Communication-Efficient Proactive MPC for Dynamic Groups with Dishonest Majorities

Karim Eldefrawy, Tancrède Lepoint, Antonin Leroux

Communication-Efficient Proactive MPC for Dynamic Groups with Dishonest Majorities

Karim Eldefrawy[1], Tancrède Lepoint[2⋆], and Antonin Leroux[3,4,5]

[1] SRI International, karim.eldefrawy@sri.com
[2] iacr@tancre.de
[3] DGA
[4] LIX, CNRS, Ecole Polytechnique, Institut Polytechnique de Paris
[5] INRIA, antonin.leroux@polytechnique.org

Abstract. Secure multiparty computation (MPC) has recently been increasingly adopted to secure cryptographic keys in enterprises, cloud infrastructure, and cryptocurrency and blockchain-related settings such as wallets and exchanges. Using MPC in blockchains and other distributed systems highlights the need to consider dynamic settings. In such dynamic settings, parties, and potentially even parameters of underlying secret sharing and corruption tolerance thresholds of sub-protocols, may change over the lifetime of the protocol. In particular, stronger threat models – in which *mobile* adversaries control a changing set of parties (up to t out of n involved parties at any instant), and may eventually corrupt *all* n *parties* over the course of a protocol's execution – are becoming increasingly important for such real world deployments; secure protocols designed for such models are known as Proactive MPC (PMPC).
In this work, we construct the first efficient PMPC protocol for *dynamic* groups (where the set of parties changes over time) secure against a *dishonest majority* of parties. Our PMPC protocol only requires $O(n^2)$ (amortized) communication per secret, compared to existing PMPC protocols that require $O(n^4)$ and only consider static groups with dishonest majorities. At the core of our PMPC protocol is a new efficient technique to perform multiplication of secret shared data (shared using a bivariate scheme) with $O(n\sqrt{n})$ communication with security against a dishonest majority without requiring pre-computation. We also develop a new efficient bivariate batched proactive secret sharing (PSS) protocol for dishonest majorities, which may be of independent interest. This protocol enables multiple dealers to contribute different secrets that are efficiently shared together in one batch; previous batched PSS schemes required all secrets to come from a single dealer.

1 Introduction

Dynamic MPC settings, where parties and parameters of the underlying secret sharing and sub-protocols can change during the execution of the protocol,

⋆ Part of this work was performed while at SRI International and Google.

Table 1. Overview of features and limitations of proactive secret sharing (PSS) and proactive MPC (PMPC) protocols.

	Type	Batching	Dynamic Groups	Dishonest Majority	Fair Reconstruct	Subprotocols Communication (amortized)
[BELO14]	PMPC	✓	✗	✗	✗	$O(1)$
[BELO15]	PSS/PMPC	✓	✓	✗	✗	$O(1)$
[DEL$^+$16]	PSS only	✗	✗	✓	✓	$O(n^4)$
[EOPY18]	PMPC	✗	✗	✓	✓	$O(n^4)$
[ELL20]	PSS only	✓	✓	✓	✓	$O(n^2)$
This work	PMPC	✓	✓	✓	✓	$O(n^2)$

have attracted a lot of attention in the past years. Some of these settings consider very powerful adversaries who can compromise dishonest majorities, i.e., active/malicious or passive/semi-honest parties that may add up to a majority. Additionally, for long-lived computation and better security guarantees, stronger threat models in which *mobile* adversaries [OY91,HJKY95] control a changing set of parties (up to t out of the n parties at any instant), and may eventually corrupt *all n parties* over the course of a protocol's execution or lifetime of confidential inputs, are becoming increasingly attractive in the real world deployments of MPC. MPC protocols withstanding such *mobile* adversaries are typically called Proactive MPC (PMPC) [OY91,HJKY95].

Related work in proactive secret sharing (PSS) and PMPC, and the different settings considered, is listed in Table 1. In the *honest majority setting*, the early work of Baron, Eldefrawy, Lampkins, and Ostrovsky [BELO14] introduces the framework to construct PMPC from PSS by computing the circuit layer by layer and (proactively) redistributing the parties' secret shares after each layer. The PMPC protocol handles batching (i.e., the secret sharing contains many secrets operated on in a coefficient-wise manner) and static groups in the honest majority setting. In a follow-up work, Baron, Eldefrawy, Lampkins, and Ostrovsky [BELO15] consider then the setting of dynamic groups. The study of PSS and PMPC in the dishonest majority setting starts with the work of Dolev, Eldefrawy, Lampkins, Ostrovsky, and Yung [DEL$^+$16], in which they present a PSS scheme (without batching) for static groups. Feasibility of constructing PMPC withstanding a dishonest majority of parties was then demonstrated by Eldefrawy, Ostrovsky, Park, and Yung in [EOPY18]. The subprotocols in the last two works have communication complexity $O(n^4)$, which significantly hinders their practicality. The schemes from [DEL$^+$16] and [EOPY18] have the additional property of ensuring fair reconstruction with the gradual sharing model from Hirt, Lucas and Maurer [HLM13]. The PSS scheme from [DEL$^+$16] was later revisited by Eldefrawy, Lepoint, and Leroux [ELL20]: they present an efficient PSS scheme with batching, fair reconstruct with no complexity overhead and dynamic groups, with security against mixed adversaries that can compromise

a majority of parties, but leave as future work to extend it to a full PMPC protocol. This naturally brings us to formulate the following open problem:

> *Can we develop a communication-efficient PMPC protocol that handles batching, with amortized communication $O(n^2)$ or less, for dynamic groups, and with security against mixed adversaries that can compromise a majority of parties?*

1.1 Contributions

In this work, we affirmatively answer this question by constructing an efficient PMPC protocol with four key properties: (i) batching, (ii) suitability for dynamic groups (iii) security against a majority of active/malicious or passive/semi-honest corruptions, and allowing (iv) fair reconstruct with no complexity overhead in the mixed adversarial setting proposed by Hirt, Lucas, and Maurer [HLM13]. Our protocol achieves computational security and the efficiency is enabled by only requiring $O(n^2)$ (amortized) communication per secret when batching $O(n)$ secrets. Our communication model assumes a broadcast channel and pairwise secure channels. Concretely, we make the following contributions:

1. We develop the *first* efficient fair PMPC for dishonest majorities and dynamic groups, with $O(n^2)$ (amortized with batches of size $\ell = n-2$) communication (in both broadcast and secure channels). Our new PMPC protocol protects secrecy of the inputs when active and passive corruptions are less than $n - 3 - \sqrt{\ell}$ at any time of the protocol. Additionally, the computation is *fair* if the number of active corruption is less than k and the number of passive corruption is less than $\min(n - k - \sqrt{\ell}, 2(n-k) - \ell)$ during the reconstruction, for $1 \leq k \leq n/3$ (cf. Theorem 1).
2. We develop a new efficient bivariate batched proactive secret sharing (PSS) Share protocol for dishonest majorities that enables multiple dealers to contribute different secrets that are shared together in one batch. Previous batched PSS schemes in the dishonest majority setting required all secrets to come from one dealer.
3. At the core of the protocol is a new efficient sub-protocol for multiplying secret-shared data (using a bivariate sharing of degree $d = n-2$ for batches of size d) with $O(n\sqrt{n})$ amortized communication, and secure when the number of corruptions (either active or passive) is less than $n - 3 - \sqrt{n-2}$ without requiring pre-computation (cf. Theorem 2). The techniques developed to this effect might be of independent interest.

1.2 Technical Overview

Previous PMPC protocols [EOPY18], proven secure in the dishonest majority setting and for static groups, builds on top of the proactive secret sharing scheme of [DEL$^+$16] by augmenting it with protocols for adding and multiplying shares to perform computation on the secret shares following the same (arithmetic)

PMPC blueprint as proposed in [BELO14]. While additions are computed locally, multiplications require using the standard GMW MPC protocol [GMW87], so as to obtain a proactive secret sharing of the multiplication of two secrets. The asymptotic communication efficiency of [EOPY18] is the same as that of [DEL$^+$16], i.e $O(n^4)$.

Similarly, we develop our new PMPC protocol for dynamic groups with dishonest majorities on top of a recent PSS protocol [ELL20] for the dishonest majority setting. However, the PSS of [ELL20] differs significantly from that of [DEL$^+$16], and extending [ELL20] to an efficient PMPC protocol with (amortized) communication $O(n^2)$ requires care and new techniques; the rest of this section summarizes the main intuition behind our construction.

Let us briefly recall the PSS construction of [ELL20]. Secrets $s_1, \ldots, s_\ell$ are secret shared among n participants $P_1, \ldots, P_n$ by a dealer, which construct a bivariate polynomial g such that:

- $g(x, \cdot)$ and $g(\cdot, y)$ are of degree at most $d \leq n-1$;
- the secrets are embedded as $g(\beta_i, \beta_i) = s_i$ for distinct β_i's;
- the secret share of party P_i is the polynomial $g(\alpha_i, \cdot)$, for distinct α_i's.

This sharing naturally supports additions: party P_i will be able to locally add its secret shares $g(\alpha_i, \cdot)$ (where g is the bivariate polynomial for $s_1, \ldots, s_\ell$) and $g'(\alpha_i, \cdot)$ (where g' is the bivariate polynomial for $s'_1, \ldots, s'_\ell$) to obtain a secret sharing of the sum of the secrets $(s_1 + s'_1, \ldots, s_\ell + s'_\ell)$. However, when extending this PSS to a PMPC protocol in a layered manner as in [BELO14,BELO15], it will be necessary to provide a Permute operation on secret shares, so that they are in the correct order for performing the arithmetic operations for that layer. In Appendix C, we adapt the protocols of [BELO14,BELO15] (inspired from [Ben64,Wak68,GHS12]) to the PSS of [ELL20].

Contribution 1: Efficient multi-dealer batched sharing protocol. We point here a subtle feature which was not present in [DEL$^+$16,EOPY18] and that usually does not manifest in the standard PSS functionality and was also lacking from [ELL20]. Multi-dealer batched sharing allows us to use the batching techniques (and the resulting improvement in communication complexity) for computations where each participant has $O(1)$ secrets, something impossible with single-dealer sharing protocols because each participant has to do at least one sharing for its secrets. We obtain this multi-dealer sharing with a simple adaptation of techniques used in [ELL20]. It suffices that all the dealers generate a classical Shamir sharing for each of their secrets and then a bivariate sharing for the whole batch is obtained by combining these univariate polynomials into a bivariate polynomial. Additionally, performing addition and multiplication on batch of secrets require permuting the secrets between consecutive layers to align them. Thus, the underlying PSS needs to be secure even when some of the shared secrets have been leaked to the adversary.

Contribution 2: Efficient multiplication of shared secrets for groups with dishonest majorities. This protocol is at the core of our contributions enabling the construction of our new communication-efficient PMPC protocol. In fact, our `Mult` protocol achieves even better than the minimal requirement of $O(n^2)$ with an amortized communication complexity of $O(n\sqrt{n})$ against up to approximately $n-\sqrt{n}$ actively corrupted participant. This improvement of $\sqrt{n}$ over the standard quadratic complexity for multiplication without precomputation in the dishonest majority setting may be of independent interest. It has the following blueprint:

1. The participants have shares for two "bivariate" secret sharings g, g' containing both ℓ secrets to be multiplied together. First, g is transformed into $\ell^{\frac{1}{2}}$ "univariate" secret sharings $f_1(\alpha_i), \ldots, f_{\ell^{\frac{1}{2}}}(\alpha_i)$, where each f_j is of degree d and contains $\ell^{\frac{1}{2}}$ secrets (and similarly for g'). This step is done by each party generating a random polynomial and using the Lagrange interpolation formula to embed the secrets from g.
2. Then, $\ell^{\frac{1}{2}}$ "blinding" bivariate polynomials h_j of degree d such that $h_j(\beta_i, \beta_i) = 0$, are generated. This step follows the classical approach of generating blinding polynomials: each party generates a random polynomial evaluating in 0 in the β_i's.
3. Next, $\ell^{\frac{1}{2}}$ bivariate polynomials $g_j^* = f_j(x)f_j'(y) + h_j(x, y)$ are computed, and party P_i learns $g_j^*(\alpha_i, \cdot)$. Note that h_j "blinds" the product of the polynomials f_j and f_j' (except in (β_i, β_i) where g_j^* will evaluate into the product of the secrets. This step uses the secure multiplication protocol introduced in [LN18] in the context of threshold ECDSA.
4. Against active corruptions, correctness of the computation is verified using additively homomorphic commitments. To verify the correctness of the multiplication operations involved in the computation of g_j^*, the participants reveal $g_j^*(\mathtt{rand}_j, \cdot)$ for some random values $\mathtt{rand}_j$. This reduces the security threshold by one while preventing an adversary to deviate from the protocol undetected.
5. Finally, all the $\ell^{\frac{1}{2}}$ bivariate secret sharings $g_j^*(\alpha_i, \cdot)$ are recombined into a single bivariate sharing $g''(\alpha_i, \cdot)$ that embeds the $\ell = \ell^{\frac{1}{2}} \cdot \ell^{\frac{1}{2}}$ secrets.

1.3 Paper Outline

The rest of the paper is organized as follows. Section 2 overviews preliminaries required for the paper. Section 3 revisits the PSS scheme of [ELL20] for the setting of PMPC and introduces a new multi-dealer batched share sub-protocol. Section 4 presents the ideal functionality and concrete instantiation of our new PMPC protocol for dynamic groups with dishonest majorities. Section 5 focuses on each subprotocols of the overall PMPC protocol and proves their security; the formal security proofs for the PMPC protocol are provided in Appendix.

2 Preliminaries

Notation. Throughout the paper, we consider a set of n parties $\mathcal{P} = \{P_1, ..., P_n\}$, connected by pairwise synchronous secure channels and authenticated broadcast channels. $\mathcal{P}$ want to securely perform computations over a finite field $\mathbb{F} = \mathbb{Z}_q$ for a prime q.

For integers a, b, we denote $[a, b] = \{k : a \leq k \leq b\}$ and $[b] = [1, b]$.[6] We denote by $\mathbb{P}_k$ the set of polynomials of degree k exactly over $\mathbb{F}$. When a variable v is drawn randomly from a set S, we denote $v \leftarrow S$.

2.1 Adversary Model

In this section, we briefly recall the *proactive* security model and the mixed adversary setting used in this work. For a more precise exposition, we refer the reader to [EOPY18, Section 2]. The adversary in this model is considered to be a *mobile* adversary that can adaptively decide which parties to (passively or actively) corrupt between predefined "refresh phases" of the protocol. The computation is thus divided into "operation phases"; for example, the circuit representing the computation can be expressed as layers followed by "refresh phases" in which a refresh protocol is performed to prevent the adversary from learning too much information. The adversary can retain all the states of a corrupted party, but once a party is uncorrupted the adversary cannot learn future states of such a previously corrupted party unless it re-corrupts the party. At any point in time, we assume that at any point during the execution protocol, the adversary controls at most N parties (passively or actively); N is called the corruption threshold, and when $N \geq n/2$, we are in the dishonest majorities setting. Finally, note that in the proactive security model, a party can be uncorrupted either because the adversary willingly releases control of said party to compromise another party while not violating the corruption threshold, or because the party was proactively rebooted to a pristine state (hence the term of proactive security); henceforth, the adversary loses control over the party. In both cases, the uncorrupted party can recover its shares with the help of the other parties using a recovery protocol, and can continue participating in the computation.

2.2 Commitment Scheme

A commitment scheme [Ped91] is a classical cryptographic primitive. The commitment to a message $m \in \mathbb{F}_p$ under randomness $r \in \mathbb{F}_p$ is written $C(m, r)$. The opening information $o(m, r)$ can be revealed to enable a verifier to check whether $C(m, r)$ was indeed a valid commitment to m. A commitment scheme is *computationally hiding* if $C(m, r)$ does not reveal information to a computationally bounded attacker. It is *perfectly binding* if a commitment $C(m_1, r)$ can never be opened with $o(m_2, r')$ when $m_1 \neq m_2$.

[6] In particular, if $a > b$, we have $[a, b] = \emptyset$.

In this paper, we use an additively homomorphic commitment scheme, i.e., there is an operation $\star$ such that $C(m_1, r_1) \star C(m_2, r_2) = C(m_1 + m_2, r_1 + r_2)$. In particular, we instantiate our protocols with the computationally hiding and perfectly binding commitment scheme $(g^m h^r, g^r) \in G^2$ where G is a group of prime order p with generator g. This protocol is secure under the hardness of the DDH problem. For any element $(g_1, g_2) \in G^2$, there exists a unique value m and randomness r such that $(g_1, g_2) = C(m, r)$. This fact will help us simplify some protocols and proofs.

Finally, we naturally extend the definition to commitments on polynomials by providing a vector of commitments for the coefficients of the polynomial. For a polynomial f, we denote by $C(f, R_f)$ the commitment to f.

2.3 Shares and Sharings

In the following, we will use two kinds of secret sharings: univariate and bivariate. In both cases, the term *sharing* is used to denote a polynomial (either univariate or bivariate). The secrets are stored in the evaluations of this sharing on publicly known points. In this context, one *share* will always refer to the information held by one participant (the evaluation of the sharing on one point in the univariate setting, or a univariate polynomial in the bivariate setting). Hence, a univariate share is a point, while a bivariate share is a univariate polynomial. With these conventions and the notations of Section 2.2, the meaning of a commitment to a *share* or to a *sharing* is clear.

More precisely, when talking about univariate sharing we refer to the classical Shamir secret sharing. Thus, a sharing f of degree d for the batch of secrets $s_1, \ldots, s_\ell$ between n participants is a univariate polynomial f of degree d that satisfies $f(\beta_j) = s_j$ for all $j \in [\ell]$ and each party P_r share is the evaluation $f(\alpha_r)$ for a set of public values $\beta_1, \ldots, \beta_\ell, \alpha_1, \ldots, \alpha_n$. In that case, it can be shown that the corruption threshold for secrecy on the $s_1, \ldots, s_\ell$ is $d + 1 - \ell$.

