresponding solution more useful. We refer the reader to Section 2.1 for an overview, and Section 4.3 for the technical details.
- *Distributing Shamir shares under different moduli.* We develop a novel technique to distribute (packed) Shamir-sharings of a vector $\boldsymbol{x}$ under two different moduli p and q, while ensuring the underlying secret is the same in both instances. Similar versions of this primitive exist already, most prominently the case of p a large prime and $q = 2$ (cf. [RW19]), but the more general case has not been studied in prior literature to the best of our knowledge. We provide an overview in Section 2.1, and present the full construction in Section 4.
- *Packed triple generation with constant communication.* Part of our preprocessing functionality generates what can basically be considered a packed triple: sharings of $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$, where $\boldsymbol{c} = \boldsymbol{a} \star \boldsymbol{b}$. A similar functionality exists in Superpack [Esc+23], but their instantiation requires $O(\ell)$ communication. We adapt the virtual party idea by Bracha [Bra87] in order to obtain a protocol with $O(1)$ communication. Note that the work of Bienstock and Yeo [BY24] also explored this problem (using also similar "virtualization ideas"), but they do achieve an overhead of $\kappa\times$.

[8] A few optimizations can be carried out if $t < n/2$. For example, we can avoid the use of OLE in our preprocessing. However, we do not dig deeper into these optimizations.

1.2 Related Work

To our knowledge, no previous work on ECDSA has considered packing multiple signing instances into a single one as we do. Consequently, the cost of signing ℓ messages with any prior threshold ECDSA protocol (even in the honest majority setting) is simply $\ell\times$ the cost of a single signing instance. The closest application of such packing ideas is in the honest majority setting for Schnorr signatures [Ben+24; GS24], which are much simpler than ECDSA. Furthermore, although SPRINT [Ben+24] does consider packing for scaling Schnorr signatures, it does not achieve constant communication per party, independent of the number of signatures. As we will see in the overview in Section 2, this stringent communication requirement poses a difficult challenge for us: we cannot even afford that all parties learn the nonces for all signatures, which means they do not even know what to "hash" in order to obtain the "challenges"! We solve this by having a single party learn the nonce and distribute the hashes, which poses difficulties when it comes to active security.

Protocols for threshold (EC)DSA signing have been studied as far back as when the original DSA was standardized [Lan95]. Works from that era mostly focused on the honest majority setting, which permits an entirely different class of techniques from dishonest majority—we therefore do not comment on subsequent honest majority works either.

More modern protocols for ECDSA have been motivated by the cryptocurrency custody domain, in which the dishonest majority setting is more common. Gennaro et al. [GGN16] and Boneh et al. [BGG19] used Threshold Additively Homomorphic Encryption (AHE) to construct dishonest majority threshold ECDSA signing, as a generalization of an earlier two-party approach by Mackenzie and Reiter [MR01]. While Threshold AHE is appealing for its low communication complexity, computing the setup parameters in a distributed fashion poses a bottleneck, as in their case it implies the notoriously difficult task of sampling a Paillier modulus using MPC.

Consequently, the more practical approach today in most regimes is to make use of pairwise Oblivious Linear Evaluation (OLE) invocations, which do not typically require expensive setup to instantiate. In this regard, the recent work of Doerner et al. [Doe+24] constitutes the state of the art, in that it shows that threshold ECDSA signing can be reduced to executing two parallel OLEs between every pair of parties, with no necessity for external authentication beyond a statistical pairwise check. Different instantiations of the OLE yield different efficiency profiles, for example low bandwidth with Class Groups [Cas+23], low computation with Oblivious Transfer (OT) [Doe+24; Hai+22], or Paillier with its own tradeoffs [Can+20]. We refer the reader to Doerner et al. [Doe+24] for details on their framework, as well as an account of the landscape of threshold ECDSA protocols.

1.3 Notation

Sometimes (particularly in Section 4) we refer to the "*signed* reduction of x modulo N", which means taking the representative in the range $[-\frac{N-1}{2}, \frac{N-1}{2}]$.

We denote by $\langle x \rangle$ additive shares $x = x_1 + \cdots + x_n$. For a positive integer N we write $[N] = \{1, \ldots, N\}$. For a vector $\boldsymbol{x}$, we use $\boldsymbol{x}[i]$ to denote the i-th entry in $\boldsymbol{x}$. We denote by $[\boldsymbol{x}]_d$ packed Shamir shares of the vector $\boldsymbol{x}$ with a degree-d polynomial, and we refer the reader to Section 3.2 in the preliminaries for the basic properties of this scheme. We omit the degree when it is clear from the context. Let G be the generator of a cyclic group $\mathbb{G}$ of order q. We use the notation $\boldsymbol{x} \cdot G$ to represent the vector $(x[1] \cdot G, \ldots, x[n] \cdot G) \in \mathbb{G}^n$. For two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, we use $\boldsymbol{x} \star \boldsymbol{y}$ to denote the component-wise product of $\boldsymbol{x}$ and $\boldsymbol{y}$. In Section 4 we will consider the ℓ_2 norm $\|\boldsymbol{x}\|_2 = \sqrt{\sum_i (\boldsymbol{x}[i])^2}$ and ℓ_∞ norm $\|\boldsymbol{x}\|_\infty = \max_i\{|\boldsymbol{x}[i]|\}$.

2 Overview of our Techniques

Our goal is to produce the ℓ signatures with a communication of $O(n)$, which amounts to $O(1)$ per party, *independently of* ℓ. An important remark is that, if each of the n parties is required to learn all of the ℓ signatures, then it is impossible to achieve this communication since each party must receive at least $\ell = \omega(1)$ messages—that is, the signatures themselves. Instead, we will let the parties get "sharings" of the signatures $([\boldsymbol{\sigma}]_{\ell-1}, [\boldsymbol{r}]_{\ell-1})$, which can be reconstructed towards any intended receiver of the signature.

First, recall that an ECDSA signature consists of $\mathbb{Z}_q$ scalars (r_x, s), where $s = (m + r_x \mathsf{sk})/k$ and r_x is the x-coordinate of nonce $k \cdot G$. We will mostly use packed Shamir shares with a degree $d = n - \ell = t + (\ell - 1)$, which ensures the ℓ secrets are kept private, and $d = \ell - 1$, which is the minimum required to encode ℓ secrets and does not provide any privacy, but the "secrets" are determined entirely by the $n - t > (\ell - 1) + 1$ honest parties, which ensures the worst an adversary can do is abort, but cannot cause reconstruction of an incorrect value. Also, local multiplication of sharings is possible except the degrees of the corresponding sharings are summed up. Most of our contributions are on the offline phase, with the online phase following the template by prior works [Abr+22; Doe+24; LN18]. Intuitively, given shares of $[\boldsymbol{\alpha} = (\boldsymbol{m} + \boldsymbol{r} \star (\mathsf{sk} \cdot \mathbf{1})) \star \boldsymbol{\phi}]_{n-1}$, and shares of $[\boldsymbol{\beta} = \boldsymbol{k} \star \boldsymbol{\phi}]_{n-1}$—where $\boldsymbol{\phi}$ is uniformly random and unknown to any party—one can simply reconstruct $\boldsymbol{s} = \boldsymbol{\alpha}/\boldsymbol{\beta}$. This is secure because $\boldsymbol{\beta}$ can be simulated by random sampling (as $\boldsymbol{\phi}$ is a one-time pad), and $\boldsymbol{\alpha}$ simulated by computing $\boldsymbol{s} \star \boldsymbol{\beta}$. In combination with the corresponding public nonces $\boldsymbol{r}$, the complete signatures can be assembled. Also observe that $\boldsymbol{\alpha}, \boldsymbol{\beta}$ can be reconstructed by simply having honest parties send their respective shares without any further authentication—assuming that parties hold a *valid* secret sharing of $\boldsymbol{\alpha}, \boldsymbol{\beta}$, any adversarial offset to the reconstruction process yields a predictable outcome, moreover the correct output $\boldsymbol{\sigma}$ is a verifiable object.

We identify that preprocessed correlated randomness of the format

$$\langle a \rangle, [\boldsymbol{\phi}]_{n-\ell}, [a \cdot \boldsymbol{\phi}]_{n-\ell}, [\boldsymbol{k} \star \boldsymbol{\phi}]_{n-\ell}, a \cdot G, [\boldsymbol{k} \cdot G]_{\ell-1}, [\boldsymbol{r}]_{\ell-1},$$

—where $a \in \mathbb{Z}_q$ is a uniformly random value—is conducive to deriving $[\boldsymbol{\alpha}]_{n-1}$ and $[\boldsymbol{\beta}]_{n-1}$ easily in the online phase. Note that unlike standard ECDSA signing

in which $k \cdot G$ is revealed to the adversary only after it chooses m, executing the preprocessing functionality as described here in an offline phase allows the adversary to choose m as a function of $k \cdot G$. This is referred to as the "pre-signature" mode of operation in the literature [Can+20; GS22], and Groth and Shoup proved in the generic group model that it does not noticeably harm security. Nonetheless, the conservative user may wish to avoid pre-signatures altogether; our efficiency claims hold even with $\mathcal{F}_{\mathsf{Prep}}$ invoked online after the messages are specified. Again, we emphasize that degree-$(\ell-1)$ sharings do not provide any privacy and they are just intended to efficiently distribute *public values*. We discuss how to generate this preprocessing later, and for now, focus on how it can be useful to compute $[\boldsymbol{\alpha}]_{n-1}$ and $[\boldsymbol{\beta}]_{n-1}$.

Note that $[\boldsymbol{\beta}]_{n-1} = [\boldsymbol{k} \star \boldsymbol{\phi}]_{n-1}$ already comes from the preprocessing directly, so we can focus on $[\boldsymbol{\alpha}]_{n-1}$ only. Note that $\boldsymbol{\alpha} = \boldsymbol{m} \star \boldsymbol{\phi} + \boldsymbol{r} \star (\mathsf{sk} \cdot \mathbf{1}) \star \boldsymbol{\phi}$. First, the parties obtain shares of the product $[\mathsf{sk} \cdot \boldsymbol{\phi}]_{n-\ell}$ by using their secret-key sharings $\langle \mathsf{sk} \rangle$, and the preprocessing $\langle a \rangle, [\boldsymbol{\phi}]_{n-\ell}, [a \cdot \boldsymbol{\phi}]_{n-\ell}$. This is done as follows, which is inspired in [Esc+23]. The parties first compute $\langle d \rangle = \langle \mathsf{sk} \rangle - \langle a \rangle$ and open d, which leaks nothing about sk since a is secret and random. Then the parties compute locally $[\mathsf{sk} \cdot \boldsymbol{\phi}]_{n-\ell} = d \cdot [\boldsymbol{\phi}]_{n-\ell} + [a \cdot \boldsymbol{\phi}]_{n-\ell}$. Continuing with the computation of $[\boldsymbol{\alpha}]_{n-1}$, we let the parties execute a protocol that derives $[\boldsymbol{r}]_{\ell-1}$ from $[\boldsymbol{k} \cdot G]_{\ell-1}$, which not only is useful towards obtaining $\boldsymbol{\alpha}$, but is also the second component of the required signatures.[9] This part turns out to be quite challenging, as addressed in this paper. Once this is done, the parties can locally compute $[\boldsymbol{\alpha}]_{n-1} = [\boldsymbol{m}]_{\ell-1} \cdot [\boldsymbol{\phi}]_{n-\ell} + [\boldsymbol{r}]_{\ell-1} \cdot [\mathsf{sk} \cdot \boldsymbol{\phi}]_{n-\ell}$, as required.

We devote the rest of this section to discussing how this correlated randomness can be generated. We begin with our first major challenge, which is to generate $[\boldsymbol{r}]_{\ell-1}$ that is consistent with $[\boldsymbol{k} \cdot G]_{\ell-1}$.

2.1 Generating $[\boldsymbol{r}]_{\ell-1}$ from $[\boldsymbol{k} \cdot G]_{\ell-1}$.

Let $\boldsymbol{r}[j]$ be the x-coordinate of the elliptic curve point $\boldsymbol{k}[j] \cdot G$. As such, $\boldsymbol{r}[j]$ lives in $\mathbb{F}_q = \mathbb{Z}_q$, the finite field over which the elliptic curve itself is defined. However, in ECDSA this is taken modulo p (recall p is the order of the curve), which happens implicitly when we write $[\boldsymbol{r}]_{\ell-1}$ since these sharings are over $\mathbb{Z}_p$. For the sake of clarity let us write explicitly $[\boldsymbol{r} \bmod p]_{\ell-1}$, which is what we intend to compute. We first note that $k \cdot G$ is *public*, and in fact, if ℓ was equal to 1 (that is, if there was no packing involved), then this process would be trivial: the parties all have $k \cdot G$ and they simply derive r as the x-component of this point. The challenge in our setting lies on the fact that $\boldsymbol{k} \cdot G$ is distributed among the parties, not for privacy, but for efficiency purposes, and the parties must somehow distribute $[\boldsymbol{r} \bmod p]_{\ell-1}$. The main issue here is that $\boldsymbol{k} \cdot G \mapsto \boldsymbol{r} \bmod p$ is a highly non-linear operation, involving two different domains $\mathbb{Z}_p$ and $\mathbb{Z}_q$, and involving non-black-box access to the elliptic curve.

[9] We note that the generation of $[\boldsymbol{r}]_{\ell-1}$ can also be considered preprocessing as it is independent of sk and $\boldsymbol{m}$, but due to its complexity and potential uses in other contexts, we isolate it.

The above must be achieved without paying more than $O(\ell)$ communication in total, which means we cannot afford that all parties learn $\boldsymbol{k} \cdot G$ entirely. However, we can allow *one* party, say P_1, to learn $\boldsymbol{k} \cdot G$ entirely, compute $\boldsymbol{r}$ locally, and then distribute $[\boldsymbol{r} \bmod p]_{\ell-1}$ back to the parties. The main challenge with this approach is achieving active security: how do we ensure that a potentially malicious P_1 performs this distribution correctly? We emphasize that the difficulty stems from the non-linear nature of the mapping $\boldsymbol{k} \cdot G \mapsto \boldsymbol{r} \bmod p$. We discuss this below.

An insecure method: trust P_1. It is tempting to allow P_1 to have full control over $\boldsymbol{r}$, and simply hope for the best. After all, if P_1 distributes an incorrect $\boldsymbol{r}^*$, then the signing protocol will simply result in non-verifying signatures. In particular, each "signature" will take the form $s^* = (m + r^* \cdot \mathsf{sk})/k$, with the difference from an actual signature being that the adversarially chosen r^* may be completely unrelated to $k \cdot G$. Given that the adversary is still given only one chance to specify the usage of the uniformly random nonce k, it may appear at first as though s^* is simply a useless one-time pad encryption of sk. However, allowing the adversary full freedom in choosing r^* opens up an avenue to forge a signature on an unauthorized message m^*. The attack works as follows: let $\delta = m/m^*$. The adversary—who controls P_1—is given r from the protocol, and rather than share it correctly, shares $r^* = \delta \cdot r$. This results in the protocol returning $s^* = (m + r^* \cdot \mathsf{sk})/k = (\delta m^* + \delta \cdot r \cdot \mathsf{sk})/k$ to the adversary. Observe that s^*/δ is a *valid* ECDSA signature on m^*, which can be chosen completely arbitrarily by the adversary, without the consent of the other parties in the MPC. This is clearly an unacceptable attack, and we must work to mitigate it.

Checking consistency of $[\boldsymbol{k \cdot G}]_{\boldsymbol{\ell-1}}$ and $[\boldsymbol{r} \bmod \boldsymbol{p}]_{\boldsymbol{\ell-1}}$. Consider that multiple such sharings as above have been distributed, and our goal is to check their consistency. Let us denote by $[\cdot]^{(N)}$ packed sharings modulo N, so the setting is that the parties have $\{[\boldsymbol{k}_i \cdot G]^{(p)}_{\ell-1}\}_{i=1}^d$ and $\{[\boldsymbol{r}_i \bmod p]_{\ell-1}\}_{i=1}^d$ for some large d, and they want to check consistency between these sharings. Here, consistency means that $\boldsymbol{r}_i[j] \bmod p$ is the x-component of the EC point $\boldsymbol{k}_i[j] \cdot G$ for all $i \in [d]$ and $j \in [\ell]$ (recall that degree-$(\ell-1)$ sharings are fully determined by the honest parties, so this statement is well defined). The biggest challenge here is that $\boldsymbol{r}_i[j]$ lives in $\mathbb{Z}_q$, and in fact EC arithmetic happens over $\mathbb{Z}_q$, but the parties have sharings modulo p. To address this, we ask P_1 to not only distribute $[\boldsymbol{r}_i \bmod p]^{(p)}_{\ell-1}$, but also share $[\boldsymbol{r}_i]^{(q)}_{\ell-1}$. The fact that the latter sharings are modulo q will allow us to check the EC arithmetic equations (which are defined over $\mathbb{Z}_q$). In a bit more detail, we sample random coefficients $\gamma_1, \ldots, \gamma_d \in \mathbb{Z}_p$ and open to all the parties the linear combination $\boldsymbol{k} \cdot G = \sum_{i=1}^d \gamma_i \cdot (\boldsymbol{k}_i \cdot G)$, which we can afford since it is only *one* vector instead of d. The goal is to let the dealer somehow "prove" that, if the parties were to follow the EC arithmetic equations on $[\boldsymbol{r}_i]^{(q)}_{\ell-1}$

for $i \in [d]$,[10] they would obtain the x-component of the public vector $\boldsymbol{k} \cdot G$. For this, we perform the linear combination using the same coefficients $(\gamma_i)_{i=1}^d$ but using the elliptic curve points $(\boldsymbol{r}_i, \cdot)$ with elliptic curve addition operation (here, we also distribute $(\ell-1)$-degree shares of the y-coordinate of the point as it is a public value). Finally, we open the resulting linear combination of points and compare its x-coordinate with the one of $\boldsymbol{k} \cdot G$. If both set of values are consistent, they should arrive to the same linear combination result with overwhelming probability. We refer the reader to Section 6 to see how this "proof" is done.

Checking consistency of $[\boldsymbol{r_i} \bmod \boldsymbol{p}]_{\boldsymbol{\ell-1}}^{(\boldsymbol{p})}$ and $[\boldsymbol{r_i}]_{\boldsymbol{\ell-1}}^{(\boldsymbol{q})}$. The above approach is worthless if P_1 can distribute sharings $[\boldsymbol{r}_i \bmod p]_{\ell-1}^{(p)}$ and $[\boldsymbol{r}_i]_{\ell-1}^{(q)}$ that are not consistent (that is, using different secrets for each sharing), since this would mean that the check we perform over $\mathbb{Z}_q$ regarding the EC equations does not translate into anything for the sharings $[\boldsymbol{r}_i \bmod p]_{\ell-1}^{(p)}$ we will actually use in our signatures. For this, we develop a novel method that allows a dealer to secret-share a vector modulo two moduli p and q, ensuring the same secret is used in both instances. We present this within the context of our work, but it can be easily generalized (*e.g.* to use non-packed secret-sharing with a higher threshold that offers privacy), and we believe this may find applications in other contexts (for instance, share conversions between different fields $\mathbb{F}_p$ and $\mathbb{F}_q$). Our approach is as follows: we instruct the dealer to secret-share $[\boldsymbol{r}_i]_{\ell-1}^{(P)}$, where P is a large modulus (with no small factors) that satisfies $pq \mid P$. Given these shares, parties can obtain $[\boldsymbol{r}_i \bmod p]_{\ell-1}^{(p)}$ and $[\boldsymbol{r}_i \bmod q]_{\ell-1}^{(q)}$ by local reduction modulo p and q respectively, which is nearly our target, with the exception being that $\boldsymbol{r}_i \bmod q$ may not match $\boldsymbol{r}_i$ if it wraps around modulo q.

To prevent wraparound, we perform a check that ensures that the secrets underlying $[\boldsymbol{r}_i]_{\ell-1}^{(P)}$ sastisfy $0 \leq \boldsymbol{r}_i[j] < q$ for all $i \in [d], j \in [\ell]$. By shifting $[\boldsymbol{z}_i]_{\ell-1}^{(P)} = [\boldsymbol{r}_i]_{\ell-1}^{(P)} - \frac{q-1}{2} \cdot \mathbf{1}$, this translates into $\|\boldsymbol{z}_i\|_\infty < \frac{q+1}{2}$. Hence, we have reduced the problem to a form of *range proof*: distributing sharings whose infinity norm fall within certain bound. We discuss our solution to this below.

Distributing $[\boldsymbol{z_i}]_{\boldsymbol{\ell-1}}^{(\boldsymbol{P})}$ such that $\|\boldsymbol{z_i}\|_\infty < \boldsymbol{B}$. We only provide a high level idea, referring the reader to Section 4 for details. We note that this primitive may also have applications beyond our work. Overall, our approach consists of taking random linear combinations of the $[\boldsymbol{z}_i]_{\ell-1}^{(P)}$ for $i \in [d]$ using carefully chosen random coefficients that do not blow up the size of the $\boldsymbol{z}_i$'s by much, but also prevent large values from becoming smaller. The parties open these linear combinations and check that they are indeed small. This way, a corrupt dealer attempting to distribute large values gets caught with high probability. However, in these methods there is usually a *slack* between what an honest dealer can prove, and what can be inferred for a potentially corrupt dealer. Due to this,

[10] Here it is important to mention that the x-component of an elliptic curve point determinens the y-component.

this approach would not work directly for us since it may be the case that the norms $\|\boldsymbol{z}_i\|_\infty$ are very close to B, and an honest dealer would not be able to prove this unless we accept corrupt dealers distributing *slightly larger* values, but this would already break the security of our overall construction.

To deal with this issue, we draw inspiration from [GHL22]. An alternative way of checking that $\|\boldsymbol{z}_i\|_\infty < B$ is checking the existence of $\boldsymbol{\alpha}_{ij}$ for $j \in [4]$ such that $(B-1)^2 \cdot \mathbf{1} - \boldsymbol{z}_i^2 = \boldsymbol{\alpha}_{i1}^2 + \boldsymbol{\alpha}_{i2}^2 + \boldsymbol{\alpha}_{i3}^2 + \boldsymbol{\alpha}_{i4}^2$, where this equality holds *over the integers*. Therefore, we let the prover distribute $[\boldsymbol{\alpha}_{ij}]_{\ell-1}^{(P)}$ for $j \in [4]$, and then the parties check the equation mentioned above over these shares: they locally compute $[\boldsymbol{o}_i]_{2(\ell-1)}^{(P)} = (B-1)^2 \cdot \mathbf{1} - [\boldsymbol{z}_i]_{\ell-1}^{(P)} \cdot [\boldsymbol{z}_i]_{\ell-1}^{(P)} - \sum_{j=1}^{4} [\boldsymbol{\alpha}_{ij}]_{\ell-1}^{(P)} \cdot [\boldsymbol{\alpha}_{ij}]_{\ell-1}^{(P)}$, sample random coefficients $\beta_1, \ldots, \beta_d \in \mathbb{Z}_P$, and reconstruct $[\boldsymbol{o}]_{2(\ell-1)}^{(P)} = \sum_{i=1}^{d} \beta_i \cdot [\boldsymbol{o}_i]_{2(\ell-1)}^{(P)}$, checking that $\boldsymbol{o} = \mathbf{0}$.[11] The above shows that the desired equations holds, but *only modulo* P, whereas we need them to hold *over the integers* to be able to conclude $\|\boldsymbol{z}_i\|_\infty < B$. For this we revisit the initial approach we discussed, which allows an honest dealer to prove certain bound which is slightly smaller than what can be guaranteed for a corrupt dealer. We use these techniques in order to ensure the sharings of all the $\boldsymbol{z}_i$'s and the $\boldsymbol{\alpha}_{ij}$'s are below a small-enough bound that ensures there is no wrap around modulo P in the quadratic equations above—note that we can choose P as large as we need. Observe that here we can indeed afford a slack, since we choose this bound to be larger than the bound B we ultimately intend to prove. We provide the details of this approach in Section 4.

2.2 Generating the preprocessing.

Now we discuss how to generate the preprocessing values. We generate a batch of λ such values, that is, $(\langle a_j \rangle, [\boldsymbol{\phi}_j]_{n-\ell}, [a_j \cdot \boldsymbol{\phi}_j]_{n-\ell}, [\boldsymbol{k}_j \star \boldsymbol{\phi}_j]_{n-\ell}, a_j \cdot G, [\boldsymbol{k}_j \cdot G]_{\ell-1})$, where $\langle a_j \rangle$ denotes additive sharing of a_j, for $j \in [\lambda]$. We define the functionality $\mathcal{F}_{\mathsf{Tuple}}$ to generate the above values, and to realize it, we first consider a scenario where there are N *virtual parties* $V_1, \ldots, V_N$ that help $P_1, \ldots, P_n$ (see Section 5). We assume that the number of corrupted parties among $V_1, \ldots, V_N$ is bounded by $(N+1)/2 - \lambda$. Later on we discuss how to concretely instantiate these virtual parties.

Preparing $(\langle \boldsymbol{a_j} \rangle, [\boldsymbol{k_j}]_{\boldsymbol{n-1}}, [\boldsymbol{\phi_j}]_{\boldsymbol{n-1}})$. For all $j \in [\lambda]$, all parties sample random values locally as their shares of $(\langle a_j \rangle, [\boldsymbol{k}_j]_{n-1}, [\boldsymbol{\phi}_j]_{n-1})$. Note that this works since, as observed in [Esc+23], degree-$(n-1)$ sharings of random secrets simply correspond to random uncorrelated shares.

Preparing $([\boldsymbol{\phi_j}]_{\boldsymbol{n-\ell}}, [\boldsymbol{a_j} \cdot \boldsymbol{\phi_j}]_{\boldsymbol{n-\ell}}, [\boldsymbol{k_j} \star \boldsymbol{\phi_j}]_{\boldsymbol{n-1}})$. This step consists of multiplying the shares generated earlier. Unfortunately, we have a dishonest majority,

[11] Here it is useful to recall that sharings of degree $2(\ell-1)$ are fully determined by the honest parties and hence can be reconstructed without errors.

and it is unclear how to achieve secure multiplication in this setting with constant communication. For instance, Superpack [Esc+23] only achieves this for the online phase, requiring linear communication in the offline phase.[12] To address this we will use packed secret-sharing with packing parameter λ among the N virtual parties, and let these parties—which contain a super-honest majority—handle the multiplication.

In more detail, recall that all parties have prepared $(\langle a_j \rangle, [\boldsymbol{k}_j]_{n-1}, [\boldsymbol{\phi}_j]_{n-1})$ for all $j \in [\lambda]$. Let us denote by $\boldsymbol{a}^{(i)}$ the vector of P_i's shares of $\langle a_1 \rangle, \ldots, \langle a_\lambda \rangle$. Each P_i generates a degree-$(N-1)/2$ packed Shamir sharing of their shares of $\langle a_1 \rangle, \ldots, \langle a_\lambda \rangle$, denoted by $[\boldsymbol{a}^{(i)}]^V_{(N-1)/2}$, and sends shares to the virtual parties $V_1, \ldots, V_N$. Then all virtual parties can locally compute $[\boldsymbol{a}]^V_{(N-1)/2} = \sum_{i=1}^n [\boldsymbol{a}^{(i)}]_{(N-1)/2}$, where $\boldsymbol{a} = (a_1, \ldots, a_\lambda)$. Similarly, each party P_i generates a degree-$(N-1)/2$ packed Shamir sharing of their shares of $[\boldsymbol{k}_1]_{n-1}, \ldots, [\boldsymbol{k}_\lambda]_{n-1}$, denoted by $[\boldsymbol{k}^{(i)}]^V_{(N-1)/2}$, and sends shares to the virtual parties $V_1, \ldots, V_N$. It is not hard to see that, by combining these sharings with adequate Lagrange coefficients, the virtual parties can locally compute $[\boldsymbol{b}_1]^V_{(N-1)/2}, \ldots, [\boldsymbol{b}_\ell]^V_{(N-1)/2}$, where $\boldsymbol{b}_i[j] = \boldsymbol{k}_j[i]$ for $i \in [\ell], j \in [\lambda]$. Following the same steps, the virtual parties also obtain $[\boldsymbol{\psi}_1]^V_{(N-1)/2}, \ldots, [\boldsymbol{\psi}_\ell]^V_{(N-1)/2}$, where $\boldsymbol{\psi}_i[j] = \boldsymbol{\phi}_j[i]$ for $i \in [\ell], j \in [\lambda]$.

