# Communication-Efficient Proactive Secret Sharing for Dynamic Groups with Dishonest Majorities

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

[1] SRI International, karim.eldefrawy@sri.com
[2] Google, tancrede@google.com
[3] École Polytechnique, antonin.leroux@polytechnique.edu

**Abstract.** In standard Secret Sharing (SS), a dealer shares a secret $s$ among $n$ parties such that an adversary corrupting no more than $t$ parties does not learn $s$, while any $t+1$ parties can efficiently recover $s$. Proactive Secret Sharing (PSS) retains confidentiality of $s$ even when a *mobile adversary* corrupts all parties over the lifetime of the secret, but no more than a threshold $t$ in each epoch (called a refresh period). Withstanding such adversaries has become of increasing importance with the emergence of settings where private (cryptographic) keys are secret shared and used to sign cryptocurrency transactions, among other applications. Feasibility of single-secret PSS for *static groups* with *dishonest majorities* was demonstrated but with a protocol that requires inefficient communication of $O(n^4)$.

In this work, we improve over prior work in three directions: *batching without incurring a linear loss in corruption threshold, communication efficiency, and handling dynamic groups*. While each of properties we improve upon appeared independently in the context of PSS and in other previous work, handling them simultaneously (and efficiently) in a single scheme faces non-trivial challenges. Some PSS protocols can handle batching of $\ell \sim n$ secrets, but all of them are for the honest majority setting. Techniques typically used to accomplish such batching decrease the tolerated corruption threshold bound by a linear factor in $\ell$, effectively limiting the number of elements that can be batched with dishonest majority. We solve this problem by reducing the threshold decrease to $\sqrt{\ell}$ instead, allowing us to deal with the dishonest majority setting when $\ell \sim n$. This is accomplished based on new bivariate-polynomials-based techniques for sharing, and refreshing and recovering of shares, that allow batching of up to $n-2$ secrets in our PSS. To tackle the efficiency bottleneck the constructed PSS protocol requires only $O(n^3/\ell)$ communication for $\ell$ secrets, i.e., an amortized communication complexity of $O(n^2)$ when the maximum batch size is used. To handle dynamic groups we develop three new sub-protocols to deal with parties joining and leaving the group.

⋆ Work was performed while the author was at SRI International.

## 1 Introduction

Secret sharing (SS) is a fundamental cryptographic primitive used to construct secure distributed protocols and systems [10,27,22,11,1,18,19,20], and in particular secure multiparty computation (MPC) [26,5,12,33,31,14,4,13,6,28,2]. In (linear) SS [35,7], a secret $s$ is encoded in a distributed form among $n$ parties such that an adversary corrupting up to $t$ parties cannot learn the secret $s$, while any $t+1$ parties can efficiently recover $s$. In some settings, SS should guarantee confidentiality of shared secrets and correctness of the computations performed on the shares, even when the protocol is run for a long time [31]. Similarly, there are settings where secret shared private keys are used sporadically over long period of time, for example to (threshold) sign cryptocurrency transactions [25,8,24,29] or in other financial applications and settings [30]. Requiring security for long durations brings forward a new challenge, as it gives a *mobile adversary* the chance to *eventually* corrupt all parties. Ensuring security against such (mobile) adversaries has therefore recently become of increased practical importance. An SS protocol that withstands such a strong adversary is called a *Proactive Secret Sharing (PSS) protocol* [31,27].

In this work, we construct an *efficient* PSS protocol with three key properties: (i) *batching*, (ii) *dynamic* groups, and (iii) *dishonest majorities*. We achieve this with new techniques based on bivariate sharing. Below, we summarize the progression from standard SS for passive adversaries, to stronger (mixed) adversary models, to PSS for dynamic groups with honest majorities, to PSS for static groups with dishonest majorities. We explain why either the tolerated threshold or the performance of existing protocols fall short in achieving the goals we strive towards.

### 1.1 Prior Work

A secret sharing (SS) protocol [35,7] typically consists of two sub-protocols: `Share` and `Reconstruct`. `Share` can be used by a dealer to share a secret $s$ among $n$ parties such that an adversary corrupting no more than $t$ parties does not learn $s$, while any $t+1$ parties can efficiently recover $s$ via `Reconstruct`. Initially, secret sharing schemes only considered (exclusively) *passive* or *active adversaries*. In the malicious setting, we say that a SS scheme is verifiable if some auxiliary information is exchanged that allows players to verify that the shares are consistent; we call the resulting scheme verifiable secret sharing (VSS). Perfectly secure or statistically secure secret sharing can only be obtained when there is a majority of honest participants. Against dishonest majorities, it is only possible to reach computationally secure protocols. Hence, all of our protocols achieve computational security.