For the bivariate sharing, we use the construction introduced in [ELL20]. A bivariate sharing g of degree d is a bivariate polynomial of degree d in both variables with $g(\beta_j, \beta_j) = s_j$ for $j \in [\ell]$. In that case, the share of the participant P_r is the univariate polynomial $g(\alpha_r, \cdot)$. For efficiency reasons in the PSS from [ELL20], it is also possible that P_r end up with the knowledge of the univariate polynomial $g(\cdot, \alpha_r)$. It was shown in [ELL20] that the corruption threshold is $d+1-\sqrt{\ell}$ for secrecy. Usually, we choose the biggest value possible for d. First, it is clear from the way the sharings are distributed that d must be smaller or equal to $n-1$. In the case of proactive secret sharing, we also require $d \leq n-2$ because the PSS functionality from [DEL$^+$16,ELL20] requires to perform regularly a `Recover` protocol where $d + 1$ participants will cooperate to recover the shares of another party. Since, there are no other constraint we usually take $d = n - 2$. In the rest of the article, we often use the fact that $d \approx n$ implicitly. When concrete security thresholds are given, either we state the formula with d or replace d by the value $n - 2$. In terms of the number of secrets ℓ, the PSS from [ELL20] requires $\ell \leq d$ and we keep this restriction in this paper.

2.4 Polynomials and Degrees of Freedom

In this section, we introduce the notion of *degree of freedom* with respect to a set of equations for a polynomial. This definition will prove useful to clarify and formalize some statements later. For the rest of this paragraph we fix f to be a polynomial of degree d (either univariate or bivariate) over a field k. We define an equation on f as an equality of the following form

$$\sum_{x \in X} f(x) = C \tag{1}$$

where X is a finite set of points ($X \subset k$ if f is univariate and $X \subset k^2$ if f is bivariate) and $C \in k$. In the special case where $X = \{x\}$, we call this the *evaluation equation* on x.

A system of equations on f is composed of several such equations as follows:

Definition 1. *A system of equations E on f is a finite set of equations*

$$E = \Big\{ \sum_{x \in X_i} f(x) = C_i \Big\}_{i \in \mathcal{I}}$$

where $X_i \subset k$ and $C_i \in k$ for all $i \in \mathcal{I} \subset \mathbb{N}$. When $\exists i$ such that $x \in X_i$ we write $f(x) \in E$.

Since polynomials of given degree d are elements of a finite vector space, it makes sense to talk about *independent equations* (in the classical sense). Hence, the dimension of a system of equations is the number of independent equations in that system. This is a terminology that we will use throughout this paper.

Definition 2. *Let E be a system of equations as per Definition 1. The degree of freedom of f with respect to E is the dimension of E subtracted from the dimension of f and is denoted by $d_f(E)$.*

In Definition 2, by dimension of f, we mean the dimension of the space in which f lives in (the dimension is $d+1$ for univariate polynomials of degree d and $(d+1)^2$ for bivariate polynomials of degree d). Another definition of the dimension could be the maximum size of an independent system of equations on f; we note that the degree of freedom is always a positive integer.

We now illustrate how this terminology helps formulate some security statements. Let us consider a univariate sharing f of m secrets and a set of corrupted parties $\{P_1, \cdots, P_t\}$ by an adversary $\mathcal{A}$. From the corruption, $\mathcal{A}$ learns the share $f(\alpha_r)$ of all corrupted parties P_r. This can be seen as a set of t equations on f. Provided, that no other equations is leaked on f, the adversary has gathered a system of t independent equations. Thus, the degree of freedom of f with respect to the system of $\mathcal{A}$ is $d + 1 - t$. We have perfect secrecy on the m secrets if m is smaller than this degree of freedom. Intuitively, this notion of degree of freedom relates to the number of secrets that can be hidden inside a polynomial. It comes especially handy when dealing with bivariate polynomials as we do in this article.

3 Proactive Secret Sharing

Constructing MPC from PSS is a natural and well-established approach. In this work, we build upon the PSS from [ELL20] to obtain efficient PMPC. Before introducing our new generic PMPC protocol (see Section 4), we need to adapt slightly the scheme from [ELL20].

In fact, the issue is not with the protocols from [ELL20] per se, but rather with the proofs and security thresholds. In the PMPC framework, operations are performed component-wise on batch of secrets, creating a sharing of $s_1 \star t_1, \ldots, s_\ell \star t_\ell$ from sharing of $s_1, \ldots, s_\ell$ and $t_1, \ldots, t_\ell$ (for the desired operation $\star$). In a generic arithmetic circuit, there is no guarantee that all the secrets are aligned before each layer of computation. That is why it is standard to use a `Permute` protocol to realign the secrets before each round. As a result, the participants will produce some sharings where secrets coming from different participants might end up in the same batch. This is why we need to ensure secrecy in the setting of a batched sharing where some of the secrets are known to the adversary. In [ELL20] where the `Share` is always performed by a single dealer and the batch of secrets are not reorganized, this situation never happens. Thus, the proofs from [ELL20] need to be updated to show that the protocols retain the desired security in this case. The updated security statements and proofs can be found in Appendix A.

In Protocol 1, we introduce an extension of the `Share` protocol to the case of multiple dealers, allowing several participants to cooperate and generate a common secret sharing of their secrets. To add more flexibility, we also make possible to add secrets in an existing sharing when the threshold for the maximal number of secrets have not been reached. This extension is quite natural given what we said above and allows us to obtain the improvement on the communication complexity due to batching even in situations where each participant has $O(1)$ secrets (which would not be possible in a single-dealer setting since each participant has to produce at least one sharing).

In the protocol below, when $\ell_1 = 0$, we assume that there is no bivariate secret sharing g. We build our `Share` protocol upon the building-block `Recover` which is part of the PSS from [ELL20]. It can be used by $d+1$ participants having shares for a bivariate sharing g of degree d to distribute a set of shares for g to another participant.

The security for Protocol 1 is stated in Lemma 1. In all the protocols in this work, we highlight the critical steps using boxes, as the full protocols includes (standard) use of commitments and openings to resist against malicious/mixed adversaries.

Protocol 1. `Share` protocol

INPUT: A subset of dealer participants $\mathcal{P}_D \subset \{P_1, \ldots, P_n\}$. A partition $\cup_{P_r \in \mathcal{P}_D} S_r$ for $\{s_{\ell_1+1}, \ldots, s_{\ell_2}\}$ where each P_r knows the elements of S_r. A

bivariate secret sharing g with commitment $C(g, R_g)$ for the batch of secrets $\{s_1, \ldots, s_{\ell_1}\}$.
OUTPUT: Distributes a bivariate sharing g' for the secrets $\{s_1, \ldots, s_{\ell_2}\}$.

1. For each $P_r \in \mathcal{P}_D$ and each $s_j \in S_r$, $\boxed{P_r \text{ samples } g_j, R_{g_j} \leftarrow \mathbb{P}_d}$ such that $\boxed{g_j(\beta_j) = s_j}$ and broadcasts the commitments $C(g_j, R_{g_j})$.
2. For all $r' \in [d+1]$, and $s_j \in S_r$ each $\boxed{P_r \in \mathcal{P}_D \text{ sends } o(g_j(\alpha_{r'}), R_{g_j}(\alpha_{r'})) \text{ to } P_{r'}}$. The receiver $P_{r'}$ broadcasts a bit indicating if the opening is correct. For each share for which an irregularity was reported, P_r broadcasts the opening. If the opening is correct, $P_{r'}$ accepts the value, otherwise P_r is disqualified and added to the set of corrupted parties B. The protocols aborts and each party outputs B.
3. Each $P_r \in \{P_1, \ldots, P_n\}$ $\boxed{\text{samples } q_{r,j}, R_{q_{r,j}} \leftarrow \mathbb{P}_d}$ with $\boxed{q_{r,j}(\beta_j) = 0}$ for all $j \in [\ell_1 + 1, \ell_2]$.
4. For all $r \in [n]$, $r' \in [d+1]$, and $s_j \in S_r$ each $\boxed{P_r \text{ sends } o(q_{r,j}(\alpha_{r'}), R_{q_{r,j}}(\alpha_{r'})) \text{ to } P_{r'}}$. The receiver $P_{r'}$ broadcasts a bit indicating if the opening is correct. For each share for which an irregularity was reported, P_r broadcasts the opening. If the opening is correct, $P_{r'}$ accepts the value, otherwise P_r is disqualified and added to the set of corrupted parties B. The protocols aborts and each party outputs B.
5. Each $P_r \in \{P_1, ..., P_{d+1}\}$ $\boxed{\text{samples } g_r, R_{g_r} \leftarrow \mathbb{P}_d}$ such that $\boxed{g_r(\beta_j) = g_j(\alpha_r) + \sum_{u=1}^{n} q_{u,j}(\alpha_r)}$ for all $j \in [\ell_1{+}1, \ell_2]$ and $\boxed{g_r(\beta_j) = g(\alpha_r, \beta_j)}$ (and the same for R_{g_r} with respect to the polynomials $R_{g_j}, R_{q_{u,j}}, R_g$) for all $j \in [\ell_1]$. P_r broadcasts the commitments g_r, R_{g_r}.
Note that this implicitly defines g' a random bivariate polynomial of degree d with $g'(\alpha_r, \cdot) = g_r(\cdot)$.
6. For all $r \in [d+1]$ and $j \in S_r$, each party locally compute the commitments $C(g_r(\beta_j), R_{g_r}(\beta_j))$, $C(g_j(\alpha_r), R_{g_j}(\alpha_r))$ and $C(q_{u,j}(\alpha_r), R_{q_{u,j}}(\alpha_r))$ for all $u \in [n]$ before verifying the relation

$$C(g_r(\beta_j), R_{g_r}(\beta_j)) = C(g_j(\alpha_r), R_{g_j}(\alpha_r)) \star_{u=1}^{n} C(q_{u,j}(\alpha_r), R_{q_{u,j}}(\alpha_r)).$$

7. For $r' \in [d+2, n]$, $\{P_1, \ldots, P_{d+1}\} \cup \{P_{r'}\}$ perform $\boxed{\texttt{Recover} \text{ on } g'}$.

We write t_P (resp. t_A) for the number of passively (resp. actively) corrupted participants. Lemma 1 informally summarizes the security of Protocol 1 but should not be considered as a formal security statement. In Appendix B, we use several such preliminary lemmas to *formally prove* Theorem 2.

Lemma 1. *(Informal) Let g be a bivariate sharing for ℓ_1 secrets $s_1, \ldots, s_{\ell_1}$ such that the adversary knows $\ell_1' \leq \ell_1$ of those shared secrets (and no other information aside from the prescribed shares) and let $s_{\ell_1+1}, \ldots, s_{\ell_2}$ be new secrets among which $\ell_2' - \ell_1 \leq \ell_2 - \ell_1$ values are known to the adversary. When $\ell_1 + \ell_2 \leq d$*

and $t_P, t_A \leq d+1-\sqrt{\ell}$*, the* `Share` *protocol above is correct and preserves secrecy of the* $(\ell_2 - \ell_2') + (\ell_1 - \ell_1')$ *secrets unknown to the adversary under the hardness of DDH. Additionally, apart from the shares of corrupted participants and the secrets already known, the adversary does not learn any other evaluation of the sharing* g'*.*

A proof for Lemma 1 can be obtained with ideas similar to the ones used in [ELL20] to prove security of their PSS scheme.

Communication Complexity: the above protocol requires $O(dn(\ell_2 - \ell_1))$ communication. Thus, it yields an amortized communication complexity of $O(n^2)$ when $d \approx n$.

4 Communication-Efficient Proactive MPC (PMPC) for Dynamic Groups with Dishonest Majorities

We follow the standard blueprint that develops PMPC based on PSS (e.g., [BELO14,EOPY18]) for arithmetic circuit. An arithmetic circuit can be divided into consecutive layers (each consisting of additions or multiplications) such that the outputs of a layer are only used once in the next layer.[7]

The outline of our PMPC protocol (Protocol 2) is similar to the one of [EOPY18]. We provide a brief summary below and refer the reader to [EOPY18] for more details. Our PMPC protocol consists of 8 sub-protocols listed below with a quick summary of their purpose. We put the tag (PSS, denoting Proactive Secret Sharing) to indicate protocols that are not introduced in this work; for those we use the construction from [ELL20].

- `Share`: Takes a batch of secrets and produces a secret sharing. (Protocol 1)
- `Refresh`: Rerandomizes a secret sharing. (PSS)
- `Recover`: Produces a share of an existing sharing. (PSS)
- `Redistribute`: Changes the number of participants for a sharing. (PSS)
- `Reconstruct`: Takes shares of a secret sharing and recovers the secrets. (PSS)
- `Add`: Performs component-wise additions of two sharings.
- `Mult:` Performs component-wise multiplications of two sharings.
- `Permute:` Takes a set of secret sharings and applies a permutation on all the secrets.

The main idea is to use secret sharings to keep the inputs private: several `Share` are performed at the beginning to create secret sharings of the inputs and all the remaining computations are performed using such secret sharings until the last layer of the circuit where `Reconstruct` is used to compute the outputs. `Refresh` and `Recover` are the two sub-protocols that make the scheme (proactively) secure against mobile adversaries. In our adversarial model, we assume

[7] Multiple uses can be handled easily by duplicating some sharings according to the circuit's requirement but we avoid them entirely to simplify the explanations

that the adversary can only change the set of corrupted participants during the `Refresh` phase. Therefore, the frequency of `Refresh` executions can be adjusted and provides a tradeoff between security and efficiency. For maximal security it can be performed after every other sub-protocol. For simplicity in Protocol 2, we refresh at every layer of the arithmetic circuit computation, and more precisely we denote R the set of layers λ after which we perform a `Refresh` operation (see Step 3e). `Recover` is used when parties are "decorrupted" and reset to a pristine default state after which they need to obtain shares to participate in the computation. The goal of `Redistribute` is to handle dynamic groups. These five protocols constitutes the PSS from [ELL20]. We extend their PSS scheme with `Add` and `Mult` protocols to evaluate the gates of the arithmetic circuits to be computed over the secret sharings. Since our PMPC protocol works with batches of secrets, we also introduce a `Permute` protocol that permutes the underlying shared secrets to align them correctly to perform `Add` and `Mult`.

To handle dynamic groups, we assume for simplicity that the dynamic changes are planned before the execution of the protocol. The `Redistribute` will be performed between consecutive layers of computations. The set $\mathcal{L}_\lambda$ is the set of leaving parties after the execution of layer λ. Similarly, $\mathcal{N}_\lambda$ is the set of new parties after layer λ. Due to the batching, there is also a need to reorder the secrets before performing the layer computation (see Appendix C). S_λ is the set of secrets after execution of layer $\lambda-1$ (i.e., after the arrival of $\mathcal{N}_\lambda$ and departure of $\mathcal{L}_\lambda$) σ_λ is the permutation to be performed on S_λ before the computation of layer λ. S_1 is just the set of inputs and σ_1 is the identity.

Protocol 2. PMPC for Dynamic Groups with Dishonest Majorities

INPUT: An arithmetic circuit C of depth d_C that has inputs $x_1, \ldots, x_n$ where each x_i is a vector of m_i values of input for the participant P_i. R is the set of layers after which a refresh phase is to be performed.

OUTPUT: $n + k$ values $y_1, \ldots, y_{n+k}$ for some $k \in \mathbb{N}$ where the total set of parties participating in the computation of C is $\{P_1, \ldots, P_{n+k}\}$. Each y_i is either the output values of the circuit Y or a special symbol $\perp$ indicating that P_i do not receive any output. We denote $\mathcal{O}$ the set of parties with non-$\perp$ output.

1. The participants label all the secrets involved in the computation and group them in batches of size ℓ. Then, the participants perform several execution of `Share` to distribute sharings of all the secrets. After this point, all the values on the input wires of C that involves participant $P_1, \ldots, P_n$ are shared among all the other parties. In particular, all the input wires of the first layer of C are shared.
2. Run the `Refresh` protocol. This corresponds to one refresh phase.
3. For each circuit layer $\lambda = 1, \ldots, d_C$:

(a) A permutation σ_λ of all the shared secrets is performed with the protocol `Permute` to align the secrets involved in all the gates of the layer λ.
(b) For each batch of addition or multiplication gates in layer λ: Compute a sharing of a batch of outputs using `Add` or `Mult`.
(c) The set of leaving participant $\mathcal{L}_\lambda$ exits the execution of the protocol using the `Redistribute` protocol. Then, the set of new parties $\mathcal{N}_\lambda$ is introduced with a new execution of `Redistribute`.
(d) All the participants may perform multiple executions of `Share` so that all the inputs of the gates of the layer $\lambda + 1$ are shared among the parties, possibly rewriting over the old secrets that will not be reused during the rest of the computation.
(e) If $\lambda \in R$, run the `Refresh` protocol.

4. At the end of the previous step, the parties are supposed to have a sharing for all the value in the output Y. We also assume that all the parties P_i with $y_i \neq \perp$ are among the set of parties at this time of the protocol. The parties in $\mathcal{O}$ perform a `Permute` protocol to regroup the output values in a set of bivariate sharings.
5. The `Reconstruct` protocol is performed several times so that all the values in Y are revealed to all $P_r \in \mathcal{O}$.

6. Two special operations may be ran during the execution of the protocol.
   - Upon receiving a message `Help!` from a party P_r, all the parties execute several times `Recover` to provide P_r sharings of all the secret values required for the later computations of the protocol. If the procedure occurs after the sharing by P_r of the value ρ_r, the other parties also reveal to P_r their share of the sharing so that P_r can compute the value ρ_r for himself.
   - If one of the participant exits the protocol without prior agreement, the remaining parties perform `Redistribute` in the corrupted mode to distribute all secrets between them with a proper sharing.