Now all virtual parties compute locally $[\boldsymbol{a} \star \boldsymbol{\psi}_j]^V_{N-1} = [\boldsymbol{a}]^V_{(N-1)/2} \cdot [\boldsymbol{\psi}_j]^V_{(N-1)/2}$ and $[\boldsymbol{b}_j \star \boldsymbol{\psi}_j]^V_{N-1} = [\boldsymbol{b}_j]^V_{(N-1)/2} \cdot [\boldsymbol{\psi}_j]^V_{(N-1)/2}$, for $j \in [\ell]$. This is precisely where we exploit the low degree $(N-1)/2$. Once this product is performed, the virtual parties can "re-share" these results towards the actual parties in a similar way as outlined above—but in the "opposite direction". Concretely, each virtual party V_i generates a random degree-$(n-\ell)$ sharing of their shares of $[\boldsymbol{\psi}_1]^V_{(N-1)/2}, \ldots, [\boldsymbol{\psi}_\ell]^V_{(N-1)/2}$, denoted by $[\boldsymbol{\psi}^{(i)}]_{n-\ell}$, and sends these shares to the parties $P_1, \ldots, P_n$. Then these parties can locally compute $[\boldsymbol{\phi}_1]_{n-\ell}, \ldots, [\boldsymbol{\phi}_\lambda]_{n-\ell}$. Following similar steps, all parties obtain $([a_j \cdot \boldsymbol{\phi}_j]_{n-\ell}, [\boldsymbol{k}_j \star \boldsymbol{\phi}_j]_{n-1})$ for all $j \in [\lambda]$.

Preparing $(a_j \cdot G, [\boldsymbol{k}_j \cdot G]_{\ell-1})$. Recall that, above, all parties have prepared $(\langle a_j \rangle, [\boldsymbol{k}_j]_{n-1})$. All parties locally multiply by G, and send their shares of $\langle a_j \cdot G \rangle, [\boldsymbol{k}_j \cdot G]_{n-1}$ to P_1. After receiving shares from all parties, P_1 reconstructs $a_j \cdot G, \boldsymbol{k}_j \cdot G$ and distributes $a_j \cdot G, [\boldsymbol{k}_j \cdot G]_{\ell-1}$ to all parties.

Verification. So far, for all $j \in [\lambda]$, all parties have prepared $(\langle a_j \rangle, [\boldsymbol{\phi}_j]_{n-\ell}, [a_j \cdot \boldsymbol{\phi}_j]_{n-\ell}, [\boldsymbol{k}_j \star \boldsymbol{\phi}_j]_{n-1}), a_j \cdot G$, $[\boldsymbol{k}_j \cdot G]_{\ell-1}$. To verify the multiplication relations, all parties locally compute $[a_j \cdot \boldsymbol{\phi}_j \cdot G]_{n-\ell}$ in two different ways: one from $a_j \cdot G$ and $[\boldsymbol{\phi}_j]_{n-\ell}$, and the other from G and $[a_j \cdot \boldsymbol{\phi}_j]_{n-\ell}$. Then the parties check that these secrets match. A similar check is done for $[\boldsymbol{k}_j \star \boldsymbol{\phi}_j]_{n-1}$.

[12] In [Esc+23] the offline phase costs linearly due to two factors: (1) the generation of triples and (2) certain per-wise masks that are required for permuting the wires. The ideas described here address the first issue in [Esc+23].

Instantiating the virtual Parties. It remains to be discussed how these virtual parties $V_1, \ldots, V_N$ are instantiated. We use the virtual parties idea from Bracha [Bra87], where each V_i will be jointly emulated by a small committee of size s sampled at random among the set of (real) parties. Each committee will run a simple dishonest majority protocol to emulate the corresponding virtual party, and hence a virtual party will be "corrupted" only if all of the parties in the committee are corrupt. As we show in Section 5.1, we can set parameters in such a way that, with high probability, there will be a super-honest majority among the virtual parties, as required in our ideas outlined above. Furthermore, the "dishonest-majority protocol" each committee uses is simple and guarantees that communication remains within our desired bounds. For secure multiplications within these committees we make use of a secure multiplication functionality $\mathcal{F}_{\mathsf{Mult}}$. Importantly, this is the *only* place in our whole solution that requires PKC-style of assumptions (since the other parts require, at most, PRGs and hash functions).

3 Preliminaries

We consider a setting where a certain amount of parties, t of which may be corrupt, hold Shamir shares of degree t of an ECDSA secret key sk. At signing time, a committee $P_1, \ldots, P_n$ of size $n = t + 2(\ell - 1) + 1$ for some $\ell \geq 1$ generates ℓ signatures. Note that the committee can locally obtain additive shares of sk from their Shamir shares. From now on, we abstract the problem in this way: n parties who hold additive shares of sk wish to compute ℓ ECDSA signatures, and we forget about the initial set of parties.

Security Definitions. We consider an *active* adversary that corrupts t out of the n parties. Recall that it may be the case that $t \geq n/2$. We aim for *security with abort*, and therefore the adversary can cause an abort at any point when interacting with a functionality. We do not write this explicitly in the description of our functionalities. We model security in the standard universal composability framework by Canetti [Can01]. We achieve "information-theoretic" security in our work in spite of having a dishonest majority. In this work we make use of the following functionalities which are not instantiated here:

- $\mathcal{F}_{\mathsf{Coin}}$ (Sampling random coins): We consider a functionality $\mathcal{F}_{\mathsf{Coin}}$ that samples a random string of arbitrary size and distributes it to the parties. This string will be interpreted differently in different instances, and this will be clear from context. See Section 3.1 for a discussion on its instantiation. In short, we can ignore its cost when analyzing communication complexity.
- $\mathcal{F}_{\mathsf{Mult}}$ (Secure dishonest majority multiplication) We assume a secure multiplication functionality that works as follows. Let V be the set of parties. $\mathcal{F}_{\mathsf{Mult}}$ receives the shares of $\langle a \rangle, \langle b \rangle$ from all parties in V. Then it reconstructs a, b and computes $c = a \cdot b$. To prepare $\langle c \rangle$, $\mathcal{F}_{\mathsf{Mult}}$ receives the shares of corrupted parties from the ideal adversary $\mathcal{S}$. Then $\mathcal{F}_{\mathsf{Mult}}$ randomly samples an additive

sharing $\langle c \rangle$ among V given the shares of corrupted parties and the secret c. Finally, $\mathcal{F}_{\mathsf{Mult}}$ distributes the shares of $\langle c \rangle$ to all parties.
This primitive is standard in dishonest majority MPC and requires computational assumptions. There are several instantiations, and we can make use of any, as long as it enjoys linear communication on $|V|$. We choose the instantiation based on pseudo-random correlator generators (PCGs) from Le Mans [RS22] to generate random triples with sublinear communication (which we ignore), and then de-randomizing them with linear communication. Concretely the communication of the de-randomization is the communication of Beaver's multiplication protocol: $4|V|$ elements. In Section 7.1, we also present the communication for the case of a more general instantiation of $\mathcal{F}_{\mathsf{Mult}}$. The functionality $\mathcal{F}_{\mathsf{Mult}}$ can be instantiated using several techniques, and this choice does not affect the benefits of our protocol. Another possible instantiation is to use generic approaches based on *oblivious transfer* (OT). Although we get the same overall improvement with both approaches, using OT-based multiplication will be slower than the PCG alternative. We use pairwise PCGs because they have simple, concrete constructions, and currently, we do not know of PCGs for generating n-wise random correlations.

– $\mathcal{F}_{\mathsf{Commit}}$ (Cryptographic commitments): We assume a commitment functionality to commit to an element in $\mathbb{Z}_p$, which will be used in the preprocessing stage. The functionality guarantees two properties: the committer can not change the value that is committed after sending the commitment (binding), and the commitment does not reveal information about the committed message (hiding).
A standard realization of the commitment functionality works as follows: let $P_1, \ldots, P_n$ be parties such that each party has a message $m_i \in \mathbb{Z}_p$ to be committed. Each party P_i samples a random element $r_i \in \{0,1\}^\kappa$, computes the commitment $c_i \stackrel{\mathsf{def}}{=} H(m_i \| r_i)$, and sends c_i to the rest of the parties. To open the commitments, each party P_i sends the opening value (m'_i, r'_i) to the rest of the parties. Once each party P_i receives (m'_j, r'_j) for $j \in [n] \setminus \{i\}$, it checks that $c_i = H(m'_i \| r'_i)$. If the checks pass, then P_i can assume $\{m'_j\}_{j \in [n] \setminus \{i\}}$ to be the committed messages. In our preprocessing, the commitment will be used mainly for each party to commit to its share by all of them calling the functionality $\mathcal{F}_{\mathsf{Commit}}$. Hence, if all parties have a share of a value, the commitment of each party to its share requires sending $n^2 \cdot \kappa$ bits. Also, opening a commitment requires sending n^2 elements in $\mathbb{Z}_p$ plus $n^2 \cdot \kappa$ bits.

3.1 Instantiating $\mathcal{F}_{\mathsf{Coin}}$ Efficiently

In our different protocols we call $\mathcal{F}_{\mathsf{Coin}}$ to sample sometimes relatively large strings. A natural instantiation of $\mathcal{F}_{\mathsf{Coin}}$ involves each party P_i committing to some r_i, then opening the commitment and letting the random value be $r = \sum_{i=1}^{n} r_i$. This, however, costs communication linearly in the size of the string, which is too much for our purposes. A solution for this is to consider a different instantiation

where the parties use the above procedure to sample only a small seed, which is expanded then with a PRG to obtain a longer string $r \leftarrow \mathsf{PRG}(\mathsf{seed})$. Sadly, this is *not* in general a secure instantiation of $\mathcal{F}_{\mathsf{Coin}}$: the simulator would need to "explain" every random string with the PRG, which is not possible. Fortunately, for the specific uses of $\mathcal{F}_{\mathsf{Coin}}$ in our work, this approach does not break security of our protocols. That is: our protocols remain secure if we modify the calls to $\mathcal{F}_{\mathsf{Coin}}$ by (1) calling $\mathcal{F}_{\mathsf{Coin}}$ to sample a *small* seed, and (2) using a PRG to expand it.

To account for this, we would need to replace all calls to $\mathcal{F}_{\mathsf{Coin}}$ by the seed-then-expand approach, which on top of complicating our protocols while adding no extra value, also makes the proofs more cumbersome. Due to this, in order to remain fully provably secure and efficient while avoiding incorporating the PRG into our protocols, we choose a different instantiation of $\mathcal{F}_{\mathsf{Coin}}$ that is provably secure, and is efficient. In short: the parties use the naive protocol from above to get a small seed, which then they expand not by using a PRG, but by inputting the seed into a *random oracle*. The RO abstraction allows us to overcome the concrete limitations of the PRG approach. A simulator that shows that this is secure in the RO-hybrid model would work as follows. The simulator emulates the seed sampling protocol, obtaining seed. Then it calls the $\mathcal{F}_{\mathsf{Coin}}$ functionality to get the output random string r. The simulator emulates RO internally, and with high probability r has not been returned as an answer previously. Therefore the simulator can *program* the RO by mapping seed to r.

3.2 Packed Shamir Secret-Sharing

Given $\boldsymbol{x} \in \mathbb{Z}_p^\ell$, we use $[\boldsymbol{x}]_d$ to denote the vector $(f(1), \ldots, f(n))$ where each party P_i has $f(i)$, $f(\mathtt{X}) \in \mathbb{Z}_p[\mathtt{X}]$ is a polynomial over $\mathbb{Z}_p$ of degree $\leq d$, and $f(-j) = x_j$ for $j \in [\ell]$.[13] We will mostly use two degrees d: $d = \ell - 1$, which is the minimum to interpolate any vector $\boldsymbol{x} \in \mathbb{Z}_p^\ell$, and $d = t + (\ell - 1) = n - \ell$, which ensures the t shares held by the corrupt parties look random and independent of the secret as long as (any) t of the evaluation points are chosen at random. Note that $[\boldsymbol{x}]_{d_1}$ and $[\boldsymbol{y}]_{d_2}$ can be locally added to obtain $[\boldsymbol{x} + \boldsymbol{y}]_{\max\{d_1, d_2\}}$, and locally multiplied to obtain $[\boldsymbol{x} \star \boldsymbol{y}]_{d_1 + d_2}$.

Note that for a sharing $[\boldsymbol{x}]_d$ with $d \leq 2(\ell - 1)$, $\boldsymbol{x}$ is determined by any $2(\ell - 1) + 1$ shares, which in particular include ones by the $n - t = 2(\ell - 1) + 1$ honest parties. For such d, when we write "parties reconstruct $[\boldsymbol{x}]_d$" we mean that they all send their shares to each other, and each party checks if the resulting shares are consistent with a polynomial of degree $\leq d$, aborting if this is not the case. Due to the above discussion, the adversary cannot cause reconstruction to an incorrect vector.

Sometimes, we use $[\cdot]_d^{(p)}$ to emphasize the modulus p. We state some properties of packed Shamir secret-sharing that are useful in the realization of our protocol:

[13] We are not using 0 as an evaluation point for notational simplicity. This is not a problem in our work where the modulus is considerably larger than the number of required evaluation points.

– Let $R, P \in \mathbb{N}$ be arbitrary values. Suppose the parties have sharings $[\boldsymbol{x}]_d^{(P)}$, and suppose that R is such that $R \mid P$ and any non-zero difference among the points $-\ell, \ldots, -1, 1, \ldots, n$ is invertible modulo R. Then, locally reducing each share modulo R leads to shares $[\boldsymbol{x} \bmod R]_d^{(R)}$. In our particular case, we will use $R = pq$, where p, q are given by the ECDSA protocol, and P will be a modulus we can choose so that $pq|P$.
– Given $[\boldsymbol{x}]_d$, the parties can obtain "Shamir-like" shares of $\boldsymbol{x} \cdot G$ by locally multiplying their shares of $\boldsymbol{x}$ by G. We denote this by $[\boldsymbol{x} \cdot G]_d$. The basic properties of Shamir secret-sharing hold: local addition is possible, local multiplication by a sharing $[\boldsymbol{y}]_{d'}$ is possible, except the degrees d and d' are added up, and $\boldsymbol{x} \cdot G$ can be reconstructed from any $\geq d + 1$ shares.
– In several parts of the paper we will need a dealer to distribute sharings $[\boldsymbol{x}]_{\ell-1}$ of degree $\ell - 1$. The procedure $\pi_{\mathsf{DistShares}}$ presented next guarantees that all parties receive consistent shares with a polynomial of degree $\ell - 1$.

Procedure 1: $\pi_{\mathsf{DistShares}}$

Input: A designated party refered to as the *dealer* has $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d \in \mathbb{Z}_p^\ell$.
Output: The parties obtain sharings $[\boldsymbol{x}_1]_{\ell-1}, \ldots, [\boldsymbol{x}_d]_{\ell-1}$ of degree $\ell - 1$.

1. For each $i \in [d]$, the dealer interpolates the unique polynomial f_i of degree $\ell - 1$ such that $f_i(-j) = x_{ij}$ for $j \in [\ell]$, and sends $f_i(h)$ to party P_h for $h \in [n]$. Denote these sharings by $[\boldsymbol{x}_i]_*$ (the $*$ emphasizes that the degree has not been checked yet).
2. The parties call $\mathcal{F}_{\mathsf{Coin}}$ to obtain random values $r_1, \ldots, r_d \in \mathbb{Z}_p$
3. The parties compute locally $[\boldsymbol{y}]_* = \sum_{i=1}^{d} r_i [\boldsymbol{x}_i]_*$
4. The parties reconstruct $[\boldsymbol{y}]_*$ by sending to each other their shares.
5. Each party checks that the shares they received is consistent with a polynomial of degree $\leq \ell - 1$. If this is not the case, they abort.

We state the following without proof, as the proof is standard an in fact similar reasonings will appear in other proofs throughout this work. We note that this holds for any modulus whose smallest prime factor is in the order of 2^κ.

Proposition 1. *If* $\pi_{\mathsf{DistShares}}$ *does not abort then each* $[\boldsymbol{x}_i]_*$ *has degree* $\leq \ell - 1$.

Simulating $\pi_{\mathsf{DistShares}}$—which will be done in multiple places on the paper—is straightforward: the simulator receives the shares for the honest parties from the adversary, and emulates $\mathcal{F}_{\mathsf{Coin}}$ by sampling the coefficients $r_1, \ldots, r_d$, and reconstructs $[\boldsymbol{y}]$ towards the adversary. If the degree is larger than $\ell - 1$, then both real and ideal worlds will result in abort with overwhelming probability.

4 Distributing Dual Moduli Shares

Recall that we work over elliptic curves of order p defined over $\mathbb{Z}_q$. One of the primitives needed in our protocol is the following: a dealer (P_1) must secret-share a vector $\boldsymbol{x} \in \mathbb{Z}_q^\ell$ both as $[\boldsymbol{x} \bmod p]_{\ell-1}^{(p)}$ and $[\boldsymbol{x}]_{\ell-1}^{(q)}$, that is, using packed Shamir

secret-sharing using moduli p and q. The way we achieve this is by secret-sharing $\boldsymbol{x}$ with Shamir SS over $\mathbb{Z}_P$, where $pq \mid P$, and then taking modulo p and q to obtain $[\boldsymbol{x} \bmod p]^{(p)}_{\ell-1}$ and $[\boldsymbol{x} \bmod q]^{(q)}_{\ell-1}$. This, however, relies on checking that the secret in $[\boldsymbol{x}]^{(P)}$ is between 0 and $q-1$, to ensure modular reduction $\boldsymbol{x} \bmod q$ yields $\boldsymbol{x}$ itself. In this section we show how to achieve this.

The operation above is abstracted by the functionality $\mathcal{F}_{\mathsf{DoubleModuli}}$. Importantly, note that there is no privacy involved! The degree $\ell-1$ is very low and does not protect the "secrets". The context in which we make use of this functionality indeed does not require privacy but efficient constant communication. An instantiation of $\mathcal{F}_{\mathsf{DoubleModuli}}$ appears in the protocol $\Pi_{\mathsf{DoubleModuli}}$ for which we present a detailed specification, and security proofs in Section 4.4. In the following section, we present an overview of this protocol.

Functionality 1: $\mathcal{F}_{\mathsf{DoubleModuli}}$

Public Parameters: Let p and q be two primes. Let ℓ be the packing parameter. Let $d \in \mathbb{Z}^+$.

1. $\mathcal{F}_{\mathsf{DoubleModuli}}$ receives $\{\boldsymbol{x}_i\}_{i=1}^{d}$ from the adversary, where each $\boldsymbol{x}_i$ is in $\mathbb{Z}_q^{\ell}$.
2. For each $i \in [d]$, $\mathcal{F}_{\mathsf{DoubleModuli}}$ interpolates the unique polynomials $f_i(\mathtt{X}) \in \mathbb{Z}_q[\mathtt{X}]$ and $g_i(\mathtt{X}) \in \mathbb{Z}_p[\mathtt{X}]$ of degree $\leq \ell-1$ such that $f_i(-j) = x_{ij}$ and $g_i(-j) = x_{ij} \bmod p \in \mathbb{Z}_p$ for $j \in [\ell]$, and sends $(f_i(h), g_i(h))$ to party P_h for $h \in [n]$.
3. For $i \in [d]$, the parties output $[\boldsymbol{x}_i]^{(q)}_{\ell-1} = (f_i(1), \ldots, f_i(n))$ and $[\boldsymbol{x}_i \bmod p]^{(p)}_{\ell-1} = (g_i(1), \ldots, g_i(n))$.

In this section we will work on batches of d vectors at a time. Since each vector has dimension ℓ, in our ECDSA context this will translate into $d \cdot \ell$ signatures.

4.1 Overview and Motivation for $\Pi_{\mathsf{DoubleModuli}}$

We begin by highlighting the difficulty of this task by presenting a simple and natural approach that unfortunately does not meet our efficiency requirements. To simplify notation use $\ell = 1$ for the purpose of this overview.[14] A standard approach to prove boundedness is to use bit decompositions: let P_1 bit-decompose the value x as $x = \sum_{i=0}^{\lceil \log q \rceil} 2^i \cdot x_i$, and then secret-share each x_i separately as $[x_i]^{(P)}$. It is possible to prove that each x_i is in $\{0,1\}$ by checking that $x_i \cdot (1-x_i) = 0$. Unfortunately, this approach leads to a communication overhead of $\times \log q$, which for practical purposes can be regarded as $\Omega(n)$ (*e.g.* for ECDSA this is roughly ≈ 256). Sadly, this would break our $O(|C|)$ communication claims.

Instead, we take as a starting point a different approach to range proofs that does not require bit decomposition. First, for purposes that will be made much clearer later, we shift the task of proving that $0 \leq x < q$ to proving

[14] This is not a very good example since in this case the degree is $\ell = 0$, meaning the dealer can simply send the "secret" directly to the parties. Ignore this for the purpose of illustration.

that $-(q-1)/2 \leq z \leq (q-1)/2$, where $z = x - (q-1)/2$. This "symmetric" formulation will turn out to be useful. We consider the sum-of-squares approach by Boudot [Bou00], which in our context can be translated as follows: to prove that $-B \leq z \leq B$, we can prove that there exist four integers $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ such that $B^2 - z^2 = \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2$, where this relation must hold *over the integers*. This multiplicative relation would be easy to handle in our case, except that our secret-sharing techniques require us to work modulo some integer P. For the sums-of-squares equation to imply boundedness of z, we need to ensure that z and the α_i's are small enough so that signed reduction modulo P has no effect, and hence the equation actualy holds over the integers. For this, it suffices to prove they are bounded in absolute value by some b, where $b \leq \sqrt{\frac{P-1}{8}}$, which ensures both the left and right hand sides of the equation are in $[-(P-1)/2, (P-1)/2]$.

At this point we have reduced the task of proving that the sharing $[z]^{(P)}$ contains $z \in [-B, B]$, to the task of checking that a few more sharings contain secrets in $[-b, b]$. This may seem redundant, but there is a crucial item that make this an easier task: we can choose b to be large enough so that it is considerably larger than the bound B we ultimately want to prove on z. This allows us to introduce some *slack*, meaning that we will be able to leverage range proof techniques where there is a mismatch between what an honest dealer can prove, and what we will be able to conclude for a corrupt dealer. Such proofs can be batched and amortized more easily, for which we follow the techniques from [GHL22] which make use of the Johnson-Lindenstrauss Lemma.[15]

4.2 Range Proofs with Slack

We begin by recalling the main result from [GHL22] that we will need in our work:

Theorem 1 (Corollary 3.3 in [GHL22]). *Fix $\delta, P \in \mathbb{Z}$, and a bound $b \leq P/45\delta$, and let $\boldsymbol{w} \in [-P/2, +P/2]^\delta$ with $\|\boldsymbol{w}\|_2 \geq b$. Let $\mathcal{D}$ be a distribution in $\{0, +1, -1\}$ such that $\mathcal{D}(1) = \mathcal{D}(-1) = 1/4$, and $\mathcal{D}(0) = 1/2$. Then*

$$\Pr_{R \leftarrow \mathcal{D}^{256 \times \delta}}[\|R \cdot \boldsymbol{w} \bmod P\|_2 < b\sqrt{30}] < 2^{-128}$$

Remark 1. We state the theorem as it appears in [GHL22]. In particular we get the statistical security of ≈ 128 that the authors were aiming for. However, in our work we are settling with a possibly much smaller parameter, so there is potentially room for concrete improvements by lowering this bound.

[15] In fact, our idea of combining these proofs with the sums-of-squares approach is inspired in [GHL22], though we adapt it from the lattice context to our setting.

Procedure 2: $\pi_{\mathsf{RangeSlack}}$

Input: The dealer P_1 has a collection of inputs $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_\delta \in \mathbb{Z}^\ell$ such that $\|\boldsymbol{z}_i\|_\infty < \frac{b\sqrt{30}}{16\delta}$ for all i. The bound b must satisfy $b \leq P/45\delta$.
Output: Parties obtain sharings $[\boldsymbol{z}_i]^{(P)}_{\ell-1}$ for all i with the guarantee that $\|\boldsymbol{z}_i\|_\infty < b$

1. Dealer distributes $\{[\boldsymbol{z}_i]^{(P)}_{\ell-1}\}_{i=1}^{\delta}$ by calling $\pi_{\mathsf{DistShares}}$
2. Let us arrange the $\boldsymbol{z}_i$'s in a $\delta \times \ell$ matrix $\boldsymbol{Z}$. Let us denote the columns of $\boldsymbol{Z}$ by $\boldsymbol{z}^j \in \mathbb{Z}^\ell$. The parties sample a random matrix $R \leftarrow \mathcal{D}^{256\times\delta}$, and let $\boldsymbol{Y} = R \cdot \boldsymbol{Z}$. Note that $\boldsymbol{Y}$ is a $256 \times \ell$ matrix, and the patries can locally obtain shares of each row $[\boldsymbol{y}_1]^{(P)}_{\ell-1}, \ldots, [\boldsymbol{y}_{256}]^{(P)}_{\ell-1}$.
3. Parties open $\boldsymbol{Y}$ by sending all the shares to each other, and take signed modulo P (that is, with representatives in $[-(P-1)/2, (P-1)/2]$)
4. Parties check that $\|\boldsymbol{y}^j\|_2 < b\sqrt{30}$ for $j \in [\ell]$, where $\boldsymbol{y}^j$ denotes the j-th column of $\boldsymbol{Y}$, and they abort if any of these checks fail.

Proposition 2. *An honest dealer can pass the check of* $\pi_{\mathsf{RangeSlack}}$. *Furthermore, if Procedure* $\pi_{\mathsf{RangeSlack}}$ *does not abort, then with overwhelming probability, it holds that* $\|\boldsymbol{z}_i\|_\infty < b$ *for* $i \in [\delta]$.

Proof. From Theorem 1 above, if $\|\boldsymbol{z}^j\|_2 \geq b$ for some $j \in [\ell]$ then with overwhelming probability it would be the case that $\|R \cdot \boldsymbol{z}^j \bmod P\|_2 \geq b\sqrt{30}$. Therefore, the only way to pass the check is if $\|\boldsymbol{z}^j\|_2 < b$ for all $j \in [\ell]$[16] in a first place. This means that $\|\boldsymbol{z}^j\|_\infty \leq \|\boldsymbol{z}^j\|_2 < b$ for all $j \in [\ell]$, and therefore $\|\boldsymbol{z}_i\| < b$ for all $i \in [\delta]$, which is what we wanted to prove.

For an honest dealer, note that the assumed bound $\|\boldsymbol{z}_i\|_\infty < \frac{b\sqrt{30}}{16\delta}$ for all $i \in [\delta]$ implies that $\|\boldsymbol{z}^j\|_\infty < \frac{b\sqrt{30}}{16\delta}$ for all $j \in [\ell]$. Since $R \in \{0, +1, -1\}^{256\times\delta}$, this implies that $\|\boldsymbol{y}^j\|_\infty \leq \delta \cdot \|\boldsymbol{z}^j\|_\infty < \delta \cdot \frac{b\sqrt{30}}{16\delta} = \frac{b\sqrt{30}}{16}$, and so $\|\boldsymbol{y}^j\|_2 \leq \sqrt{256} \cdot \|\boldsymbol{y}^j\|_\infty < 16 \cdot \frac{b\sqrt{30}}{16} = b\sqrt{30}$. This is the bound checked in the protocol, which shows an honest dealer can pass the check.

4.3 Sums-of-Squares Range Proof

So far we have managed to ensure that the parties obtain sharings $[\boldsymbol{z}_i]^{(P)}_{\ell-1}$ with $\|\boldsymbol{z}_i\|_\infty < b$. However, our ultimate goal is to prove that $\|\boldsymbol{z}_i\|_\infty < B$ for some $B \leq b$, where *we cannot afford a slack*. Note that this is equivalent (cf. [Bou00; GHL22]) to the existence of $\boldsymbol{\alpha}_{ij}$ for $j \in \{1, 2, 3, 4\}$ such that

$$(B-1)^2 \cdot \mathbf{1} - \boldsymbol{z}_i^2 = \boldsymbol{\alpha}_{i1}^2 + \boldsymbol{\alpha}_{i2}^2 + \boldsymbol{\alpha}_{i3}^2 + \boldsymbol{\alpha}_{i4}^2. \tag{1}$$

Our approach is simple and it is inspired by [GHL22]: let the dealer provide shares of the $\boldsymbol{\alpha}_{ij}$'s, and let the parties use the range proof *with slack* from the

[16] It must be noted that the original theorem is stated where $\boldsymbol{w}$ is a vector, but here it is a $\delta \times \ell$ matrix. Applying the union bound degrades the 2^{-128} bound by only $\log(\ell)$ bits.

previous section to check that these secrets are "somewhat bounded", meaning that, even if they are larger than the desired B, they are not "too large". In particular, the dealer proves they are low enough so that taking signed-modulo P on both sides of Eq. (1) does not result in wraparound. At this point the parties can use a protocol to check Eq. (1) mod P, and due to the property above this check will translate over the integers, therefore showing the desired bound on $\boldsymbol{z}_i$.