*The mixed adversarial model and gradual secret sharing.* In [28], Hirt, Maurer, and Lucas introduced the concept of mixed adversaries in SS and multiparty computation (MPC), to capture the trade-off between passive and active adversaries. In particular, they develop an MPC protocol using *gradual* VSS against

mixed adversaries that corrupt $k$ parties actively out of less than $n-k$ corrupted parties total. One of the main benefits of gradual SS is to ensure *fairness*, i.e., if corrupted parties can deny the output of a protocol to the set of honest parties, then they cannot learn the secret themselves. The key idea is to additively share the secret $s$ (i.e., $s = \sum_{i=1}^{d} s_i$) and then linearly share each of the $s_i$ to the parties under polynomial of (gradually) increasing degrees $i = 1$ to $i = d$. In the `Reconstruct` protocol, the parties open the shares gradually, from $i = d$ to $i = 1$ and incorrect parties cannot deviate without being detected.

*Proactive secret sharing (PSS).* It may be desirable to guarantee the security of standard (and gradual) SS throughout the entire lifetime of the secret. The notion of proactive security was first suggested by Ostrovsky and Yung [31], and applied to SS in [27]. It protects against a mobile adversary that can change the subset of corrupted parties over time. Such an adversary could eventually gain control of all parties over a long enough period, but is limited to corrupting no more than $t$ parties during the same time period. In this work, we use the definition of PSS from [28,17]: in addition to `Share` and `Reconstruct`, a PSS scheme contains a `Refresh` and a `Recover` sub-protocols. `Refresh` produces new shares of $s$ from an initial set of shares. An adversary who controls a subset of the parties before the refresh and the remaining subset of parties after, will not be able to reconstruct the value of $s$. `Recover` is required when one of the participant is rebooted to a clean initial state. In this case, the `Recover` protocol is executed by all other parties to provide shares to the rebooted party. Ideally such rebooting is performed sequentially for randomly chosen parties at a predetermined rate – hence the "proactive" security idea. In addition, `Recover` could be executed after an active corruption is detected.

*Dynamic proactive secret sharing.* PSS initially only considered static groups, Dynamic Proactive Secret Sharing (DPSS) schemes are both proactively secure and allow the set of parties to dynamically change over time [16,34,37,38,3]. In other words, DPSS handles the redistribution of shares when the number of parties increases or decreases, respectively with the `Increase` and a `Decrease` sub-protocols. The authors in [3] extended the PSS introduced in [2] with ideas from [15,14,13] to produce a DPSS scheme for honest majorities. We will follow a similar approach to extend our PSS to the dynamic setting.

### 1.2 Limitations of Prior Work

Our goal is to address limitations and open problems left by prior PSS work, as shown in Table 1. First, the only PSS in the dishonest majority setting [17,21] assumes a static group of parties, i.e., unchanged during the secret lifetime. In this work, PSS protocol for *dynamic* groups with dishonest majorities. As for any secret sharing against dishonest majorities, security is only computational. Second, [17,21] do not explicitly handle batching of secrets [23], i.e., sharing, refreshing, and recovering shares of several secrets in parallel. While the authors in [17] mention a batched version of their PSS, the paper does not provide any

**Table 1.** Overview of features and limitations of current proactive secret sharing (PSS) protocols. The communication complexity is amortized over the number of secrets handled by the schemes. Note that batching is briefly mentioned in [17], but no technical details are provided. A detailed comparison of complexity of involved sub-protocols and tolerated corruption thresholds is provided in Table 2 and Appendix E.

| | PSS | Batching | Fairness | Dynamic Groups | Dishonest Majority | Communication (amortized) |
|---|---|---|---|---|---|---|
| [2] | ✓ | ✓ | ✗ | ✗ | ✗ | $O(1)$ |
| [3] | ✓ | ✓ | ✗ | ✓ | ✗ | $O(1)$ |
| [17,21] | ✓ | ✗ | ✓ | ✗ | ✓ | $O(n^4)$ |
| This work | ✓ | ✓ | ✓ | ✓ | ✓ | $O(n^2)$ |

**Table 2.** Comparison of amortized communication complexity of sub-protocols in this work and existing PSS schemes in the dishonest majority setting for $\ell$ secrets. The complexities stated in the column "Dynamic" are the worst-case complexities of three sub-protocols required to handle dynamic groups (see Section 5). We note that the complexity of the `Recover` sub-protocol is per party, and this is the bottleneck since it has to be repeated $n$ times, once when each party is (eventually) rebooted. This explaines the $O(n^4)$ overall complexity.