Remark 1. For easy of exposition, we assumed that all the outputs (represented by the set Y) are revealed to all the participants in $\mathcal{O}$. In reality, the protocols often require that each participant obtain a different output. We can apply standard techniques to modify Protocol 2 in order to handle this functionality. For a given sharing containing several outputs that are to be revealed to different participants it suffices that each of these participants generate a new sharing of zeroes and a random value at the index of the desired output. Then, each of these sharings can be added to the initial sharing with `Add`. Finally, the participants perform `Reconstruct` on the sharing obtained after all the operations. Thus, the participants will learn values $s_1 + \rho_1, \ldots, s_\ell + \rho_\ell$ where the ρ_i are the random values. The participants that is supposed to get the value s_i will be able ρ_i since he was the one to generate it, but the other participants will not learn anything on s_i.

In Appendix D, we prove Theorem 1, which shows that Protocol 2 securely realizes the ideal functionality $\text{IDEAL}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{PMPC}}$ (defined in Appendix D). The thresholds are computed by taking $d = n - 2$ (which is the maximum possible value).

Theorem 1. *For a circuit $\mathcal{C}$ where the minimum number of participants is n_0. When the size batch is $\ell = n_0 - 2$, and assuming the hardness of DDH, the protocol Π^{PMPC} introduced in Protocol 2 securely realizes the ideal process $\text{IDEAL}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{PMPC}}_{unfair}$ against any adversary bounded at any given time by the multi-threshold $T(\ell) = \left\{(n-3-\sqrt{\ell}, n-3-\sqrt{\ell})\right\}$ with n the number of participants at that time. Additionally, when the adversary is also bounded by the multi-threshold $T_f(\ell) = \{(k, \min(n-k-\sqrt{\ell}, 2(n-k)-\ell, 1 \leq k \leq n/3\}$, Π^{PMPC} securely realizes $\text{IDEAL}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{PMPC}}_{fair}$.*

5 Subprotocols for PMPC

In Sections 5.1 to 5.4, we introduce the sub-protocols used in the `Mult` protocol described at a high level in Section 1.2. Next, we introduce the full multiplication protocol in Section 5.5. We defer the treatment of reordering the secrets in Appendix C as it follows essentially from previous work. The security statements can be found in the Appendix.

5.1 Bivariate to Univariate Sharing

The subprotocol from this section efficiently transforms a (bivariate) secret sharing of a batch of ℓ secrets $s_1, \ldots, s_\ell$ into several univariate sharings of smaller size. For simplicity, we treat the case where $\ell = vm$ and construct v univariate sharings of m secrets each. This procedure is described in Protocol 3.

We denote g the bivariate sharing of degree d and $(f_k)_{k\in[v]}$ the univariate sharings of degree d. We write $I_k = [(k-1)m + 1, km]$ and each f_k will be a sharing for s_j for all $j \in I_k$. We also denote $\Pi_r(x) = \prod_{\substack{u\in[d+1]\\u\neq r}} \frac{x-\alpha_u}{\alpha_r-\alpha_u}$ for the Lagrange polynomial.

Protocol 3. `ReshareBivariateToUnivariate`

INPUT: A set $\mathcal{P} = \{P_1, \ldots, P_n\}$ holding a bivariate sharing for a batch of $\ell = mv$ secrets $s_1 = g(\beta_1, \beta_1), \ldots, s_\ell = g(\beta_\ell, \beta_\ell)$ and the commitment to this sharing $C(g, R_g)$.

OUTPUT: For all $k \in [v]$, each party P_r holds its share of the univariate sharing for the batch of secrets $\{s_j\}_{j\in I_k}$ along with a commitment $C(f_k, R_{f_k})$ to that sharing.

1. For $k \in [v]$:

(a) For $r \in [d+1]$, P_r samples polynomials $\boxed{q_r, R_{q_r} \leftarrow \mathbb{P}_d}$ such that

$$g(\alpha_r, \beta_j)\Pi_r(\beta_j) = q_r(\beta_j)\,, \quad R_g(\alpha_r, \beta_j)\Pi_r(\beta_j) = R_{q_r}(\beta_j)\,, \quad \forall j \in I_k\,.$$

P_r broadcast $C(q_r, R_{q_r})$.

(b) For all $j \in I_k$, each participant computes locally $C(q_r(\beta_j), R_{q_r}(\beta_j))$ and $C(g(\alpha_r, \beta_j), R_g(\alpha_r, \beta_j))$ and verifies that

$$C(q_r(\beta_j), R_{q_r}(\beta_j)) = \Pi_r(\beta_j) \cdot C(g(\alpha_r, \beta_j), R_g(\alpha_r, \beta_j))$$

If the verification fails, P_r is added to the list of corrupted participant B. The protocol aborts and each participant outputs B.

(c) For $r \in [d+1]$ and $r' \in [n]$, P_r sends $\boxed{o(q_r(\alpha_{r'}), R_{q_r}(\alpha_{r'})) \text{ to } P_{r'}}$. The receiver $P_{r'}$ broadcasts a bit indicating if the opening is correct. For each share for which an irregularity was reported, P_r broadcasts the opening. If the opening is correct, $P_{r'}$ accepts the value, otherwise P_r is disqualified and added to the set of corrupted parties B. The protocols aborts and each party outputs B.

(d) Each party P_r sets its share as $\boxed{f_k(\alpha_r) = \sum_{u=1}^{d+1} q_u(\alpha_r)}$ and locally compute the commitment to f_k.

5.2 Blinding Bivariate Mask Generation

The goal of the `BlindingBivariateGeneration` protocol is for the participants to share and generate a bivariate sharing h of the m values $0, \ldots, 0$. Hence, it will verify $h(\beta_j, \beta_j) = 0$ for $j \in I$, where $I \subset [\ell]$ has size m (typically the I_k defined for Protocol 3). This sharing will be used to blind some values during the multiplication protocol and that is why we will sometimes call this sharing a blinding mask. As this protocol is designed to be used during the multiplication, we need that h verifies some very specific conditions on its values and their distribution to the participants. This condition depends on some threshold t and is very ad hoc but will allow us to quantify the "amount of randomness" we need from the blinding mask. We write $C_{\texttt{mult}}(t)$ this condition. For a subset $T \subset [n]$, we write E_T the biggest system of independent equations gathered by an adversary $\mathcal{A}$ corrupting each P_r for $r \in T$. The condition $C_{\texttt{mult}}(t)$ can be expressed as follows:

Definition 3. *A bivariate sharing h of degree d for m secrets is said to satisfy the condition $C_{\texttt{mult}}(t)$ for $t \leq n$ if for every subset $T \subset [n]$ of size t:*

- *For any $u \notin T$, the degree of freedom of $h(\alpha_u, \cdot)$ with respect to the adversary's knowledge is $d + 2 - m$.*
- *For a system of equation E on h, let us write $X_E = \{x, \exists y \text{ such that } h(x, y) \in E\}$. For any $u \notin T$, any value z and any system of equations E such that $\#X_E \leq d - t$ and $\alpha_u \notin X_E$, the evaluation equation of $h(\alpha_u, z)$ is independent of $E \cup E_T$.*

Protocol 4. `BlindingBivariateGeneration`

INPUT: A set of index $I \subset [\ell]$ of size m.
OUTPUT: A bivariate sharing of $0, \ldots, 0$ at the points $(\beta_j)_{j \in I}$ is distributed to the parties along with the commitment to this sharing.

1. For $r \in [n]$, P_r samples $q_r, R_{q_r} \leftarrow \mathbb{P}_d$ such that $q_r(\beta_j) = 0$ for $j \in I$ and broadcast the commitment $C(q_r, R_{q_r})$.
2. For $j \in I$, each P_r broadcasts the openings $o(0, R_{q_r}(\beta_j))$. If one of the opening is not correct, the corresponding participant is added to the list of corrupted participant B.
3. For $r \in [n]$, $r' \in [n]$, P_r sends to $P_{r'}$ the opening $o(q_r(\alpha_{r'}), R_{q_r}(\alpha_r))$. If the opening is not correct, the corresponding participant is added to the list of corrupted participant B.
4. For $r' \in [n]$, $P_{r'}$ computes $q(\alpha_{r'}) = \sum_{r=1}^{n} q_r(\alpha_{r'})$, $R_q(\alpha_{r'}) = \sum_{r=1}^{n} R_{q_r}(\alpha_{r'})$, and locally compute the commitment $C(q, R_q)$.
5. For $r \in [d+1]$, P_r samples $h(\alpha_r, \cdot), R_h(\alpha_r, \cdot) \leftarrow \mathbb{P}_d$ such that $h(\alpha_r, \beta_j) = q(\alpha_r)$, $R_h(\alpha_r, \beta_j) = R_q(\alpha_r)$ and broadcast the commitment $C(h(\alpha_r, \cdot), R_h(\alpha_r, \cdot))$.
   (*Note that this implicitly defines random bivariate polynomials h, R_h of degree d.*)
6. For $j \in I$ and $r \in [d+1]$, parties locally compute $C(h(\alpha_r, \beta_j), R_h(\alpha_r, \beta_j))$, $C(q(\alpha_r), R_q(\alpha_r))$ and check if $C(h(\alpha_r, \beta_j), R_h(\alpha_r, \beta_j)) = C(q(\alpha_r), R_q(\alpha_r))$. If one commitment does not satisfy the equation, the corresponding participant is added to the list of corrupted participant B.
7. For $r' \in [d+2, n]$, $\{P_1, \ldots, P_{d+1}\} \cup \{P_{r'}\}$ perform `Recover` on h.

5.3 Bivariate Product

The `BivariateProduct` protocol aims at creating a bivariate sharing $g^\star$ for the multiplication of the secrets contained in two univariate sharings f, f' under the blinding bivariate mask h. To distribute $g^\star(x, y) = f(x)f'(y) + h(x, y)$, each pair of participants is going to interact. The core of this exchange is a zero-knowledge multiplication protocol, denoted `ZK-Mult`, described in Appendix E.1 and based on [LN18, Section 6.2]. When composed with the key generation KeyGen of the Paillier encryption, this protocol `ZK-mult` securely realizes the $\mathcal{F}_{prod}$ functionality (see Fig. 1). The protocol is performed by two participants P_1, P_2 where the input for P_1 is a key pair for the Paillier encryption scheme and a value x while P_2 has input the corresponding public key and two values y, δ. At the end of the protocol, P_1 learns $x \cdot y + \delta$. This subprotocol will be repeated to compute product of polynomials.

Figure 1. Functionality $\mathcal{F}_{prod}$

$\mathcal{F}_{prod}$ interacts with two parties P_1 and P_2.
Upon reception of x from P_1 and y, δ from P_2, $\mathcal{F}_{prod}$ sends $xy + \delta$ to P_1 and the special symbol $\perp$ to P_2.

Protocol 5. `BivariateProduct`

INPUT: A set I of indices of size m and a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of participants holding two univariate sharings f, f' to batches $(s_i)_{i \in I}$, $(s'_i)_{i \in I}$ of m secrets and one bivariate sharing h to m zeroes and the commitments to these sharings $C(f, R_f), C(f', R_{f'}), C(h, R_h)$.
OUTPUT: The bivariate sharing of $g^\star(x, y) = f(x)f'(y) + h(y, x)$ and the commitment $C(g^\star, R_{g^\star})$ (with $R_{g^\star}(x, y) = f(x)R_{f'}(y) + R_h(y, x)$).

1. For all $r \in [n]$ and $r' \in [d+1]$,
   (a) P_r samples $(pk_r, sk_r), (pk_{r,R}, sk_{r,R}) \leftarrow \mathsf{KeyGen}$, produces two NIZK proofs of key generation and broadcasts all the public keys and corresponding proofs.
   (b) $P_{r'}$ verifies the proofs and if one of the verification fails, P_r is added to the set of corrupted participants.
   (c) $\boxed{P_r \text{ and } P_{r'} \text{ perform } \texttt{ZK-Mult}}$ on inputs $pk_r, sk_r, f(\alpha_r)$ for P_r and $pk_r, f'(\alpha_{r'}), h(\alpha_{r'}, \alpha_r)$ for $P_{r'}$. $\boxed{P_r \text{ obtains output } g^\star(\alpha_r, \alpha_{r'})}$.
   (d) $\boxed{P_r \text{ and } P_{r'} \text{ perform } \texttt{ZK-Mult}}$ on inputs $pk_{r,R}, sk_{r,R}, f(\alpha_r)$ for P_r and $pk_{r,R}, R_{f'}(\alpha_{r'}), R_h(\alpha_{r'}, \alpha_r)$ for $P_{r'}$. $\boxed{P_r \text{ obtains output } R_{g^\star}(\alpha_r, \alpha_{r'})}$.
   (e) P_r computes locally the commitments $C(f'(\alpha_{r'}), R_{f'}(\alpha_{r'}))$ and $C(h(\alpha_{r'}, \alpha_r), R_h(\alpha_{r'}, \alpha_r))$ and checks that

$$C(g^\star(\alpha_r, \alpha_{r'}), R_{g^\star}(\alpha_r, \alpha_{r'})) =$$

$$\Big(f(\alpha_r) \cdot C(f'(\alpha_{r'}, R_{f'}(\alpha_{r'}))\Big) \star C(h(\alpha_{r'}, \alpha_r), R_h(\alpha_{r'}, \alpha_r))$$

   If the verification fails, $P_{r'}$ is added to the list of corrupted participants B.
2. $\boxed{P_r \text{ broadcast}}$ the commitment $\boxed{C(g^\star(\alpha_r, \cdot), R_{g^\star}(\alpha_r, \cdot))}$ for all $r \in [d+1]$.

5.4 Random Evaluation for Commitment Verification

The goal of Protocol 6 is to generate a random value `rand` used to ensure correctness of the shared commitments in Protocol 5 by revealing $f(\texttt{rand})$ in clear.

Protocol 6. `CommitmentVerification`

INPUT: Two univariate sharings f, f' and one bivariate sharings h distributed among a set $\mathcal{P} = \{P_1, \ldots, P_n\}$. Another bivariate sharing $g^\star(x, y) = f(x)f'(y) + h(y, x)$ distributed among the participant along with the commitment $C(g, R_g)$ for some bivariate polynomials g, R_g
OUTPUT: A bit $b \in \{0, 1\}$.

1. For $r \in [n]$, $\boxed{P_r \text{ samples } v_r, R_{v_r} \leftarrow \mathbb{F}}$ and broadcast $C(v_r, R_{vr})$.
2. For $r \in [n]$, $\boxed{P_r \text{ broadcasts } o(v_r, R_{v_r})}$. If the verification fails, P_r is added to the list of corrupted participant B.
   Then, every party computes $\boxed{\texttt{rand} = \sum_{u=1}^{n} v_u}$. If `rand` is any values of the protocol (α_r or β_j), go back to step 1.
3. For $r \in [n]$, $\boxed{P_r \text{ samples } \hat{f}_r, R_{\hat{f}_r} \leftarrow \mathbb{P}_d}$ such that $\boxed{\hat{f}_r(\texttt{rand}) = 0}$ and broadcasts $C(\hat{f}_r, R_{\hat{f}_r})$.
4. For $r \in [n]$, $\boxed{P_r \text{ sends to all } P_{r'}}$ the opening to $\boxed{o(\hat{f}_r(\alpha_{r'}), R_{\hat{f}_r}(\alpha_{r'}))}$. The receiver P_r broadcasts a bit indicating if the opening is correct. For each share for which an irregularity was reported,P_r broadcasts the opening. If the opening is correct, $P_{r'}$ accepts the value, otherwise P_r is disqualified and added to the set of corrupted parties B. The protocols aborts and each party outputs B.
5. Each P_r broadcasts the openings $o(0, R_{\hat{f}_r}(\texttt{rand}))$. If one of the opening is not correct, the corresponding participant is added to the list of corrupted participant B.
6. Set $\hat{f} = f + \sum_{u=1}^{n} \hat{f}_u$ and $R_{\hat{f}} = R_f + \sum_{u=1}^{n} R_{\hat{f}_u}$. For $r \in [n]$,
   $\boxed{P_r \text{ broadcasts the opening } o(\hat{f}(\alpha_r), R_{\hat{f}}(\alpha_r))}$.
7. For all $r \in [n]$, each party computes locally $C(\hat{f}(\alpha_r), R_{\hat{f}}(\alpha_r))$ and verifies that the opening is correct. If the verification fails, P_r is added to the list of corrupted participant B.
8. Each party computes the values $\boxed{\hat{f}(\texttt{rand}) \text{ by interpolation}}$.
9. For all $r' \in [d+1]$, each participants computes locally the commitments $C(f'(\alpha_{r'}, R_{f'}(\alpha_{r'}), C(h(\alpha_{r'}, \alpha_r), R_h(\alpha_{r'}, \alpha_r))$ and $C(g(\texttt{rand}, \alpha_{r'}), R_g(\texttt{rand}, \alpha_{r'}))$ and checks that

$$C(g(\texttt{rand}, \alpha_{r'}), R_g(\texttt{rand}, \alpha_{r'})) =$$

$$\Big(f(\texttt{rand}) \ \cdot \ C(f'(\alpha_{r'}, R_{f'}(\alpha_{r'}))\Big) \ \star \ C(h(\alpha_{r'}, \alpha_r), R_h(\alpha_{r'}, \alpha_r))$$

   If one of the verifications fails the participants broadcasts a 0, otherwise it broadcasts a 1.

10. If one 0 was broadcasted in the previous step, the protocols outputs $b = 0$, otherwise it outputs $b = 1$.

5.5 The Multiplication Protocol

We are now ready to introduce the whole protocol performing the multiplication of two bivariate sharings g, g' containing ℓ secrets each. The output g'' is a bivariate sharing for the ℓ pair-wise products of secrets in g, g'. We have also a value m which must respect the bound $m \leq d - t_P$ to obtain security. For simplicity we assume that $\ell = mv$. The generalization for any ℓ is straightforward.