For the rest of this section we let P be a positive integer such that $pq \mid P$ and also the smallest prime factor of P is in the order of 2^κ, where κ is the statistical security parameter.

Procedure 3: π_{RangeSOS}

Input: The dealer P_1 has a collection of inputs $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d \in \mathbb{Z}^\ell$ such that $\|\boldsymbol{z}_i\|_\infty < B$ for all i. Set $b = \sqrt{(P-1)/8}$, and assume $b \leq \frac{P}{45 \cdot 5d}$.[a] Assume $B \leq \frac{b\sqrt{30}}{16 \cdot 5d}$.
Output: Parties obtain sharings $[\boldsymbol{z}_i]^{(P)}_{\ell-1}$ for all $i \in [\delta]$ with the guarantee that $\|\boldsymbol{z}_i\|_\infty < B$

1. The dealer finds $\boldsymbol{\alpha}_{ij}$ for $i \in [d]$ and $j \in [4]$ such that Eq. (1) holds.
2. Note that since $\|\boldsymbol{z}_i\|_\infty < B$ then $\|\boldsymbol{\alpha}_{ij}\|_\infty < B$. The parties call Procedure $\pi_{\mathsf{RangeSlack}}$ (using $\delta = 5d$) to obtain $\{([\boldsymbol{z}_i]^{(P)}_{\ell-1}, [\boldsymbol{\alpha}_{i1}]^{(P)}_{\ell-1}, [\boldsymbol{\alpha}_{i2}]^{(P)}_{\ell-1}, [\boldsymbol{\alpha}_{i3}]^{(P)}_{\ell-1}, [\boldsymbol{\alpha}_{i4}]^{(P)}_{\ell-1})\}_{i=1}^d$. It can be checked that, from $B \leq \frac{b\sqrt{30}}{16 \cdot 5d}$ and also $b \leq \frac{P}{45 \cdot 5d}$, it follows that $\|\boldsymbol{z}_i\|_\infty < \frac{b\sqrt{30}}{16\delta}$ and $b \leq \frac{P}{45\delta}$ with $\delta = 5d$, which are the inequalities required to be able to call $\pi_{\mathsf{RangeSlack}}$ in a first place. By the guarantees of $\pi_{\mathsf{RangeSlack}}$ every vector has infinity norm strictly less than b.
3. The parties call $\mathcal{F}_{\mathsf{Coin}}$ to sample $\beta_1, \ldots, \beta_d \in \mathbb{Z}_P$, and compute locally $[\boldsymbol{o}]^{(P)}_{2(\ell-1)} = \sum_{i=1}^d \beta_i \cdot [\boldsymbol{o}_i]^{(P)}_{2(\ell-1)}$, where $[\boldsymbol{o}_i]^{(P)}_{2(\ell-1)} = (B-1)^2 \cdot \mathbf{1} - [\boldsymbol{z}_i]^{(P)}_{\ell-1} \cdot [\boldsymbol{z}_i]^{(P)}_{\ell-1} - \sum_{j=1}^4 [\boldsymbol{\alpha}_{ij}]^{(P)}_{\ell-1} \cdot [\boldsymbol{\alpha}_{ij}]^{(P)}_{\ell-1}$.
4. The parties reconstruct $\boldsymbol{o} \leftarrow [\boldsymbol{o}]^{(P)}_{2(\ell-1)}$ and check that $\boldsymbol{o} = \mathbf{0}$. The parties abort if this is not the case.
5. If no abort is produced, output $\{[\boldsymbol{z}_i]^{(P)}_{\ell-1}\}_{i=1}^d$

[a] This will follow directly for large P and for moderate d.

Proposition 3. *Procedure π_{RangeSOS} succeeds for an honest dealer. Furthermore, if the procedure does not abort, then with overwhelming probability, it holds that $\|\boldsymbol{z}_i\|_\infty < B$ for all $i \in [d]$.*

Proof. The statement for an honest dealer is easy to verify by inspection. Now, suppose that the call to π_{SOSCheck} in line 3 succeeds. By Proposition 4.3, we have that the vectors $\boldsymbol{z}_i, \boldsymbol{\alpha}_{i1}, \boldsymbol{\alpha}_{i2}, \boldsymbol{\alpha}_{i3}, \boldsymbol{\alpha}_{i4}$ for $i \in [d]$ satisfy Eq. (1) modulo P with overwhelming probability. Now, assuming that the call to $\pi_{\mathsf{RangeSlack}}$ in line 2 succeeds, by Proposition 2 we have that $\|\boldsymbol{w}\|_\infty < b$ for all symbols $\boldsymbol{w} \in \cup_{i=1}^\delta \{\boldsymbol{z}_i, \boldsymbol{\alpha}_{i1}, \boldsymbol{\alpha}_{i2}, \boldsymbol{\alpha}_{i3}, \boldsymbol{\alpha}_{i4}\}$. This implies that $\|B^2 \cdot \mathbf{1} - \boldsymbol{z}_i^2\|_\infty < B^2 + b^2 \leq 2b^2 = (P-1)/4 < (P-1)/2$ (here we use the fact that $B \leq b$) and also

$\|\boldsymbol{\alpha}_{i1}^2 + \boldsymbol{\alpha}_{i2}^2 + \boldsymbol{\alpha}_{i3}^2 + \boldsymbol{\alpha}_{i4}^2\|_\infty < 4b^2 = (P-1)/2$. We see then that both left and right hand sides of Eq. (1) do not change in value when taken sign-modulo P, and therefore checking the equality over the integers is equivalent to checking it modulo P.

Now we show that, if at least one such equalities does not hold (modulo P), then step 4 would lead to an abort. Suppose that $\boldsymbol{o} = \mathbf{0}$, and yet $\exists i_0 \in [d]$ such that $\boldsymbol{o}_{i_0} \not\equiv 0 \bmod P$. Due to this, there must exist $j_0 \in [\ell]$ and a prime factor ρ of P such that $\boldsymbol{o}_{i_0}[j_0] \not\equiv 0 \bmod \rho$, which means $\boldsymbol{o}_{i_0}[j_0]$ is invertible modulo ρ. Since $\boldsymbol{o}[j_0] \equiv 0 \bmod \rho$, this means that $\beta_{i_0} \equiv \boldsymbol{o}_i[j_0]^{-1} \cdot \left(-\sum_{i \neq i_0} \beta_i \cdot \boldsymbol{o}_i[j_0]\right) \bmod \rho$. Since β_{i_0} is chosen independently of the terms on the right-hand side, this congruence can only be satisfied with probability $1/\rho$, which is negligible from our assumption that P only has big prime factors. From this we obtain that $\|\boldsymbol{z}_i\|_\infty < B$ for all $i \in [d]$.

4.4 Final Protocol

We put together the pieces described so far in the protocol below. The communication complexity of $\Pi_{\mathsf{DoubleModuli}}$ is discussed in Section 4.5.

Protocol 4: $\Pi_{\mathsf{DoubleModuli}}$

Input: The dealer P_1 has a collection of inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d \in \mathbb{Z}_q^\ell$. Let $B = \frac{q-1}{2} + 1$. Let P be such that $P \geq \frac{8(B \cdot 16 \cdot 5d)^2}{30} + 1$.

Output: Parties obtain sharings $([\boldsymbol{x}_i \bmod p]_{\ell-1}^{(p)}, [\boldsymbol{x}_i]_{\ell-1}^{(q)})$ for all $i \in [d]$.

1. P_1 sets $\boldsymbol{z}_i = \boldsymbol{x}_i - \frac{q-1}{2} \cdot \mathbf{1}$. Note that $\|\boldsymbol{z}_i\|_\infty < B$ where $B = \frac{q-1}{2} + 1$.
2. Let $b = \sqrt{(P-1)/8}$. Note that $P \geq \frac{8(B \cdot 16 \cdot 5d)^2}{30} + 1$ directly implies that $B \leq \frac{b\sqrt{30}}{16 \cdot 5d}$. Additionally, $b \leq \frac{P}{45 \cdot 5d}$ is equivalent to $P \geq \frac{(45 \cdot 5d)^2}{8}$, which can be checked to be implied already by the assumed bound on P. These inequalities are the premises needed in π_{RangeSOS}, so the parties can call this procedure to obtain $[\boldsymbol{z}_i]_{\ell-1}^{(P)}$ for all $i \in [d]$, with the guarantee that $\|\boldsymbol{z}_i\|_\infty < B$, or equivalently $\|\boldsymbol{z}_i\|_\infty \leq B - 1$.
3. Parties compute locally $[\boldsymbol{x}_i]_{\ell-1}^{(P)} = [\boldsymbol{z}_i]_{\ell-1}^{(P)} + \frac{q-1}{2}\mathbf{1}$. Note that $\|\boldsymbol{z}_i\|_\infty \leq B - 1$ implies that $-(B-1) + \frac{q-1}{2} \leq x_{ij} \leq (B-1) + \frac{q-1}{2}$, which is the same as $x_{ij} \in [0, q)$, for all $i \in [d], j \in [l]$.
4. Since $p \mid P$ and $q \mid P$, parties can locally take modulo p and q to obtain $[\boldsymbol{x}_i \bmod p]_{\ell-1}^{(p)}$ and $[\boldsymbol{x}_i \bmod q]_{\ell-1}^{(q)}$ respectively. Since $0 \leq x_{ij} < q$, $\boldsymbol{x}_i \bmod q = \boldsymbol{x}_i$.
5. Parties output the pairs $\{([\boldsymbol{x}_i \bmod p]_{\ell-1}^{(p)}, [\boldsymbol{x}_i]_{\ell-1}^{(q)})\}_{i=1}^d$.

Theorem 2. *Protocol $\Pi_{\mathsf{DoubleModuli}}$ (which uses procedures $\pi_{\mathsf{RangeSlack}}$ and π_{RangeSOS}) implements functionality $\mathcal{F}_{\mathsf{DoubleModuli}}$ in the $\mathcal{F}_{\mathsf{Coin}}$-hybrid model.*

Proof. We describe the simulator $\mathcal{S}$. We are only interested in the case in which the dealer is corrupt, since otherwise the proof is straightforward. The simulator

emulates a set of $n - t = 2\ell + 1$ virtual honest parties towards the adversary. We note that in line 2 of $\Pi_{\mathsf{DoubleModuli}}$ the procedure π_{RangeSOS} is called on the d vectors $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d$. π_{RangeSOS} itself calls, on line 2, the procedure $\pi_{\mathsf{RangeSlack}}$ on $5d$ inputs $\{([\boldsymbol{z}_i]^{(P)}_{\ell-1}, [\boldsymbol{\alpha}_{i1}]^{(P)}_{\ell-1}, [\boldsymbol{\alpha}_{i2}]^{(P)}_{\ell-1}, [\boldsymbol{\alpha}_{i3}]^{(P)}_{\ell-1}, [\boldsymbol{\alpha}_{i4}]^{(P)}_{\ell-1})\}_{i=1}^{d}$. For notational simplicity, let us denote these vectors by $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{5d}$. We show how to simulate this call to $\pi_{\mathsf{RangeSlack}}$.

Simulating the calls to $\pi_{\mathsf{RangeSlack}}$. Consider one of these calls to $\pi_{\mathsf{RangeSlack}}$ with input $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{5d} \in \mathbb{Z}^{\ell}$. First, $\mathcal{S}$ receives from the dealer the honest parties' shares of $[\boldsymbol{w}_i]^{(P)}_{*}$. If at least one of these sharings has degree $> \ell - 1$, then $\mathcal{S}$ aborts. By Proposition 1, this happens in the real world with overwhelming probability. Otherwise, all the degrees are $\leq \ell - 1$ and therefore the underlying secrets can be extracted by $\mathcal{S}$ as they are determined by the honest parties' shares.

Now, $\mathcal{S}$ emulates steps 2 and 3 of $\pi_{\mathsf{RangeSlack}}$ as follows: $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Coin}}$ by sending a random $R \leftarrow \mathcal{D}^{256\times 5d}$ to the corrupt parties, and then reconstructs $[\boldsymbol{y}_1]^{(P)}_{\ell-1}, \ldots, [\boldsymbol{y}_{256}]^{(P)}_{\ell-1}$ towards the corrupt parties; $\mathcal{S}$ can do this as it knows both R and the honest parties' shares of the $\boldsymbol{x}_i$'s. Then $\mathcal{S}$ aborts if it does not hold that $\|\boldsymbol{y}^j\|_2 < b\sqrt{30}$, where $\boldsymbol{y}^j$ is the j-th column of the matrix $[\boldsymbol{y}_1 \mid \cdots \mid \boldsymbol{y}_{256}]^{\intercal}$ (taken sign-modulo P), as in the protocol. From Proposition 2, in the real world either the parties abort, or $\|\boldsymbol{w}_i\|_\infty < b$ with overwhelming probability for every $i \in [5d]$. Since $\mathcal{S}$ plays the protocol with the same distribution, the same holds for the ideal world, and therefore the two worlds are indistinguishable thus far. If there is no abort, we conclude that $\|\boldsymbol{w}_i\|_\infty < b$ for all $i \in [5d]$.

Simulating the rest of π_{RangeSOS}. Jumping to lines 3 and 4 in π_{RangeSOS}, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Coin}}$ one more time, obtaining $\beta_1, \ldots, \beta_d$, and opens $[\boldsymbol{o}]^{(P)}_{2(\ell-1)}$ as in the protocol, aborting if $\boldsymbol{o} \neq \boldsymbol{0}$. From Proposition 4.3, if the protocol does not result in abort then the shares the honest parties hold $[\boldsymbol{z}_i]^{(P)}_{\ell-1}$ must satisfy that $\|\boldsymbol{z}_i\|_\infty \leq B - 1$.

Simulating the rest of $\Pi_{\mathsf{DoubleModuli}}$. Once π_{RangeSOS} has been simulated, the rest of the proof is rather straightforward. If no abort has been issued, $\mathcal{S}$ computes $\boldsymbol{x}_i := \boldsymbol{z}_i + \frac{q-1}{2}\boldsymbol{1}$ and sends $\boldsymbol{x}_i$ for $i \in [d]$ to the functionality $\mathcal{F}_{\mathsf{DoubleModuli}}$, which distribute sharings $([\boldsymbol{x}_i]^{(q)}_{\ell-1}, [\boldsymbol{x}_i \bmod p]^{(p)}_{\ell-1})$ to the honest parties in the ideal world. This is the same the honest parties output in the real world: since $pq \mid Q$, the properties of Shamir secret-sharing outlined in Section 3.2 imply that the reduction modulo p and q lead to secrets $\boldsymbol{x}_i \bmod p$ and $\boldsymbol{x}_i \bmod q$ respectively. Furthermore, given that $\|\boldsymbol{z}_i\|_\infty \leq B - 1$, every entry in $\boldsymbol{x}_i$ lies between 0 and $q - 1$, so $\boldsymbol{x}_i \bmod q = \boldsymbol{x}_i$, as required. □

4.5 Communication Complexity of $\Pi_{\mathsf{DoubleModuli}}$

Recall that for our approach to work P must be larger than $\frac{8(B\cdot 16\cdot 5d)^2}{30} + 1$. This means elements in $\mathbb{Z}_P$ have bit-length $\approx \log((qd)^2) = 2\log(qd) = 2\log(q) +$

$2\log(d)$. Note that $2\log(q)$ is in the order of 512, while $d = k/\ell$ where k is the number of desired signatures, which means $\log(d)$ is comparatively much smaller (for example, for 10 million signatures $\log(k)$ is only ≈ 23, without subtracting from it $\log \ell$). Hence, we approximate the bit-length of P as $\approx 2\log(q)$, that is, the cost of sharing elements in $\mathbb{Z}_P$ is about only *twice* of sharing elements over $\mathbb{Z}_q$.

Now we compute the protocol costs. We ignore the calls to $\mathcal{F}_{\mathsf{Coin}}$ since it is only called a constant amount of times and, as pointed out in Section **??**, the length of the sampled strings does not affect the communication complexity. The cost of $\Pi_{\mathsf{DoubleModuli}}$ is the following:

- Step 1 in $\pi_{\mathsf{RangeSlack}}$ requires the dealer to distribute d packed sharings over $\mathbb{Z}_P$. This costs $d \cdot n \cdot 2\log q$ bits.
- Step 3 in $\pi_{\mathsf{RangeSlack}}$ requires the parties to reconstruct 256 values over $\mathbb{Z}_P$, which costs $n^2 \cdot 256 \cdot 2\log(q)$ bits.
- $\pi_{\mathsf{RangeSlack}}$ is called 5 times in step 2 of π_{RangeSOS}, so the cost of this step is $5 \cdot \left(d \cdot n \cdot 2\log q + n^2 \cdot 256 \cdot 2\log(q)\right)$.
- Step 3 of π_{RangeSOS} requires reconstructing the sharing $\boldsymbol{o}$, which is comparatively smaller than the above complexities.

The total is $\approx 2\log q \cdot \left(5dn + 512n^2\right) = \log q \cdot \left(10dn + 1024n^2\right)$ bits. If d is large enough, then this is $O(n \cdot d)$, as required. Specifically, if $d = \Theta(n)$, say $n = \gamma \cdot d$, then this communication per party (*i.e.* after dividing by n) is

$$\log q \cdot (10 + 1024\gamma) \cdot d,$$

which crucially is *independent of* n.

5 Generation of Tuples

Towards generating our ECDSA preprocessing we will need to generate sharings $\langle a_j \rangle$, $[\boldsymbol{\phi}_j]_{n-\ell}$, $[a_j \cdot \boldsymbol{\phi}_j]_{n-\ell}$, $[\boldsymbol{\beta}_j = \boldsymbol{k}_j * \boldsymbol{\phi}_j]_{n-1}$, $a_j \cdot G$, $[\boldsymbol{k}_j \cdot G]_{\ell-1}$, for $j \in [\lambda]$, where λ is certain parameter. This is abstracted by the functionality $\mathcal{F}_{\mathsf{Tuple}}$ below, where we use matrix notation to collect the λ sharings above into different columns.

Functionality 2: $\mathcal{F}_{\mathsf{Tuple}}$

Public Parameters: Let n be the number of parties, t be the corruption threshold. Let $\ell = (n - t + 1)/2$. Let λ be a public parameter.

1. $\mathcal{F}_{\mathsf{Tuple}}$ samples a random vector $\boldsymbol{a}$ of size λ and two random matrices $\boldsymbol{B}, \boldsymbol{\Phi}$ of size $\ell \times \lambda$.
2. Let $\boldsymbol{A}$ be the matrix of size $\ell \times \lambda$ where each row is equal to $\boldsymbol{a}$. Then $\mathcal{F}_{\mathsf{Tuple}}$ computes $\boldsymbol{C} = \boldsymbol{A} \star \boldsymbol{\Phi}$ and $\boldsymbol{D} = \boldsymbol{B} \star \boldsymbol{\Phi}$.
3. For a vector $\boldsymbol{a}$, we use a_j to denote the j-th entry of $\boldsymbol{a}$. For a matrix $\boldsymbol{A}$, we use $\boldsymbol{A}_{\star,j}$ to denote the j-th column of $\boldsymbol{A}$. From $j = 1$ to λ, $\mathcal{F}_{\mathsf{Tuple}}$ does the following to generate $(\langle a_j \rangle, [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}, [\boldsymbol{C}_{\star,j}]_{n-\ell}, [\boldsymbol{D}_{\star,j}]_{n-1})$.

(a) For $\langle a_j \rangle$, $\mathcal{F}_{\mathsf{Tuple}}$ receives from $\mathcal{S}$ shares of corrupted parties and samples a random additive sharing $\langle a_j \rangle$ based on the secret a_j and the shares of corrupted parties.
(b) For $[\boldsymbol{\Phi}_{\star,j}]_{n-\ell}$, $\mathcal{F}_{\mathsf{Tuple}}$ receives from $\mathcal{S}$ shares of corrupted parties and samples a random degree-$(n-\ell)$ packed Shamir sharing of $\boldsymbol{\Phi}_{\star,j}$ based on the secrets $\boldsymbol{\Phi}_{\star,j}$ and the shares of corrupted parties. $\mathcal{F}_{\mathsf{Tuple}}$ prepares $[\boldsymbol{C}_{\star,j}]_{n-\ell}$ in a similar way.
(c) For $[\boldsymbol{D}_{\star,j}]_{n-1}$, $\mathcal{F}_{\mathsf{Tuple}}$ receives from $\mathcal{S}$ shares of corrupted parties and samples a random degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{D}_{\star,j}$ based on the secrets $\boldsymbol{D}_{\star,j}$ and the shares of corrupted parties.

4. $\mathcal{F}_{\mathsf{Tuple}}$ computes $\boldsymbol{a} \cdot G$ and $\boldsymbol{B} \cdot G$. From $j = 1$ to λ, $\mathcal{F}_{\mathsf{Tuple}}$ computes $a_j \cdot G$ and $[\boldsymbol{B}_{\star,j} \cdot G]_{\ell-1}$.
5. $\mathcal{F}_{\mathsf{Tuple}}$ distributes $(\langle a_j \rangle, [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}, [\boldsymbol{C}_{\star,j}]_{n-\ell}, [\boldsymbol{D}_{\star,j}]_{n-1}), a_j \cdot G, [\boldsymbol{B}_{\star,j} \cdot G]_{\ell-1}$ to all parties for all $j \in [\lambda]$.

5.1 Generation of Virtual Parties

In the actual protocol, we make use of the party virtualization technique [Bra87] to boost the corruption threshold. Each virtual party will be emulated by a small committee of size s. We will simply choose N random committees by generating a random coin via $\mathcal{F}_{\mathsf{Coin}}$ and then all parties use the same random coin to decide the committees. We say a virtual party V_i is honest if its committee contains at least one honest party and we say V_i is corrupted if all parties in its committee are corrupted.

We first analyse the probability of randomly choosing N random committees such that there exists a set of t corrupted parties such that at least $(N+1)/2-\lambda+1$ virtual parties are corrupted.

For a fixed set of t corrupted parties, the probability that a virtual party is corrupted is bounded by $\frac{t^s}{n^s}$. We choose the smallest s such that $\frac{t^s}{n^s} \leq \frac{1}{2} \cdot ((N+1)/2-\lambda)/N$. Let $X_i \in \{0,1\}$ denote the random variable where $X_i = 1$ means that the V_i is corrupted and vise versa. Let $X = X_1 + \cdots + X_N$. Then the probability that there are at least $(N+1)/2-\lambda+1$ corrupted committees among all N committees is $\Pr[X > (N+1)/2-\lambda]$. By the Chernoff bound,

$$\Pr[X > E[X] + m] < e^{-2m^2/N},$$

for all $m > 0$. Since $E[X] = \frac{t^s}{n^s} \cdot N \leq \frac{1}{2} \cdot ((N+1)/2-\lambda)$, taking $m = \frac{1}{2} \cdot ((N+1)/2-\lambda)$, we have

$$\Pr[X > (N+1)/2-\lambda] < e^{-2m^2/N} = e^{-\frac{(N+1-2\lambda)^2}{8N}}.$$

Then, by union bound, the probability that there exists a set of t corrupted parties such that at least $(N+1)/2-\lambda+1$ virtual parties are corrupted is

$$\binom{n}{t} \cdot e^{-\frac{(N+1-2\lambda)^2}{8N}} \leq e^{n-\frac{(N+1-2\lambda)^2}{8N}}.$$

Parameter Choice. To achieve the desired efficiency, we may set $\lambda = N/4$. Then when $N = 32(n + \kappa)$, the probability that there exists a set of t corrupted parties such that at least $(N + 1)/2 - \lambda + 1$ virtual parties are corrupted is bounded by $2^{-\kappa}$. As for the committee size s, recall $t/n \approx (n - 2\ell)/n$ and that we need s with $\frac{t^s}{n^s} \leq \frac{1}{2} \cdot ((N + 1)/2 - \lambda)/N$. For our current choice of λ we have $\frac{1}{2} \cdot ((N + 1)/2 - \lambda)/N \approx 1/8$, so it suffices to set s so that $(t/n)^s \leq 1/8$. We may choose $s = \log_{t/n}(1/8) \approx \log(1/8)/\log((n - 2\ell)/n)$. We analyze the resulting communication complexity of our protocol in Section 5.4.

Emulation of a Virtual Party. When using a committee to emulate a virtual party, we maintain the invariant that the data that should be learnt by the virtual party is additively shared among the parties in the committee. To be more concrete, we elaborate how to emulate the virtual party in our concrete construction.

- When a real party P_i needs to distribute a degree-$(N - 1)/2$ packed Shamir sharing to virtual parties, say $[\boldsymbol{a}^{\star,i}]^V_{(N-1)/2}$, P_i additively shares the value that should be sent to V_j to the parties in the committee V_j.
- When a virtual party V_j needs to multiply his two local values x, y, the parties in V_j with additive sharings $\langle x \rangle, \langle y \rangle$ invoke $\mathcal{F}_{\mathsf{Mult}}$. Note that this is the *only* place in our whole protocol OLE is needed.[17]
- When a virtual party V_j needs to distribute a degree-$(n-\ell)$ (or degree-$(n-1)$) packed Shamir sharing of $\boldsymbol{\Phi}^{j,\star}$ to real parties, each party in V_j computes a degree-$(n - \ell)$ (or degree-$(n - 1)$) packed Shamir sharing of his additive shares of $\boldsymbol{\Phi}^{j,\star}$ and distributes the shares to real parties.

5.2 Realization of $\mathcal{F}_{\mathsf{Tuple}}$

Now we are ready to present the protocol for $\mathcal{F}_{\mathsf{Tuple}}$.

Protocol 5: Π_{Tuple}

Generation of Virtual Parties: Let N, λ, s be agreed parameters. All parties invoke $\mathcal{F}_{\mathsf{Coin}}$ to generate a random seed and use it to randomly choose N subsets of parties where each subset is of size s. We use V_i to denote the i-th subset of parties.

Step 1. All parties run the following steps twice to generate

$$\big(\langle a_j \rangle, [\boldsymbol{A}_{\star,j}]_{n-\ell}, [\boldsymbol{B}_{\star,j}]_{n-\ell}, [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}, [\boldsymbol{C}_{\star,j}]_{n-\ell}, [\boldsymbol{D}_{\star,j}]_{n-1}\big),$$

and

$$\big(\langle a'_j \rangle, [\boldsymbol{A}'_{\star,j}]_{n-\ell}, [\boldsymbol{B}'_{\star,j}]_{n-\ell}, [\boldsymbol{\Phi}'_{\star,j}]_{n-\ell}, [\boldsymbol{C}'_{\star,j}]_{n-\ell}, [\boldsymbol{D}'_{\star,j}]_{n-1}\big),$$

[17] In particular, if we start with an honest majority we can setup honest majority committees and hence get rid of OLE

where $\boldsymbol{A}_{\star,j} = a_j \cdot \mathbf{1}$ and $\boldsymbol{A}'_{\star,j} = a'_j \cdot \mathbf{1}$.