| | $\ell$ | `PSS` `Share` | `PSS` `Reconstruct` | `PSS` `Refresh` | `PSS` `Recover` | Dynamic `Redistribute` | Overall |
|---|---|---|---|---|---|---|---|
| [17,21] | 1 | $O(n^2)$ | $O(n^2)$ | $O(n^3)$ | $O(n^3)$ | – | $O(n^4)$ |
| **This work** | $n-2$ | $O(n)$ | $O(n^2)$ | $O(n)$ | $O(n)$ | $O(n^2)$ | $\boldsymbol{O(n^2)}$ |
| **This work** | 1 | $O(n)$ | $O(n^3)$ | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ | $\boldsymbol{O(n^3)}$ |

detail on the effect of batching on the communication complexity nor on the security impact in the mixed adversary setting. In this work, we introduce a notion of batched PSS scheme that retain fairness against mixed adversaries. Third, [17,21] has a large communication complexity, $O(n^4)$, and leaves as an open problem how to reduce the communication bottleneck in the PSS (due to the `Refresh` and `Recover` sub-protocols) to $O(n^2)$ or $O(n^3)$. Moreover, the *fairness* additional feature of [17] is costly in terms of communication and it is not clear how this additional cost can be handled. We solve these open questions by reducing the communication complexity to $O(n^2)$ in the batched setting, and $O(n^3)$ in the single secret setting. In theses improvements, we obtain *fairness* with no additional cost in asymptotic complexity.

### 1.3 Our Contributions

In this work, we develop a new efficient computationally-secure PSS protocol for dynamic groups with dishonest majorities. To achieve this, we proceed in the following steps.

*(1) Batched proactive secret sharing (without linear reduction in the corruption threshold).* The main feature of this work is a new PSS scheme of $O(n^2)$ amortized communication complexity, improving by a quadratic factor the complexity of the best known construction for dishonest majority [17]. This improvement is mainly obtained through a bivariate polynomials based batching, which deviates from how secret sharing is performed in all previous PSS schemes for dishonest majorities. While bivariate polynomials have been used before for secret sharing, we devise a new way to compute hiding bivariate polynomial (used in Protocol 2) that will enable an improvement in the communication complexity. It is well-known that linear secret sharing with threshold $t$ can be extended to share $\ell$ secrets $s_1, \ldots, s_\ell$ by sampling a random polynomial $f$ of degree $t$ such that $f(\beta_i) = s_i$ for public values $\beta_1, \ldots, \beta_\ell$ and distributing shares $f(\alpha_i)$ for $i = 1, \ldots, n$ to the $n$ parties. However, in order to learn no information about $(s_1, \ldots, s_\ell)$ an adversary must learn at most $t - \ell + 1$ evaluations of $f$, which yields a secret sharing scheme with threshold $t - \ell + 1$. To remove this linear dependency, we revisit the idea of using secret sharing with bivariate polynomials (e.g., [36]). Our new way to construct secret sharing from bivariate polynomials preserves secrecy with a corruption threshold of $t - \sqrt{\ell} + 1$ for any $\ell \leq n - 2$. This yields a batched PSS scheme with sublinear reduction in the corruption threshold.[4] Since gradual secret sharing consists of a ladder of polynomial shares, the same linear dependency in the number of secrets being batched applies to the mixed adversarial model considered in [28], which then becomes secure against mixed adversaries that corrupt $k$ parties actively out of less than $n - k - \ell$ corrupted parties total. Similarly, we introduce the notion of Batched gradual VSS, a batched generalization of gradual VSS [28] which is secure against adversaries corrupting either $t - \lfloor\sqrt{\ell}\rfloor$ parties passively, or $(n - \lfloor\sqrt{\ell}\rfloor)/2 - 1$ parties actively, or $k$ parties actively out of $n - k - \lfloor\sqrt{\ell}\rfloor$. The gradual Secret Sharing aims at obtaining fairness during the reconstruction. We note that our gradual bacthed PSS obtains this fairness property without any additional asymptotic cost.

*(2) Accommodating dynamic groups.* We extend the batched PSS scheme above with a new `Redistribute` protocol. It is built from three new efficient protocols to handle dynamic groups: `Increase`, `Decrease`, and `DecreaseCorrupt`. The $\texttt{Increase}_k$ protocol addresses the case where the number of parties increases: it efficiently transforms the shares for $n$ parties held by $P_1, \ldots, P_n$ to shares for $n + k$ parties distributed to $P_1, \ldots, P_{n+k}$, and has $O(k(n + k))$ communication complexity per secret. Similarly, $\texttt{Decrease}_k$ produces shares for a sharing between $n - k$ parties and has a complexity per secret of $O(k(n - k))$; however it requires the $k$ leaving parties to participate in the protocol. For the case where some of the parties are leaving due to permanent corruption or physical destruction, we propose a third protocol `DecreaseCorrupt`. This protocol produces new

[4] In Appendix C, we give a self-contained description of the special case of $\ell = 1$ that improves the communication complexity of the gradual VSS of [17] by $O(n)$ without any decrease in the corruption threshold.

shares for the remaining parties without interacting with the leaving party (in this case we can handle only $k = 1$).