Protocol 7. `Mult`

INPUT: A set $\mathcal{P} = \{P_1, \ldots, P_n\}$ holding two bivariate sharings g, g' for two batches of $\ell = mv$ secrets $s_1, \ldots, s_\ell$ and $s'_1, \ldots, s'_\ell$ and the commitment to these sharings $C(g, R_g), C(g', R_{g'})$.

OUTPUT: Each party holds its shares for the bivariate sharing g'' for the batch of ℓ secrets $s_1 s'_1, \ldots, s_\ell s'_\ell$ along with the commitment $C(g'', R_{g''})$

1. The participants perform `ReshareBivariateToUnivariate` on g and g'. We write $(f_k)_{k\in[v]}$ and $(f'_k)_{k\in[v]}$ the outputs of the two executions.
2. For $k \in [v]$,
   (a) Set $I_k = [1+(k-1)m, km]$ and execute `BlindingBivariateGeneration` on input I_k. We write h_k the bivariate sharing obtained in output.
   (b) Participants perform `BivariateProduct` on the sharings f_k, f'_k, h_k and set I_k to obtain $g^\star_k$ a bivariate sharing with a bivariate commitment $C(g_k, R_{g_k})$.
   (c) Participants execute `CommitmentVerfication` on sharings $f_k, f'_k, h_k, g^\star_k$. If the output is 0, the protocol aborts and each party outputs B the set of corrupted participants.
   (d) P_r interpolates $g^\star_k(\alpha_r, \beta_j)$ for $j \in I_k$.
3. For $r \in [d+1]$, P_r samples $g''(\alpha_r, \cdot), R_{g''}(\alpha_r, \cdot) \leftarrow \mathbb{P}_d$ with $g''(\alpha_r, \beta_j) = g^\star_k(\alpha_r, \beta_j)$ and $R_{g''}(\alpha_r, \beta_j) = R_{g^\star_k}(\alpha_r, \beta_j)$ for all $k \in [v]$ and $j \in I_k$ and broadcast the commitment $C(g'', R_{g''})$.
   (*Note that this implicitly defines a bivariate polynomial g'' of degree d.*)
4. For all $r \in [d+1]$, each participant $P_{r'}$ computes $C(g_k(\alpha_r, \beta_j), R_{g_k}(\alpha_r, \beta_j))$ and $C(g''(\alpha_r, \beta_j), R_{g''}(\alpha_r, \beta_j))$ and checks equality for all $k \in [v]$ and $j \in I_k$. If one of the verification fails for index r, P_r is added to the set of corrupted participants B. The protocol aborts and each participant outputs B.
5. For $r' \in [d+2, n]$, $\{P_1, \ldots, P_{d+1}\} \cup \{P_{r'}\}$ perform `Recover` on g''.

Communication complexity: The complexity of `ReshareBivariateToUnivariate` is $O(n^2 v)$ and $O(n^2)$ for `BlindingBivariateGeneration`, `BivariateProduct`

and `CommitmentVerification`. Thus, the overall complexity of the two first steps is $O(n^2 v)$. The final `Recover` step is performed in $O(n^2)$ when $n-d = O(1)$. Overall, this is $O(n^2 v)$ and $O(n^2/m)$ amortized. When $\ell = n - 2$ and $m \approx \sqrt{\ell}$, we obtain the claimed amortized complexity of $O(n\sqrt{n})$.

Theorem 2. *When $\ell \leq d$ and $m = \lceil \sqrt{\ell} \rceil$ and assuming the hardness of DDH, the protocol Π^{MULT} introduced in Protocol 7 securely realizes the ideal process $\text{IDEAL}_{mixed}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{MULT,\ell}}$ against any adversary bounded by the multi-threshold $T(\ell)$ where we have $T(\ell) = \left\{(d-1-\sqrt{\ell}, d-1-\sqrt{\ell})\right\}$.*

The ideal functionality $\text{IDEAL}_{mixed}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{MULT,\ell}}$ and the proof of Theorem 2 can be found in Appendix B.

References

BELO14. Joshua Baron, Karim Eldefrawy, Joshua Lampkins, and Rafail Ostrovsky. How to withstand mobile virus attacks, revisited. In *PODC*, pages 293–302. ACM, 2014.

BELO15. Joshua Baron, Karim Eldefrawy, Joshua Lampkins, and Rafail Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In *ACNS*, volume 9092 of *LNCS*, pages 23–41. Springer, 2015.

Ben64. Václav E. Beneš. Optimal rearrangeable multistage connecting networks. *The Bell System Technical Journal*, 43(4):1641–1656, July, 1964.

BGRS18. F Baldimtsi, S Goldberg, L Reyzin, and O Sagga. Certifying rsa public keys with an efficient nizk. Cryptology ePrint Archive, Report 2018/057, 2018.

DEL$^+$16. Shlomi Dolev, Karim Eldefrawy, Joshua Lampkins, Rafail Ostrovsky, and Moti Yung. Proactive secret sharing with a dishonest majority. In *SCN*, volume 9841 of *LNCS*, pages 529–548. Springer, 2016.

ELL20. Karim Eldefrawy, Tancrède Lepoint, and Antonin Leroux. Communication-efficient proactive secret sharing for dynamic groups with dishonest majorities. In *ACNS (1)*, volume 12146 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2020.

EOPY18. Karim Eldefrawy, Rafail Ostrovsky, Sunoo Park, and Moti Yung. Proactive secure multiparty computation with a dishonest majority. In *SCN*, volume 11035 of *LNCS*, pages 200–215. Springer, 2018.

GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT*, pages 465–482, 2012.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *STOC*, pages 218–229. ACM, 1987.

HJKY95. Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO*, pages 339–352, 1995.

HLM13. Martin Hirt, Christoph Lucas, and Ueli Maurer. A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In *CRYPTO (2)*, volume 8043 of *LNCS*, pages 203–219. Springer, 2013.

LN18. Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *ACM Conference on Computer and Communications Security*, pages 1837–1854. ACM, 2018.

OY91. Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *PODC*, pages 51–59. ACM, 1991.

Ped91. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

Wak68. Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, 1968.

A Proofs of PSS

In this section, we present the updated proofs for the sub-protocols introduced in [ELL20] in the setting where the adversary might know some of the secrets given in the sharing.

A.1 Recover

Protocol 8. `Recover`

INPUT: A set $\mathcal{P} = \{P_1, \ldots, P_{d+1}\}$ with respective shares $\{g(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}$ and a recovering party P_{r_C}.
OUTPUT: Each party P_r for $r \in [d+1] \cup \{r_C\}$ obtains $\{g'(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}$, where $g'(\beta_j, \beta_j) = g(\beta_j, \beta_j)$ for all $j \in [\ell]$.

1. For $r \in [d+1]$, P_r broadcasts the commitments to $\{g(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}$. Each broadcast commitment consistency is locally verified; if consistency fails, P_r broadcasts a complaining bit and the protocol aborts.
2. For $r \in [d+1]$, $\boxed{P_r \text{ samples } f_r \leftarrow \mathbb{P}_d}$ such that $\boxed{f_r(\alpha_{r_C}) = 0}$, then broadcasts commitments of $f_r(\alpha_{r'})$ for all $r' \in [d+1]$, and then $\boxed{\text{sends}}$ an opening to the commitment of $\boxed{f_r(\alpha_{r'}) \text{ to each } P_{r'}}$.
3. Each party verifies that $f_{r'}(\alpha_{r_C})$ opens to 0 for every $r' \in [d+1]$. When the opening fails, $P_{r'}$ is disqualified and added to the set of corrupted parties B, and the protocol aborts and each party outputs B.
4. For $r \in [d+1]$, $\boxed{P_r \text{ locally computes } f_{r'}(\alpha_r), r' \in [d+1]}$ and broadcasts a complaining bit indicating if the opening is correct. For each share $f_{r'}(\alpha_r)$, for which an irregularity was reported, $P_{r'}$ broadcasts the opening. If the opening is correct, P_r accepts the value, otherwise $P_{r'}$ is disqualified and added to the set of corrupted parties B. The protocols aborts and each party outputs B.
5. For $r \in [d+1]$, $\boxed{P_r \text{ sends to } P_{r_C}}$ openings to the values $\boxed{g(\alpha_r, \alpha_{r'}) + f_{r'}(\alpha_r)}$ for all $r' \in [d+1]$. P_{r_C} is able to compute locally a commitment to the values $g(\alpha_r, \alpha_{r'}) + f_{r'}(\alpha_r)$ and for each r' broadcasts a bit indicating if the opening was correct.
6. For each share $g(\alpha_r, \alpha_{r'}) + f_{r'}(\alpha_r)$, for which an irregularity was reported, P_r broadcasts the opening. If the opening is correct, P_{r_C} accepts the value, otherwise P_r is disqualified and added to the set of corrupted parties B. The protocols aborts and each party outputs B.
7. $\boxed{P_{r_C} \text{ locally interpolates } g(\alpha_{r_C}, \alpha_{r'})}$ for all $r' \in [d+1]$.

Lemma 2. *(Informal) When $\ell \leq d$ and the adversary knows $\ell' \leq \ell$ secrets in the sharing g in input,* `Recover` *preserves secrecy on the $\ell - \ell'$ secrets when $(t_P, t_A) \leq \{(d+1-\sqrt{\ell}, d+1-\sqrt{\ell})\}$. The only information obtained by the*

adversary on the sharing are the shares of the corrupted participants and the leaked secrets.

Proof. As can easily be seen, the critical case is when P_{r_C} is corrupted. We set $T = [t_P - 1]$, wlog we can assume the set of indices of the corrupted parties is obtained as $\{r_C\} \cup T$. As part of the bivariate sharing distribution, we can assume that the adversary already knows $g(\alpha_r, \cdot)$, $g(\cdot, \alpha_r)$ for corrupted P_r and $g(\beta_j, \beta_j)$ for $j \in [\ell']$. In this setting, we know from the proofs of the original PSS in [ELL20] that we have information theoretic security on the $\ell - \ell'$ other secrets.

We need to verify that the equations $g(\alpha_r, \alpha_u) + f_u(\alpha_r)$ seen by the adversary for $r, u \in [d+1]^2$ are not sufficient to learn any new evaluations of g.

When either r or u is in T, the adversary already knows both $g(\alpha_r, \alpha_u)$ and $f_u(\alpha_r)$. This is also true for $g(\alpha_{r_C}, \alpha_u)$ and $f_u(\alpha_{r_C}) = 0$. We write E_0 the set of all these evaluation equations. For the adversary, the set of new interesting equations are the $g(\alpha_r, \alpha_u) + f_u(\alpha_r)$ where both P_r and P_u are honest. Let us write E this set of $(d + 2 - t_P)^2$ equations.

For each u, the interpolation formula allows to express $g(\alpha_{r_C}, \alpha_u) + f_u(\alpha_{r_C}) = \sum_{r=1}^{d+1} \Pi_r(\alpha_{r_C})(g(\alpha_r, \alpha_u) + f_u(\alpha_r))$. Rewriting this equation gives $g(\alpha_{d+1}, \alpha_u) + f_u(\alpha_{d+1}) = \frac{1}{\Pi_{d+1}(\alpha_{r_C})}(g(\alpha_{r_C}, \alpha_u) + f_u(\alpha_{r_C}) - \sum_{r=1}^{d} \Pi_r(\alpha_{r_C})(g(\alpha_r, \alpha_u) + f_u(\alpha_r)))$. This proves that for all $u \in [t_P, d+1]$, the evaluation equation on $g(\alpha_{d+1}, \alpha_u) + f_u(\alpha_{d+1})$ can be derived from the equations in E_0 and the other equations in E. Thus, we can reduce the set of independent equations inside E to be $g(\alpha_r, \alpha_u) + f_u(\alpha_r)$ with $u \in [t_P, d+1]$ and $r \in [t_P, d]$. The degree of freedom of each f_u with respect to E_0 is $d+1-(t_P-1)-1$ for $u \in [t_P, d+1]$. With that, we get that the degree of freedom of $g(\alpha_r, \cdot)$ is $d + 1 - t_P$ for $r \in [t_P, d]$ with respect to $E_0 \cup E$.

In the end, if we take the $d + 1$ participants $\{P_1, \ldots, P_d\} \cup \{P_{r_C}\}$, we see that we obtain the usual distribution of a bivariate secret sharing between $d+1$ parties among which t_P are corrupted. In this case, we know there is information theoretic security on the $\ell - \ell'$ secrets previously unknown to the adversary.

A.2 Reconstruct

Up to this point, we have shown that the additional information of several secrets leaked to the adversary was not a threat to the security of our scheme. Unfortunately that is not the case of the fairness of the `Reconstruct` protocol. Fairness is the property that the adversary cannot deny the output to the honest participants and still compute the output for himself. Against active adversary, it was shown that fairness cannot be obtained in dishonest majority. However, Hirt, Lucas, and Maurer [HLM13] introduced the gradual sharing setting where fairness is obtained when $t_A \leq n/2$ and $t_P \leq n - t_A$. In [ELL20], this idea was adapted to their bivariate sharing to obtain a gradual threshold of $t_A \leq n/2$ and $t_P \leq n - \sqrt{ell} - t_A$. In Lemma 3, we show that the gradual threshold is further decreased to $t_P \leq \min(n - t_A - \sqrt{\ell}, 2(n - t_A) - \ell)$. Usually we have $n \approx \ell$ and so the threshold becomes $t_P \leq \min(n - t_A - \sqrt{n}, n - 2t_A)$. Since $t_A \leq t_P$ we must

have $t_A \leq n - 2t_A \Rightarrow t_A \leq n/3$. While this reduces the range of threshold we can tolerate, this is only affecting the fairness threshold (for which we couldn't handle an active majority anyway).

Protocol 9. `Reconstruct`

INPUT: A set $\mathcal{P} = \{P_1, \ldots, P_n\}$ with respective shares $\{g(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}$. A (public) set of nonzero values $(\lambda_k)_{1 \leq k \leq d}$ such that $\lambda_1 + \cdots + \lambda_d = 0$ and $\lambda_1 + \ldots + \lambda_i \neq 0$ for all $i < d$.
OUTPUT: Values $s_j = g(\beta_j, \beta_j)$ for $j \in [\ell]$ to all parties in $\mathcal{P}$.

1. *Initialization*: Set $B = \emptyset$,$i = d$ and the number of remaining parties as $N = n$. Each party in $\mathcal{P}$ sets locally $\boxed{s_j = 0 \text{ for all } j \in [\ell]}$.
2. *First step* $(i = d)$:
   (a) Without loss of generality, assume $\mathcal{P} = \{P_1, \ldots, P_N\}$.
   For $r \in [d]$, $\boxed{P_r \text{ samples } Q_{d-1}(\alpha_r, \cdot) \leftarrow \mathbb{P}_{d-1}}$ and broadcast commitments to $\{Q_{d-1}(\alpha_r, \alpha_{r'})\}_{r' \in [d]}$.
   Note that this implicitly defines Q_{d-1} a random bivariate polynomial of degree $d-1$.
   (b) $\boxed{\text{Using } \texttt{Recover}, P_1, \ldots, P_d \text{ reveal } \{Q_{d-1}(\alpha_{d+1}, \alpha_{r'})\}_{r' \in [d+1]}}$ to P_{d+1}.
   If `Recover` aborts with output B', sets $B = B \cup B'$, $N = N - |B'|$ and $\mathcal{P} = \mathcal{P} \setminus B'$. If $N > d$, go to step (a), otherwise the protocol aborts and outputs B.
   (c) Denote $\boxed{g_d = g + \lambda_d Q_{d-1}}$. For $r \in [d+1]$, P_r $\boxed{\text{locally updates}}$ their shares to $\boxed{\{g_d(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}}$ using the $Q_{d-1}(\alpha_r, \alpha_{r'})$'s, and broadcasts commitments thereof.
3. *Gradual Reconstruction*: $\boxed{\text{While } i \geq 2}$:
   (a) Wlog, assume $\mathcal{P} = \{P_1, \ldots, P_N\}$. For $r \in [i+1]$, P_r $\boxed{\text{broadcasts}}$ openings to $\boxed{\{g_i(\alpha_r, \alpha_{r'})\}_{r' \in [i+1]}}$, and all parties locally verify the openings. Let B' denote the parties with incorrect openings. Each party sets $B = B \cup B'$, $N = N - |B'|$ and $\mathcal{P} = \mathcal{P} \setminus B'$. If $N > i$, go the step (b), otherwise the protocol aborts and outputs B.
   (b) $\boxed{\text{For } r \in [i+1, N], P_r \text{ interpolates its shares}}$ $\{g_i(\alpha_r, \alpha_{r'})\}_{r' \in [i+1]}$.
   Then, computes the values $\{Q_{i-1}(\alpha_r, \alpha_{r'})\}_{r' \in [i]}$.
   Note that we have the invariant $g_i + \cdots + g_d = g + (\lambda_d + \cdots + \lambda_i) Q_{i-1}$.
   (c) $\boxed{\text{All parties interpolate } g_i}$ and update $\boxed{s_j \leftarrow s_j + g_i(\beta_j, \beta_j)}$.
   $\boxed{\text{Set } i \leftarrow i - 1.}$
   (d) If $i = 1$, sets $Q_0 = 0$ and go to Step (f).
   Else, for $r \in [i]$, $\boxed{P_r \text{ samples } Q_{i-1}(\alpha_r, \cdot) \leftarrow \mathbb{P}_{i-1}}$ and broadcast commitments to $\{Q_{i-1}(\alpha_r, \alpha_{r'})\}_{r' \in [i+1]}$.

Note that this implicitly defines Q_{i-1} a random bivariate polynomial of degree $i-1$.