1. **Preparing** $(\langle a_j \rangle, [\boldsymbol{B}_{\star,j}]_{n-1}, [\boldsymbol{\Phi}_{\star,j}]_{n-1})$: For all $j \in \{1, \ldots, \lambda\}$, each party samples three random field elements as his shares of $(\langle a_j \rangle, [\boldsymbol{B}_{\star,j}]_{n-1}, [\boldsymbol{\Phi}_{\star,j}]_{n-1})$ respectively.
2. **Preparing** $[\boldsymbol{a}]^V_{(N-1)/2}$, $[\boldsymbol{B}_{j,\star}]^V_{(N-1)/2}$, **and** $[\boldsymbol{\Phi}_{j,\star}]^V_{(N-1)/2}$:
   (a) Each party P_i generates a degree-$(N-1)/2$ packed Shamir sharing of his shares of $\langle a_1 \rangle, \ldots, \langle a_\lambda \rangle$, denoted by $[\boldsymbol{a}^{(\star,i)}]^V_{(N-1)/2}$. Then for all $k \in \{1, \ldots, N\}$, P_i additively shares the k-th share of $[\boldsymbol{a}^{(\star,i)}]^V_{(N-1)/2}$ to parties in V_k.
   Finally, parties in V_k emulates the computation of the virtual party V_k to compute the k-th share of $[\boldsymbol{a}]^V_{(N-1)/2} = \sum_{i=1}^n [\boldsymbol{a}^{(\star,i)}]^V_{(N-1)/2}$.
   (b) Each party P_i generates a degree-$(N-1)/2$ packed Shamir sharing of his shares of $[\boldsymbol{B}_{\star,1}]_{n-1}, \ldots, [\boldsymbol{B}_{\star,\lambda}]_{n-1}$, denoted by $[\boldsymbol{B}^{(\star,i)}]^V_{(N-1)/2}$. Then for all $k \in \{1, \ldots, N\}$, P_i additively shares the k-th share of $[\boldsymbol{B}^{(\star,i)}]^V_{(N-1)/2}$ to parties in V_k.
   Finally, for all $j \in \{1, \ldots, \ell\}$, parties in V_k emulates the computation of the virtual party V_k to compute the k-th share of $[\boldsymbol{B}_{j,\star}]^V_{(N-1)/2} = \sum_{i=1}^n c_j^{(i)} \cdot [\boldsymbol{B}^{(\star,i)}]^V_{(N-1)/2}$ where $\{c_j^{(i)}\}_{i=1}^n$ are proper Lagrange coefficients.
   Following the same steps, parties in V_k obtain the k-th shares of $[\boldsymbol{\Phi}_{j,\star}]^V_{(N-1)/2}$ for all $j \in \{1, \ldots, \ell\}$.
3. **Computing** $([\boldsymbol{C}_{j,\star}]^V_{N-1}, [\boldsymbol{D}_{j,\star}]^V_{N-1})$: For all $j \in \{1, \ldots, \ell\}$, parties in V_k invoke $\mathcal{F}_{\mathsf{Mult}}$ with their additive sharings of the k-th share of $[\boldsymbol{a}]^V_{(N-1)/2}$ and the k-th share of $[\boldsymbol{\Phi}_{j,\star}]^V_{(N-1)/2}$. The result is viewed as an additive sharing of the k-th share of $[\boldsymbol{C}_{j,\star}]^V_{N-1}$.
   Following the same step, parties in V_k invoke $\mathcal{F}_{\mathsf{Mult}}$ with their additive sharings of the k-th share of $[\boldsymbol{B}_{j,\star}]^V_{(N-1)/2}$ and the k-th share of $[\boldsymbol{\Phi}_{j,\star}]^V_{(N-1)/2}$. The result is viewed as an additive sharing of the k-th share of $[\boldsymbol{D}_{j,\star}]^V_{N-1}$.
4. **Preparing** $([\boldsymbol{A}_{\star,j}]_{n-\ell}, [\boldsymbol{B}_{\star,j}]_{n-\ell}, [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}, [\boldsymbol{C}_{\star,j}]_{n-\ell}, [\boldsymbol{D}_{\star,j}]_{n-1})$:
   (a) For each $j \in \{1, \ldots, \ell\}$, all parties set $[\boldsymbol{A}_{j,\star}]^V_{N-1} = [\boldsymbol{a}]^V_{(N-1)/2}$, $[\boldsymbol{B}_{j,\star}]^V_{N-1} = [\boldsymbol{B}_{j,\star}]^V_{(N-1)/2}$, and $[\boldsymbol{\Phi}_{j,\star}]^V_{N-1} = [\boldsymbol{\Phi}_{j,\star}]^V_{(N-1)/2}$.
   (b) For each V_k, parties in V_k emulates the virtual party V_k to generate a random degree-$(n-\ell)$ packed Shamir sharing of the k-th shares of $[\boldsymbol{A}_{1,\star}]^V_{N-1}, \ldots, [\boldsymbol{A}_{\ell,\star}]^V_{N-1}$, denoted by $[\boldsymbol{A}^{(k,\star)}]_{n-\ell}$. To achieve this, recall that parties in V_k have additive sharings of the k-th share of $[\boldsymbol{A}_{1,\star}]^V_{N-1}, \ldots, [\boldsymbol{A}_{\ell,\star}]^V_{N-1}$. They first generate random additive sharings (by locally sampling random field elements as their shares) and then emulate the sharing algorithm of the degree-$(n-\ell)$ packed Shamir sharing scheme. Since the sharing algorithm is linear, parties in V_k eventually obtain an additive sharing for each share in $[\boldsymbol{A}^{(k,\star)}]_{n-\ell}$. Then parties in V_k send their additive shares of the i-th share of $[\boldsymbol{A}^{(k,\star)}]_{n-\ell}$ to P_i.
   Finally, all parties locally compute $[\boldsymbol{A}_{\star,j}]_{n-\ell}$.
   (c) Following the above step, all parties obtain $[\boldsymbol{B}_{\star,j}]_{n-\ell}, [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}, [\boldsymbol{C}_{\star,j}]_{n-\ell}, [\boldsymbol{D}_{\star,j}]_{n-1}$.

Step 2. All parties check the correctness of the computation via the sacrifice technique.

1. All parties invoke $\mathcal{F}_{\mathsf{Coin}}$ to receive two random values r_1, r_2.
2. For all $j \in \{1, \ldots, \lambda\}$, all parties locally compute $r_1 \cdot [\boldsymbol{A}_{\star,j}]_{n-\ell} + [\boldsymbol{A}'_{\star,j}]_{n-\ell}$ and send their shares to P_1. Then P_1 reconstructs $r_1 \cdot \boldsymbol{A}_{\star,j} + \boldsymbol{A}'_{\star,j}$ and distributes $[r_1 \cdot \boldsymbol{A}_{\star,j} + \boldsymbol{A}'_{\star,j}]_{\ell-1}$ to all parties.
   Following the same step, all parties also obtain $[r_1 \cdot \boldsymbol{B}_{\star,j} + \boldsymbol{B}'_{\star,j}]_{\ell-1}$ and $[r_2 \cdot \boldsymbol{\Phi}_{\star,j} + \boldsymbol{\Phi}'_{\star,j}]_{\ell-1}$.
3. All parties invoke $\mathcal{F}_{\mathsf{Coin}}$ to generate random values $\chi_1, \ldots, \chi_{3\lambda}$. Then all parties locally compute

$$\begin{aligned}[\boldsymbol{x}]_{\ell-1} = \sum_{j=1}^{\lambda} (&\chi_j \cdot [r_1 \cdot \boldsymbol{A}_{\star,j} + \boldsymbol{A}'_{\star,j}]_{\ell-1} \\ &+ \chi_{j+\lambda} \cdot [r_1 \cdot \boldsymbol{B}_{\star,j} + \boldsymbol{B}'_{\star,j}]_{\ell-1} \\ &+ \chi_{j+2\lambda} \cdot [r_2 \cdot \boldsymbol{\Phi}_{\star,j} + \boldsymbol{\Phi}'_{\star,j}]_{\ell-1}).\end{aligned} \tag{2}$$

   They send their shares of $[\boldsymbol{x}]_{\ell-1}$ to every other party and each party P_i verifies that $[\boldsymbol{x}]_{\ell-1}$ is a degree-$(\ell-1)$ packed Shamir sharing. If not, P_i aborts.
4. For all $j \in \{1, \ldots, \lambda\}$, all parties locally compute

$$\begin{aligned}[\boldsymbol{O}^{(1)}_{\star,j}]_{n-1} = \\ &[r_1 \cdot \boldsymbol{A}_{\star,j} + \boldsymbol{A}'_{\star,j}]_{\ell-1} \cdot [r_2 \cdot \boldsymbol{\Phi}_{\star,j} + \boldsymbol{\Phi}'_{\star,j}]_{\ell-1} \\ &- [r_1 \cdot \boldsymbol{A}_{\star,j} + \boldsymbol{A}'_{\star,j}]_{\ell-1} \cdot [\boldsymbol{\Phi}'_{\star,j}]_{n-\ell} \\ &- [r_2 \cdot \boldsymbol{\Phi}_{\star,j} + \boldsymbol{\Phi}'_{\star,j}]_{\ell-1} \cdot [\boldsymbol{A}'_{\star,j}]_{n-\ell} \\ &+ [\boldsymbol{C}'_{\star,j}]_{n-\ell} - r_1 \cdot r_2 \cdot [\boldsymbol{C}_{\star,j}]_{n-\ell}\end{aligned}$$

   and

$$\begin{aligned}[\boldsymbol{O}^{(2)}_{\star,j}]_{n-1} = \\ &[r_1 \cdot \boldsymbol{B}_{\star,j} + \boldsymbol{B}'_{\star,j}]_{\ell-1} \cdot [r_2 \cdot \boldsymbol{\Phi}_{\star,j} + \boldsymbol{\Phi}'_{\star,j}]_{\ell-1} \\ &- [r_1 \cdot \boldsymbol{B}_{\star,j} + \boldsymbol{B}'_{\star,j}]_{\ell-1} \cdot [\boldsymbol{\Phi}'_{\star,j}]_{n-\ell} \\ &- [r_2 \cdot \boldsymbol{\Phi}_{\star,j} + \boldsymbol{\Phi}'_{\star,j}]_{\ell-1} \cdot [\boldsymbol{B}'_{\star,j}]_{n-\ell} \\ &+ [\boldsymbol{D}'_{\star,j}]_{n-1} - r_1 \cdot r_2 \cdot [\boldsymbol{D}_{\star,j}]_{n-1}.\end{aligned}$$

5. All parties invoke $\mathcal{F}_{\mathsf{Coin}}$ to generate random values $u_1, \ldots, u_{2\lambda}$. Then all parties locally compute

$$[\boldsymbol{o}]_{n-1} = \sum_{j=1}^{\lambda} (u_j \cdot [\boldsymbol{O}^{(1)}_{\star,j}]_{n-1} + u_{j+\lambda} \cdot [\boldsymbol{O}^{(2)}_{\star,j}]_{n-1}.$$

6. All parties use $\mathcal{F}_{\mathsf{Commit}}$ to commit their shares of $[\boldsymbol{o}]_{n-1}$.
7. All parties open their commitments to every party P_i. Then P_i checks whether $\boldsymbol{o}$ is a vector of 0. If not, P_i aborts.

Step 3. Prepare $\langle a_j \rangle, a_j \cdot G, \boldsymbol{B}_{\star,j} \cdot G$.

1. **Preparing Random Additive Sharings of** 0: All parties repeat the following steps $\lceil \lambda/(2\ell-1) \rceil$ times to prepare λ random additive sharings of 0 among the real parties.
   (a) Each party P_i shares a random additive sharing of 0, denoted by $\left\langle o^{(i)} \right\rangle$.
   (b) All parties agree on a Vandermonde matrix $\boldsymbol{M}$ of dimension $(2\ell-1) \times n$.
   (c) All parties compute $(\langle o_1 \rangle, \ldots, \langle o_{2\ell-1} \rangle) = \boldsymbol{M} \cdot \left(\left\langle o^{(1)} \right\rangle, \ldots, \left\langle o^{(n)} \right\rangle \right)$.
2. **Preparing** $(a_j \cdot G, [\boldsymbol{B}_{\star,j} \cdot G]_{\ell-1})$: For all $j \in [\lambda]$, all parties do the following.
   (a) All parties locally compute $\langle a_j \cdot G \rangle$ and $[\boldsymbol{B}_{\star,j} \cdot G]_{n-1}$ by locally multiplying their shares of $(\langle a_j \rangle, [\boldsymbol{B}_{\star,j}]_{n-1})$ with G. Then all parties send their shares of $\langle a_j \cdot G \rangle$ and $[\boldsymbol{B}_{\star,j} \cdot G]_{n-1}$ to P_1.
   (b) P_1 reconstructs $a_j \cdot G$ and $\boldsymbol{B}_{\star,j} \cdot G$. Then P_1 sends $a_j \cdot G$ and distributes $[\boldsymbol{B}_{\star,j} \cdot G]_{\ell-1}$ to all parties.
   (c) All parties invoke $\mathcal{F}_{\mathsf{Coin}}$ to generate $\chi_1, \ldots, \chi_\lambda$. Then all parties locally compute
   $$[\boldsymbol{b} \cdot G]_{\ell-1} = \sum_{j=1}^{\lambda} \chi_j \cdot [\boldsymbol{B}_{\star,j} \cdot G]_{\ell-1}.$$
   (d) All parties send their shares of $[\boldsymbol{b} \cdot G]_{\ell-1}$ to every party. Then each party P_i checks whether $[\boldsymbol{b} \cdot G]_{\ell-1}$ is a valid degree-$(\ell-1)$ packed Shamir sharing. If not, P_i aborts.
   (e) All parties refresh $\langle a_j \rangle$ by computing $\langle a_j \rangle := \langle a_j \rangle + \langle o_j \rangle$.
3. **Verification**:
   (a) For all $j \in [\lambda]$, all parties locally compute $[\tilde{\boldsymbol{C}}_{\star,j} \cdot G]_{n-\ell} = (a_j \cdot G) \cdot [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}$. All parties also locally compute $[\boldsymbol{C}_{\star,j} \cdot G]_{n-\ell}$ from $[\boldsymbol{C}_{\star,j}]_{n-\ell}$. Then for each $i \in \{1, \ldots, \ell\}$, all parties compute $\left\langle \tilde{C}_{i,j} \cdot G \right\rangle$ from $[\tilde{\boldsymbol{C}}_{\star,j} \cdot G]_{n-\ell}$, compute $\langle C_{i,j} \cdot G \rangle$ from $[\boldsymbol{C}_{\star,j} \cdot G]_{n-\ell}$, and set $\left\langle \Delta^C_{i,j} \cdot G \right\rangle = \langle C_{i,j} \cdot G \rangle - \left\langle \tilde{C}_{i,j} \cdot G \right\rangle$.
   (b) For all $j \in [\lambda]$, all parties locally compute $[\tilde{\boldsymbol{D}}_{\star,j} \cdot G]_{n-1} = [\boldsymbol{B}_{\star,j} \cdot G]_{\ell-1} \cdot [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}$. All parties also locally compute $[\boldsymbol{D}_{\star,j} \cdot G]_{n-1}$ from $[\boldsymbol{D}_{\star,j}]_{n-1}$. Then for each $i \in \{1, \ldots, \ell\}$, all parties compute $\left\langle \tilde{D}_{i,j} \cdot G \right\rangle$ from $[\tilde{\boldsymbol{D}}_{\star,j} \cdot G]_{n-1}$, compute $\langle D_{i,j} \cdot G \rangle$ from $[\boldsymbol{D}_{\star,j} \cdot G]_{n-1}$, and set $\left\langle \Delta^D_{i,j} \cdot G \right\rangle = \langle D_{i,j} \cdot G \rangle - \left\langle \tilde{D}_{i,j} \cdot G \right\rangle$.
   (c) All parties prepare a random additive sharing of 0 as follows:
      – Each P_i distributes a random additive sharing of 0, denoted by $\langle o_i \rangle$ to all parties.
      – All parties locally compute $\langle o \rangle = \sum_{i=1}^{n} \langle o_i \rangle$.
   (d) All parties invoke $\mathcal{F}_{\mathsf{Coin}}$ to generate $\chi_1, \ldots, \chi_{2\lambda}$. Then all parties locally compute
   $$\langle \Delta \cdot G \rangle = \langle o \cdot G \rangle + \sum_{j=1}^{\lambda} \chi_j \cdot \left\langle \Delta^C_{i,j} \cdot G \right\rangle + \sum_{j=1}^{\lambda} \chi_{\lambda+j} \cdot \left\langle \Delta^D_{i,j} \cdot G \right\rangle.$$

(e) All parties use $\mathcal{F}_{\mathsf{Commit}}$ to commit their shares of $\langle \Delta \cdot G \rangle$.
(f) All parties open their commitments to every party P_i. Then P_i checks whether $\Delta \cdot G = 1$. If not, P_i aborts.

The communication complexity of Π_{Tuple} is studied in Section 5.4.

Theorem 3. *Protocol Π_{Tuple} securely realizes $\mathcal{F}_{\mathsf{Tuple}}$ in the $(\mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{Mult}})$-hybrid model.*

5.3 Proof of Theorem 3

We start with the construction of an ideal adversary $\mathcal{S}$. $\mathcal{S}$ starts running the code of $\mathcal{A}$ and simulating the behaviors of honest parties and internal functionalities.

In the beginning, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{Coin}}$ to generate a random seed and decides each subset V_k of parties. Then $\mathcal{S}$ counts the number of subsets such that all parties in the subset are corrupted, i.e., the number of corrupted virtual parties. If the number of corrupted virtual parties is larger than $(N+1)/2 - \lambda$, $\mathcal{S}$ aborts. Without loss of generality, assume that the first $(N-1)/2 + \lambda$ virtual parties are honest. We further assume that the first party in the committee of each honest virtual party V_k is honest, denoted by $P_{V_k[0]}$. In the following, when emulating each virtual party V_k, $\mathcal{S}$ will generate all messages sent to honest parties in V_k except $P_{V_k[0]}$ when $k \leq (N-1)/2 + \lambda$, and to all honest parties in V_k when $k > (N-1)/2 + \lambda$. Then $\mathcal{S}$ will honestly follow the protocol for those honest parties. Effectively, we assume that only the first $(N-1)/2 + \lambda$ virtual parties are honest and each honest virtual party only contains a single honest real party. Let $\mathcal{C}^V = \{V_i \mid i > (N-1)/2 + \lambda\}$ and $\mathcal{H}^V = \{V_i \mid i \leq (N-1)/2 + \lambda\}$. We also use $\mathcal{C}$ to denote the set of real corrupted parties and $\mathcal{H}$ to denote the set of real honest parties.

Simulation of Step 1. In Step 1.1, $\mathcal{S}$ does nothing. In Step 1.2, for each degree-$(N-1)/2$ packed Shamir sharing dealt by an honest party P_i, $\mathcal{S}$ samples a random additive sharing and distributes it to parties in $V_k \in \mathcal{C}^V$. Then $\mathcal{S}$ samples a random value and sends it to each party in $V_k \in \mathcal{H}^V$ except $P_{V_k[0]}$. For each degree-$(N-1)/2$ packed Shamir sharing dealt by a corrupted party P_i, $\mathcal{S}$ records the value received from P_i for each $P_{V_k[0]}$ where $V_k \in \mathcal{H}^V$.

In Step 1.3, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{Mult}}$ as follows: When $\mathcal{F}_{\mathsf{Mult}}$ is invoked by parties in $V_k \in \mathcal{H}^V$, $\mathcal{S}$ records the values received from parties except $P_{V_k[0]}$. Then $\mathcal{S}$ samples a random value for each honest party except $P_{V_k[0]}$ and sends it to this honest party as his share of $\langle c \rangle$. $\mathcal{S}$ also sends back the shares of $\langle c \rangle$ to corrupted parties in V_k. When $\mathcal{F}_{\mathsf{Mult}}$ is invoked by parties in $V_k \in \mathcal{C}^V$, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{Mult}}$.

Recall that

$$[\boldsymbol{a}]^V_{(N-1)/2} = \sum_{i=1}^{n} [\boldsymbol{a}^{(\star,i)}]^V_{(N-1)/2}.$$

Let $[\boldsymbol{a}]^{V,\mathcal{H}}_{(N-1)/2} = \sum_{i\in\mathcal{H}}[\boldsymbol{a}^{(\star,i)}]^{V}_{(N-1)/2}$ and $[\boldsymbol{a}]^{V,\mathcal{C}}_{(N-1)/2} = \sum_{i\in\mathcal{C}}[\boldsymbol{a}^{(\star,i)}]^{V}_{(N-1)/2}$. Then $[\boldsymbol{a}]^{V}_{(N-1)/2} = [\boldsymbol{a}]^{V,\mathcal{H}}_{(N-1)/2} + [\boldsymbol{a}]^{V,\mathcal{C}}_{(N-1)/2}$. $\mathcal{S}$ computes the shares of $[\boldsymbol{a}]^{V,\mathcal{H}}_{(N-1)/2}$ of corrupted virtual parties and determines $[\boldsymbol{a}]^{V,\mathcal{C}}_{(N-1)/2}$ as follows: When simulating $\mathcal{F}_{\mathsf{Mult}}$ to compute $[\boldsymbol{C}_{1,\star}]^{V}_{N-1}$,

- For each corrupted virtual party V_k, $\mathcal{S}$ receives the additive sharing of the k-th share of $[\boldsymbol{a}]^{V}_{(N-1)/2}$. $\mathcal{S}$ reconstructs the secret and compute the k-th share of $[\boldsymbol{a}]^{V,\mathcal{C}}_{(N-1)/2}$ by subtracting the k-th share of $[\boldsymbol{a}]^{V,\mathcal{H}}_{(N-1)/2}$.
- For each honest virtual party V_k, $\mathcal{S}$ receives all but the first party's shares of the additive sharing of the k-th share of $[\boldsymbol{a}]^{V}_{(N-1)/2}$. Then for all but the first party, $\mathcal{S}$ computes their shares of the additive sharing of the k-th share of $[\boldsymbol{a}]^{V,\mathcal{C}}_{(N-1)/2}$ by subtracting the parts for $[\boldsymbol{a}]^{V,\mathcal{H}}_{(N-1)/2}$ which are explicitly generated by $\mathcal{S}$. For $P_{V_k[0]}$, $\mathcal{S}$ computes the additive sharing of the k-th share of $[\boldsymbol{a}]^{V,\mathcal{C}}_{(N-1)/2}$. Then $\mathcal{S}$ reconstructs the k-th share of $[\boldsymbol{a}]^{V,\mathcal{C}}_{(N-1)/2}$.

$\mathcal{S}$ uses the first $(N+1)/2$ shares to determine $[\boldsymbol{a}]^{V,\mathcal{C}}_{(N-1)/2}$ and sets $\delta[\boldsymbol{A}_{1,\star}]^{V,\mathcal{C}}_{(N-1)/2}$ to be the difference between the actual shares computed above and $[\boldsymbol{a}]^{V,\mathcal{C}}_{(N-1)/2}$, which is a vector of N values and the first $(N+1)/2$ entries are 0. $\mathcal{S}$ follows the same process to determine $\delta[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$, which is the difference between the actual shares extracted out when computing $[\boldsymbol{C}_{j,\star}]^{V}_{N-1}$ and $[\boldsymbol{a}]^{V,\mathcal{C}}_{(N-1)/2}$ determined above. Note that when $j \geq 2$, the first $(N+1)/2$ entries of $\delta[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$ may not be 0.

Similarly, $\mathcal{S}$ defines $[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{H}}_{(N-1)/2}$, $[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$ and determines $[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$, $\delta[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$ when simulating $\mathcal{F}_{\mathsf{Mult}}$ to compute $[\boldsymbol{C}_{j,\star}]^{V}_{N-1}$.

When simulating $\mathcal{F}_{\mathsf{Mult}}$ to compute $[\boldsymbol{D}_{j,\star}]^{V}_{N-1}$, $\mathcal{S}$ defines $[\boldsymbol{B}_{j,\star}]^{V,\mathcal{H}}_{(N-1)/2}$, $[\boldsymbol{B}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$ and determines $[\boldsymbol{B}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$, $\delta[\boldsymbol{B}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$. In addition, $\mathcal{S}$ defines $\tilde{\delta}[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$ to be the difference between the actual shares of $[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$ and $[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$ determined above.

Now for $[\boldsymbol{C}_{j,\star}]^{V}_{N-1}$, we may write it as

$$\begin{aligned}
[\boldsymbol{C}_{j,\star}]^{V}_{N-1} &= ([\boldsymbol{a}]^{V}_{(N-1)/2} + \delta[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}) \\
&\quad \cdot ([\boldsymbol{\Phi}_{j,\star}]^{V}_{(N-1)/2} + \delta[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}) \\
&= [\boldsymbol{a} \star \boldsymbol{\Phi}_{j,\star}]^{V}_{N-1} + [\boldsymbol{a}]^{V}_{(N-1)/2} \cdot \delta[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2} \\
&\quad + \delta[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2} \cdot [\boldsymbol{\Phi}_{j,\star}]^{V}_{(N-1)/2} \\
&\quad + \delta[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2} \cdot \delta[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}.
\end{aligned}$$

Note that $[\boldsymbol{a}]^{V}_{(N-1)/2}$ is determined by the shares of corrupted virtual parties, which have been generated explicitly, and the secrets $\boldsymbol{a}$. Thus $[\boldsymbol{a}]^{V}_{(N-1)/2} \cdot \delta[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$ can be viewed as a degree-$(N-1)$ packed Shamir sharing where each secret is a linear combination of values in $\boldsymbol{a}$ and the linear combination is

solely determined by $\delta[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$. Similarly, $\delta[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2} \cdot [\boldsymbol{\Phi}_{j,\star}]^{V}_{(N-1)/2}$ can be viewed as a degree-$(N-1)$ packed Shamir sharing where each secret is a linear combination of values in $\boldsymbol{\Phi}_{j,\star}$ and the linear combination is solely determined by $\delta[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$. Finally, $\delta[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2} \cdot \delta[\boldsymbol{\Phi}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$ can be viewed as a constant degree-$(N-1)$ packed Shamir sharing. Thus, $\mathcal{S}$ computes vectors $\boldsymbol{e}^{C}_{j,i,1}, \boldsymbol{e}^{C}_{j,i,2}$ and a constant $e^{C}_{j,i}$ such that $C_{j,i} = a_i \cdot \Phi_{j,i} + \boldsymbol{e}^{C}_{j,i,1} \cdot \boldsymbol{a} + \boldsymbol{e}^{C}_{j,i,2} \cdot \boldsymbol{\Phi}_{j,\star} + e^{C}_{j,i}$.

Similarly, $\mathcal{S}$ computes vectors $\boldsymbol{e}^{D}_{j,i,1}, \boldsymbol{e}^{D}_{j,i,2}$ and a constant $e^{D}_{j,i}$ such that $D_{j,i} = B_{j,i} \cdot \Phi_{j,i} + \boldsymbol{e}^{D}_{j,i,1} \cdot \boldsymbol{\Phi}_{j,\star} + \boldsymbol{e}^{D}_{j,i,2} \cdot \boldsymbol{B}_{j,\star} + e^{D}_{j,i}$.

In Step 1.4, for all $j \in \{1, \ldots, \ell\}$, $\mathcal{S}$ defines $[\boldsymbol{A}_{j,\star}]^{V,\mathcal{H}}_{N-1} = [\boldsymbol{a}]^{V,\mathcal{H}}_{(N-1)/2}$ and then defines $[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{N-1}$ as follows:

- $\mathcal{S}$ sets the k-th share of $[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{N-1}$ to be the same as that of $[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2} + \delta[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{(N-1)/2}$ for $k \in \{1, \ldots, N-\lambda\}$.
- $\mathcal{S}$ computes $[\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{N-1}$ by using the secrets $\boldsymbol{a}^{\mathcal{C}}$ and the shares from the first $N-\lambda$ virtual parties determined above.

Next $\mathcal{S}$ sets $[\boldsymbol{A}_{j,\star}]^{V}_{N-1} = [\boldsymbol{A}_{j,\star}]^{V,\mathcal{H}}_{N-1} + [\boldsymbol{A}_{j,\star}]^{V,\mathcal{C}}_{N-1}$. When preparing $[\boldsymbol{A}^{(k,\star)}]_{n-\ell}$,

- If $V_k \in \mathcal{H}^V$, note that each party essentially distributes a degree-$(n-\ell)$ packed Shamir sharing of his additive shares of the k-th shares of $[\boldsymbol{A}_{1,\star}]^{V}_{N-1}, \ldots, [\boldsymbol{A}_{\ell,\star}]^{V}_{N-1}$. $\mathcal{S}$ generates random values as shares of corrupted parties and send them to corrupted parties on behalf of $P_{V_k[0]}$. For other parties in V_k, $\mathcal{S}$ has learnt their additive shares. $\mathcal{S}$ receives the honest parties' shares of the degree-$(n-\ell)$ packed Shamir sharing dealt by these parties and then compute the whole sharing by using the honest parties' shares and the secrets, and setting the first $n-4\ell+2$ corrupted parties' shares to be 0.
- If $V_k \in \mathcal{C}^V$, $\mathcal{S}$ receives and computes the honest parties' shares of $[\boldsymbol{A}^{(k,\star)}]_{n-\ell}$. Then $\mathcal{S}$ computes the whole sharing by using the honest parties' shares and the secrets $\boldsymbol{A}^{(k,\star)}$ determined above, and setting the first $n-4\ell+2$ corrupted parties' shares to be 0.

$\mathcal{S}$ follows the protocol to compute the corrupted parties' shares of $[\boldsymbol{A}_{\star,j}]_{n-\ell}$.

$\mathcal{S}$ simulates the steps for preparing $[\boldsymbol{B}_{\star,j}]_{n-\ell}, [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}, [\boldsymbol{C}_{\star,j}]_{n-\ell}, [\boldsymbol{D}_{\star,j}]_{n-1}$ in a similar way.