(e) Using `Recover`, $P_1, \ldots, P_i$ enable P_{i+1} to obtain evaluations of $\{Q_{i-1}(\alpha_r, \alpha_{r'})\}_{r' \in [i+1]}$. If `Recover` aborts with output B', sets $B = B \cup B'$, $N = N - |B'|$ and $\mathcal{P} = \mathcal{P} \setminus B'$. If $N > i$, go to step (d), otherwise the protocol aborts and outputs B.

(f) Denote $g_i = \lambda_i Q_i + \left(\sum_{k=1}^{i-1} \lambda_k\right) \cdot (Q_i - Q_{i-1})$. For $r \in [i+1]$, P_r locally updates its shares to $\{g_i(\alpha_r, \alpha_{r'})\}_{r' \in [i+1]}$ and broadcast commitments to these values.

4. *Last Step* ($i = 1$):
Wlog, assume $\mathcal{P} = \{P_1, \ldots, P_N\}$. Each party $P_r \in \mathcal{P}$ broadcasts openings to $g_1(\alpha_r, \alpha_1)$ and $g_1(\alpha_r, \alpha_2)$. If there are at least 2 correct set of openings, all parties compute $g_1(\beta_j, \beta_j)$ for all $j \in [\ell]$ and set $s_j \leftarrow s_j + g_1(\beta_j, \beta_j)$; otherwise the protocol aborts.

Lemma 3. *(Informal) If $\ell \leq d$ and $t_P \leq \min(n - t_A - \sqrt{\ell}, 2(n - t_A) - \ell)$, then the `Reconstruct` protocol provides information theoretic security on the $\ell - \ell'$ secrets unknown to the adversary when the protocol aborts.*

Proof. Fairness

First, we stress that B is contained in the set of actively corrupted participants. Thus, we have $|B| \leq t_A$ and $t_P \leq \min(n - t_A - \sqrt{\ell}, 2(n - t_A) - \ell) \leq \min(n - |B| - \ell, 2(n - |B|) - \ell)$. The correctness of `Reconstruct` relies on the invariant $g_i + \cdots + g_d = g + (\lambda_d + \cdots + \lambda_i)Q_{i-1}$. At any given index i_0, the evaluations $g_i(\beta_j, \beta_j)$ have been reveled for $i_0 \leq i \leq d$ and the value $s_j = g(\beta_j, \beta_j)$ is protected by the value $Q_{i_0-1}(\beta_j, \beta_j)$. The fairness of `Reconstruct`, which was proven in [ELL20], follows from the fact that if the protocol aborts at index i_0 we have the bound $n - |B| < i_0$ which implies $t_P < i_0 - \sqrt{\ell}$ when $t_P \leq n - |B| - \sqrt{\ell}$. This is the usual threshold for a bivariate sharing of degree $i_0 - 1$ and so the values of $Q_{i_0-1}(\beta_j, \beta_j)$ are information theoretically hidden.

We now want to assess what changes are introduced by the fact that the adversary knows some of the secrets s_j. From the equation above we see that the knowledge of s_j will ensure that the adversary will learn the additional evaluations $Q_i(\beta_j, \beta_j)$ for all $i_0 - 1 \leq i \leq d$. Thus, for each i, the adversary will learn ℓ' additional evaluations on each Q_i, the problem is that this quantity is independant of each i. Thus, there is necessarily a moment where these ℓ' values will be enough for the adversary to break information theoretic security on the remaining $\ell - \ell'$ secrets. In particular, the concern comes from the diagonal polynomial $Q_{i_0-1}(x, x)$. The degree of this polynomial is $2i_0$. If we write $t_P^{i_0}$ the number of corrupted participants among $P_1, \ldots, P_{i_0}$, we get that the degree of freedom of this polynomial is $2 * i_0 + 1 - (t_P^{i_0} + \ell')$. The usual bound $t_P \leq n - |B| - \sqrt{\ell}$ is not enough to ensure information theoretic security on the $\ell - \ell'$

secrets. This is why we need the new bound $t_P \leq 2(n-|B|)-\ell$. Then, we see that the degree of freedom of $Q_{i_0-1}(x,x)$ can be lower bounded by $\ell - \ell'$. When the degree of freedom of $Q_{i_0-1}(x,x)$ is big enough, the usual proof of secrecy for the bivariate sharing can be applied to Q_{i_0-1} in order to show that the $\ell-\ell'$ secrets are information theoretically hidden when $t_P \leq \min(n-|B|-\sqrt{\ell}, 2(n-|B|)-\ell)$. □

A.3 Refresh

Protocol 10. `Refresh`

INPUT: A set $\mathcal{P} = \{P_1, \ldots, P_n\}$ with respective shares $\{g(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}$.
OUTPUT: Each party $P_r \in \mathcal{P}$ obtains $\{g'(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}$, where $g'(\beta_j, \beta_j) = g(\beta_j, \beta_j)$ for all $j \in [\ell]$.

1. For $r \in [d]$, $\boxed{P_r \text{ samples } R(\alpha_r, \cdot) \leftarrow \mathbb{P}_{d-1}}$ and broadcasts homomorphic commitments to the values $\{R(\alpha_r, \alpha_{r'})\}_{r' \in [d]}$.
   Note that this implicitely defines a bivariate polynomial $R(x,y)$ of degree $d-1$.
2. For $i \in \{d+1, ..., n\}$, $\{P_i\} \cup \{P_1, ..., P_d\}$ perform $\boxed{\texttt{Recover}}$ to provide P_i with the shares $\boxed{R(\alpha_i, \alpha_{r'}) \text{ for } r' \in [d]}$.
   Note that the first step of `Recover` is unnecessary since each P_i already knows the homomorphic commitments to R.
3. For $r \in [n]$, $\boxed{P_r \text{ samples } h_r \leftarrow \mathbb{P}_d}$, and broadcasts commitments to the coefficients of $h_r(\alpha_{r'})$ for all $r' \in [d+1]$. $\boxed{P_r \text{ sends to } P_{r'}}$ an opening of the commitment to $\boxed{h_r(\alpha_{r'})}$ for all $r' \in [d+1]$.
4. For $r \in [n]$, P_r locally verifies the commitments and for each r' broadcasts a bit indicating if the opening was correct. For every irregularity on $h_{r'}(\alpha_r)$, $P_{r'}$ broadcasts the opening. If the opening is correct, P_r accepts the value, otherwise $P_{r'}$ is disqualified and added to the set of corrupted parties B. The protocols aborts and each party outputs B.
5. For $r \in [n]$, $\boxed{P_r \text{ computes } h(\alpha_r) = \sum_{r'=1}^{n} h_{r'}(\alpha_r)}$.
6. For $r \in [n]$, for all $r' \in [d+1]$, $\boxed{P_r \text{ computes}}$
   $\boxed{g'(\alpha_r, \alpha_{r'}) = g(\alpha_r, \alpha_{r'}) + (\alpha_r - \alpha_{r'}) \cdot R(\alpha_r, \alpha_{r'}) + h(\alpha_r) \cdot \prod_{j \in [\ell]} (\alpha_{r'} - \beta_j)}$.

Lemma 4. *(Informal) `Refresh` operates correctly to refresh the bivariate sharing of $\ell \leq d$ secrets among which ℓ' are known to the adversary without revealing the $\ell - \ell'$ secrets when the bound $t_P \leq d + 1 - \sqrt{\ell}$ is verified at all times. The only information obtained by the adversary on the new sharings are the shares of the corrupted participants and the leaked secrets.*

Proof. From the `Refresh` protocol, we see that $g(x,y)$ is replaced by $g'(x,y) = g(x,y) + (x-y)R(x,y) + h(x)\prod_{j\in[\ell]}(y-\beta_j)$. Both R and h are completely random

polynomials. In the proof of `Refresh` provided in [ELL20], it was shown that `Refresh` provided information theoretic security on the ℓ values $g(\beta_j, \beta_j)$ for $j \in [\ell]$. From the formula $g'(x,y) = g(x,y) + (x-y)R(x,y) + h(x)\prod_{j\in[\ell]}(y-\beta_j)$ it is clear that the additional evaluation equations $g(\beta_j, \beta_j)$ for $j \in [\ell']$ will not help the adversary discover new evaluations of either g or g'.

A.4 Redistribute

We now treat the case of `Redistribute`. This protocol modifies the number of participants involved in a sharing given in input. At the beginning the set of participant is $\{P_1, ..., P_{n^{(\omega)}}\}$ and at the end the set is $\{P_1, ..., P_{n^{(\omega+1)}}\}$. We index every parameters $(d, t_P ...)$ with the phases $\omega, \omega+1$ to make the distinction.

Protocol 11. `Redistribute`

INPUT: A set $\mathcal{P} = \{P_1, ..., P_{n^{(\omega)}}\}$ with respective shares $\{g(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}$ of degree d

OUTPUT: A set $\mathcal{P}' = \{P_1, ..., P_{n^{(\omega+1)}}\}$ of parties with respective shares $\{g'(\alpha_r, \alpha_{r'})\}_{r' \in [d'+1]}$ of degree d'

1. For $j \in [\ell]$:
   (a) Each $P_r \in \mathcal{P}$ sets $\boxed{f_j(\alpha_r) = g(\alpha_r, \beta_j)}$.
   (b) If $\mathcal{P}_{L_C} \neq \emptyset$: $\mathcal{P}_L \cup \mathcal{P}_\cap$ perform $\boxed{\texttt{DecreaseCorrupt}}$. The output is evaluations of $\boxed{f_j^1}$ a polynomial of degree $d-1$.
   (c) $\mathcal{P}_L$ and $\mathcal{P}_\cap$ uses $\boxed{\texttt{Decrease to share } f_j^2}$ a polynomial of degree $d - |\mathcal{P}_L| - |\mathcal{P}_{L_C}|$ among the participants in $\mathcal{P}_\cap$.
   (d) $\mathcal{P}_\cap$ and $\mathcal{P}_N$ perform $\boxed{\texttt{Increase to produce shares of } f_j^3}$ a polynomial of degree $d' = d - |\mathcal{P}_L| - |\mathcal{P}_{L_C}| + |\mathcal{P}_N|$ among every participant in $\mathcal{P}'$.
2. Each $P_r \in \{P_1, ..., P_{d'+1}\} \subset \mathcal{P}'$ samples $\boxed{g'(\alpha_r, .) \leftarrow \mathbb{P}_{d'}}$ such that $\boxed{\forall j \in [\ell],\, g'(\alpha_r, \beta_j) = f_j^3(\alpha_r)}$ and broadcasts the associated homomorphic commitments.
   Note that this implicitly defines a bivariate polynomial g' of degree d'.
3. Each $P_r \in \mathcal{P}'$ verifies that the commitments to g' were constructed consistently from the commitments to the f_j^3 polynomials. For each party r', P_r broadcasts a bit indicating if the commitments are correct. Each failing $P_{r'}$ is disqualified and added to the set of corrupted parties B. The protocol aborts and each party outputs B.
4. $\boxed{\{P_1, ..., P_{d'+1}\} \text{ uses } \texttt{Recover} \text{ to share } g' \text{ to } \{P_{d'+2}, \ldots, P_{n^{(\omega+1)}}\}}$.

Lemma 5. *(Informal)* `Redistribute` *transforms a sharing of ℓ secrets between $n^{(\omega)}$ participants into a sharing between $n^{(\omega+1)}$ participants. If $\ell \leq \min(d^{(\omega)}, d^{(\omega+1)})$*

and ℓ' secret are known to the adversary, then the adversary cannot find any information on the $\ell - \ell'$ remaining secrets when $t_P^{(\omega)} \leq d^{(\omega)} + 1 - \sqrt{\ell}$ and $t_P^{(\omega+1)} \leq d^{(\omega+1)} + 1 - \sqrt{\ell}$. The only information obtained by the adversary on the new sharings are the shares of the corrupted participants and the leaked secrets.

Proof. The protocol `Redistribute` makes use of the three sub-protocols `Increase`, `Decrease`, `DecreaseCorrupt` on univariate sharings and works by applying successively these three protocols on each $f_j(\cdot) = g(\cdot, \beta_j)$. It is clear from there that knowing $g(\beta_j, \beta_j)$ for j$\in [\ell']$ will not help learning any new evaluation of g'.

B Security Proof of the Multiplication Protocol

In this section we are going to prove that the protocol `Mult` introduced in Protocol 7 is secure against a mixed malicious adversary (see Theorem 2). We start by proving a series of informal lemmas on the protocols of Section 5, culminating in a proof of Lemma 11 on the security of the `Mult` protocol. As for Lemma 1. The goal of these lemmas and their proofs is to summarize the important technical ideals behind the security. It can be seen as a preliminary to the full simulator-based proof of Theorem 2 that follows.

Lemma 6. *(Informal) When $\ell = mv \leq d$, the `ReshareBivariateToUnivariate` protocol computes a correct resharing from a bivariate sharing of ℓ secrets to v univariate sharing of m secrets. When $(t_P, t_A) \leq (d + 1 - \max(m, \sqrt{\ell}), d + 1 - \max(m, \sqrt{\ell}))$, the degree of freedom of each new sharing f_k with respect to the adversary's knowledge is $d + 1 - t_P - m_k$ where $m_k \leq m$ is the number of secrets already known to the adversary before the protocol. The privacy of all the $m - m_k$ remaining secrets is preserved under the hardness of DDH.*

Lemma 6 can be easily be proven using the ideas from [ELL20].

Lemma 7. *(Informal) The protocol `BlindingBivariateGeneration` successfully distribute a bivariate sharing of m zeroes to the participants. When $m \leq d$ and $t_P \leq d + 1 - m$, the resulting bivariate sharing h (and the associted randomness R_h) satisfies $C_{\texttt{mult}}(t_P)$ under the hardness of DDH.*

Proof. For simplicity of notations we assume that $I = [m]$ in this proof. First, let us prove that the protocol correctly distributes a sharing of m zeroes. It is clear from the construction that $q(\beta_j) = 0$ for all $j \in I$. Any participant P_r trying to share a polynomial q_r not having all the β_j as roots would be caught due to the homomorphic commitments. For the same reason, all the participants are able to compute correct commitments to the coefficient of a bivariate sharing h of m zeroes at the points (β_j, β_j) for $j \in I$.

The more difficult issue, here, is to prove that h verifies the $C_{\texttt{mult}}(t_P)$ condition (the same reasoning applies to prove the condition is also verified by R_h). Let us start by enumerating the equations in E_T. Wlog we can assume that $T = [t_P]$. First, each corrupted participant P_r knows $h(\alpha_r, \cdot)$ by construction of the sharing. This provides $t_P(d + 1)$ independent evaluation equations to E_T. Since

every participant P_r with $r \in [d+1]$ generates independently its own $h(\alpha_r, \cdot)$, any new independent equation can only come from the way h is constructed in `BlindingBivariateGeneration`. Hence, $\mathcal{A}$ knows that $h(x, \beta_j) = q(x)$ for all x and $j \in I$. This is a maximum of $(d+1)m$ new equations, but in fact some of them are not independent with the equations $\mathcal{A}$ already has. Indeed, when $x = \alpha_r$ with $r \in [t_P]$, this equation can be obtained from the knowledge of $h(\alpha_r, \cdot)$. This leaves at most $m(d+1-t_P)$ new independant equations. The fact that $q(\beta_j) = 0$ provide m^2 new free evaluation equations by replacing x with all the β_j for $j \in I$. Removing these from the $(d+1-t_P)m$ equalities, we have still $(d+1-t_P-m)m$ to take into account. Since, $d+1-t_P \geq m$, this number is positive. From the construction of q, we see that its degree of freedom is exactly $d+1-t-m$ with respect to the system constituted of evaluation equations on α_r and β_j for $r \in T$ and $j \in I$. So the evaluations of q on $d+1-t_P-m$ distinct values x (different from α_r with $r \in [t_P]$ or β_j with $j \in I$) are completely unknown to $\mathcal{A}$. For these values x, the equalities $h(x, \beta_j) = q(x)$ are not equations on h as defined in Section 2.4 (since $q(x)$ is unknown). However, eliminating $q(x)$ by pairing these equations two by two, we obtain the equivalent system $h(x, \beta_j) - h(x, \beta_{j+1}) = 0$ for $j \in [m-1]$. Thus, we obtain $(m-1)(d+1-t_P-m)$ new independent equations. Since our polynomials are of degree d, we cannot extract any new independent equations.

We are now ready to prove that h verifies $C_{\texttt{mult}}(t_P)$. We start by the first item. Since the honest P_u generates all the $h(\alpha_u, \cdot)$ independently of the other participants, we can obtain $d_{h(\alpha_u,\cdot)}(E_T)$ by counting the number of equations in E_T involving $h(\alpha_u, \cdot)$. We saw that there were only $m-1$ such equations which proves that $d_{h(\alpha_u,\cdot)}(E_T) = d+2-m$.

The second item is going to hold for the same reason. First, let us point that when $E = \emptyset$, the result is clear. Indeed, we cannot use the $m-1$ equations $h(\alpha_u, \beta_j) - h(\alpha_u, \beta_{j+1}) = 0$ for $j \in [m-1]$ to express the evaluation equation on (α_u, z). Similarly, adding any system of equations E with $\#X_E \leq d - t_P$ and $\alpha_u \notin E$ will not help. Indeed, since $\#X_E \leq d - t_P$, the additional knowledge of E will only help learn information on at most $d - t_P$ new $h(x, \cdot)$. We remind the reader that the adversary knows the t_P polynomial $h(x, \cdot)$ for $x = \alpha_r$ when $r \in T$. In total this is a total of at most $t + d - t$ different x. Since $t_P + d - t_P < d + 1$, this is not enough to obtain meaningful information on $h(\alpha_u, \cdot)$ by the bivariate polynomial structure of h. Indeed, we can write $h(x, y) = \sum_{r=1}^{d+1} \Pi_r(x) h(\alpha_r, y)$ with the interpolation formula (we can replace $\alpha_1, \ldots, \alpha_{d+1}$ by any set of $d+1$ points). This formula implies that $h(\alpha_u, \cdot)$ is independent of any other d polynomials $h(x, \cdot)$. Thus, $h(\alpha_u, z)$ remains information theoretically hidden with respect to $E \cup E_T$. This finishes to prove that our bivariate sharing verifies $C_{\texttt{mult}}(t_P)$.

Remark 2. Proving the correctness of the Protocol 5 is a bit tricky. Indeed, unlike all the other protocols, correctness cannot be ensured from the additively homomorphic commitments because of the multiplications. In all the interactions between pairs P_r and $P_{r'}$, the receiver P_r is going to be able to check correctness of the values $g^\star(\alpha_r, \alpha_{r'})$ and $R_{g^\star(\alpha_r, \alpha_{r'})}$ with the commitments since

he knows $f(\alpha_r)$ in the clear. However, it is impossible for every other participant to compute this commitment. That is why we need the last step where P_r broadcasts $C(g^\star(\alpha_r, \cdot), R_{g^\star}(\alpha_r, \cdot))$. There is no way for the participants to verify the correctness of this commitment without further interactions and that is why we cannot prove correctness directly. This is the purpose of Protocol 6. In Lemma 8, we only treat secrecy. In Lemma 9, we formulate the correctness statement for Protocol 5.

Lemma 8. *(Informal) When $m < d$, the adversary knows $m_1 < m$ of the secrets shared in f, $m'_1 < m$ of the secrets shared in f' (and no other information), the corruption threshold satisfies the inequality $t_P, t_A \leq (d + 1 - m)$, the bivariate sharing h (along with the commitment randomness R_h) in input satisfies $C_{\texttt{mult}}(t_P)$ (as defined in Definition 3) and the composition of* `KeyGen` *with* `ZK-Mult` *privately realizes $\mathcal{F}_{prod}$, the protocol* `BivariateProduct` *does not leak information about the $(s_j)_{j\in I}$, $(s'_j)_{j\in I}$ and $(s_j s'_j)_{j\in I}$ which the adversary did not already know under the hardness of DDH.*

Proof. By the security of `ZK-Mult` during the exchange between P_r and $P_{r'}$, the receiver P_r learns $g^\star(\alpha_r, \alpha_{r'})$ and only that value, while $P_{r'}$ does not learn anything even when one of the two is maliciously corrupted. The same holds for $R_{g^\star}$. From this and the outline of the protocol, we need to focus on the recovery of the secrets in f'. Indeed, from the exchange between each P_r and $P_{r'}$, the receiver P_r learns a value $g^\star(\alpha_r, \alpha_{r'})$ involving $f'(\alpha_{r'})$ and that's it. Thus, no information is leaked about the secrets in f.

Let us now focus on f'. Before the execution of `BivariateProduct`, the degree of freedom of f' with respect to the adversary's knowledge is $d+1-t_P-m'_1 \geq m-m'_1$. Without loss of generality, we can assume that the set of corrupted indices is $T = [t_P]$. We are going to see the information received by the corrupted parties during `BivariateProduct` as a system of linear equations (here the term equation refer to classical equations and not equations on a polynomial as defined in Section 2.4). The unknowns will be $f'(\alpha_{r'})$ for $r' \in [t_P + 1, d + 1 - m'_1]$ and some evaluations of h. Above all, we want to keep the evaluations of f' secret. More precisely, the adversary learns $t_P(d + 1 - t_P)$ equations $g^\star(\alpha_r, \alpha_{r'}) = f(\alpha_r)f'(\alpha_{r'}) + h(\alpha_{r'}, \alpha_r)$ for all corrupted P_r and honest $P_{r'}$.

We are going to show that we have information theoretic secrecy on our secrets for any subsystem of these $t_P(d+1-t_P)$ equations. Write $\mathcal{E}$ such a system of $e \leq t_P(d+1-t_P)$ equations. If i is the number of f' secret evaluations involved in $\mathcal{E}$, we are going to show that the difference Δ between the number of unknowns in $\mathcal{E}$ and e is bigger than $\min(i, d+1-t_P-m'_1)$. If this holds for any subsystem, it proves our point. In fact, we are going to prove that $\Delta = \min(i, d+1-t_P-m'_1)$.

We are going to show this equality by induction. We start by looking at the case $i = 1$, where $1 \leq e \leq t_P$. Wlog we can assume that the e equations are $g^\star(\alpha_r, \alpha_{t_P+1})$ for $r \in [e]$. The unknowns of $\mathcal{E}$ are $f'(\alpha_{t_P+1})$ and $h(\alpha_{t_P+1}, \alpha_r)$ for $r \in [e]$, but we need to check that the adversary cannot add some equations to the system with his knowledge on h. By the first item of $C_{\texttt{mult}}$, the degree of freedom of $h(\alpha_{t_P+1}, \cdot)$ with respect to the adversary's knowledge is $d + 2 - m$.

Since, $e \leq d+1-m \leq d+2-m$, the e evaluations of h are independent and this proves that $\Delta = e+1-e = 1$. We now assume that the equality holds for i and we show that this imply the inequality for $i+1 \leq d+1-t_P-m'_1$. Wlog, we say that the $i+1$ values involved are $\alpha_{t_P+1}, \ldots, \alpha_{t_P+i+1}$. We decompose $\mathcal{E} = E_i \cup E_{i+1}$ where E_{i+1} is the set of equations involving $f'(\alpha_{t_P+i+1})$ and E_i is the set of remaining equations. We write $e_{i+b} = \#E_{i+b}$ for $b \in \{0,1\}$ and we have $e = e_i + e_{i+1}$. By applying the second item of $C_{\texttt{mult}}$ to E_i (with $i+1 \leq d+1-t_P$ we have that $\#X_{E_i} \leq d-t_P$) and $u = \alpha_{t_P+i+1}$, we see that any value of $h(\alpha_{t_P+1+i}, \cdot)$ is independent of $E_i \cup E_T$. This proves that the unknowns of E_{i+1} are independent of those of E_i. Thus, the number of unknowns in E is the sum of the number of unknowns in E_i and E_{i+1}. By applying the case $i = 1$, we see that the number of unknowns of E_{i+1} is $1 + e_{i+1}$. By applying our induction hypothesis on E_i we see that the number of unknowns in this system is $i + e_i$. Thus, the number of unknowns in E is $1 + e_i + i + e_{i+1}$, which shows that $\Delta = i + 1 + (e_i + e_{i+1}) - e = i + 1$. Finally, we show the result for $d+1-t_P-m'_1 < i+1 \leq d+1-t_P$. In this case, the $i+1-(d+1-t_P-m'_1)$ values $f'(\alpha_j)$ for $d+1-t_P-m'_1 < j \leq i+1$ can be seen as equations involving the $d+1-t_P-m'_1$ secret evaluation of f'. Following the reasoning for our induction step when $i+1 \leq d+1-t_P-m'_1$, we can still divide E in two sets of equations E_i and E_{i+1} of sizes e_i and e_{i+1}. Applying the first item of $\mathcal{C}_{\texttt{mult}}$ we can show that there are $e_{i+1} + d + 1 - t_P - m'_1$ unknowns in E_{i+1} and $e_i + d_1 - t_P - m'_1$ unknowns in E_i by the induction hypothesis. Putting the two systems together, we can remove the common unknowns (which are the $d+1-t_P-m'_1$ secret evaluations of f') and see that $\Delta = e_i + e_{i+1} + d + 1 - t_P - m'_1$.

The same reasoning applies to $R_{g^\star}$ with the condition verified by R_h. So the random $R_{f'}$ has the same level of secrecy as f' and the homomorphic commitments preserve the secrecy with the hiding property of the commitment scheme. This finishes to prove that privacy of $(s'_j)_{j\in I}$ is preserved during `BivariateProduct`. Finally, we want to show the same thing for $(s_j s'_j)_{j\in I}$. All the values $g^\star(\alpha_r, \alpha_{r'})$ for honest P_r and $P_{r'}$ are unknown to the adversary. This is the usual bivariate sharing setting. There are more than m blinding terms unknown to the adversary that are protecting the secrets in $g^\star$. Additionally, the same holds for the evaluations of $R_{g^\star}$ and the hiding property of the commitment scheme preserves secrecy of the commitments.

Lemma 9. *(Informal) If the protocol* `BivariateProduct` *executes without any abort, every honest participant P_r got the correct polynomials $g^\star(\alpha_r, \cdot)$ and $R_{g^\star}(\alpha_r, \cdot)$.*

Proof. We need to prove that every participant can check the consistency of its exchange with $P_{r'}$ for all $r' \in [d+1]$. The value $f(\alpha_r)$, that is known to P_r, is enough to locally compute the commitment $f(\alpha_r) \cdot C(f'(\alpha_{r'}), R_{f'}(\alpha_{r'})) \star C(h(\alpha_{r'}, \alpha_r), R_h(\alpha_{r'}, \alpha_r)) = C(f(\alpha_r)f'(\alpha_{r'}) + h(\alpha_{r'}, \alpha_r)), f(\alpha_r)R_{f'}(\alpha_{r'}) + R_h(\alpha_{r'}, \alpha_r))$. Thus, by the perfectly binding property of the commitment scheme, the values $g^\star(\alpha_r, \alpha_{r'})$ and $R_{g^\star}(\alpha_r, \alpha_{r'})$ are correct.

Lemma 10. *(Informal) If $m < d$ and $t_P, t_A \leq d-m$ The protocol* `CommitmentVerification` *preserves the secrecy of the batch of m secrets (s_i) shared in f under the hard-*

ness of DDH. If the output is 1*, the set of commitments* $C(g(\cdot,\beta_j), R_g(\cdot,\beta_j))$ *is equal to* $C(g^\star, R_{g^\star})$ *(where* $g^\star(x,y) = f(x)f'(y) + h(y,x)$ *and* $R_{g^\star}(x,y) = f(x)R_{f'}(y) + R_h(y,x)$ *with probability bigger than* $1 - d/q$*.*

Proof. For secrecy, note that during the protocol, the value $f(\texttt{rand})$ will be revealed to all parties; henceforth information theoretic secrecy of the s_i's can only hold when $t_P \leq d-m$. The technique from steps 1–8 used to reveal the latter value follows that of [DEL+16]. By construction, we have $\hat{f} = f + \sum_{u\in[n]} \hat{f}_u$. Assume $t_P \geq 1$. For every honest P_u, $\hat{f}_u$ has a degree of liberty of $d - m \geq 1$ with respect to the adversary's knowledge, and there are at least $d+1-t_P > m$ such polynomials. Henceforth, the m values $\hat{f}(\beta_j)$ are distributed as uniformly random values for the adversary. Finally, the hiding property of the commitment scheme guarantees computational secrecy.

We are now going to show that when the protocol outputs 1, the commitment $C(g, R_g)$ cannot be different anything else than $C(g^\star, R_{g^\star})$ with probability higher than d/q. We can see g, R_g as equal to $g^\star + \Delta, R_{g^\star} + R_\Delta$ for some bivariate error polynomials Δ, R_Δ. We want to show that the probability of $\Delta \neq 0$ is smaller than d/q.

First, the homomorphic property ensures that the parties obtain the correct value $f(\texttt{rand})$ as it allows the participants to check every operation performed leading to this computation. Since each $\hat{f}_u(\texttt{rand}) = 0$, we have $\hat{f}(\texttt{rand}) = f(\texttt{rand})$. Thus, the adversary needs that $\Delta(\texttt{rand}, \cdot) = 0$ as the commitment scheme is perfectly binding. This implies that $\Delta(x,y) = (x - \texttt{rand})\Delta'(x,y)$. The adversary can ensure that $\Delta \neq 0$ verifies this equation for at most d values $\texttt{rand}$. Thus, the value $\texttt{rand}$ being uniformly distributed in $\mathbb{Z}_q$, the adversary has at most probability d/q to cheat successfully.

We conclude with Lemma 11 by combining all the above results.

Lemma 11. *(Informal) If* $\ell = mv \leq d$ *and* $t_P, t_A \leq \min(d-m, d+1-\sqrt{\ell})$*, then when* ℓ_1 *secrets are leaked in the sharing* g *and* ℓ'_1 *secrets are leaked in the sharing* g'*, the protocol* `Mult` *preserves secrecy of the secrets* $(s_i)_{i\in[\ell-\ell_1]}, (s'_i)_{i\in[\ell-\ell'_1]}$ *under the hardness of DDH. The protocol distributes a bivariate a sharing* g'' *of* $(s_i s'_i)_{i\in[\ell]}$*. Additionally, the adversary only learns the shares of the corrupted parties for* g'' *along with the values* $g''(\beta_j, \beta_j)$ *where both* s_j *and* s'_j *are leaked secrets. In the end of the protocol, when the protocol, has not aborted, the participants hold a valid sharing of the secrets* $s_1 s'_1, \ldots, s_\ell s'_\ell$ *with overwhelming probabilities.*

Proof. (Lemma 11) Since $t_P, t_A \leq d + 1 - \sqrt{\ell}$, no information is leaked on the secret through the various bivariate sharings. Wlog we can assume that $P_1, \ldots, P_{t_P}$ are the corrupted participants as the adversary has clearly nothing to gain by corrupting P_r when $r \in [d+2, n]$. The secrecy on all the $s_i, s'_i, s_i s'_i$ is preserved throughout the executions of `ReshareBivariateToUnivariate` , `BlindingBivariateGeneration`,
`BivariateProduct` and `CommitmentVerification` because of Lemmas 6 to 8

and 10. Then, the generation of g'' is performed locally. The distribution of their shares to the remaining participants P_r with $r \in [d+2, n]$ is made with `Recover`, does not reveal anything to the adversary by secrecy of `Recover`. The same if true for the execution of `Refresh`. Hence, secrecy is preserved. We also need to verify that no other evaluation of g'' is leaked. This is the purpose of the `Refresh` step at the end. Indeed, after step 3, we have $g''(\cdot, \beta_j) = f_k(\cdot)f'_k(\beta_j) + h(\beta_j, \cdot)$. The security properties on h proven in Lemma 7 are not enough to prove the result and in practice if we look at the concrete Protocol 4, it is quite clear that the result does not hold. With `Refresh`, we replace $g''(x, y)$ by $g''(x, y) + (x - y)R'(x, y) + h(x)\prod_{j\in[\ell}(y - \beta_j)$ where h and R are random polynomials. From there, it is easy to see that every value of g'' is replaced by a new random value except at (β_j, β_j) for $j \in [\ell]$. Thus, it proves that the adversary will know his shares $g(\alpha_r, \cdot)$ for corrupted participants and the secrets $g''(\beta_j, \beta_j)$ for leaked s_j, s'_j and nothing else.

For correctness, the executions of `ReshareBivariateToUnivariate`, `BlindingBivariateGeneration`, `BivariateProduct` are correct due to Lemmas 6, 7 and 9. After the execution of these 3 steps, the participants have obtained a bivariate sharing $g^\star_k$ and randomness $R_{g^\star_k}$ along with the commitments $C(g_k, R_{g_k})$ to some bivariate polynomial g_k under randomness R_{g_k} for all k. If we assume that `CommitmentVerification` passes, we know that $g_k(\cdot, \beta_j) = f(\cdot)f'(\beta_j) + h(\beta_j\cdot)$ with probability higher than $1 - d/q$ (that we consider overwhelming) by Lemma 10. Thus, $g''(\beta_j, \beta_j) = f(\beta_j)f'(\beta_j) = s_j s'_j$ for all $j \in [\ell]$ with overwhelming probability.

Let us first define the ideal process for the mixed-adversary model for the multiplication.

Figure 2. Ideal Process for the Extended Multiplication protocol in the mixed adversary model.

In this ideal process, the environment $\mathcal{Z}$ will provide the parties and an ideal adversary $\mathcal{S}$ with inputs of its choice. Throughout the protocol the parties will interact with an ideal functionality $\mathcal{F}_{MULT,\ell}$ that will play the role of a trusted third party that will compute new shares for the multiplication. Upon reception of the parties' inputs that are the sharings of two batches of secrets $s_1, \ldots, s_\ell$ and $s'_1, \ldots, s'_\ell$, the functionality will output a new sharing of $s_1 s'_1, \ldots, s_\ell s'_\ell$.

The ideal process.
INITIALIZATION
1. $\mathcal{Z}$ invokes the adversary $\mathcal{S}$ with an auxiliary input z along with the sets of actively and passively corrupted parties $\mathcal{P}_A$ and $\mathcal{P}_P$.
2. $\mathcal{Z}$ invokes the parties P_r and their inputs x_r, x'_r.
INPUTS
3. Each party sends his input to the ideal functionality.

INPUT CORRUPTION
4. $\mathcal{F}_{MULT,\ell}$ sends x_r, x'_r to $\mathcal{S}$ when $P_r \in \mathcal{P}_P$.
COMPUTATION
5. $\mathcal{F}_{MULT,\ell}$ evaluates the shares to find the batches of secrets $s_1, \ldots, s_\ell$ and $s'_1, \ldots, s'_\ell$. Then, it computes $s_1 s'_1, \ldots, s_\ell s'_\ell$ and produces a new bivariate sharing for which the share for participant P_r is y_r.
OUTPUT DISTRIBUTION
6. $\mathcal{F}_{MULT,\ell}$ sends the output y_r for corrupted participants P_r to $\mathcal{S}$.
7. $\mathcal{S}$ sends either `abort` to $\mathcal{F}_{MULT,\ell}$ and $\mathcal{F}_{MULT,\ell}$ aborts and outputs $\perp$ or `continue` and the functionality proceeds to the next step.

8. $\mathcal{F}_{MULT,\ell}$ sends y_i to each party P_i.

Outputs. Each honest party outputs whatever they received from F_{MULT}. The adversary outputs a value $v_{\mathcal{S}}$ that may be arbitrarily computed from the information he obtained during the execution of the protocol. After observing the outputs of all parties and of the adversary, the environment outputs a bit $b_{\mathcal{Z}}$.