Simulation of Step 2. Following the simulation in Step 1. $\mathcal{S}$ has computed the corrupted parties shares of $[\boldsymbol{A}_{\star,j}]_{n-\ell}$, $[\boldsymbol{B}_{\star,j}]_{n-\ell}$, $[\boldsymbol{\Phi}_{\star,j}]_{n-\ell}$, $[\boldsymbol{C}_{\star,j}]_{n-\ell}$, $[\boldsymbol{D}_{\star,j}]_{n-1}$) and $[\boldsymbol{A}'_{\star,j}]_{n-\ell}$, $[\boldsymbol{B}'_{\star,j}]_{n-\ell}$, $[\boldsymbol{\Phi}'_{\star,j}]_{n-\ell}$, $[\boldsymbol{C}'_{\star,j}]_{n-\ell}$, $[\boldsymbol{D}'_{\star,j}]_{n-1}$. $\mathcal{S}$ has also computed

$$\{(\boldsymbol{e}^{C}_{i,j,1}, \boldsymbol{e}^{C}_{i,j,2}, e^{C}_{i,j}), (\boldsymbol{e}^{D}_{i,j,1}, \boldsymbol{e}^{D}_{i,j,2}, e^{D}_{i,j})\}_{i,j}$$

and

$$\{(\boldsymbol{e}^{C'}_{i,j,1}, \boldsymbol{e}^{C'}_{i,j,2}, e^{C'}_{i,j}), (\boldsymbol{e}^{D'}_{i,j,1}, \boldsymbol{e}^{D'}_{i,j,2}, e^{D'}_{i,j})\}_{i,j}.$$

In Step 2.1, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{Coin}}$ to generate r_1, r_2. Then $\mathcal{S}$ randomly samples a vector such that all entries are identical as $r_1 \cdot \boldsymbol{A}_{\star,j} + \boldsymbol{A}'_{\star,j}$. $\mathcal{S}$ also

samples two random vectors as $r_1 \cdot \boldsymbol{B}_{\star,j} + \boldsymbol{B}'_{\star,j}$ and $r_2 \cdot \boldsymbol{\Phi}_{\star,j} + \boldsymbol{\Phi}'_{\star,j}$. Based on the shares of corrupted parties and the secrets, $\mathcal{S}$ generates the whole sharings $[r_1 \cdot \boldsymbol{A}_{\star,j} + \boldsymbol{A}'_{\star,j}]_{n-\ell}, [r_1 \cdot \boldsymbol{B}_{\star,j} + \boldsymbol{B}'_{\star,j}]_{n-\ell}, [r_2 \cdot \boldsymbol{\Phi}_{\star,j} + \boldsymbol{\Phi}'_{\star,j}]_{n-\ell}$.

$\mathcal{S}$ honestly follows the protocol in Step 2.2 and Step 2.3. If any sharing distributed by P_1 is not a valid degree-$(\ell-1)$ packed Shamir sharing but the check in Step 2.3 passes, $\mathcal{S}$ aborts.

Let $\delta(r_1 \cdot \boldsymbol{A}_{\star,j} + \boldsymbol{A}'_{\star,j})$ be the difference between the secrets of the sharing distributed by P_1 and the one computed by $\mathcal{S}$. Similarly, we define $\delta(r_1 \cdot \boldsymbol{B}_{\star,j} + \boldsymbol{B}'_{\star,j})$ and $\delta(r_2 \cdot \boldsymbol{\Phi}_{\star,j} + \boldsymbol{\Phi}'_{\star,j})$. Then, we have

$$
\begin{aligned}
O^{(1)}_{i,j} &= \delta(r_1 \cdot A_{i,j} + A'_{i,j}) \cdot (r_2 \cdot \Phi_{i,j} + \Phi'_{i,j}) \\
&\quad + (r_1 \cdot A_{i,j} + A'_{i,j}) \cdot \delta(r_2 \cdot \Phi_{i,j} + \Phi'_{i,j}) \\
&\quad + \ \delta(r_1 \cdot A_{i,j} + A'_{i,j}) \cdot \delta(r_2 \cdot \Phi_{i,j} + \Phi'_{i,j}) \\
&\quad - \delta(r_1 \cdot A_{i,j} + A'_{i,j}) \cdot (\Phi'_{i,j} + r_2 \cdot \Phi_{i,j}) \\
&\quad - \ \delta(r_2 \cdot \Phi_{i,j} + \Phi'_{i,j}) \cdot (a'_j + r_1 \cdot a_j) \\
&\quad + (\boldsymbol{e}^{C'}_{i,j,1} \cdot (\boldsymbol{a}' + r_1 \cdot \boldsymbol{a}) \\
&\quad + \boldsymbol{e}^{C'}_{i,j,2} \cdot (\boldsymbol{\Phi}'_{i,\star} + r_2 \cdot \boldsymbol{\Phi}_{i,\star}) + e^{C'}_j) \\
&\quad + \ \delta(r_1 \cdot A_{i,j} + A'_{i,j}) \cdot r_2 \cdot \Phi_{i,j} \\
&\quad + \delta(r_2 \cdot \Phi_{i,j} + \Phi'_{i,j}) \cdot r_1 \cdot a_j \\
&\quad - \ (r_1 \cdot r_2 \cdot \boldsymbol{e}^{C}_{i,j,1} + r_1 \cdot \boldsymbol{e}^{C'}_{i,j,1}) \cdot \boldsymbol{a} \\
&\quad - (r_1 \cdot r_2 \cdot \boldsymbol{e}^{C}_{i,j,2} + r_2 \cdot \boldsymbol{e}^{C'}_{i,j,2}) \cdot \boldsymbol{\Phi}_{i,\star} - r_1 \cdot r_2 \cdot e^{C}_j.
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
O^{(2)}_{i,j} &= \delta(r_1 \cdot B_{i,j} + B'_{i,j}) \cdot (r_2 \cdot \Phi_{i,j} + \Phi'_{i,j}) \\
&\quad + (r_1 \cdot B_{i,j} + B'_{i,j}) * \delta(r_2 \cdot \Phi_{i,j} + \Phi'_{i,j}) \\
&\quad + \ \delta(r_1 \cdot B_{i,j} + B'_{i,j}) \cdot \delta(r_2 \cdot \Phi_{i,j} + \Phi'_{i,j}) \\
&\quad - \delta(r_1 \cdot B_{i,j} + B'_{i,j}) \cdot (\Phi'_{i,j} + r_2 \cdot \Phi_{i,j}) \\
&\quad - \ \delta(r_2 \cdot \Phi_{i,j} + \Phi'_{i,j}) \cdot (B'_{i,j} + r_1 \cdot B_{i,j}) \\
&\quad + (\boldsymbol{e}^{D'}_{i,j,1} \cdot (\boldsymbol{B}'_{i,\star} + r_1 \cdot \boldsymbol{B}_{i,\star}) \\
&\quad + \boldsymbol{e}^{D'}_{i,j,2} \cdot (\boldsymbol{\Phi}'_{i,\star} + r_2 \cdot \boldsymbol{\Phi}_{i,\star}) + e^{D'}_j) \\
&\quad + \ \delta(r_1 \cdot B_{i,j} + B'_{i,j}) \cdot r_2 \cdot \Phi_{i,j} \\
&\quad + \delta(r_2 \cdot \Phi_{i,j} + \Phi'_{i,j}) \cdot r_1 \cdot B_{i,j} \\
&\quad - \ (r_1 \cdot r_2 \cdot \boldsymbol{e}^{D}_{i,j,1} + r_1 \cdot \boldsymbol{e}^{D'}_{i,j,1}) \cdot \boldsymbol{B}_{i,\star} \\
&\quad - (r_1 \cdot r_2 \cdot \boldsymbol{e}^{D}_{i,j,2} + r_2 \cdot \boldsymbol{e}^{D'}_{i,j,2}) \cdot \boldsymbol{\Phi}_{i,\star} - r_1 \cdot r_2 \cdot e^{D}_j.
\end{aligned}
$$

$\mathcal{S}$ checks whether the coefficients of $a_j, B_{i,j}, \Phi_{i,j}$ are all 0. If not, $\mathcal{S}$ randomly samples $\boldsymbol{a}, \boldsymbol{B}, \boldsymbol{\Phi}$ and compute the shares of honest parties accordingly. Then $\mathcal{S}$ honestly follows the rest of Step 2. If the final check passes, $\mathcal{S}$ aborts.

If the coefficients of $a_j, B_{i,j}, \Phi_{i,j}$ are all 0 but there exists $j' \neq j$ such that the j'-th entry of any of $\boldsymbol{e}^C_{i,j,1}, \boldsymbol{e}^C_{i,j,2}, \boldsymbol{e}^D_{i,j,1}, \boldsymbol{e}^D_{i,j,1}$ is not zero, or the j-th entry of $\boldsymbol{e}^C_{i,j,1}$ is not equal to the j-th entry of $\boldsymbol{e}^D_{i,j,1}$, $\mathcal{S}$ aborts.

Otherwise, $\mathcal{S}$ computes $\boldsymbol{O}^{(1)}_{\star,j}, \boldsymbol{O}^{(2)}_{\star,j}$. Then $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{Coin}}$ to generate $u_1, \ldots, u_{2\lambda}$. Finally, $\mathcal{S}$ computes $\boldsymbol{o}$ and generates a random degree-$(n-1)$ packed Shamir sharing as $[\boldsymbol{o}]_{n-1}$ based on the secrets and the shares of corrupted parties. $\mathcal{S}$ honestly follows the rest of Step 2.

Simulation of Step 3. In Step 3.1, for each additive sharing of 0 distributed by an honest party, $\mathcal{S}$ samples random values as shares of corrupted parties and send them to corrupted parties. For each additive sharing of 0 distributed by a corrupted party, $\mathcal{S}$ receives the shares of honest parties. Then $\mathcal{S}$ sets the first $t-1$ corrupted parties' shares to be 0 and computes the whole additive sharing of 0.

In Step 3.2, for each $\langle a_j \rangle$, $\mathcal{S}$ sets the share of the first corrupted party to be $a^{\mathcal{C}}_j$, which is the j-th secret of $[\boldsymbol{a}]^{V,\mathcal{C}}_{(N-1)/2}$, and sets the rest of corrupted parties' shares to be 0. For each $[\boldsymbol{B}_{\star,j}]_{n-1}$, $\mathcal{S}$ determines a degree-$(n-1)$ packed Shamir sharing of $[\boldsymbol{B}^{\mathcal{C}}_{\star,j}]_{n-1}$ as follows:

- $\mathcal{S}$ sets the secrets to be $\boldsymbol{B}^{\mathcal{C}}_{\star,j}$, which is the j-th column of $\boldsymbol{B}^{\mathcal{C}}$ that is determined by the secrets of $[\boldsymbol{B}_{1,\star}]^{V,\mathcal{C}}_{(N-1)/2}, \ldots, [\boldsymbol{B}_{\ell,\star}]^{V,\mathcal{C}}_{(N-1)/2}$. Then $\mathcal{S}$ sets honest parties' shares and the first $n-3\ell+1$ corrupted parties' shares to be 0. Next, $\mathcal{S}$ computes the whole sharing $[\boldsymbol{B}^{\mathcal{C}}_{\star,j}]_{n-1}$ based on the values determined above. Finally, $\mathcal{S}$ sets the corrupted parties' shares of $[\boldsymbol{B}_{\star,j}]_{n-1}$ to be the same as those of $[\boldsymbol{B}^{\mathcal{C}}_{\star,j}]_{n-1}$.

Then, $\mathcal{S}$ checks that

- For all $i \in \{1, \ldots, \ell\}$, the j-th entry of $\boldsymbol{e}^C_{i,j,2}$ are all identical.

If true, $\mathcal{S}$ prepares the shares of corrupted parties that should be sent to $\mathcal{F}_{\mathsf{Tuple}}$ as follows.

1. For $\langle a_j \rangle$, $\mathcal{S}$ computes the shares of corrupted parties after the refresh and adds the first corrupted party's share by the j-th entry of $\boldsymbol{e}^C_{1,j,2}$.
2. For $[\boldsymbol{\Phi}_{\star,j}]_{n-\ell}$, $\mathcal{S}$ generates a degree-$(n-\ell)$ packed Shamir sharing of the vector that contains the j-th entry of $\boldsymbol{e}^C_{1,j,1}, \ldots, \boldsymbol{e}^C_{\ell,j,1}$ given that the shares of honest parties and the first $n-4\ell+2$ corrupted parties are set to be 0. Then $\mathcal{S}$ adds each corrupted party's share to his share of $[\boldsymbol{\Phi}_{\star,j}]_{n-\ell}$
3. For $[\boldsymbol{C}_{\star,j}]_{n-\ell}$, $\mathcal{S}$ generates a degree-$(n-\ell)$ packed Shamir sharing of the vector where the i-th entry is the difference between the j-th entry of $\boldsymbol{e}^C_{i,j,1} * \boldsymbol{e}^C_{i,j,2}$ and $e^C_{i,j}$ given that the shares of honest parties and the first $n-4\ell+2$ corrupted parties are set to be 0. Then $\mathcal{S}$ adds each corrupted party's share to his share of $[\boldsymbol{C}_{\star,j}]_{n-\ell}$.
4. For $[\boldsymbol{D}_{\star,j}]_{n-1}$, $\mathcal{S}$ generates a degree-$(n-1)$ packed Shamir sharing of the vector where the i-th entry is the difference between the j-th entry of $\boldsymbol{e}^D_{i,j,1} * \boldsymbol{e}^D_{i,j,2}$

and $e^D_{i,j}$ given that the shares of honest parties and the first $n - 3\ell + 1$ corrupted parties are set to be 0. Then $\mathcal{S}$ adds each corrupted party's share to his share of $[\boldsymbol{D}_{\star,j}]_{n-1}$.

Then $\mathcal{S}$ provides the shares of corrupted parties to $\mathcal{F}_{\mathsf{Tuple}}$ and receives $\tilde{\boldsymbol{a}} \cdot G, \tilde{\boldsymbol{B}}_{\star,j} \cdot G$ where $\tilde{\boldsymbol{B}}_{\star,j} \cdot G$ is reconstructed from corrupted parties shares of $[\boldsymbol{B}_{\star,j} \cdot G]_{\ell-1}$. $\mathcal{S}$ computes $a_j \cdot G$ by subtracting the j-th entry of $\boldsymbol{e}^C_{1,j,2} \cdot G$ from $\tilde{a}_j \cdot G$. $\mathcal{S}$ also computes $\boldsymbol{B}_{\star,j} \cdot G$ by subtracting the vector that contains the j-th entry of $\boldsymbol{e}^D_{1,j,2} \cdot G, \ldots, \boldsymbol{e}^D_{\ell,j,2} \cdot G$ from $\tilde{\boldsymbol{B}}_{\star,j} \cdot G$. Finally, $\mathcal{S}$ samples a random additive sharing $\langle a_j \cdot G \rangle$ based on the secret and the shares of corrupted parties. $\mathcal{S}$ also samples a random degree-$(n-1)$ packed Shamir sharing $[\boldsymbol{B}_{\star,j} \cdot G]_{n-1}$ based on the secrets and the shares of corrupted parties. $\mathcal{S}$ follows the rest of Step 3.2. If the sharings distributed by P_1 are not in the right degree but the check in Step 3.2.(d) passes, $\mathcal{S}$ aborts.

If the check fails, $\mathcal{S}$ generates random values as honest parties' shares of $\langle a_j \rangle, [\boldsymbol{B}_{\star,j}]_{n-1}$. Then $\mathcal{S}$ follows the rest of Step 3.2. If the sharings distributed by P_1 are not in the right degree but the check in Step 3.2.(d) passes, $\mathcal{S}$ aborts.

In Step 3.3, when the check in Step 3.2 passes, $\mathcal{S}$ checks the following:

- For all $j \in \{1, \ldots, \lambda\}$, $a_j \cdot G$ distributed by P_1 is the same as $\tilde{a}_j \cdot G$ received from the functionality.
- For all $j \in \{1, \ldots, \lambda\}$, $\boldsymbol{B}_{\star,j} \cdot G$ distributed by P_1 is the same as $\tilde{\boldsymbol{B}}_{\star,j} \cdot G$ extracted out from the corrupted parties' shares received from the functionality.

If true, $\mathcal{S}$ sets $\Delta^C_{i,j} = \Delta^D_{i,j} = 0$. When preparing a random additive sharing of 0, $\mathcal{S}$ follows the same strategy as that in Step 5 and computes the corrupted parties' shares of $\langle o \rangle$. Then $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{Coin}}$ and samples a random additive sharing of $\langle 0 \cdot G \rangle$ as $\langle \Delta \cdot G \rangle$ given the shares of corrupted parties. $\mathcal{S}$ honestly follows the rest of steps.

If either the check in Step 3.2 fails or the check in Step 3.3 fails, $\mathcal{S}$ generates a random degree-$(n-\ell)$ packed Shamir sharing as $[\boldsymbol{\Phi}_{\star,j}]_{n-\ell}$ given the shares of corrupted parties. $\mathcal{S}$ reconstructs $\boldsymbol{\Phi}$. Then $\mathcal{S}$ computes $\boldsymbol{C} \cdot G$ by

$$C_{j,i} \cdot G = (a_i \cdot G) \cdot \Phi_{j,i} + \boldsymbol{e}^C_{j,i,1} \cdot (\boldsymbol{a} \cdot G) + \boldsymbol{e}^C_{j,i,2} \cdot \boldsymbol{\Phi}_{j,\star} \cdot G + e^C_{j,i} \cdot G.$$

Similarly, $\mathcal{S}$ computes $\boldsymbol{D} \cdot G$ by

$$\begin{aligned} D_{j,i} \cdot G =& (B_{j,i} \cdot G) \cdot \Phi_{j,i} + \boldsymbol{e}^D_{j,i,1} \cdot (\boldsymbol{B}_{j,\star} \cdot G) \\ &+ \boldsymbol{e}^D_{j,i,2} \cdot \boldsymbol{\Phi}_{j,\star} \cdot G + e^D_{j,i} \cdot G. \end{aligned}$$

Next, $\mathcal{S}$ generates a random degree-$(n-\ell)$ packed Shamir sharing as $[\boldsymbol{C}_{\star,j} \cdot G]_{n-\ell}$ given the secrets and the shares of corrupted parties. $\mathcal{S}$ also generates a random degree-$(n-1)$ packed Shamir sharing as $[\boldsymbol{D}_{\star,j} \cdot G]_{n-1}$ given the secrets and the shares of corrupted parties. $\mathcal{S}$ honestly follow the rest of Step 3.3. If the final verification passes, $\mathcal{S}$ aborts.

Finally, $\mathcal{S}$ outputs the view of the adversary.

Hybrid Arguments. We show that the output in the ideal world is computationally indistinguishable from that in the real world.

$\mathtt{Hyb}_0$: In this hybrid, we consider the real-world execution. The output contains the view of the adversary and the output of honest parties.

$\mathtt{Hyb}_1$: In this hybrid, whenever an honest party distributes a random sharing (including degree-$(n-1)$ packed Shamir sharings, degree-$(n-\ell)$ packed Shamir sharings, additive sharings), $\mathcal{S}$ first samples random values as the shares of corrupted parties and then generates the shares of honest parties. By the property of the secret sharing schemes, $\mathtt{Hyb}_1$ and $\mathtt{Hyb}_0$ are identically distributed.

$\mathtt{Hyb}_2$: In this hybrid, $\mathcal{S}$ computes the shares of corrupted parties as well as the error vectors and constants as described above. This does not change the behaviors of honest parties. $\mathtt{Hyb}_2$ and $\mathtt{Hyb}_1$ are identically distributed.

$\mathtt{Hyb}_3$: In this hybrid, $\mathcal{F}_{\mathsf{Mult}}$ is simulated by $\mathcal{S}$ and $\mathcal{S}$ delays the generation of honest parties' shares for honest dealers until they are needed. In particular, this means that the shares related to $\langle a_j \rangle, [\boldsymbol{B}_{\star,j}]_{n-1}, [\boldsymbol{\Phi}_{\star,j}]_{n-1}$ are only prepared at the beginning of Step 2. $\mathtt{Hyb}_3$ and $\mathtt{Hyb}_2$ are identically distributed.

$\mathtt{Hyb}_4$: In this hybrid, $\mathcal{S}$ changes the way of generating the honest parties' sharesof $\langle a_j \rangle, [\boldsymbol{B}_{\star,j}]_{n-1}$. $\mathcal{S}$ samples a random value as a_j and samples a random vector as $\boldsymbol{B}_{\star,j}$. Then $\mathcal{S}$ computes $a_j^{\mathcal{H}} = a_j - a_j^{\mathcal{C}}$ and $\boldsymbol{B}_{\star,j}^{\mathcal{H}} = \boldsymbol{B}_{\star,j} - \boldsymbol{B}_{\star,j}^{\mathcal{C}}$. Finally, $\mathcal{S}$ generates a random additive sharing of $a_j^{\mathcal{H}}$ given that all corrupted parties' shares are 0, and generates a random degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{B}_{\star,j}^{\mathcal{H}}$ given that all corrupted parties shares are 0. $\mathcal{S}$ sets honest parties' shares of $\langle a_j \rangle, [\boldsymbol{B}_{\star,j}]_{n-1}$ to be those of $\langle a_j^{\mathcal{H}} \rangle, [\boldsymbol{B}_{\star,j}^{\mathcal{H}}]_{n-1}$. $\mathtt{Hyb}_4$ and $\mathtt{Hyb}_3$ are identically distributed.

$\mathtt{Hyb}_5$: In this hybrid, $\mathcal{S}$ no longer generates the shares of honest parties related to $[\boldsymbol{\Phi}_{\star,j}]_{n-1}$. Instead, $\mathcal{S}$ samples a random vector as $\boldsymbol{\Phi}_{\star,j}$, computes $\boldsymbol{\Phi}_{\star,j}^{\mathcal{H}} = \boldsymbol{\Phi}_{\star,j} - \boldsymbol{\Phi}_{\star,j}^{\mathcal{C}}$, computes the whole sharing $[\boldsymbol{\Phi}_{\star,j}]_{(N-1)/2}^{V,\mathcal{H}}$ based on the secrets and the shares of corrupted parties, and finally computes the shares of $[\boldsymbol{\Phi}_{\star,j}]_{(N-1)/2}^{V}$ of honest parties. To argue that $\mathtt{Hyb}_4$ and $\mathtt{Hyb}_3$ are identically distributed, it is sufficient to show that $\boldsymbol{\Phi}_{\star,j}^{\mathcal{H}}$ is identically distributed in both hybrids. Note that in $\mathtt{Hyb}_4$, it is just a random vector. In $\mathtt{Hyb}_3$, honest parties first sample their shares of $[\boldsymbol{\Phi}_{\star,j}]_{n-1}$, which fully determine $\boldsymbol{\Phi}^{\mathcal{H}}$. In particular, $\boldsymbol{\Phi}^{\mathcal{H}}$ is a uniform matrix. Thus, $\mathtt{Hyb}_5$ and $\mathtt{Hyb}_4$ are identically distributed.

$\mathtt{Hyb}_6$: In this hybrid, $\mathcal{S}$ changes the generation of honest parties' shares of $[\boldsymbol{C}_{\star,j}]_{n-\ell}$. After generating $\boldsymbol{a}, \boldsymbol{B}, \boldsymbol{\Phi}$, $\mathcal{S}$ computes $C_{j,i} = a_i \cdot \Phi_{j,i} + \boldsymbol{e}_{j,i,1}^{C} \cdot \boldsymbol{a} + \boldsymbol{e}_{j,i,2}^{C} \cdot \boldsymbol{\Phi}_{j,\star} + e_{j,i}^{C}$. Then $\mathcal{S}$ generates $[\boldsymbol{C}_{\star,j}]_{n-\ell}$ based on the secrets $\boldsymbol{C}_{j,\star}$ and the shares of corrupted parties computed by $\mathcal{S}$. According to the way $\mathcal{S}$ generates the shares of corrupted parties and computes the matrices, $\mathtt{Hyb}_6$ and $\mathtt{Hyb}_5$ are identically distributed.

$\mathtt{Hyb}_7$: In this hybrid, $\mathcal{S}$ changes the generation of honest parties' shares of $[\boldsymbol{D}_{\star,j}]_{n-1}$. After generating $\boldsymbol{a}, \boldsymbol{B}, \boldsymbol{\Phi}$, $\mathcal{S}$ computes $D_{j,i} = B_{j,i} \cdot \Phi_{j,i} + \boldsymbol{e}_{j,i,1}^{D} \cdot \boldsymbol{\Phi}_{j,\star} + \boldsymbol{e}_{j,i,2}^{D} \cdot \boldsymbol{B}_{j,\star} + e_{j,i}^{D}$. Then $\mathcal{S}$ samples a random $[\boldsymbol{D}_{\star,j}]_{n-1}$ based on the secrets $\boldsymbol{D}_{j,\star}$ and the shares of corrupted parties computed by $\mathcal{S}$. Note that

– According to the way $\mathcal{S}$ generates the shares of corrupted parties and computes the matrices, $\boldsymbol{D}_{\star,j}$ is identical to that in $\mathtt{Hyb}_6$.
– When $V_k \in \mathcal{H}^V$, $P_{V_k[0]}$ is an honest party. Thus, in $\mathtt{Hyb}_6$, each $[\boldsymbol{D}^{(k,\star)}]_{n-1}$ is a random degree-$(n-1)$ packed Shamir sharing given the shares of corrupted parties and the secrets. Thus, by the property of polynomials, $[\boldsymbol{D}_{\star,j}]_{n-1}$ is also a random degree-$(n-1)$ packed Shamir sharing given the shares of corrupted parties and the shares.

In summary, $\mathtt{Hyb}_7$ and $\mathtt{Hyb}_6$ are identically distributed.

$\mathtt{Hyb}_8$: In this hybrid, $\mathcal{S}$ changes the generation of honest parties' shares of

$$[\boldsymbol{A}_{\star,j}]_{n-\ell}, [\boldsymbol{B}_{\star,j}]_{n-\ell}, [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}.$$

After generating $\boldsymbol{a}, \boldsymbol{B}, \boldsymbol{\Phi}$, $\mathcal{S}$ generates $[\boldsymbol{A}_{\star,j}]_{n-\ell}, [\boldsymbol{B}_{\star,j}]_{n-\ell}, [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}$ based on the secrets $\boldsymbol{A}_{\star,j} = a_j \cdot \boldsymbol{1}, \boldsymbol{B}_{\star,j}, \boldsymbol{\Phi}_{j,\star}$ and the shares of corrupted parties computed by $\mathcal{S}$. According to the way $\mathcal{S}$ generates the shares of corrupted parties, the distributions of $\mathtt{Hyb}_8$ and $\mathtt{Hyb}_7$ are identical.

$\mathtt{Hyb}_9$: In this hybrid, $\mathcal{S}$ changes the way of generating $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\Phi}, \boldsymbol{A}', \boldsymbol{B}', \boldsymbol{\Phi}'$. After generating r_1, r_2 in Step 2.1, $\mathcal{S}$ randomly samples a vector such that all entries are identical as $r_1 \cdot \boldsymbol{A}_{\star,j} + \boldsymbol{A}'_{\star,j}$. $\mathcal{S}$ also samples two random vectors as $r_1 \cdot \boldsymbol{B}_{\star,j} + \boldsymbol{B}'_{\star,j}$ and $r_2 \cdot \boldsymbol{\Phi}_{\star,j} + \boldsymbol{\Phi}'_{\star,j}$. Then $\mathcal{S}$ randomly samples $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\Phi}$ and computes $\boldsymbol{A}', \boldsymbol{B}', \boldsymbol{\Phi}'$. The distributions of $\mathtt{Hyb}_9$ and $\mathtt{Hyb}_8$ are identical.

$\mathtt{Hyb}_{10}$: In this hybrid, $\mathcal{S}$ simulates Step 2 as described above. We argue the following three points:

– If there exists $j' \neq j$ such that the j'-th entry of any of $\boldsymbol{e}^C_{i,j,1}, \boldsymbol{e}^C_{i,j,2}, \boldsymbol{e}^D_{i,j,1}, \boldsymbol{e}^D_{i,j,1}$ is not zero, or the j-th entry of $\boldsymbol{e}^C_{i,j,1}$ is not equal to the j-th entry of $\boldsymbol{e}^D_{i,j,1}$, then the coefficients of $a_j, B_{i,j}, \Phi_{i,j}$ are not all 0 with overwhelming probability.
– If the coefficients of $a_j, B_{i,j}, \Phi_{i,j}$ are not all 0, then the check in Step 2 fails with overwhelming probability.
– If neither of the above happens, honest parties' shares of $[\boldsymbol{o}]_{n-1}$ are identically distributed to those in $\mathtt{Hyb}_9$ with overwhelming probability.