For simplicity, in the theorem below we assume that ℓ is a perfect square so that we can take the parameter m to be the exact square root of ℓ (recall that we assumed the factorization $\ell = mv$ for Protocol 7). The extension for a generic ℓ is easy.

Theorem 2. *When $\ell \leq d$ and $m = \lceil \sqrt{\ell} \rceil$ and assuming the hardness of DDH, the protocol Π^{MULT} introduced in Protocol 7 securely realizes the ideal process* $\text{IDEAL}_{mixed}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{MULT,\ell}}$ *against any adversary bounded by the multi-threshold $T(\ell)$ where we have $T(\ell) = \left\{(d-1-\sqrt{\ell}, d-1-\sqrt{\ell})\right\}$.*

We are going to construct a simulator $\mathcal{S}$ that will emulate the view of an adversary $\mathcal{A}$ in the real execution of Protocol 7. In the following, when we say $\mathcal{S}$ inputs `abort` to $\mathcal{F}_{MULT,\ell}$ it refers to Step 6 of the ideal process.

In the Simulator described in Fig. 3, we recall that the set of actively (resp. passively) corrupted parties is $\mathcal{P}_A$ (resp. $\mathcal{P}_P$) and that $\mathcal{P}_A \subset \mathcal{P}_P$. The set of honest participant is denoted $\mathcal{H}$.

Figure 3. Simulator for the Multiplication a malicious adversary

1. $\mathcal{S}$ receives the inputs x_r and new shares y_r for all the corrupted parties $P_r \in \mathcal{P}_P$ from $\mathcal{F}_{MULT,\ell}$.
2. $\mathcal{S}$ generates random shares for the set of honest x_r, x'_r for all honest parties $P_r \in \mathcal{H}$..
3. The simulator initialize $\mathcal{A}$ commitments to the shares of all the participants and opening of these commitments for shares belonging to corrupted participants.

4. $\mathcal{S}$ perform the `mult` protocol with $\mathcal{A}$ on behalf of the honest parties using the shares and commitments it generated during the previous steps.
5. If at any point during the execution, the protocol aborts, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{MULT,\ell}$.
6. At the end of the protocol, if no `abort` was sent, the simulator sends `continue`.
7. $\mathcal{S}$ sends `continue` to $\mathcal{F}_{MULT,\ell}$ and outputs whatever $\mathcal{A}$ outputs.
8. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Proof. We want to show that the output sent by $\mathcal{S}$ to $\mathcal{F}_{MULT,\ell}$ is indistinguishable from the one produced by the adversary in the real execution of the protocol. Since $\mathcal{S}$ forwards to the ideal functionality whatever $\mathcal{A}$ outputs, it suffices to show that the simulator's execution of the protocol on behalf's of the honest parties is indistinguishable from the real executions. First, it is clear that the initialization step is performed correctly since $\mathcal{S}$ simply transmits the shares obtained from $\mathcal{F}_{MULT,\ell}$. Then, $\mathcal{S}$ generates random shares on behalf of the honest parties and from then behaves as the honest participants would. By Lemma 11, there is computational secrecy on the secrets during the execution of Protocol 7 under the DDH assumption underlying the hiding property of the commitment scheme. Thus, the adversary cannot distinguish between the real execution and the simulated one where the honest parties shares have been replaced by random values. This concludes the proof that the output of the adversary is correctly distributed. Due to the correctness of the protocol (also showed in Lemma 11), we have that either the protocol aborts or the participants end up with a correct sharing of the batch of secrets $s_1 s'_1, \ldots, s_\ell, s'_\ell$. Finally, due to the execution of `Refresh` (see [ELL20] for more details on this protocol and its security) at the end of Protocol 7, the output shares are indistinguishable from a random sharing of $s_1 s'_1, \ldots, s_\ell s'_\ell$. This shows that the honest parties outputs are similarly distributed in both the real and ideal executions. Hence, we have shown indistinguishability of all the elements sent to the environment in both the ideal and real executions. Thus, the output $b_{\mathcal{Z}}$ are also indistinguishable.

C Permuting Packed Secrets

In this section, we present a `Permute` (Protocol 14) protocol that allows to perform any permutation π on a given set of secrets $s_1, \ldots, s_L$ divided in several batches of size ℓ. This protocol uses two sub-protocol `PermuteBetweenBlocks` (Protocol 12) and `PermuteWithinBlock` (Protocol 13). We are going to adapt protocols from [BELO15] which were inspired from the works of [Ben64] and [Wak68] on networks and relied on results proved in [GHS12]. Let us suppose that we have a total of $L = a\ell$ secrets that are shared in a bivariate polynomials $\left(g^m\right)$ for all $m \in [a]$ after the computation of some layers of an arithmetic circuit C. To apply a given permutation on these secrets, we rely on the following lemma.

Lemma 12. *Let π be a permutation on L elements, where L is a power of two. Each element is given a index written with a binary representation of size $log(L)$. Then π can be decomposed as $\pi = \pi_1 \circ \pi_2 \circ \ldots \circ \pi_{2log(L)-1}$. $\forall\ 1 \leq k \leq log(L)$, π_k only swaps elements with indexes whose only difference is in the k-th bit, $\forall\ log(L) \leq k \leq 2log(L) - 1$, π_k only swaps elements with indexes whose only difference is in the $2log(L) - k$th bit.*

The sub-permutations π_k are composed of transpositions that swaps 2 elements that have always the same distance, denoted $\delta(\pi_k)$, a power of 2. In all the following we are going to consider that ℓ and L are powers of 2. That is fine since we can always extend the number of secret to a power of two by adding random secrets. If we write the L secrets $s_1^1, \ldots, s_\ell^1, \ldots, s_1^a, \ldots, s_\ell^a$ then if $\delta(\pi_k) < \ell$, π_k is a permutation within a block and if $\ell \leq \delta(\pi_k)$ then π_k is a permutation between blocks. In the following we introduce two protocols `PermuteWithinBlocks` (Protocol 13) and `PermuteBetweenBlocks` (Protocol 12) treating these two cases. Then, we will introduce a `Permute` protocol (Protocol 14) that performs the whole permutation on the $a\ell$ secrets.

Permutation between blocks We present a protocol that performs a permutation between blocks π of the form we introduced earlier. Such a permutation swaps elements having the same index between all pairs of blocks $(b, b + \frac{\delta(\pi)}{\ell})\mod a$ for all $b \in [a]$. To do that, we will use our `Mult` protocol. For $J \subset [\ell]$ we denote as sh_J a bivariate sharing of the batch of values $\left(\mathbb{1}_{j \in J}\right)_{j \in [\ell]}$. In practice we assume that for each $J \subset [\ell]$, the parties know such a bivariate polynomial sh_J, it is known by every participant and can follow any generic construction from J. For a given $m \in [a]$, we denote J_m the set of elements $j \in [\ell]$ such that the transposition $(b+j, b+\frac{\delta(\pi)}{\ell}+j)$ is contained in π_k. The idea is that the updated bivariate polynomial g^b is obtained by adding the bivariate polynomials obtained from the multiplication of the batches in g^b and in $sh_{J_b^C}$ and the multiplication of the batches in $g_{b+\frac{\delta(\pi)}{\ell}}$ and in sh_{J_b}. The updated $g_{b+\frac{\delta(\pi)}{\ell}}$ is obtained by swapping g_b and $g_{b+\frac{\delta(\pi)}{\ell}}$ in the previous sentence.

Protocol 12. `PermuteBetweenBlocks`

INPUT: The set of parties have $(g_b)_{b\in[a]}$, bivariate sharings of a batches of ℓ secrets and the associated commitments $C(g_b, R_{g_b})_{b\in[a]}$. A permutation π of the $a\ell$ secrets, swapping pairs of secrets in sharings distant by $\frac{\delta(\pi)}{\ell}\ mod\ a$.
OUTPUT: The set of parties have $(g_b^\pi)_{b\in[a]}$, bivariate sharings of the a batches of ℓ secrets permuted by π along with the commitments $C(g_b^\pi, R_{g_b^\pi})_{b\in[a]}$.

1. For each $b \in [a]$ such that the $\delta(\pi)$-th bit in the binary representation of $b-1$ is zero:
   (a) The parties perform the `Mult` protocol on the 4 pairs of bivariate sharing (g_b, sh_{J_b}), $(g_{b+\frac{\delta(\pi)}{\ell}}, sh^C_{J_b})$, $(g_b, sh^C_{J_b})$, $(g_{b+\frac{\delta(\pi)}{\ell}}, sh_{J_b})$ to

obtain the four bivariate sharings g_1, g_2, g_3, g_4 along with commitments to these sharings.

(b) Using the `Add` protocol, the parties compute the updating sharings $g_b^\pi = g_3 + g_4$ and $g_{b+\frac{\delta(\pi)}{\ell}}^\pi = g_1 + g_2$ and the corresponding bivariate commitments.

Lemma 13. *(Informal) When $\ell = mv \leq d$ and $t_P, t_A \leq \min(d-m, d+1-\sqrt{\ell})$, the `PermuteBetweenBlocks` protocol is correct and the adversary cannot learn any of the $a\ell$ secrets that he didn't know before the execution of the protocol.*

Proof. The correctness of our protocols `Mult` ensures that g_1, g_2, g_3, g_4 are correct sharings. Then, we just need to verify that the formulae for g_b^π and $g_{b+\frac{\delta(\pi)}{\ell}}^\pi$ are correct. If an index j is in J_b, then by definition the secrets of index j shared in g_b and $gb + \frac{\delta(\pi)}{\ell}$ must be swapped. In this case, the secret of index j in g_3 is 0 and the secret of same index in g_4 is the j-th secret of $g_{b+\frac{\delta(\pi)}{\ell}}$. Thus, the j-th secret stored in $g_3 + g_4$ is indeed the correct value. Conversely, when $j \notin J_b$, the secret stored in $g_3 + g_4$ is the j-th secret of g_b. Similarly, we can show the correctness of the updated $g_{b+\frac{\delta(\pi)}{\ell}}^\pi$.

The secrecy follows directly from the secrecy property of the `Add` and `Mult` protocols. The fact that half of the sharing is known for sh^{J_m} does not change anything, it can be easily verified. □

Communication complexity: The communication complexity is the same as the `Mult` protocol multiplied by the number of blocks so it is $O(an^2v)$. Amortized by the number of secrets, we obtain $O(n^2/m)$ as in `Mult`.

Permutation within blocks. This paragraph presents a protocol to perform permutations π with $\delta(\pi) < \ell$, theses permutations can be decomposed as several permutations operating on one block. We give below a protocol that perform such a permutation of the secrets on one block shared by one bivariate polynomial g. This protocol will take as input a permutation π of $[\ell]$ and a sharing of a batch of secret $s_1, \ldots, s_\ell$ and will output a sharing for the batch of secrets $s_{\pi(1)}, \ldots, s_{\pi(\ell)}$. The idea is that, it is easy for a set of $d+1$ participant to transform a sharing of $s_1, \ldots, s_\ell$ to a sharing of $s_{\pi(1)}, \ldots, s_{\pi(\ell)}$. If we note g^π this new bivariate sharing, the relations $g^\pi(\alpha_r, \beta_j) = g(\alpha_r, \beta_{\pi(j)}) \frac{\Pi_r(\beta_{\pi(j)})}{\Pi_r(\beta_j)}$ for all $r \in [d+1]$ ensures the correctness of the computation. We remind the notation $\Pi_u(x) = \prod_{1 \leq v \leq d+1, v \neq u} \frac{x-\alpha_v}{\alpha_u - \alpha_v}$ for $u \in [d+1]$.

Protocol 13. `PermuteWithinBlock`

INPUT: The set of parties have the shares of g a bivariate sharing of a batch of ℓ secrets $s_1, \ldots, s_\ell$ along with the commitment $C(g, R_g)$. A permutation π of the ℓ secrets.

OUTPUT: A bivariate sharing g_π for the permuted batch of secrets $s_{\pi(1)}, \ldots, s_{\pi(\ell)}$ together with the commitment $C(g^\pi, R_{g^\pi})$.

1. For $r \in [d+1]$, P_r samples $g^\pi(\alpha_r, \cdot), R_{g^\pi}(\alpha_r, \cdot) \leftarrow \mathbb{P}_d$ verifying $g^\pi(\alpha_r, \beta_j) = g(\alpha_r, \beta_{\pi(j)}) \frac{\Pi_r(\beta_{\pi(j)})}{\Pi_r(\beta_j)}$ and $R_{g^\pi}(\alpha_r, \beta_j)) = R_g(\alpha_r, \beta_{\pi(j)}) \frac{\Pi_r(\beta_{\pi(j)})}{\Pi_r(\beta_j)}$ for all $j \in [\ell]$. P_r broadcasts the commitment $C(g^\pi(\alpha_r, \cdot), R_{g^\pi}(\alpha_r, \cdot))$.
2. For $u \in [n]$ and $r \in [d+1]$, P_u computes locally the commitments $C(g(\alpha_r, \beta_{\pi(j)}), R_g(\alpha_r, \beta_{\pi(j)}))$, $C(g^\pi(\alpha_r, \beta_j), R_{g^\pi}(\alpha_r, \beta_j))$ and checks the equality

$$\frac{\Pi_r(\beta_{\pi(j)}}{\Pi_r(\beta_j)} \cdot C(g(\alpha_r, \beta_{\pi(j)}), R_g(\alpha_r, \beta_{\pi(j)})) = C(g^\pi(\alpha_r, \beta_j), R_{g^\pi}(\alpha_r, \beta_j)).$$

   If one of the verification fails for index r, P_r is added to the set of corrupted participants B. The protocol aborts and each participant outputs B.
3. For all $r' \in [d+2, n]$, $\{P_1, \ldots, P_{d+1}\} \cup \{P_{r'}\}$ perform `Recover` on g^π.

Lemma 14. *(Informal) When $\ell \leq d$ a,d $t_P, t_A \leq d + 1 - \sqrt{\ell}$ and ℓ' secrets are leaked to the adversary, the `PermuteWithinBlocks` protocol is correct and the adversary cannot learn anythin of the $\ell - \ell'$ secrets still unknown to the adversary.*

Proof. We need to check that g^π is a correct sharing of $s_{\pi(1)}, \ldots, s_{\pi(\ell)}$. Let us look at the values $g^\pi(\beta_j, \beta_j)$ for $j \in [\ell]$. From Lagrange's interpolation formula we have

$$\begin{aligned} g^\pi(\beta_j, \beta_j) &= \sum_{u=1}^{d+1} q^\pi(\alpha_u, \beta_j) \Pi_u(\beta_j) \\ &= \sum_{u=1}^{d+1} \frac{\Pi_u(\beta_{\pi(j)})}{\Pi_u(\beta_j)} g(\alpha_u, \beta_{\pi(j)}) \Pi_u(\beta_j) \\ &= s_{\pi(j)} \,. \end{aligned}$$

This proves the correctness of the formula. The fact that $P_1, \ldots, P_{d+1}$ follow a similar equation to produce R_{g^π} ensures that each participant P_u can compute locally valid commitments to $C(g^\pi(\cdot, \beta_j), R_{g^\pi}(\cdot, \beta_j))$ for all $j \in [\ell]$. The transformation is linear and the computationally binding property of the commitment guarantees that no misbehavior can go undetected.

Apart from commitments, the only communication happens during the execution `Recover` and the secrecy for `PermuteWithinBlock` follows from `Recover`'s security (see Lemma 2). The communication complexity if $O(n^2)$ or $O(n^2/\ell)$ amortized.

□

Performing an arbitrary permutation. In this paragraph, we introduce a `Permute` protocol that performs a permutation π on $L = a\ell$ values divided in a batches of ℓ secrets. Without loss of generality, we assume that A, a and ℓ are powers of 2. Using the decomposition of π with Lemma 12, we perform several `PermuteBetweenBlocks` and `PermuteWithinBlock`.

Protocol 14. `Permute`

INPUT: The set of parties have $(g_b)_{b\in[a]}$, bivariate sharings of a batches of ℓ secrets and the associated commitments $C(g_b, R_{g_b})_{b\in[a]}$. A permutation π of the $a\ell$ secrets decomposed as $\pi = \pi_1 \circ \pi_2 \circ \ldots \circ \pi_{2log(L)-1}$.
OUTPUT: The set of parties have $(g_b^\pi)_{b\in[a]}$ and the associated commitments $C(g_b^\pi, R_{g_b^\pi})_{b\in[a]}$, bivariate sharings of the a batches of ℓ secrets permuted by π.

1. For $i = 2log(A) - 1$ down to $i = 1$:
   (a) If $\delta(\pi_i) < \ell$ replace each sharing g_b and the corresponding commitments by the result of `PermuteWithinBlock` on g_b under the permutation π_i.
   (b) If $\ell \leq \delta(\pi_i)$, execute `PermuteBetweenBlocks` with the permutation π_i and replace the set of sharings $(g_b)_{b\in[a]}$ and associated commitments by the output of the protocol.

Communication complexity: The overall complexity amortized complexity is the same as `PermuteBetweenBlocks` so $O(n^2/m)$ or $O(n\sqrt{n})$ when $m =\approx \sqrt{n}$.

D Proof for the Full PMPC Protocol

We finally prove the security of Protocol 2 that we denote PMPC. For simplicity of explanations, we do not provide a full step-by-step description of the simulator in the proof, but rather a sketch of proof to argue why all our secure sub-protocols can be safely composed to constitute Protocol 2.

We start by presenting the full Dynamic PMPC ideal functionality for generic computation of a function f. To take into consideration the Dynamic groups property we are going to assume that f takes n inputs $x_1, \ldots, x_n$ and produces $n + k$ outputs $y_1, \ldots, y_{n+k}$ for $k \in \mathbb{N}$. When f is randomized, the input includes a random seed r as well. The initial set of participant is $\mathcal{P}_0 = \{P_1, \ldots, P_n\}$ and the total set of parties involved, at any time, in the computation is $\mathcal{P}_1 = \{P_1, \ldots, P_{n+k}\}$. In fact, we present two flavours for our ideal functionality, one with fairness and one without. We write $\text{IDEAL}_{\texttt{xxx}}^{\mathcal{Z},\mathcal{S},\mathcal{F}PMPC}$ for this ideal functionality where `xxx` $=$ `fair` for the former and `unfair` for the latter.

Figure 4. Ideal Process $\text{IDEAL}_{\texttt{xxx}}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{PMPC}}$ for the Generic Dynamic PMPC in the Mixed Adversary Model.

In this ideal process, the environment $\mathcal{Z}$ will provide the parties and an ideal adversary $\mathcal{S}$ with inputs of its choice. Throughout the protocol the parties and the adversary $\mathcal{S}$ will interact with an ideal functionality $\mathcal{F}_{PMPC}$ that will play the role of a trusted third party that will compute the evaluation of f on the desired input and output it to the parties. When `xxx` $=$ `unfair`,

the adversary $\mathcal{S}$ has the additional ability to decide if the honest parties obtain an output.

The ideal process:

1. INITIALIZATION: $\mathcal{Z}$ invokes the adversary $\mathcal{S}$ with an auxiliary input z.
2. $\mathcal{Z}$ invokes the parties P_r and their inputs x_r.
3. $\mathcal{F}$ initializes the sets of passively (resp. actively) corrupted parties $\mathcal{P}_P = \emptyset$ (resp. $\mathcal{P}_A = \emptyset$) and sets $t_P = t_A = 0$.
4. INPUTS : Each party sends his input to the ideal functionality.
5. INPUT CORRUPTION: $\mathcal{S}$ sends to $\mathcal{F}$ a message (τ, i) where $\tau \in \{\texttt{passive,active,}\}$ and $i \in [n] \cup \{\perp\}$.
6. If $i \neq \perp$ and $(t_P, t_A) \leq T_s$:
   - $\mathcal{F}$ sends `Yes` to $\mathcal{S}$.
   - If $\tau \in \{\texttt{passive,active}\}$: Update $\mathcal{P}_P = \mathcal{P}_P \cup \{P_i\}$ and $t_P = t_P + 1$. $\mathcal{F}$ sends x_i to $\mathcal{S}$.
   - If $\tau \in \{\texttt{active}\}$: Update $\mathcal{P}_A = \mathcal{P}_A \cup \{P_i\}$ and $t_A = t_A + 1$.
   - Go back to the beginning of the INPUT CORRUPTION phase.
7. Else, if $i \neq \perp$ then $\mathcal{F}$ sends `No` to $\mathcal{S}$. Go back to the beginning of INPUT CORRUPTION.
8. $\mathcal{F}$ samples a random bit-string r of appropriate length.
9. COMPUTATION : $\mathcal{F}$ computes $(y_1, \ldots, y_{n+k}) = f(x_1, \ldots, x_n)$ if f is deterministic, and $(y_1, \ldots, y_{n+k}) = f(x_1, \ldots, x_n; r)$ otherwise.
10. OUTPUT CORRUPTION : $\mathcal{S}$ sends message `StartOutputPhase` to $\mathcal{F}$ that resets $\mathcal{P}_P = \emptyset$ and $\mathcal{P}_A = \emptyset$ and $t_P = t_A = 0$.
11. $\mathcal{S}$ sends to $\mathcal{F}$ a message (τ, i) where $\tau \in \{\texttt{passive,active}\}$ and $i \in [n+k] \cup \{\perp\}$.
12. If $i \neq \perp$ and $(t_P, t_A) \leq T_s$:
    - $\mathcal{F}$ sends `Yes` to $\mathcal{S}$.
    - If $\tau \in \{\texttt{passive,active}\}$: Update $\mathcal{P}_P = \mathcal{P}_P \cup \{P_i\}$ and $t_P = t_P + 1$. $\mathcal{F}$ sends y_i to $\mathcal{S}$.
    - If $\tau \in \{\texttt{active}\}$: Update $\mathcal{P}_A = \mathcal{P}_A \cup \{P_i\}$ and $t_A = t_A + 1$.
    - Go back to the beginning of the OUTPUT CORRUPTION phase.
13. Else, if $i \neq \perp$ then $\mathcal{F}$ sends `No` to $\mathcal{S}$. Go back to the beginning of OUTPUT CORRUPTION.
14. OUTPUTS: $\mathcal{S}$ sends either `abort` to $\mathcal{F}$ and $\mathcal{F}$ aborts and outputs $\perp$ or `continue` and the functionality proceeds to the next step.
15. For each $P_r \in \mathcal{P}_P$, $\mathcal{F}$ sends the output y_r to $\mathcal{F}$.
16. If `xxx= unfair`, $\mathcal{F}$ sends the message `fair?` to $\mathcal{S}$, who answers with `Yes` or `No`. In the affirmative case, the functionality proceeds to the next step. Otherwise, $\mathcal{F}$ aborts and outputs $\perp$.
17. $\mathcal{F}$ sends its output y_r to each participant P_r for $r \in [1, n+k]$.

Outputs. Each honest party outputs whatever they received from F. The adversary outputs a value $v_{\mathcal{S}}$ that may be arbitrarily computed from the information he obtained during the execution of the protocol. After observing the outputs of all parties and of the adversary, the environment outputs a bit $b_{\mathcal{Z}}$.

Theorem 1. *For a circuit $\mathcal{C}$ where the minimum number of participants is n_0. When the size batch is $\ell = n_0 - 2$, and assuming the hardness of DDH, the protocol Π^{PMPC} introduced in Protocol 2 securely realizes the ideal process* $\text{IDEAL}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{PMPC}}_{unfair}$ *against any adversary bounded at any given time by the multi-threshold $T(\ell) = \left\{(n-3-\sqrt{\ell}, n-3-\sqrt{\ell})\right\}$ with n the number of participants at that time. Additionally, when the adversary is also bounded by the multi-threshold $T_f(\ell) = \{(k, \min(n-k-\sqrt{\ell}, 2(n-k)-\ell, 1 \leq k \leq n/3\}$, Π^{PMPC} securely realizes* $\text{IDEAL}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{PMPC}}_{fair}$.

Proof. (Sketch) The Generic PMPC protocol is essentially a sequential execution of a mix of `Share`, `Refresh`, `Recover`, `Redistribute`, `Mult`, `Add`, `Permute` and `Reconstruct`. All of those protocols were proven secure when the adversary is bounded by $T_s(\ell), T_c(\ell)$ (see Lemma 1 and Theorem 2 and the ideal functionalities and theorems in [ELL20]). We also remind the reader that the `Add` protocol can be performed locally and is secure. So we just need to verify that the composition of all these sub-protocols is not breaking the security. First, note that by definition, the set of corrupted parties is assumed to change only during the `Refresh` phase (see [ELL20,DEL+16] for the corruption model). By definition of the `Refresh` ideal functionality, the protocol is producing a new random sharing of the batch of secrets and distributes it to the participants. More generally, in `Mult, Redistribute, Permute` we showed (and it was proven for `Refresh,Recover` in [ELL20]) that the sharings produced were indistinguishable from new random sharings of the desired secrets. Thus, the correctness and secrecy of each of these sub-protocols are preserved throughout the execution. We have also verified in Appendix A that all our protocols were secure even when some of the secrets were known. Thus, the composition with `Permute` is not problematic.

By composing the simulators of `Share, Resfresh, Recover, Redistribute, Permute, Add, Mult` we can simulate Π_{PMPC} up to the last `Reconstruct` step. To deal with this final step, we need to make a distinction between the `fair` and `unfair` ideal functionalities. In any case, it was proven in [ELL20] that `Reconstruct` is correct when $T_c(\ell)$ is respected (secrecy is irrelevant at that point). Thus, we can prove that Π_{PMPC} securely realizes $\text{IDEAL}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{PMPC}}_{\texttt{unfair}}$. When the additional fairness threshold $T_f(\ell)$ is respected, it was shown in [ELL20] that we have also fairness. Thus, we can show that $\text{IDEAL}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{PMPC}}_{\texttt{fair}}$ is securely realized by our generic PMPC protocol when the adversary is also bounded by the fairness threshold.

E Sub-protocols for the multiplication from the literature

In this section, we introduce the protocols from [LN18] that we use in Protocol 7.

E.1 Zero Knowledge Multiplication

The goal of this section is to introduce the `ZK-Mult` protocol that is used in the `Mult` protocol against malicious adversary. It relies on several Zero-Knowledge Proofs that are going to be given afterwards in Appendix E.2. The `ZK-Mult` protocol is a small adaptation of the multiplication protocol of [LN18]. It introduces the interaction between a pair of parties P_i, P_j. Part of the public key provided by P_i is a value N_i, in this case the plaintext space for the paillier encryption scheme The Zero-knwoledge proofs can be found in [LN18]. The protocol relies on parameter t, l, s that can be tailored depending on the required level of security and on the size of q and N_i (see [LN18]). We assume that these parameters are fixed and known by both participants.

Protocol 15. `ZK-Mult`

INPUT: P_i inputs x_i,pk_i,sk_i. P_j inputs $y_j, \delta_{j\to i}, pk_i$
OUTPUT: P_i obtains $x_i y_j + \delta_{j\to i} \bmod q$

1. P_i computes $c_i = \mathtt{Enc}_{pk_i}(x_i)$.
2. P_i prepares a non-interactive-zero-knowledge range proof $\pi_{i\to j}$ proving that $x_i \in \mathbb{Z}_q$
   with soundness as long as $x_i \in [-2^{t+l} \cdot q, 2^{t+l} \cdot q]$
3. P_i sends $c_i, \pi_{i\to j}$ to P_j
4. P_j receives $c_i, \pi_{i\to j}$ and verifies $\pi_{i\to j}$.
5. P_j samples a random $\Delta_{j\to i} \in Z_{q^2 2^{t+l+s}}$ such that $\Delta_{j\to i} = \delta_{j\to i} \bmod q$.
6. P_j computes $c_{j\to j} = (c_i \oplus 2^{t+l} \cdot q) \odot y_j \oplus \Delta_{j\to i} = \mathtt{Enc}_{pk_i}(x_i \cdot y_j + 2^{t+l} \cdot q\dot{y}_j + \Delta_{j\to i})$.
7. P_j prepares a non-interactive-zero-knowledge proof $\pi_{j\to i}$ for the pair $c_{j\to i}, \Delta_{j\to i}$ proving that $y_j \in \mathbb{Z}_q$, $\Delta_{j\to i} \in Z_{q^2 2^{t+l+s}}$ and $c_{j\to j} = (c_i \oplus 2^{t+l} \cdot q) \odot y_j \oplus \Delta_{j\to i}$
   with soundness as long as $y_j \in [-2^{t+l} \cdot q, 2^{t+l} \cdot q]$ *and* $\Delta_{j\to j} \in [-2^{2t+2l+s} \cdot q^2, 2^{2t+2l+s} \cdot q^2]$
8. P_j sends $c_{j\to i}, \pi_{j\to i}$ to P_i
9. P_i verifies $\pi_{j\to i}$
10. P_i decrypts $c_{j\to i}$ and adds $(x_i + 2^{t+l} \cdot q)2^{t+l} \cdot q + 2^{2t+2l+s} \cdot q^2 \bmod N$ to obtain a value z_i
11. P_i outputs $z_i \bmod q$

E.2 Zero-Knowledge Proofs

In this section we give the Zero-Knowledge Proofs used in Protocol 15. These proofs are exactly the ones given in [LN18] and are only presented here for completeness.

Proof of correct Paillier generation. The proof used to prove that the paillier keys were generated correctly is the one described in [BGRS18].

Proof of ZK equality for Paillier and Pedersen. Efficient range proofs are based on commitments, the first step of the proof is to show that the committed value is the one encrypted under paillier. As for every commitments in this paper we use Pedersen commitment scheme. The sigma protocol is for the relation :

$$\mathcal{R}_{eq} = \left\{ \left((C, \tilde{C}, N, \tilde{N}, g, h), (x, r, \rho) \right) \middle| \; C = (1+N)^x r^N \bmod N^2 \; \wedge \; \tilde{C} = g^x h^\rho \bmod \tilde{N} \right\}$$

where $g, h \in \mathbb{Z}^\star_{\tilde{N}}$ are elements involved in the commitments $\tilde{C}$, N is the parameter for paillier encryption, $x \in \mathbb{Z}_q$ is the message encrypted under C and r, ρ are the randomness for the encryption and the commitment respectively. The following has security parameters t, l. The following sigma protocol can be transformed into a ZKP using Fiat-Shamir transform (this will be the case for all the sigma protocol presented below). The prover is P and the verifier is V.

1. **Prover's P first message:** P chooses random $\alpha \in \mathbb{Z}_{q2^{t+\ell}}, \beta \in \mathbb{Z}_N$ and $\gamma \in \mathbb{Z}_{\tilde{N}}$. Then, P computes $A = (1+N)^\alpha \beta^N \bmod N^2$ and $\beta = g^\alpha h^\gamma \bmod \tilde{N}$, and sends (A, B) to V.
2. **Verifier V's challenge:** V chooses a random $e \in \{0,1\}^{2t}$ and sends it to P.
3. **P's second message:** P computes $z_1 = \alpha + ex$ (over the integers), $z_2 = \beta r^e \bmod N$, and $z_3 = \gamma + \rho e$. P sends (z_1, z_2, z_3) to V.
4. **V's verification:** V accepts if and only if the following equations hold

$$(1+N)^{z_1} z_2^N = AC^e \bmod N^2 \quad \text{and} \quad g^{z_1} h^{z_3} = B\tilde{C}^e \bmod \tilde{N}$$

Range proof for Pedersen with Slack For this proof, there are 3 security parameters t, l, s (t, l are the same as before). We take the same notation for the elements involved in the Pedersen Commitments. The goal of this proof is to show that the committed value x lies in $\mathbb{Z}_q$. We describe a Sigma protocol for the relation

$$\mathcal{R}_{\texttt{PedRange}} = \left\{ \left((\tilde{C}, \tilde{N}, g, h), (x, \rho) \right) \middle| \; \tilde{C} = g^x h^\rho \bmod \tilde{N} \;\; \wedge \;\; x \in \mathbb{Z}_q \right\}$$

In fact the soundness of the protocol can only force that $x \in [-2^{t+l}q, 2^{t+l}q]$, and that is why the protocol is called slack.

1. **P's first message:** P chooses samples $a \to \mathbb{Z}_{2^{t+l}q}$ and $\alpha \to \mathbb{Z}_{\tilde{N}}$ and computes the commitment $A = g^a h^\alpha \bmod \tilde{N}$ and sends it to V.
2. **V's challenge:** V chooses a random $e \to \mathbb{Z}_{2^t}$ and sends it to P.
3. **P's second message:** P computes $z_1 = a + xe$ (over the integers) and $z_2 = \alpha + e\rho$. P sends (z_1, z_2) to V.
4. **V's verification:** V verifies that $z_1 \in [2^t q, 2^{t+l} q)$ and that $g^{z_1} h^{z_2} = A\tilde{C}^e \bmod \tilde{N}$.

Range proof for Paillier with Slack As described above, the range proof for Paillier combines the two previous proofs. First, the prover generates a Pedersen commitment to the encrypted value x. Then, after proving to the verifier that the committed value is indeed x using the proof of equality, the prover generates the Range proof for Pedersen with Slack that is described above.

ZK Proof of Pailler-Pedersen Range-Bounded Affine Operation Here, we describe the Sigma protocol for showing that a value D was generated by carrying on a ciphertext C a homomorphic affine operation using values y, δ such that when x was encrypted in C, D carries the encryption of $xy + \delta$. The proof works similarly to the proofs we described earlier and involves Pedersen Commitment's The relation we consider is

$$\mathcal{R}_{\texttt{AffineRange}} = \left\{ \left((C, D, N), (y, \delta) \right) \middle| \ D = C^y (1+N)^\delta \bmod N^2 \wedge y \in \mathbb{Z}_q \wedge \delta \in \mathbb{Z}_{q^2 2^{t+l+s}} \right\}$$

1. **Verifier first message:** The verifier sends the prover P parameteres $\tilde{N}, g, h$ for Pedersen commitments.
2. **First prover message:** P samples $\alpha \to \mathbb{Z}_{q 2^{t+l}}$, $\beta \to \mathbb{Z}_{q^2 2^{2t+2l+s}}$, $\rho_1, \rho_2, \rho_3, \rho_4 \to \mathbb{Z}_{\tilde{N}}$. Then, P computes $A = C^\alpha (1+N)^\beta \bmod N^2$ along with $B_1 = g^\alpha h_1^\rho \bmod \tilde{N}$, $B_2 = g^\beta h_2^\rho \bmod \tilde{N}$, $B_3 = g^y h_3^\rho \bmod \tilde{N}$, $B_4 = g^\delta h_4^\rho \bmod \tilde{N}$. P sends (A, B_1, B_2, B_3, B_4) to V.
3. **Verifier challenge :** V sends the prover a random $e \in \mathbb{Z}_t$.
4. **Second prover message :** P computes $z_1 = \alpha + ey$, $z_2 = \beta + e\delta$, $z_3 = \rho_1 + e\rho_3$, $z_4 = \rho_2 + e\rho_4$ and sends (z_1, z_2, z_3, z_4) to the verifier.
5. **Proof Verification:** V verifies that
   (a) $z_1 \in [2^t q, 2^{t+l} q)$.
   (b) $z_2 \in [q^2 2^{3t+l+s}, q^2 2^{3t+2l+s})$
   (c) $C^{z_1} (1+N)^{z_2} = A D^e \bmod N^2$
   (d) $g^{z_1} h^{z_3} = B_1 B_3^e \bmod \tilde{N}$
   (e) $g^{z_2} h^{z_4} = B_2 B_4^e \bmod \tilde{N}$