For the first point, note that if there exists $j' \neq j$ such that the j'-th entry of $\boldsymbol{e}^C_{i,j,1}$ is not zero, then when r_1, r_2 are randomly sampled, the coefficient of a'_j in $O^{(1)}_{i,j}$ is not zero. The same holds for $\boldsymbol{e}^C_{i,j,2}, \boldsymbol{e}^D_{i,j,1}, \boldsymbol{e}^D_{i,j,1}$. If the j-th entry of $\boldsymbol{e}^C_{i,j,1}$ is not equal to the j-th entry of $\boldsymbol{e}^D_{i,j,1}$, then when r_1, r_2 are randomly sampled, the coefficient of a_j in $O^{(1)}_{i,j}$ and the coefficient of $B_{i,j}$ in $O^{(2)}_{i,j}$ are different with overwhelming probability, which means that at least one of them is non-zero.

For the second point, since $\boldsymbol{a}, \boldsymbol{B}, \boldsymbol{\Phi}$ are secret values that are unknown to corrupted parties, if the coefficients of $a_j, B_{i,j}, \Phi_{i,j}$ are not all 0, then with overwhelming probability, at least one entry of $\boldsymbol{o}$ is uniformly random and not known to corrupted parties. In this case, the probability that the verification in Step 2 passes is negligible.

For the last point, when the coefficients of $a_j, B_{i,j}, \Phi_{i,j}$ are all 0, $O^{(1)}_{i,j}, O^{(2)}_{i,j}$ can be fully computed by $\mathcal{S}$ without using $\boldsymbol{a}, \boldsymbol{B}, \boldsymbol{\Phi}$. Furthermore, since $[\boldsymbol{D}'_{\star,j}]_{n-1}$

is a random degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{D}'_{\star,j}$, $[\boldsymbol{O}^{(2)}_{\star,j}]_{n-1}$ is also a random degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{O}^{(2)}_{\star,j}$. Thus, with overwhelming probability, $[\boldsymbol{o}]_{n-1}$ is a random degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{o}$.

In summary, $\mathtt{Hyb}_{10}$ is statistically close to $\mathtt{Hyb}_9$.

$\mathtt{Hyb}_{11}$: In this hybrid, $\mathcal{S}$ randomly samples $\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{B}}$ and computes $\boldsymbol{a}, \boldsymbol{B}$. Note that in both $\mathtt{Hyb}_{11}$ and $\mathtt{Hyb}_{10}$, $\boldsymbol{a}, \boldsymbol{B}$ are uniformly random.. Thus, $\mathtt{Hyb}_{11}$ and $\mathtt{Hyb}_{10}$ are identically distributed.

$\mathtt{Hyb}_{12}$: In this hybrid, $\mathcal{S}$ performs the check in Step 3.2 and Step 3.3 as described above. If the check fails but the final verification passes, $\mathcal{S}$ aborts. We show that $\mathtt{Hyb}_{12}$ and $\mathtt{Hyb}_{11}$ are statistically indistinguishable. We focus on the case where at least one condition in the checks does not match.

First, consider the case where $\bar{\boldsymbol{a}} \cdot G$ returned by P_1 is different from $\tilde{\boldsymbol{a}} \cdot G$. Suppose the j-th entry does not match. Then there exists i such that $\bar{a}_j - a_j$ is different from the j-th entry of $\boldsymbol{e}^C_{i,j,2}$. In this case, $\Delta^C_{i,j}$ is masked by the difference between $\bar{a}_j - a_j$ and the j-th entry of $\boldsymbol{e}^C_{i,j,2}$ times $\Phi_{i,j}$. Since $\Phi_{i,j}$ is a random element that is only used in Step 3.3, the probability that the final check passes is negligible.

Similarly, when $\bar{\boldsymbol{B}} \cdot G$ returned by P_1 is different from $\tilde{\boldsymbol{B}} \cdot G$, then there exists i such that $\bar{B}_{i,j} - B_{i,j}$ is different from the j-th entry of $\boldsymbol{e}^D_{i,j,2}$. In this case, $\Delta^D_{i,j}$ is masked by the difference between $\bar{B}_{i,j} - B_{i,j}$ and the j-th entry of $\boldsymbol{e}^D_{i,j,2}$ times $\Phi_{i,j}$. Since $\Phi_{i,j}$ is a random element that is only used in Step 3.3, the probability that the final check passes is negligible.

Now assume that $\bar{\boldsymbol{a}} = \tilde{\boldsymbol{a}}$ and $\bar{\boldsymbol{B}} = \tilde{\boldsymbol{B}}$. If the j-th entries of $\boldsymbol{e}^C_{1,j,2}, \ldots, \boldsymbol{e}^C_{\ell,j,2}$ are not all identical, then there exists i such that $\tilde{a}_j - a_j$ is different from the j-th entry of $\boldsymbol{e}^C_{i,j,2}$. Following the same argument, the probability that the final check passes is negligible.

In summary, $\mathtt{Hyb}_{12}$ and $\mathtt{Hyb}_{11}$ are statistically close.

$\mathtt{Hyb}_{13}$: In this hybrid, instead of sampling $\tilde{a}, \tilde{\boldsymbol{B}}, \boldsymbol{\Phi}$, $\mathcal{S}$ provides the shares of corrupted parties to $\mathcal{F}_{\mathsf{Tuple}}$. This corresponds to the ideal world. Note that in $\mathtt{Hyb}_{12}$, when using the shares computed by $\mathcal{S}$, we have the correct multiplication relations as computed by $\mathcal{F}_{\mathsf{Tuple}}$. Thus, $\mathtt{Hyb}_{13}$ and $\mathtt{Hyb}_{12}$ are identically distributed.

5.4 Communication Complexity of Π_{Tuple}

Cost Analysis. We list the concrete cost of each step below. The cost of step 1 has taken the repetition into the consideration.

- Step 1.2: $6 \cdot N \cdot n \cdot s$ elements of communication.
- Step 1.3: $4 \cdot N \cdot \ell$ invocations of $\mathcal{F}_{\mathsf{Mult}}$.
- Step 1.4: $10 \cdot N \cdot n \cdot s$ elements of communication.
- Step 2: 3 invocations of $\mathcal{F}_{\mathsf{Coin}}$ to generate $5\lambda + 1$ random values, 1 invocation of $\mathcal{F}_{\mathsf{Commit}}$ for committing and opening.
- Step 3.1: $(\lambda + n) \cdot n$ elements of communication.
- Step 3.2: 1 invocation of $\mathcal{F}_{\mathsf{Coin}}$ to generate λ random values and $4\lambda \cdot n + n^2$ elements of communication.
- Step 3.3: 1 invocation of $\mathcal{F}_{\mathsf{Coin}}$ to generate 2λ random values, 1 invocation of $\mathcal{F}_{\mathsf{Commit}}$ for committing and opening.

Improvements by using PRGs. We note that in our construction, we frequently ask a party to distribute random sharings to other parties. For every pair of parties (P_i, P_j), they may agree on a short PRG seed in advance and whenever P_i (or P_j) needs to send a random value to P_j (or P_i), they can locally expand the seed without any interaction. Applying this improvement, we obtain the following concrete cost.

- Step 1.2: $3 \cdot N \cdot n$ elements of communication.
- Step 1.3: $4 \cdot N \cdot \ell$ invocations of $\mathcal{F}_{\mathsf{Mult}}$.
- Step 1.4: $10 \cdot N \cdot (2\ell - 1) \cdot s$ elements of communication.
- Step 2: 3 invocations of $\mathcal{F}_{\mathsf{Coin}}$ to generate $5\lambda + 1$ random values, 1 invocation of $\mathcal{F}_{\mathsf{Commit}}$ for committing and opening the commitments.
- Step 3.1: No interaction.
- Step 3.2: 1 invocation of $\mathcal{F}_{\mathsf{Coin}}$ to generate λ random values and $4\lambda \cdot n + n^2$ elements of communication.
- Step 3.3: 1 invocation of $\mathcal{F}_{\mathsf{Coin}}$ to generate 2λ random values, 1 invocation of $\mathcal{F}_{\mathsf{Commit}}$ for committing and opening the commitments.

Recall from Section 3 that we ignore the cost of $\mathcal{F}_{\mathsf{Coin}}$. Also, the cost of $\mathcal{F}_{\mathsf{Commit}}$ to compute a commitment is $n^2 \cdot \kappa$ bits and for the opening is n^2 elements in $\mathbb{Z}_p$ plus $n^2\kappa$ bits. This sums up to $3Nn + 10N(2\ell - 1)s + 4\lambda n + 3n^2$ elements in $\mathbb{Z}_p$ plus $2n^2\kappa$ bits, plus $4N\ell$ invocations to $\mathcal{F}_{\mathsf{Mult}}$.

We now count this in terms of λ. Recall that in the context of ECDSA signatures, $\lambda \cdot \ell$ signatures are produced, and our goal is to achieve a communication per party that is in the order of $O(\lambda)$, which is independent of ℓ. Recall we take $\lambda = N/4$ and $N = 32(n + \kappa)$, so $n = (\lambda/8) - \kappa$. Incorporating this, the communication per party (*i.e.* dividing by n) in number of elements from $\mathbb{Z}_p$ without including the calls to $\mathcal{F}_{\mathsf{Mult}}$ nor the bits from opening the commitment is bounded by

$$\begin{aligned}
&3N + \frac{10N(2\ell - 1)s}{n} + 2n + 4\lambda + n \\
&\leq 3(4\lambda) + 10 \cdot (4\lambda) \cdot 2 \cdot \frac{\ell s}{n} + 3\left(\frac{\lambda}{8} - \kappa\right) + 4\lambda \\
&< \lambda\left(16.37 + 80\frac{\ell s}{n}\right) - 3\kappa.
\end{aligned}$$

Recall that $s = \log_{t/n}(1/8) \approx \log_{1-2\ell/n}(1/8)$. With this, we can show that $\frac{\ell}{n} \cdot s < 1.04$, which shows that the communication per party without considering the calls to $\mathcal{F}_{\mathsf{Mult}}$ and the bits from opening the commitment is bounded by $99.57\lambda - 3\kappa$. Regarding $\mathcal{F}_{\mathsf{Mult}}$, let $C(\mathcal{F}_{\mathsf{Mult}}, s)$ be the cost of one call to $\mathcal{F}_{\mathsf{Mult}}$ among s parties. Hence, adding the cost of the calls to $\mathcal{F}_{\mathsf{Mult}}$ and the missing bits from the commitments to the previous computation, we obtain that the cost *per party* in bits of Π_{Tuple} is bounded by

$$
\begin{aligned}
&\log(p)\left[99.57\lambda - 3\kappa + \frac{4N\ell}{n} C(\mathcal{F}_{\mathsf{Mult}}, s)\right] \\
&+ 2\left(\frac{\lambda}{8} - \kappa\right)\kappa \\
&= \log(p)\left[99.57\lambda - 3\kappa + \frac{16\lambda\ell}{\lambda/8 - \kappa} C(\mathcal{F}_{\mathsf{Mult}}, s)\right] \\
&+ 2\left(\frac{\lambda}{8} - \kappa\right)\kappa
\end{aligned}
\tag{3}
$$

In particular, using the PCG-based instantiation from [RS22], discussed in Section 3, this costs $4s$ elements. Hence, the $4N\ell$ invocations to $\mathcal{F}_{\mathsf{Mult}}$ have an overall cost

$$4N\ell \cdot 4s = 16(N\ell s) = 16 \cdot (4\lambda) \cdot \ell s = 64\lambda\ell s.$$

After dividing by n, the cost *per party* becomes $64\lambda\frac{\ell s}{n} < 66.56\lambda$. Hence, the total cost of $\mathcal{F}_{\mathsf{Tuple}}$, accounting for the calls to $\mathcal{F}_{\mathsf{Mult}}$ and using PCGs, is bounded by

$$\log(p) \cdot (166.13\lambda - 3\kappa) + 2\left(\frac{\lambda}{8} - \kappa\right)\kappa \tag{4}$$

bits.

6 Instantiating the Preprocessing

In this section, we use all of the tools that we proposed in previous sections to present a protocol that generates the preprocessing material for the signing process. This generation is formalized in the functionality $\mathcal{F}_{\mathsf{Prep}}$,which is in charge of generating all the shares needed for signing. This preprocessing generates the following elements: $[\boldsymbol{z}]_{n-1}$, where $\boldsymbol{z} = \mathbf{0} \in \mathbb{Z}_p^\ell$, $\langle a \rangle, a \cdot G, [\boldsymbol{\phi}]_{n-\ell}, [a \cdot \boldsymbol{\phi}]_{n-\ell}, [\boldsymbol{\beta} = \boldsymbol{k} * \boldsymbol{\phi}]_{n-1}$, and $[\boldsymbol{r}_{\mathsf{x}}]_{\ell-1}$, where $(\boldsymbol{r}_{\mathsf{x}}[i], \cdot) \stackrel{\mathsf{def}}{=} \boldsymbol{k}[i] \cdot G$, for all $i \in [\ell]$. Here, $a \in \mathbb{Z}_p$, and $\boldsymbol{\phi}, \boldsymbol{k} \in \mathbb{Z}_p^\ell$ are random elements. The instantiation of $\mathcal{F}_{\mathsf{Prep}}$ is described in the protocol Π_{Prep} presented in Section 6.2. Next, we present an overview of this protocol.

Functionality 3: $\mathcal{F}_{\mathsf{Prep}}$

Public Parameters: Let n be the number of parties, t be the corruption threshold, and $(\mathbb{G}, G, p)$ characterize an elliptic curve over $\mathbb{Z}_q$ generated by G, of prime order p. Let $n = t + 2(\ell - 1) + 1$.

- Upon receiving (init) from all parties, $\mathcal{F}_{\mathsf{Prep}}$ samples $\mathsf{a} \leftarrow \mathbb{Z}_p$, $\boldsymbol{\phi} \leftarrow \mathbb{Z}_p^\ell$, and $\boldsymbol{k} \leftarrow \mathbb{Z}_p^\ell$.
- Define $\boldsymbol{z} = \mathbf{0}$, $\boldsymbol{\beta} = \boldsymbol{k} * \boldsymbol{\phi}$, and $(\boldsymbol{r}_{\mathsf{x}}, \boldsymbol{r}_{\mathsf{y}}) = \boldsymbol{k} \cdot G$
- $\mathcal{F}_{\mathsf{Prep}}$ samples secret sharings $[\boldsymbol{z}]_{n-1}$, $\langle a \rangle$, $a \cdot G$, $[\boldsymbol{\phi}]_{n-\ell}$, $[a \cdot \boldsymbol{\phi}]_{n-\ell}$, $[\boldsymbol{r}_{\mathsf{x}}]_{\ell-1}$, and $[\boldsymbol{\beta}]_{n-1}$ and sends the appropriate share to each party as private output.

6.1 Overview of our Instantiation of $\mathcal{F}_{\mathsf{Prep}}$

Consider a group of λ preprocessing tuples to be generated. That is, we will generate enough preprocessing material for signing $\ell \cdot \lambda$ messages in one run. First, we consider a functionality $\mathcal{F}_{\mathsf{Tuple}}$ (see Section 5) that returns $(\langle a_j \rangle, [\boldsymbol{\phi}_j]_{n-\ell}, [a_j \cdot \boldsymbol{\phi}_j]_{n-\ell}, [\boldsymbol{k}_j \star \boldsymbol{\phi}_j]_{n-\ell}, a_j \cdot G, [\boldsymbol{k}_j \cdot G]_{\ell-1})$ for $j \in [\lambda]$, which is a big portion of what $\mathcal{F}_{\mathsf{Prep}}$ produces. What remains is to let the parties obtain sharings of $\boldsymbol{r}_{\mathsf{x}}^{(j)}$ such that $(\boldsymbol{r}_{\mathsf{x}}^{(j)}, \cdot) = \boldsymbol{k}_j \cdot G$ (we also need to show how to obtain degree-$(n-1)$ shares of zero, but this is an easier task that we do not discuss in this overview).

We will let P_1 receive $\boldsymbol{k}_j \cdot G$ in the clear, compute the corresponding coordinate points $(\boldsymbol{r}_{\mathsf{x}}^{(j)}, \boldsymbol{r}_{\mathsf{y}}^{(j)}) \in \mathbb{F}_q^2$, and then distribute shares $[\boldsymbol{r}_{\mathsf{x}}^{(j)}]_{\ell-1}$. The main challenge is to check that the distribution of $\boldsymbol{r}_{\mathsf{x}}^{(j)}$ is consistent with $\boldsymbol{k}_j \cdot G$. To address this, we let the parties sample random coefficients $\gamma_1, \ldots, \gamma_\lambda \in \mathbb{Z}_2$, and compute and open the linear combination $\boldsymbol{k} \cdot G \leftarrow \sum_{i=1}^{\lambda} \gamma_i \cdot [\boldsymbol{k}_i \cdot G]_{\ell-1}$. We can use a low degree to perform this check, since the values being checked do not need to be private in this step. Intuitively, we use packed Shamir shares to "compress" a vector of ℓ values into a single share. Given that the underlying values are fully determined by the honest parties' shares, a corrupt dealer is fully committed to its adversarial errors and has no room to adaptively react to the given $(\gamma_i)_{i=1}^{\lambda}$.

Let us write $\boldsymbol{k} \cdot G = (\boldsymbol{r}_{\mathsf{x}}, \boldsymbol{r}_{\mathsf{y}}) \in \mathbb{F}_q^2$. What the parties need to check now is that, if they were to compute this linear combination using the elliptic curve equations from the plain ECDSA scheme on their shares $[\boldsymbol{r}_{\mathsf{x}}^{(j)}]_{\ell-1}$, then they would arrive at the same pair $(\boldsymbol{r}_{\mathsf{x}}, \boldsymbol{r}_{\mathsf{y}})$. Unfortunately, the parties have shares of the $\boldsymbol{r}_{\mathsf{x}}^{(j)}$ modulo p, and the circuit defining the linear combination is an arithmetic circuit over $\mathbb{Z}_q$. This is where our techniques from Section 4 are relevant: the dealer can secret-share $\boldsymbol{r}_{\mathsf{x}}^{(j)}$ both mod p and mod q, we use $\mathbb{Z}_q$ to check the addition circuit, and we use $\mathbb{Z}_p$ as the output of the preprocessing. In more detail, P_1 distributes have shares $\{[\boldsymbol{r}_{\mathsf{x}}^{(j)} \bmod p]_{\ell-1}^{(p)}\}_{j=1}^{\lambda}$ and $\{[\boldsymbol{r}_{\mathsf{x}}^{(j)}]_{\ell-1}^{(q)}\}_{j=1}^{\lambda}$ using $\mathcal{F}_{\mathsf{DoubleModuli}}$. P_1 also distributes $\{[\boldsymbol{r}_{\mathsf{y}}^{(j)}]_{\ell-1}^{(q)}\}_{j=1}^{\lambda}$ (using $\pi_{\mathsf{DistShares}}$), which will be useful for the check. The "arithmetic circuit" over $\mathbb{Z}_q$ defining the linear combination from above is given by

$$(\boldsymbol{r}_{\mathsf{x}}, \boldsymbol{r}_{\mathsf{y}}) \stackrel{\mathsf{def}}{=} \gamma_1 \cdot (\boldsymbol{r}_{\mathsf{x}}^{(1)}, \boldsymbol{r}_{\mathsf{y}}^{(1)}) +_{\mathsf{ec}} \cdots +_{\mathsf{ec}} \gamma_\lambda \cdot (\boldsymbol{r}_{\mathsf{x}}^{(\lambda)}, \boldsymbol{r}_{\mathsf{y}}^{(\lambda)}), \tag{5}$$

where $+_{\mathsf{EC}}$ denotes the sum of ℓ points in an elliptic curve. This can be described as a circuit $O(\lambda)$ EC addition gates; we refer to Figure 1 for a pictorial representation of this. The goal for the dealer is to prove that this circuit, on inputs $\{([\boldsymbol{r}_{\mathsf{x}}^{(j)}]_{\ell-1}, [\boldsymbol{r}_{\mathsf{y}}^{(j)}]_{\ell-1})\}_{j=1}^{\lambda}$, produces $[\boldsymbol{r}_{\mathsf{x}}]_{\ell-1}, [\boldsymbol{r}_{\mathsf{y}}]_{\ell-1})$ as output. Although we could in principle let the parties run some protocol to compute the circuit themselves, we will exploit the fact that the dealer knows the inputs and hence all *intermediate values* in order to accelerate this check. To illustrate how we do this let us recall how each of these $+_{\mathsf{EC}}$ gates work. Let $(\boldsymbol{x}_i, \boldsymbol{y}_i), (\boldsymbol{u}_i, \boldsymbol{v}_i) \in (\mathbb{Z}_q^\ell)^2$ be the inputs and $(\boldsymbol{z}_i, \boldsymbol{w}_i) \in (\mathbb{Z}_q^\ell)^2$ be the outputs of the i-th addition gate $+_{\mathsf{EC}}$. From the specification of the stand-alone ECDSA, we see that the output is indeed the EC-sum of the two inputs if and only if the following relations hold

over $\mathbb{Z}_q$:[18]

$$\begin{aligned}(\boldsymbol{y}_i - \boldsymbol{v}_i)^2 - (\boldsymbol{x}_i + \boldsymbol{u}_i + \boldsymbol{z}_i) * (\boldsymbol{x}_i - \boldsymbol{u}_i)^2 = 0,\\ (\boldsymbol{y}_i - \boldsymbol{v}_i) * (\boldsymbol{x}_i - \boldsymbol{z}_i) + (\boldsymbol{w}_i - \boldsymbol{y}_i) * (\boldsymbol{x}_i - \boldsymbol{u}_i) = 0.\end{aligned} \tag{6}$$

Notice that the first equation is a degree-3 equation. We add an extra constraint in order to lower this to degree-2. Let $\boldsymbol{a}_i \stackrel{\text{def}}{=} (\boldsymbol{x}_i - \boldsymbol{u}_i)^2$. We can rewrite the above set of equations to include the new constraint as

$$\begin{bmatrix}\boldsymbol{A}_i\\ \boldsymbol{B}_i\\ \boldsymbol{C}_i\end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix}(\boldsymbol{y}_i - \boldsymbol{v}_i)^2 - (\boldsymbol{x}_i + \boldsymbol{u}_i + \boldsymbol{z}_i) * \boldsymbol{a}_i\\ (\boldsymbol{y}_i - \boldsymbol{v}_i) * (\boldsymbol{x}_i - \boldsymbol{z}_i) + (\boldsymbol{w}_i - \boldsymbol{y}_i) * (\boldsymbol{x}_i - \boldsymbol{u}_i)\\ \boldsymbol{a}_i - (\boldsymbol{x}_i - \boldsymbol{u}_i)^2\end{bmatrix} = \begin{bmatrix}\mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}. \tag{7}$$

The dealer can use $\pi_{\mathsf{DistShares}}$ to secret share all the new intermediate sharings $[\boldsymbol{a}_j]^{(q)}_{\ell-1}$, where j ranges over the number of gates, so at this point the parties have degree-$(\ell-1)$ shares of all the terms appearing in Eq. (7). Since the equations have all degree-2, the parties can *locally* obtain degree-$2(\ell-1)$ sharings of each $\boldsymbol{A}_i, \boldsymbol{B}_i, \boldsymbol{C}_i$. Note that this degree is still low enough so that the underlying "secrets" are fully determined by the honest shares. Here, the parties could simply reconstruct all these sharings, and check that indeed they all equal $\mathbf{0}$, but this would be too expensive. Instead, we sample random coefficients and only open a random linear combination.

A detailed specification of the instantiation of $\mathcal{F}_{\mathsf{Prep}}$ is presented in Π_{Prep} in Section 6.2 with its security proof and communication complexity.

6.2 Instantiation of $\mathcal{F}_{\mathsf{Prep}}$

We now provide the deails of the intuition given above. This is given in Protocol Π_{Prep} below.

Protocol 6: Π_{Prep}

Public parameters: Let n be the number of parties, t be the corruption threshold, and $(\mathbb{G}, G, p)$ characterize an elliptic curve generated by G, of prime order p. Let $n = t + 2(\ell-1) + 1$. Let $\lambda \in \mathbb{N}$ be a public parameter.

Output: $\langle a_j \rangle$, $a_j \cdot G$, $[\boldsymbol{\phi}_j]_{n-\ell}$, $[a_j \cdot \boldsymbol{\phi}_j]_{n-\ell}$, $[\boldsymbol{r}_{\mathsf{x}}^{(j)} \bmod p]_{\ell-1}$, and $[\boldsymbol{\beta}_j] = [\boldsymbol{k}_j \star \boldsymbol{\phi}_j]_{n-1}$, for $j \in \lambda$, where $\boldsymbol{k}_j \cdot G = \left(\boldsymbol{r}_{\mathsf{x}}^{(j)}, \cdot\right)$.

1. All parties invoke $\mathcal{F}_{\mathsf{Tuple}}$ with the public parameter λ to obtain $\langle a_j \rangle$, $[\boldsymbol{\phi}_j]_{n-\ell}$, $[a_j \cdot \boldsymbol{\phi}_j]_{n-\ell}$, $[\boldsymbol{\beta}_j] = [\boldsymbol{k}_j \star \boldsymbol{\phi}_j]_{n-1}$, $a_j \cdot G$, $[\boldsymbol{k}_j \cdot G]_{\ell-1}$, for $j \in [\lambda]$.

[18] Note that these equations correspond to the case of when we are adding two EC points with *different* x-coordinates. In our case this is guaranteed to be the case with overwhelming probability. Even if this was not the case, this would not be an issue as P_1 simply uses a slightly different equation in this case (and informs the parties of this change).

2. For $j \in [\lambda]$, all parties reconstruct $[\boldsymbol{k}_j \cdot G]_{\ell-1}$ towards P_1 so that P_1 obtains $\boldsymbol{k}_j \cdot G$ in the clear.
3. Steps 3-8 are repeated κ times. All parties invoke $\mathcal{F}_{\mathsf{Coin}}$ to obtain public random binary coefficients $\{\gamma_j\}_{j=1}^{\lambda}$ from $\mathbb{Z}_2$. Then, party P_1 computes $\boldsymbol{k} \cdot G \stackrel{\mathsf{def}}{=} \sum_{j=1}^{\lambda} \gamma_j (\boldsymbol{k}_j \cdot G)$. Let $(\boldsymbol{r}_{\mathsf{x}}, \boldsymbol{r}_{\mathsf{y}}) \stackrel{\mathsf{def}}{=} \boldsymbol{k} \cdot G \in \mathbb{Z}_q^2$ and $\left(\boldsymbol{r}_{\mathsf{x}}^{(j)}, \boldsymbol{r}_{\mathsf{y}}^{(j)}\right) \stackrel{\mathsf{def}}{=} \boldsymbol{k}_j \cdot G \in \mathbb{Z}_q^2$, for $j \in [\lambda]$.[a]
4. Party P_1 distributes the following shares to the rest of the parties:
   - $\left\{\left([\boldsymbol{r}_{\mathsf{x}}^{(j)} \bmod p]_{\ell-1}^{(p)}, [\boldsymbol{r}_{\mathsf{x}}^{(j)}]_{\ell-1}^{(q)}\right)\right\}_{j=1}^{\lambda}$ using $\mathcal{F}_{\mathsf{DoubleModuli}}$.
   - $\left\{[\boldsymbol{r}_{\mathsf{y}}^{(j)}]_{\ell-1}^{(q)}\right\}_{j=1}^{\lambda}$ using $\pi_{\mathsf{DistShares}}$.
   - Shares of the intermediate values modulo q that appear in Eq. (7). This is done using $\pi_{\mathsf{DistShares}}$.
5. The parties reconstruct $\boldsymbol{k} \cdot G$ from $[\boldsymbol{k} \cdot G]_{\ell-1}^{(p)} = \sum_{j=1}^{\lambda} \gamma_j [\boldsymbol{k}_j \cdot G]_{\ell-1}^{(p)}$. Then the parties let $(\boldsymbol{r}_{\mathsf{x}}, \boldsymbol{r}_{\mathsf{y}}) = \boldsymbol{k} \cdot G$.
6. Once all parties receive the inputs and outputs of each gate in the circuit depicted in Fig. 1, they represent the correctness of the computation of the circuit as m sets of quadratic constraints of the form $\boldsymbol{A}_i = \boldsymbol{0}$, $\boldsymbol{B}_i = \boldsymbol{0}$ and $\boldsymbol{C}_i = \boldsymbol{0}$, for $i \in [m]$, where $\boldsymbol{A}_i$, $\boldsymbol{B}_i$ and $\boldsymbol{C}_i$ are quadratic expressions on the input and output wires of each gate in the circuit representation depicted in Fig. 1, as explained in Equation (7). Hence, all parties can locally compute $[\boldsymbol{A}_i]_{2\ell-2}$, $[\boldsymbol{B}_i]_{2\ell-2}$, and $[\boldsymbol{C}_i]_{2\ell-2}$.
7. The parties call $\mathcal{F}_{\mathsf{Coin}}$ to sample $\alpha_i, \beta_i, \gamma_i \in \mathbb{Z}_q$ and compute locally

$$[\boldsymbol{S}]_{2\ell-2} \stackrel{\mathsf{def}}{=} \sum_{i=1}^{m} (\alpha_i [\boldsymbol{A}_i]_{2\ell-2} + \beta_i [\boldsymbol{B}_i]_{2\ell-2} + \gamma_i [\boldsymbol{C}_i]_{2\ell-2}). \tag{8}$$

   Here, m is the number of $+_{\mathsf{EC}}$-gates in the circuit from Eq.(5), as explained in the prior text.
8. The parties reconstruct $\boldsymbol{S} \leftarrow [\boldsymbol{S}]_{2\ell-2}$ and check that $\boldsymbol{S} = \boldsymbol{0}$. The parties abort if this is not the case.
9. **Preparing Random Packed Sharings of 0**: All parties repeat the following steps $\lceil \lambda/(2\ell-1) \rceil$ times to prepare λ random packed sharings of $\boldsymbol{0}$ among the parties.
   (a) Each party P_i shares a random packed sharing of $\boldsymbol{0}$, denoted by $[\boldsymbol{o}^{(i)}]_{n-1}$.
   (b) All parties agree on a Vandermonde matrix $\boldsymbol{M}$ of dimension $(2\ell-1) \times n$.
   (c) All parties compute $([\boldsymbol{o}_1]_{n-1}, \ldots, [\boldsymbol{o}_{2\ell-1}]_{n-1}) = \boldsymbol{M} \cdot \left([\boldsymbol{o}^{(1)}]_{n-1}, \ldots, [\boldsymbol{o}^{(n)}]_{n-1}\right)$.
10. Parties output $\langle a_j \rangle$, $a_j \cdot G$, $[\boldsymbol{\phi}_j]_{n-\ell}$, $[a_j \cdot \boldsymbol{\phi}_j]_{n-\ell}$, $[\boldsymbol{r}_{\mathsf{x}}^{(j)} \bmod p]_{\ell-1}$, $[\boldsymbol{\beta}_j] = [\boldsymbol{k}_j \star \boldsymbol{\phi}_j]_{n-1}$, and $[\boldsymbol{o}_j]_{n-1}$ for $j \in \lambda$.

[a] This only needs to be done once for the whole κ iterations.

Theorem 4. *The protocol Π_{Prep} securely realizes $\mathcal{F}_{\mathsf{Prep}}$ in the $(\mathcal{F}_{\mathsf{Tuple}}, \mathcal{F}_{\mathsf{DoubleModuli}}, \mathcal{F}_{\mathsf{Coin}})$-hybrid.*

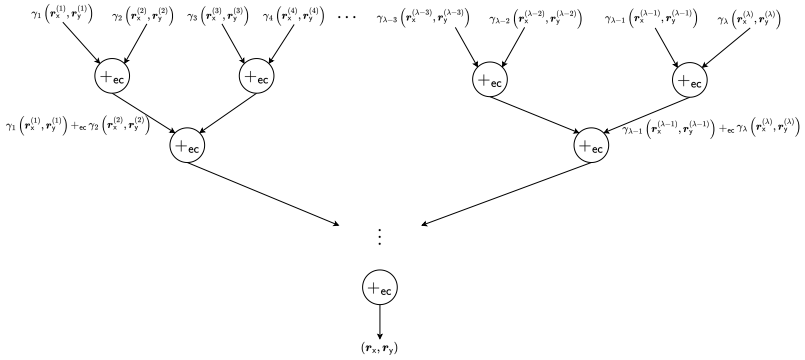


Fig. 1: Circuit to check the consistency in Π_{Prep}.

Proof. We define a simulator $\mathcal{S}$ that interacts with $\mathcal{F}_{\mathsf{Prep}}$ and with the corrupt parties. The simulator emulates internally a set of honest parties, and also emulates the functionalities $\mathcal{F}_{\mathsf{Tuple}}$, $\mathcal{F}_{\mathsf{DoubleModuli}}$ and $\mathcal{F}_{\mathsf{Coin}}$. There are two main cases to distinguish. If P_1 is honest then most of the proof is straightforward, and the challenging aspect is when P_1 is corrupt. Hence, we focus on this case from now on.

For step 1 in Π_{Prep}, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Tuple}}$ by producing the tuple $\langle a_j \rangle$, $[\boldsymbol{\phi}_j]_{n-\ell}$, $[a_j \cdot \boldsymbol{\phi}_j]_{n-\ell}$, $[\boldsymbol{\beta}_j] = [\boldsymbol{k}_j \star \boldsymbol{\phi}_j]_{n-1}$, $a_j \cdot G$, $[\boldsymbol{k}_j \cdot G]_{\ell-1}$, for $j \in [\lambda]$. Then $\mathcal{S}$ emulates the reconstruction of $[\boldsymbol{k}_j \cdot G]_{\ell-1}$ towards P_1, as in step 2. For step 3, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Coin}}$ as specified, which defines the circuit from Eq. (5).

In step 4, the simulator will receive the secrets $\left\{\boldsymbol{r}_{\mathsf{x}}^{(j)}\right\}_{j=1}^{\lambda}$ from P_1 as part of the emulation of $\mathcal{F}_{\mathsf{DoubleModuli}}$, and $\mathcal{S}$ distributes the required sharings

$$\left\{\left([\boldsymbol{r}_{\mathsf{x}}^{(j)} \bmod p]_{\ell-1}^{(p)}, [\boldsymbol{r}_{\mathsf{x}}^{(j)}]_{\ell-1}^{(q)}\right)\right\}_{j=1}^{\lambda}.$$

Note that at this point $\mathcal{S}$ knows if P_1 is attempting to cheat, as $\mathcal{S}$ knows the $\boldsymbol{k}_i \cdot G$. In this step $\mathcal{S}$ also emulates the procedure $\pi_{\mathsf{DistShares}}$, and in the process $\mathcal{S}$ learns $\left\{[\boldsymbol{r}_{\mathsf{y}}^{(j)}]_{\ell-1}^{(q)}\right\}_{j=1}^{\lambda}$, along with degree-$(\ell-1)$ shares of all the intermediate values appearing in Eq.(7). Note that, once again, $\mathcal{S}$ can easily detect at this point if the circuit from Eq.(5) fails.

In step 5, $\mathcal{S}$ emulates the reconstruction of $\boldsymbol{k} \cdot G$ towards the corrupt parties from $[\boldsymbol{k} \cdot G]_{\ell-1}^{(p)} = \sum_{j=1}^{\lambda} \gamma_j [\boldsymbol{k}_j \cdot G]_{\ell-1}^{(p)}$. Step 6 is non-interactive. For steps 7-8, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Coin}}$ and computes $[\boldsymbol{S}]_{2\ell-2}$ as specified. We claim that the probability that there exists one i_0 for which either $\boldsymbol{A}_{i_0}$, $\boldsymbol{B}_{i_0}$ or $\boldsymbol{C}_{i_0}$ is not zero, and simultaneously $\boldsymbol{S} = \boldsymbol{0}$, is negligible. To see this suppose w.l.o.g. that $\boldsymbol{A}_{i_0} \neq \boldsymbol{0}$, say $\boldsymbol{A}_{i_0}[j_0] \neq 0$, and yet $\boldsymbol{S} = \boldsymbol{0}$. From Eq.(8), we can write $\alpha_{i_0} = (\boldsymbol{A}_{i_0}[j_0])^{-1} \cdot \left(-\sum_{i \neq i_0} \alpha_i \boldsymbol{A}_i[j_0] + \sum_i \beta_i \boldsymbol{B}_i[j_0] + \sum_i \gamma_i \boldsymbol{C}_i[j_0]\right)$. Since the left-hand

side is uniform in $\mathbb{Z}_q$, and the right-hand side follows an independent distribution from α_{i_0}, we conclude this can only be satisfied with negligible probability $1/q$.

Note that this holds both in the real and ideal worlds, and hence the two worlds are indistinguishable to the environment thus far. Furthermore, since (in case of no abort) $\boldsymbol{A}_i = \mathbf{0}$, $\boldsymbol{B}_i = \mathbf{0}$ and $\boldsymbol{C}_i = \mathbf{0}$ for all $i \in [m]$, as explained in Section 6.1 this implies that Eq. (6) holds and therefore the $+_{\mathsf{EC}}$ gates are computed correctly. In other words, the circuit from Eq. (5) holds. Let us write the EC point given by the coordinates $(\boldsymbol{r}_{\mathsf{x}}^{(j)}, \boldsymbol{r}_{\mathsf{y}}^{(j)})$ as $\boldsymbol{k}'_j \cdot G$. The fact that Eq. (5) holds means can be translated into $\sum_{j=1}^{\lambda} \gamma_j \cdot (\boldsymbol{k}_j \cdot G) = \sum_{j=1}^{\lambda} \gamma_j \cdot (\boldsymbol{k}'_j \cdot G)$, which implies

$$\sum_{j=1}^{\lambda} \gamma_j \cdot \boldsymbol{k}_j = \sum_{j=1}^{\lambda} \gamma_j \cdot \boldsymbol{k}'_j. \tag{9}$$

Now, we claim this implies that for all $j \in [\lambda]$, the $[\boldsymbol{r}_{\mathsf{x}}^{(j)}]_{\ell-1}$ distributed by P_1 corresponds to the x-coordinate of $[\boldsymbol{k}_j \cdot G]_{\ell-1}$. For the sake of contradiction suppose that there exists $j_0 \in [\lambda]$ such that this is not true. In particular, $\boldsymbol{k}_{j_0} \cdot G \neq \boldsymbol{k}'_{j_0} \cdot G$, and hence $\boldsymbol{k}_{j_0} \neq \boldsymbol{k}'_{j_0}$. Let $i_0 \in [\ell]$ be such that $\boldsymbol{k}_{j_0}[i_0] \neq \boldsymbol{k}'_{j_0}[i_0]$. From Eq. (9) we see that

$$\gamma_{j_0} = (\boldsymbol{k}'_{j_0}[i_0] - \boldsymbol{k}_{j_0}[i_0])^{-1} \cdot \left(\sum_{j \neq j_0} \gamma_j \cdot (\boldsymbol{k}_j[i_0] - \boldsymbol{k}'_j[i_0]) \right).$$

As before, the LHS is uniformly random over $\mathbb{Z}_p$ and independent of the RHS, and hence this can only happen with negligible probability $1/p$. We see then that, indeed, the $[\boldsymbol{r}_{\mathsf{x}}^{(j)}]_{\ell-1}$ distributed by P_1 corresponds to the x-coordinate of $[\boldsymbol{k}_j \cdot G]_{\ell-1}$. Therefore, the shares that the parties output in the ideal world follow the same distribution as in the real one. For the case of the shares of $\mathbf{0}$, the proof is similar to that of Theorem 3, which exploits the fact that Vandermonde matrices are full rank. We omit this portion of the proof. □

6.3 Communication Complexity of Π_{Prep}

The concrete cost of each step of the protocol is:

- Step 1: 1 invocation of $\mathcal{F}_{\mathsf{Tuple}}$.
- Step 2: $\lambda \cdot n$ elements in $\mathbb{Z}_p$
- Step 3: κ invocations of $\mathcal{F}_{\mathsf{Coin}}$ to obtain λ random elements in $\mathbb{Z}_2$.
- Step 4.1: κ invocations of $\mathcal{F}_{\mathsf{DoubleModuli}}$ to share λ elements in $\mathbb{Z}_q^{\ell}$.
- Step 4.2: 1 invocation of $\pi_{\mathsf{DistShares}}$ to share λ elements in $\mathbb{Z}_q$.
- Step 4.3: κ invocation to $\pi_{\mathsf{DistShares}}$ to share $2\lambda - 2$ in $\mathbb{Z}_q$.
- Step 5: κn^2 elements in $\mathbb{Z}_p$.
- Step 7: κ invocation of $\mathcal{F}_{\mathsf{Coin}}$ to sample $3\lambda - 1$ random elements in $\mathbb{Z}_q$
- Step 8: κn^2 elements in $\mathbb{Z}_p$
- Step 9: $\lceil \frac{\lambda}{2\ell-1} \rceil \cdot n^2$ elements in $\mathbb{Z}_p$.

Now let us compute the concrete communication complexity for Π_{Prep}. We assume that $\log p \approx \log q$ which is standard in practice for ECDSA. As we saw in Section 5.4, the communication complexity per party of calling $\mathcal{F}_{\mathsf{Tuple}}$ is bounded by $\log(p) \cdot \left(97.45\lambda - 2\kappa + \frac{16\lambda\ell}{\lambda/8-\kappa} C(\mathcal{F}_{\mathsf{Mult}}, s)\right) + 2\left(\frac{\lambda}{8} - \kappa\right)\kappa$ bits, where $C(\mathcal{F}_{\mathsf{Mult}}, s)$ is the cost of one call to $\mathcal{F}_{\mathsf{Mult}}$ among s parties. Note this assumes that $\lambda = 8(n+\kappa)$. Also, $\mathcal{F}_{\mathsf{DoubleModuli}}$ has a communication complexity of $\approx \log p \cdot \left(10\lambda n + 1024n^2\right)$ bits, and distributing d elements using $\pi_{\mathsf{DistShares}}$ has a complexity of $\log p \cdot (dn + n^2)$ bits.

Organizing this all together and considering that $n = (\lambda/8) - \kappa$, the communication complexity *per party* for Π_{Prep} is bounded by:

$$
\begin{aligned}
&\log p \cdot \Bigg[99.57\lambda - 3\kappa + \frac{16\lambda\ell}{\lambda/8-\kappa} C(\mathcal{F}_{\mathsf{Mult}}, s) + \lambda \\
&+ \kappa \cdot (10\lambda + 1024n + (2\lambda - 2) + 3n) \\
&+ \lambda + n + \left(\frac{\lambda}{2\ell - 1}\right) \cdot n\Bigg] + 2\left(\frac{\lambda}{8} - \kappa\right)\kappa \\
&= \log p \cdot \Bigg[101.57\lambda - 3\kappa + \frac{16\lambda\ell}{\lambda/8-\kappa} C(\mathcal{F}_{\mathsf{Mult}}, s) \\
&+ \kappa \cdot (12\lambda + 1027n - 2) \\
&+ n + \left(\frac{\lambda}{2\ell - 1}\right) \cdot n\Bigg] + 2\left(\frac{\lambda}{8} - \kappa\right)\kappa
\end{aligned}
$$

bits.

Optimization. In the current specification of the protocol, we have that the preparation of shares of $\mathbf{0}$ costs $\lceil \lambda/(2\ell - 1) \rceil \cdot n^2$. However we can optimize the communication by using PRGs. Specifically, each parties agrees on seeds with the first $n - \ell$ parties and then send shares to the rest of the ℓ parties. At the moment of generating the shares of zero, the parties expand the seeds to generate $n - \ell$ shares, which in conjunction with the ℓ zero constraints determine the remaining ℓ shares, which is the only thing the dealer must send. Therefore, the communication complexity of generating shares of zero will be $\lceil \lambda/(2\ell - 1) \rceil \cdot n \cdot \ell$. This gets rid of the extra factor of n in the last summand.

Including the above optimization, we have that the communication complexity per party of Π_{Prep} is bounded by the following number of bits:

$$
\begin{aligned}
&\log p \cdot \Bigg(102.69\lambda - 4\kappa + \frac{16\lambda\ell}{\lambda/8-\kappa} C(\mathcal{F}_{\mathsf{Mult}}, s) \\
&+ \kappa \cdot (140.37\lambda - 1027\kappa - 2)\Bigg) \\
&+ 2\left(\frac{\lambda}{8} - \kappa\right)\kappa
\end{aligned}
$$

Using PCGs for the multiplications, we have a complexity bounded by

$$\begin{aligned}&\log p \cdot (169.25\lambda + \kappa \cdot (140.37\lambda - 1027\kappa - 6)) \\ &\quad + 2\left(\frac{\lambda}{8} - \kappa\right)\kappa\end{aligned} \tag{10}$$

which is in particular independent of ℓ, as desired. We point out that Π_{Prep} actually needs a batch of $\lambda \cdot \ell$ signatures, and we obtain the above communication by dividing the total costs of Π_{Prep}—which are for $\lambda \cdot \ell$ signatures—by λ. Note that the batch size $\lambda \cdot \ell$ is not an issue for Π_{Prep} since this protocol is executed in the offline phase, which is independent of both the key and the message. Preprocessing for many signatures can be generated, and in the online phase only ℓ simultaneous signatures are needed.

In Section 6.4 we discuss a more communication-efficient instantiation of Π_{Prep} that avoids the $\kappa\times$ overhead, although it uses more heavy cryptographic machinery (namely, SNARKs).

6.4 A More Efficient Check for Π_{Prep} using SNARKs

The techniques presented in Section 6, although concretely efficient, suffer from a multiplicative overhead of $\times\kappa$, which may be too high in cases where $n \leq \kappa$. We remark that this is the *only* component of our entire threshold ECDSA construction that suffers from such overhead, and we leave it as relevant future work to remove this overhead while making use of simple information-theoretic techniques as in our current construction. However, to illustrate that this overhead is not precisely inherent, we do sketch in this section a solution that avoids this extra factor, albeit it requires the use of SNARKs. These are inherently heavier than the IT techniques as above, but due to the rapid progress in SNARK implementations and optimizations, we suspect this approach can actually be highly efficient in practice.

Let us begin by isolating the main limiting factor that leads to our $\kappa\times$ overhead. The "circuit" that must be proven is defined by Eq. (5). The right-hand side has inputs $(\boldsymbol{r}_{\mathsf{x}}^{(i)}, \boldsymbol{r}_{\mathsf{y}}^{(i)})$ for $i \in [\lambda]$, and the goal is to prove that the output equals the left-hand side $(\boldsymbol{r}_{\mathsf{x}}, \boldsymbol{r}_{\mathsf{y}})$. Our approach to checking this is to let the dealer distribute sharings of all "intermediate values" in the circuit, and then check that this is done correctly, which ultimately translates to the quadratic constraints in Eq. (7). We use bits $\gamma_1, \ldots, \gamma_\lambda \in \mathbb{Z}_2$ as the challenges as this way the associated circuit has size proportional to λ, but we only get $1/2$ soundness in exchange; this requires us to repeat κ times, which is the root of the overhead.

Alternatively, one can choose larger coefficients $\gamma_1, \ldots, \gamma_\lambda \in \mathbb{Z}_q$ in order to obtain soundness $1/q$ in one go, but this increases the circuit size by $\log q\times$ since each scalar multiplication must be done via the double-and-add method, which requires $\log q$ additions ultimately. The main overhead here comes from requiring the dealer to distribute shares of all intermediate values. Our approach consists of using SNARK-like techniques in order to.

Using a Generic SNARK. Consider a uniform random string (expanded from a small seed) parsed as $\beta_1 \cdot G, \ldots, \beta_\lambda \cdot G$ which is given as setup. The parties locally compute $[\boldsymbol{c}]_{\ell-1} = \sum_{i=1}^{\lambda} [\boldsymbol{r}_{\mathsf{x}}^{(i)}]_{\ell-1} \cdot (\beta_i \cdot G)$ and reconstruct this vector. Note that $\boldsymbol{c}$ is a non-hiding version of Pedersen commitments and hence acts as a way of binding the dealer to $(\boldsymbol{r}_{\mathsf{x}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{x}}^{(\lambda)})$. Now, the dealer uses a SNARK to prove the NP-relation consisting of all tuples

$$(\underbrace{(\{\boldsymbol{r}_{\mathsf{x}}^{(i)}\}_{i=1}^{\lambda}, \{\boldsymbol{r}_{\mathsf{y}}^{(i)}\}_{i=1}^{\lambda})}_{\text{witness}}, \underbrace{(\boldsymbol{c}, (\boldsymbol{r}_{\mathsf{x}}, \boldsymbol{r}_{\mathsf{y}}))}_{\text{statement}})$$

such that $\boldsymbol{c} = \sum_{i=1}^{\lambda} \boldsymbol{r}_{\mathsf{x}}^{(i)} \cdot (\beta_i \cdot G)$ and and Eq. (5) holds.

Succinctness implies that the size of the proof π is logarithmic in the size of the circuit, which in our case is $\log q \cdot \lambda$. The dealer can send this proof to all parties directly, each of which verifies locally. If the proof passes, soundness implies that indeed there exists $(\{\boldsymbol{s}_{\mathsf{x}}^{(i)}\}_{i=1}^{\lambda}, \{\boldsymbol{s}_{\mathsf{y}}^{(i)}\}_{i=1}^{\lambda})$ such that $\boldsymbol{c} = \sum_{i=1}^{\lambda} \boldsymbol{s}_{\mathsf{x}}^{(i)} \cdot (\beta_i \cdot G)$ and Eq. (5) holds. It is easy to prove that if there is at least one of the $\boldsymbol{s}_{\mathsf{x}}^{(i_0)}$ that differs from the shared value $[\boldsymbol{r}_{\mathsf{x}}^{(i_0)}]_{\ell-1}$, then the binding of the commitment would be broken.

Note that, if we know what type of SNARK will be used, we may be able to optimize this approach even further. For example, suppose that a SNARK constructed by compiling the Poly-IOP model with a polynomial commitment scheme is used. Depending on the polynomial commitment, we may be able to perform some optimizations. For instance, if the scheme is KZG [KZG10], then we can let the parties compute a KZG-commitment to $\{\boldsymbol{r}_{\mathsf{x}}^{(i)}\}_{i=1}^{\lambda}$ homomorphically, instead of the Pedersen-like presented above. This way, the dealer has the starting point for the associated Poly-IOP, and we do not involve any other extra commitment schemes.

Another example where optimizations are possible is if we use an IOP-based SNARK such as FRI [Ben+20]. We can let the dealer commit to the intermediate circuit values using a Merkle hash as usual, but for the input wires we can let the IOP be instantiated by the parties opening the queried items from the shares $(\{[\boldsymbol{r}_{\mathsf{x}}^{(i)}]_{\ell-1}\}_{i=1}^{\lambda}, \{[\boldsymbol{r}_{\mathsf{y}}^{(i)}]_{\ell-1}\}_{i=1}^{\lambda})$. Since there will be a sublinear amount of queries, the communication is not negatively impacted.

7 Our Final Threshold ECDSA Protocol

We use the tools developed in the previous sections to construct our threshold ECDSA signing protocol. For simplicity we assume that the number of signing parties n is equal to the total number of parties in the signing pool N, but it is trivial to accommodate $n < N$ by using Shamir shares for the secret key instead of additive shares. We begin by specifying the ECDSA signing functionality.

Functionality 4: $\mathcal{F}_{\mathsf{ECDSA}}$

Public Parameters: Let n be the number of parties, t be the corruption threshold, and $(\mathbb{G}, G, p)$ characterize an elliptic curve generated by G, of prime order p. Let $n = t + 2(\ell - 1) + 1$. All outputs are adversarially delayed.

- **Key Verification**. Upon receiving $(\mathsf{init}, \mathsf{pk})$ from each party P_i, $\mathcal{F}_{\mathsf{ECDSA}}$ returns $(\mathsf{public\text{-}key\text{-}verified})$ to all parties. The adversary is notified of each $(\mathsf{init}, \mathsf{pk})$ sent by an honest party.
- **Generate Nonces**. Upon receiving $(\mathsf{gen\text{-}nonces})$ from all parties, if $(\mathsf{priv\text{-}nonces}, \boldsymbol{k})$ does not already exist in memory, $\mathcal{F}_{\mathsf{ECDSA}}$ samples $\boldsymbol{k} \leftarrow \mathbb{Z}_p^l$, sets $\boldsymbol{r}_{\mathsf{x}} = (\boldsymbol{k}[i] \cdot G)_{i \in [l]}$, and returns $(\mathsf{nonces}, \boldsymbol{r}_{\mathsf{x}})$ to all parties. $\mathcal{F}_{\mathsf{ECDSA}}$ additionally stores $(\mathsf{priv\text{-}nonces}, \boldsymbol{k})$ in memory.
- **Sign**. Upon receiving $(\mathsf{sign}, \boldsymbol{m}, \mathsf{sk}_i)$ from each Party P_i, define $\mathsf{sk} = \sum_{i \in [n]} \mathsf{sk}_i$. If $(\mathsf{priv\text{-}nonces}, \boldsymbol{k})$ exists in memory and $\mathsf{pk} = \mathsf{sk} \cdot G$, then $\mathcal{F}_{\mathsf{ECDSA}}$ computes $\boldsymbol{s} = ((\mathsf{SHA}(\boldsymbol{m}_i) + \boldsymbol{r}_{\mathsf{x}} \cdot \mathsf{sk})/\boldsymbol{k}_i)_{[i \in [l]]}$ and returns $(\mathsf{signatures}, \boldsymbol{s})$ to P_1. Finally, $\mathcal{F}_{\mathsf{ECDSA}}$ removes $(\mathsf{priv\text{-}nonces}, \boldsymbol{k})$ from memory.

We now give our protocol to realize the above functionality. Our protocol has three stages: a preprocessing phase, a message-independent phase, and an online phase in which the message to be signed is required.

Protocol 7: Π_{Sign}

- **Key Verification.** Parties jointly hold additive shares $\langle \mathsf{sk} \rangle$, and wish to confirm that $\mathsf{pk} = \mathsf{sk} \cdot G$ is agreed upon. They exchange pk with each other and verify that they match.
- **Generate Nonces.** Parties invoke $\mathcal{F}_{\mathsf{Prep}}$ to obtain $[\boldsymbol{z}]_{n-1}$, $\langle a \rangle, a \cdot G, [\boldsymbol{\phi}]_{n-\ell}, [a \cdot \boldsymbol{\phi}]_{n-\ell}, [\boldsymbol{\beta} = \boldsymbol{k} * \boldsymbol{\phi}]_{n-1}$, and $[\boldsymbol{r}_{\mathsf{x}}]_{\ell-1}$, where $(\boldsymbol{r}_{\mathsf{x}}[i], \cdot) \stackrel{\mathsf{def}}{=} \boldsymbol{k}[i] \cdot G$, for all $i \in [\ell]$.
- **Message-independent phase.** Computation of $[\mathsf{sk} \cdot \boldsymbol{\phi}]_{n-\ell}$:
  1. The parties locally compute $\langle d \rangle \stackrel{\mathsf{def}}{=} \langle \mathsf{sk} \rangle - \langle a \rangle$.
  2. Parties reveal d by sending their shares to P_1. Then, P_1 reconstructs d and sends the revealed value to all the parties.
  3. Parties check consistency of d by verifying $\mathsf{pk} - a \cdot G = d \cdot G$
  4. The parties compute the following linear combination: $[\mathsf{sk} \cdot \boldsymbol{\phi}]_{n-\ell} = d \cdot [\boldsymbol{\phi}]_{n-\ell} + [a \cdot \boldsymbol{\phi}]_{n-\ell}$.
- **Online phase.** To sign a vector of messages $\boldsymbol{m}$, the parties proceed as follows.
  1. Each party derives its share of a canonical $[\mathsf{SHA}(\boldsymbol{m})]_{\ell-1}$.
  2. The parties locally compute $[\boldsymbol{\alpha}]_{n-1} \stackrel{\mathsf{def}}{=} [\boldsymbol{z}]_{n-1} + [\mathsf{SHA}(\boldsymbol{m})]_{\ell-1} * [\boldsymbol{\phi}]_{n-\ell} + [\boldsymbol{r}_{\mathsf{x}}]_{\ell-1} * [\mathsf{sk} \cdot \boldsymbol{\phi}]_{n-\ell}$.
  3. Let R be the intended receiver of the signatures. Each party sends its shares of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ to R, who then interpolates them to obtain $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ respectively, and computes the vector of putative signatures $\boldsymbol{s} = \boldsymbol{\alpha}/\boldsymbol{\beta}$. R outputs $(\boldsymbol{r}_{\mathsf{x}}, \boldsymbol{s})$ if it is a vector of valid ECDSA signatures, and aborts if not.

Theorem 5. *Protocol Π_{Sign} securely instantiates the functionality $\mathcal{F}_{\mathsf{ECDSA}}$ in the ($\mathcal{F}_{\mathsf{Prep}}$)-hybrid model.*

Proof. We sketch how the above protocol can be simulated by making a series of changes through hybrid experiments.

- In the first hybrid experiment, $d \leftarrow \mathbb{Z}_p$ is sampled first, following which $a \cdot G = \mathsf{pk} - d \cdot G$ and $a = \mathsf{sk} - d$ are set on behalf of $\mathcal{F}_{\mathsf{Prep}}$. This hybrid experiment is distributed identically to the real protocol, as the order in which these values are defined is not visible to the distinguisher, and the distribution of a is unchanged.
- In the second hybrid experiment, $\boldsymbol{\beta} \leftarrow \mathbb{Z}_p$ is sampled first—implicitly by first sampling each honest party's share of $\boldsymbol{\beta}$ uniformly—and $\boldsymbol{\phi}$ is defined (component-wise) as $\boldsymbol{\beta}/\boldsymbol{k}$. This change does not induce a change in the distribution of any values either, and so this hybrid experiment is identical to the last.
- In the third hybrid experiment, $\boldsymbol{k}$ and $\boldsymbol{r}_{\mathsf{x}}$ (as relevant in $\mathcal{F}_{\mathsf{Prep}}$) and $\boldsymbol{s}$ (the vector of signatures on messages $\boldsymbol{m}$ with nonces $\boldsymbol{r}_{\mathsf{x}}$) are computed within the scope of a functionality $\mathcal{F}_{\mathsf{ECDSA}}$. This is merely a syntactic change. Note that $\boldsymbol{s}$ matches $\boldsymbol{\alpha}/\boldsymbol{\beta}$ as computed by P_1 in the final step in the event that signing is successful, as there is a unique $\boldsymbol{s}$ that comprises verifying signatures on $\boldsymbol{m}$ with nonces $\boldsymbol{r}_{\mathsf{x}}$.
- In the third and final hybrid experiment, $\boldsymbol{\alpha}$ is defined to be the component-wise product $\boldsymbol{s} \cdot \boldsymbol{\beta}$, where $\boldsymbol{s}$ comes from $\mathcal{F}_{\mathsf{ECDSA}}$. The honest parties' shares of $\boldsymbol{\alpha}$ are sampled uniformly conditioned on their reconstruction with corrupt parties' shares inducing $\boldsymbol{\alpha}$. This hybrid experiment is distributed identically to the last; $[\boldsymbol{\alpha}]_{n-1}$ is a uniformly random secret sharing of $\boldsymbol{\alpha}$ in the previous hybrid as $[\boldsymbol{z}]_{n-1}$ is a uniformly random additive one-time pad.

In summary, the honest parties' values in the adversary's view in the protocol consist only of their shares of d, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta}$ as revealed during their respective reconstructions, and each of these values are computed without touching honest parties' private inputs in the final hybrid experiment. The adversary's view in Π_{Sign} is therefore perfectly simulated in the $\mathcal{F}_{\mathsf{Prep}}$ hybrid model. □

We highlight that we achieve standard security with abort, but the adversary cannot cause aborts conditioned on honest parties' *secrets*, as these are simulatable in the proof. The adversary may indeed cause an abort conditioned on *public* input as the message $\boldsymbol{m}$, and this action is allowed by $\mathcal{F}_{\mathsf{ECDSA}}$. To our knowledge, essentially all threshold signing protocols in the security with abort model permit the adversary to halt signature generation conditioned on the message.

Remark 2 (Signing under multiple keys). Our protocol can be easily extended to sign ℓ messages $\boldsymbol{m} = (m_1, \ldots, m_\ell)$ under ℓ different secret keys $\mathbf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$, adding minimal cost. This requires the N parties to start with degree-t Shamir shares $[\mathsf{sk}_i \mid i]_t$ for $i = 1, \ldots, \ell$, where $[x \mid i]_t$ is shorthand for Shamir shares where the secret is in position $-i$. See Section 7.2 for details.

7.1 Communication Complexity of Π_{Sign}

The communication complexity per party for the protocol Π_{Sign} is bounded by

$$\begin{aligned}
&\log p \cdot \Bigg(102.69\lambda - 4\kappa + \frac{16\lambda\ell}{\lambda/8 - \kappa} C(\mathcal{F}_{\mathsf{Mult}}, s) \\
&+ \kappa \cdot (140.37\lambda - 1027\kappa - 2) \Bigg) \\
&+ \log p \cdot \Bigg(\underbrace{2\lambda}_{\text{Message independent}} + \underbrace{2\lambda}_{\text{Message dependent}} \Bigg) \\
&+ \underbrace{2\left(\frac{\lambda}{8} - \kappa\right)\kappa}_{\text{Commitments}} \\
&= \log p \cdot \Bigg(106.69\lambda - 4\kappa + \frac{16\lambda\ell}{\lambda/8 - \kappa} C(\mathcal{F}_{\mathsf{Mult}}, s) \\
&+ \kappa \cdot (140.37\lambda - 1027\kappa - 2) \Bigg) + 2\left(\frac{\lambda}{8} - \kappa\right)\kappa,
\end{aligned}$$

where $C(\mathcal{F}_{\mathsf{Mult}}, s)$ is the cost of one call of $\mathcal{F}_{\mathsf{Mult}}$ among s parties. We discussed some optimizations in Section 6.3, where we also present potential solutions for the κ overhead based on SNARKs. However, as we show in Section 8, even with this factor, we outperform prior works.

Using PCGs, the communication complexity is bounded by

$$\begin{aligned}
&\log p \cdot (173.25\lambda + \kappa \cdot (140.37\lambda - 1027\kappa - 6)) \\
&\quad + 2\left(\frac{\lambda}{8} - \kappa\right)\kappa
\end{aligned} \tag{11}$$

7.2 Signing under Multiple Keys

Our protocol can be easily extended to sign ℓ messages $\boldsymbol{m} = (m_1, \ldots, m_\ell)$ under ℓ different secret keys $\mathbf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$. For our signing protocol we require the set of n signing parties to have packed shares $[\mathbf{sk}]_{n-\ell}$. This can be obtained from the shares of the global N parties if they start with degree-t Shamir shares $[\mathsf{sk}_i \mid i]_t$ for $i = 1, \ldots, \ell$, where $[x \mid i]_t$ is shorthand for Shamir shares where the secret is in position $-i$. Note that, if $\boldsymbol{e}_i$ is the ℓ-dimensional vector with all 0s except for 1 in the i-th position, then $[\boldsymbol{e}_i]_{\ell-1} \cdot [\mathsf{sk}_i \mid i]_t$ is a degree-$(t + (\ell - 1))$ packed sharing where the secrets has all 0s, except for the i-th entry which is sk_i. Hence, parties can locally compute packed shares $[\mathbf{sk}]_{t+(\ell-1)} = \sum_{i=1}^{\ell} [\boldsymbol{e}_i]_{\ell-1} \cdot [\mathsf{sk}_i \mid i]_t$.

The changes with respect to our main protocol for one key are simple. For the online phase, we need to use more than one mask a in order to hide all different keys. This is reflected in the preprocessing in that we need to generate more such masks. We discuss details below.

Preprocessing for Multiple Keys The main change with respect to $\mathcal{F}_{\mathsf{Prep}}$ is that, instead of generating one $\langle a \rangle, a \cdot G$, we need to prepare $[\boldsymbol{a}]_{n-\ell}$ together with $[\boldsymbol{a} \cdot G]_{\ell-1}$. These changes are reflected in Functionality $\mathcal{F}_{\mathsf{PrepManyKeys}}$ below.

Functionality 5: $\mathcal{F}_{\mathsf{PrepManyKeys}}$

Public Parameters: Let n be the number of parties, t be the corruption threshold, and $(\mathbb{G}, G, p)$ characterize an elliptic curve over $\mathbb{Z}_q$ generated by G, of prime order p. Let $n = t + 2(\ell - 1) + 1$.

- Upon receiving (init) from all parties, $\mathcal{F}_{\mathsf{PrepManyKeys}}$ samples $\boldsymbol{a} \leftarrow \mathbb{Z}_p^\ell$, $\boldsymbol{\phi} \leftarrow \mathbb{Z}_p^\ell$, and $\boldsymbol{k} \leftarrow \mathbb{Z}_p^\ell$.
- Define $\boldsymbol{z} = \boldsymbol{0}$, $\boldsymbol{\beta} = \boldsymbol{k} \star \boldsymbol{\phi}$, and $(\boldsymbol{r}_{\mathsf{x}}, \boldsymbol{r}_{\mathsf{y}}) = \boldsymbol{k} \cdot G$
- $\mathcal{F}_{\mathsf{PrepManyKeys}}$ samples secret sharings $[\boldsymbol{z}]_{n-1}$, $[\boldsymbol{a}]_{n-\ell}$, $[\boldsymbol{a} \cdot G]_{\ell-1}$, $[\boldsymbol{\phi}]_{n-\ell}$, $[\boldsymbol{a} \star \boldsymbol{\phi}]_{n-\ell}$, $[\boldsymbol{r}_{\mathsf{x}}]_{\ell-1}$, and $[\boldsymbol{\beta}]_{n-1}$ and sends the appropriate shares to each party as private output.

We note that our protocol Π_{Tuple} can be easily modified to achieve $\mathcal{F}_{\mathsf{PrepManyKeys}}$ as follows.

1. In Step 1.1, we change to prepare $[\boldsymbol{A}_{\star,j}]_{n-1}$ for all $j \in \{1, \ldots, \lambda\}$ in the same way as that for $[\boldsymbol{B}_{\star,j}]_{n-1}$ and $[\boldsymbol{\Phi}_{\star,j}]_{n-1}$.
2. In Step 1.2.(a), we change to prepare $[\boldsymbol{A}_{j,\star}]^V_{(N-1)/2}$ in the same way as Step 1.2.(b) for $[\boldsymbol{B}_{j,\star}]^V_{(N-1)/2}$ and $[\boldsymbol{\Phi}_{j,\star}]^V_{(N-1)/2}$.
3. In Step 1.3, $[\boldsymbol{C}_{j,\star}]^V_{N-1}$ is computed by multiplying $[\boldsymbol{A}_{j,\star}]^V_{(N-1)/2}$ and $[\boldsymbol{\Phi}_{j,\star}]^V_{(N-1)/2}$.
4. Step 1.4, Step 2, and Step 3.1 remain unchanged.
5. In Step 3.2, $[\boldsymbol{A}_{\star,j} \cdot G]_{\ell-1}$ is prepared (and checked) in the same way as that for $[\boldsymbol{B}_{\star,j} \cdot G]_{\ell-1}$. Note that $[\boldsymbol{A}_{\star,j} \cdot G]_{\ell-1}$ and $[\boldsymbol{B}_{\star,j} \cdot G]_{\ell-1}$ can be checked together by generating 2λ random values using one invocation of $\mathcal{F}_{\mathsf{Coin}}$.
6. In Step 3.3, we change to compute $[\tilde{\boldsymbol{C}}_{\star,j} \cdot G]_{n-1} = [\boldsymbol{A}_{\star,j} \cdot G]_{\ell-1} \cdot [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}$, which is the same as that for computing $[\tilde{D}_{\star,j} \cdot G]_{n-1} = [\boldsymbol{B}_{\star,j} \cdot G]_{\ell-1} \cdot [\boldsymbol{\Phi}_{\star,j}]_{n-\ell}$. Then the rest of steps remain unchanged.

The cost of the new protocol for $\mathcal{F}_{\mathsf{PrepManyKeys}}$ remains the same as the original Π_{Tuple}.

In Section 8, we present a detailed comparison of the communication of our work with [Doe+24] using concrete measurements along with a graphical visualization of the comparison.

Online Phase for Multiple Keys. The changes with respect to Protocol Π_{Sign} are the following. First, in step 1 of the message-independent phase we mask and open using packed secret-sharing instead of additive secret-sharing. In step 2, P_1 sends back *shares* of the masked vector $\boldsymbol{d}$, instead of sending it in the clear. Checking consistency is slightly more involved than in the single-key case, since the parties do not have $\boldsymbol{d}$ in the clear. However, we assume they do have all the public keys $\mathbf{pk}$ in the clear, which enables the following generalization of the single-key check: Parties locally compute $[\mathbf{pk}]_{\ell-1} - [\boldsymbol{a} \cdot G]_{\ell-1}$, which should be

degree-$(\ell-1)$ sharings of $\boldsymbol{d} \cdot G$. Since the parties have received $[\boldsymbol{d}]_{\ell-1}$ from P_1, they can locally compute $[\boldsymbol{d} \cdot G]_{\ell-1}$ and compare it against the shares computed above. Note that degree-$(\ell-1)$ sharings are *deterministic*, which is what enables this check.

Now, in step 4 a crucial difference is that the parties obtain degree-$(n-1)$ sharings of $[\mathsf{sk} \star \boldsymbol{\phi}]_{n-1}$, instead of degree-$(n-\ell)$ sharings as in Π_{Sign}. This causes trouble later in step 2 of the online phase, where these sharings must be multiplied by degree-$(\ell-1)$ sharings. To address this we can perform *degree reduction*, which converts sharings from degree $n-1$ to sharings of degree $n-\ell$. At a high level, this works by having the parties sample sharings of random values of both degree $n-1$ and $n-\ell$, mask the desired high-degree sharings with these random values, open this towards one party, and then "undo the mask" using the low degree. We refer the reader to [Esc+23; GPS21; GPS22] for details.

The protocol is described as Protocol $\Pi_{\mathsf{SignManyKeys}}$ below.

Protocol 8: $\Pi_{\mathsf{SignManyKeys}}$

Key Shares. Parties start with Shamir shares $[\mathsf{sk}]_{n-\ell}$, where $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$.

Generate Nonces. Parties invoke $\mathcal{F}_{\mathsf{PrepManyKeys}}$ to obtain $[\boldsymbol{z}]_{n-1}$, $[\boldsymbol{a}]_{n-\ell}, [\boldsymbol{a} \cdot G]_{\ell-1}, [\boldsymbol{\phi}]_{n-\ell}, [\boldsymbol{a} \star \boldsymbol{\phi}]_{n-\ell}, [\boldsymbol{\beta} = \boldsymbol{k} \star \boldsymbol{\phi}]_{n-1}$, and $[\boldsymbol{r}_{\mathsf{x}}]_{\ell-1}$, where $(\boldsymbol{r}_{\mathsf{x}}[i], \cdot) \stackrel{\mathsf{def}}{=} \boldsymbol{k}[i] \cdot G$, for all $i \in [\ell]$.

Message-independent phase. Computation of $[\mathsf{sk} \star \boldsymbol{\phi}]_{n-\ell}$:

1. The parties locally compute $[\boldsymbol{d}]_{n-\ell} \stackrel{\mathsf{def}}{=} [\mathsf{sk}]_{n-\ell} - [\boldsymbol{a}]_{n-\ell}$.
2. Parties send their shares of $[\boldsymbol{d}]_{n-\ell}$ to P_1 who reconstructs $\boldsymbol{d}$, and then sends shares $[\boldsymbol{d}]_{\ell-1}$.
3. Parties check consistency by computing locally $[\mathsf{pk}]_{\ell-1} - \boldsymbol{a} \cdot G$, and also $[\boldsymbol{d} \cdot G]_{\ell-1} = [\boldsymbol{d}]_{\ell-1} \cdot G$. Each party checks their resulting shares in both cases are the same, and aborts if this is not the case.
4. The parties locally compute the following:
$$[\mathsf{sk} \cdot \boldsymbol{\phi}]_{n-1} = [\boldsymbol{d}]_{\ell-1} \cdot [\boldsymbol{\phi}]_{n-\ell} + [\boldsymbol{a} \cdot \boldsymbol{\phi}]_{n-\ell} \tag{12}$$
5. Parties perform *degree reduction* (see cf. [Esc+23; GPS21; GPS22]) to convert $[\mathsf{sk} \cdot \boldsymbol{\phi}]_{n-1}$ into $[\mathsf{sk} \cdot \boldsymbol{\phi}]_{n-\ell}$.

Online phase. To sign a vector of messages $\boldsymbol{m}$, the parties proceed as follows.

1. Each party derives its share of a canonical $[\mathsf{SHA}(\boldsymbol{m})]_{\ell-1}$.
2. The parties locally compute
$$\begin{aligned}[\boldsymbol{\alpha}]_{n-1} \stackrel{\mathsf{def}}{=} [\boldsymbol{z}]_{n-1} + [\mathsf{SHA}(\boldsymbol{m})]_{\ell-1} \star [\boldsymbol{\phi}]_{n-\ell} \\ + [\boldsymbol{r}_{\mathsf{x}}]_{\ell-1} \star [\mathsf{sk} \cdot \boldsymbol{\phi}]_{n-\ell}.\end{aligned} \tag{13}$$
3. Let R be the intended receiver of the signatures. Each party sends its shares of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ to R, who then interpolates them to obtain $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ respectively, and computes the vector of putative signatures $\boldsymbol{s} = \boldsymbol{\alpha}/\boldsymbol{\beta}$. R outputs $(\boldsymbol{r}_{\mathsf{x}}, \boldsymbol{s})$ if it is a vector of valid ECDSA signatures, and aborts if not.

8 Comparison with [Doe+24]

In this section, we compare the communication of our protocol with that of [Doe+24], which we refer to as DKLs23. Their protocol is set in the standard $t+1$-party setting, and their communication complexity grows with the number of signatures. In both cases, we consider an instantiation of $\mathcal{F}_{\mathsf{Mult}}$ using PCGs. We present in Table 1 the total communication (across all parties) of the offline and online phases of our protocol to sign ℓ messages, and compare them with these from [Doe+24]. We present the results for different values of t and ℓ, which implicitly define the number of parties n in our protocol as $n = t + 2(\ell - 1) + 1$. In contrast, [Doe+24] needs less parties, namely $t + 1$.

t	ℓ	n	Offline phase (KB)		Online phase (KB)	
			Ours	DKLs23 [Doe+24]	**Ours**	DKLs23 [Doe+24]
10	2	13	794.57	131.65 (×**0.17**)	0.83	1.41 (×**1.69**)
	3	15	984.47	197.47 (×**0.20**)	0.96	2.11 (×**2.20**)
	5	19	1400.95	329.12 (×**0.23**)	1.22	3.52 (×**2.89**)
25	3	30	2737.30	1103.23 (×**0.40**)	1.92	4.99 (×**2.60**)
	7	38	3833.26	2574.21 (×**0.67**)	2.43	11.65 (×**4.79**)
	11	46	5000.90	4045.18 (×**0.81**)	2.94	18.30 (×**6.22**)
50	6	61	7324.12	8489.66 (×**1.16**)	3.90	19.58 (×**5.02**)
	13	75	9599.35	18394.27 (×**1.92**)	4.80	42.43 (×**8.84**)
	21	91	12282.55	29713.82 (×**2.42**)	5.82	68.54 (×**11.77**)
75	8	90	12112.84	25137.15 (×**2.08**)	5.76	38.91 (×**6.76**)
	19	112	15893.48	59700.74 (×**3.76**)	7.17	92.42 (×**12.89**)
	31	136	20101.30	97406.46 (×**4.85**)	8.70	150.78 (×**17.32**)
100	11	121	17463.17	61042.78 (×**3.50**)	7.74	71.10 (×**9.18**)
	26	151	22761.29	144282.94 (×**6.34**)	9.66	168.06 (×**17.39**)
	41	181	28129.05	227523.10 (×**8.09**)	11.58	265.02 (×**22.88**)
200	21	241	38984.28	461541.02 (×**11.84**)	15.42	270.14 (×**17.51**)
	51	301	49928.92	1120885.34 (×**22.45**)	19.26	656.06 (×**34.06**)
	81	361	60922.83	1780229.66 (×**29.22**)	23.10	1041.98 (×**45.10**)
400	41	481	82990.38	3586505.50 (×**43.22**)	30.78	1052.22 (×**34.18**)
	101	601	105114.48	8835050.14 (×**84.05**)	38.46	2592.06 (×**67.39**)
	161	721	127268.37	14083594.78 (×**110**)	46.14	4131.90 (×**89.54**)

Table 1: Total communication (across all parties) to sign ℓ elements using our protocol vs. [Doe+24] instantiating $\mathcal{F}_{\mathsf{Mult}}$ using PCGs. Various combinations of t and ℓ are displayed, and reported values are in kilobytes. Our protocol uses $n = t + 2(\ell - 1) + 1$ parties while [Doe+24] uses $t + 1$. We use $\kappa = 40$.

We see that, when it comes to the online phase, our protocol *always* involves less communication, with factors increasing drastically already for very small combinations of t and ℓ. For example, for $t = 10$, adding 3 parties to our protocol leads to 1.69× less communication for signing 2 messages. For $t = 50$, adding 11 parties leads to 5× improvements, and for $t = 100$ adding 21 parties leads to 9× less communication. Although unusual in practical settings, as means of exploration we analyze the case when t is very large. For $t = 400$, we see that

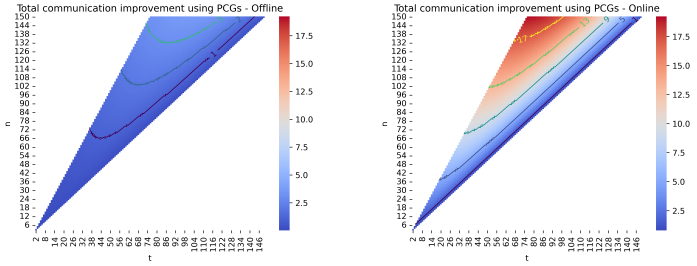


Fig. 2: Improvement factor of the offline (left) and online (right) phases compared to DKLs23 [Doe+24]. We are comparing our work with DKLs23 using pre-signing, which means that their work uses the first two rounds as the preprocessing.

adding 20% more parties (81) leads to our protocol being 34× lighter, and adding a bit over 300 parties leads to almost 90× reduction in communication.

Naturally, the more parties are added, the bigger the improvement is, like for $t = 100$ where adding 81 parties leads to $> 20\times$ improvements for signing 41 messages. That said, when so many parties are added the benefits in communication may be reduced due to the fact that our protocol would use many more parties than DKLs23. We believe that when not so many parties are added, that is, when ℓ is small relative to t, our protocol has the most potential of practically outperforming DKLs23, but even for larger ℓ this potential still exists.

In terms of the offline phase, our protocol is far more complex than DKLs23 in order to allow for the packed signing, and therefore its offline phase is not always better. However, the regime where DKLs23 outperform us is rather reduced. Specifically, only if t is very small our offline phase has more communication than DKLs23, like $t \leq 30$. Furthermore, this gap is not too big. For larger t our offline phase not only matches that of DKLs23, but it actually outperforms it rather quickly. We remark however that our offline phase has about 24 rounds, which in large distributed settings may constitute a more noticeable bottleneck than communication itself. Nevertheless, since the offline phase is executed once and possibly "overnight" before the message are known, one can reasonably expect this price to be quickly offset by the benefits in the online phase.

Finally, we also present in Fig. 2 a more visual representation of the improvement factor of our offline and online phases with respect to DKLs23. These graphs put into perspective the regime for ℓ and n for which our protocol is better. For the offline phase, note that for any t we do not need to add too many parties until our offline phase matches that of DKLs23, as shown by the contour line corresponding to the ratio = 1. For the online phase we note that the improvement ratios quickly go up as we add more parties.

References

[Abr+22] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. "Low-Bandwidth Threshold ECDSA via Pseudorandom Correlation Generators". In: 2022, pp. 2554–2572. DOI: 10.1109/SP46214.2022.9833559.

[Ben+20] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. "DEEP-FRI: Sampling Outside the Box Improves Soundness". In: 2020, 5:1–5:32. DOI: 10.4230/LIPIcs.ITCS.2020.5.

[Ben+24] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. "SPRINT: High-Throughput Robust Distributed Schnorr Signatures". In: *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part V*. Ed. by Marc Joye and Gregor Leander. Vol. 14655. Lecture Notes in Computer Science. Springer, 2024, pp. 62–91. DOI: 10.1007/978-3-031-58740-5_3. URL: https://doi.org/10.1007/978-3-031-58740-5_3.

[BGG19] Dan Boneh, Rosario Gennaro, and Steven Goldfeder. "Using Level-1 Homomorphic Encryption to Improve Threshold DSA Signatures for Bitcoin Wallet Security". In: 2019, pp. 352–377. DOI: 10.1007/978-3-030-25283-0_19.

[Bou00] Fabrice Boudot. "Efficient Proofs that a Committed Number Lies in an Interval". In: 2000, pp. 431–444. DOI: 10.1007/3-540-45539-6_31.

[Bra87] Gabriel Bracha. "An O(log n) expected rounds randomized byzantine generals protocol". In: *J. ACM* 34.4 (1987), 910–920. ISSN: 0004-5411. DOI: 10.1145/31846.42229. URL: https://doi.org/10.1145/31846.42229.

[BY24] Alexander Bienstock and Kevin Yeo. "Two Levels are Better than One: Dishonest Majority MPC with O(|C|) Total Communication". In: *Cryptology ePrint Archive* (2024).

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.

[Can+20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts". In: 2020, pp. 1769–1787. DOI: 10.1145/3372297.3423367.

[Cas+23] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. "Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts proactive and adaptive security". In: *Theor. Comput. Sci.* 939 (2023), pp. 78–104. DOI: 10.1016/J.TCS.2022.10.016. URL: https://doi.org/10.1016/j.tcs.2022.10.016.

[Dal+20] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. "Securing DNSSEC Keys via Threshold ECDSA

from Generic MPC". In: 2020, pp. 654–673. DOI: 10.1007/978-3-030-59013-0_32.

[Doe+24] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. "Threshold ECDSA in Three Rounds". In: *45th IEEE Symposium on Security and Privacy, SP 2024*. IEEE, 2024.

[Esc+23] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, Yifan Song, and Chenkai Weng. "SuperPack: Dishonest Majority MPC with Constant Online Communication". In: 2023, pp. 220–250. DOI: 10.1007/978-3-031-30617-4_8.

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. "Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security". In: 2016, pp. 156–174. DOI: 10.1007/978-3-319-39555-5_9.

[GHL22] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. "Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties". In: 2022, pp. 458–487. DOI: 10.1007/978-3-031-06944-4_16.

[GPS21] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. "Unconditional Communication-Efficient MPC via Hall's Marriage Theorem". In: 2021, pp. 275–304. DOI: 10.1007/978-3-030-84245-1_10.

[GPS22] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. "Sharing Transformation and Dishonest Majority MPC with Packed Secret Sharing". In: 2022, pp. 3–32. DOI: 10.1007/978-3-031-15985-5_1.

[GS22] Jens Groth and Victor Shoup. "On the Security of ECDSA with Additive Key Derivation and Presignatures". In: 2022, pp. 365–396. DOI: 10.1007/978-3-031-06944-4_13.

[GS24] Jens Groth and Victor Shoup. "Fast Batched Asynchronous Distributed Key Generation". In: *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part V*. Ed. by Marc Joye and Gregor Leander. Vol. 14655. Lecture Notes in Computer Science. Springer, 2024, pp. 370–400. DOI: 10.1007/978-3-031-58740-5_13. URL: https://doi.org/10.1007/978-3-031-58740-5_13.

[Hai+22] Iftach Haitner, Nikolaos Makriyannis, Samuel Ranellucci, and Eliad Tsfadia. "Highly Efficient OT-Based Multiplication Protocols". In: 2022, pp. 180–209. DOI: 10.1007/978-3-031-06944-4_7.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11.

[Lan95] Susan K. Langford. "Threshold DSS Signatures without a Trusted Party". In: 1995, pp. 397–409. DOI: 10.1007/3-540-44750-4_32.

[LN18] Yehuda Lindell and Ariel Nof. "Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocur-

rency Custody". In: 2018, pp. 1837–1854. DOI: 10.1145/3243734.3243788.

[MR01] Philip D. MacKenzie and Michael K. Reiter. "Two-Party Generation of DSA Signatures". In: 2001, pp. 137–154. DOI: 10.1007/3-540-44647-8_8.

[RS22] Rahul Rachuri and Peter Scholl. "Le Mans: Dynamic and Fluid MPC for Dishonest Majority". In: 2022, pp. 719–749. DOI: 10.1007/978-3-031-15802-5_25.

[RW19] Dragos Rotaru and Tim Wood. "MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security". In: 2019, pp. 227–249. DOI: 10.1007/978-3-030-35423-7_12.

List of Functionalities, Protocols and Procedures

Functionalities

Functionality 1: $\mathcal{F}_{\mathsf{DoubleModuli}}$ 16
Functionality 2: $\mathcal{F}_{\mathsf{Tuple}}$ 22
Functionality 3: $\mathcal{F}_{\mathsf{Prep}}$ 38
Functionality 4: $\mathcal{F}_{\mathsf{ECDSA}}$ 47
Functionality 5: $\mathcal{F}_{\mathsf{PrepManyKeys}}$ 50

Protocols

Protocol 4: $\Pi_{\mathsf{DoubleModuli}}$ 20
Protocol 5: Π_{Tuple} 24
Protocol 6: Π_{Prep} 40
Protocol 7: Π_{Sign} 47
Protocol 8: $\Pi_{\mathsf{SignManyKeys}}$ 51

Procedures

Procedure 1: $\pi_{\mathsf{DistShares}}$ 15
Procedure 2: $\pi_{\mathsf{RangeSlack}}$ 18
Procedure 3: π_{RangeSOS} 19