*(3) Efficient communication.* The techniques used in the previous techniques were also carefully designed to limit the communication complexity. As shown in Table 2, our (fully) batched PSS for dynamic groups with dishonest majorities has an overall complexity[5] of $O(n^2)$ per secret.

### 1.4 Intuition Behind New Techniques for the Proactive Setting

We summarize here the main intuition behind the techniques that enable our performance and threshold improvements outlined in Section 1.3.

*(1) Addressing the Recover Bottleneck in PSS.* As mentioned above, the real bottleneck of PSS is the `Recover` protocol. This protocol is costly in itself and is performed regularly on each of the participants (adding a $O(n)$ complexity factor to the overall communication complexity). The main issue is the inability to efficiently generate a set of blinding polynomials. We revisit the `Recover` protocol to overcome this limitation by optimizing the number of blinding polynomials generated. This improvement is made possible by the use of bivariate polynomials. The `Recover` protocol is also a necessary building block for the `Refresh` protocol and the "gradual" `Reconstruct` protocol (see item 3 below), as it enables a subset of participants to generate random polynomials and share them with the rest of the parties.

*(2) Batching with Bivariate Polynomials:* Batching $O(n)$ secrets saves $O(n)$ in communication complexity, but usually reduces the threshold by a linear factor proportional to the number of batched secrets. This severely limits the number of elements one can batch. We use bivariate polynomials to perform sharing (of a batch of secrets) instead of univariate polynomials. As mentioned above, the real bottleneck in this protocol is the generation of blinding polynomials in `Recover` that protects the secrets without changing their values. We develop a new technique to generate these polynomials in $O(n^2)$ with the number of hiding values being quadratic in $n - t_P$. To have information theoretic security for the batched secrets we need this term $(n - t_P)^2$ to be greater than $\ell$. This leads to only a sub-linear $\sqrt{\ell}$ reduction in the threshold, as opposed to linear in $\ell$. Note that our generation of hiding bivariate polynomial is optimal. Indeed, this hiding polynomial has degree $d$ and the data size of a bivariate polynomial of this degree is $O(n^2)$ when we take $d = O(n)$ (in practice, we take $d = n - 2$ for maximum security). Hence, our technique cannot yield a protocol with better communication complexity than $O(n^2)$.

[5] For simplicity of notation we often write complexity instead of amortized complexity.

*(3) Gradual Property only Needed in Reconstruction.* We also observe that, in previous work, the "gradual" feature of the underlying secret sharing scheme (to withstand dishonest majorities) is critically used during the `Reconstruct` operation only. We will therefore work only with regular shares. To recreate a gradual secret sharing, we develop a new (gradual technique at the core of the) `Reconstruct` protocol that creates directly a ladder of blinding polynomials that sums to 0, adds the shares of the first element of the ladder, and then gradually reveals everything while preserving confidentiality of the secrets. At the bottom layer, what is revealed is the actual secret because all the blinding polynomials of the ladder add up to 0. This enables us to save an additional factor (after batching) of $O(n)$ in `Reconstruct`. This results in a final complexity of $O(n^2)$ for the `Reconstruct` which was the bottleneck as shown in Table 2. This also implies that we can obtain *fairness* during the reconstruction without increasing the communication complexity.

*(4) Dealing with Dynamic Groups.* In linear sharing via polynomials, the degree $d$ is the main security parameter. This degree $d$ can be computed from the value of $n$. In the case of dishonest majority, the value $d = n - 2$ is close to $n$. Any change in the number of participant implies the necessity to change the degree of the sharing polynomials. The extension to dynamic groups with the `Redistribute` protocol is achieved with two sub-protocols `Increase` and `Decrease` that respectively increases and decreases the degree of the sharing given as input without changing the value of the secret. To obtain efficient protocols, we will take polynomials of the correct degree as close as possible from the input polynomials.

### 1.5 Outline

This paper is organized as follows: Section 2 overviews preliminaries required for the rest of the paper. Section 3 gives the formal definition of batched dynamic proactive secret sharing, i.e., multi-secret PSS for dynamic groups with dishonest majorities. Section 4 proposes a concrete efficient instantiation of a batched (static) proactive secret sharing using buivariate polynomials. Section 5 then extends the above PSS scheme to deal with dynamic groups. Full correctness and security proofs of the protocols are provided in the Supplementary Material.

## 2 Preliminaries

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 share and proactively maintain a confidential secret $s$ 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 $v \leftarrow S$.

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

### 2.1 Mixed Adversaries

We first recall the model of mixed adversaries from [28]; we consider a central adversary $\mathcal{A}$ with polynomially bounded computation power who corrupts some parties passively (i.e., $\mathcal{A}$ learns the view of a $P_i$) and actively (i.e., $\mathcal{A}$ makes a $P_i$ behave arbitrarily) during a stage $\sigma$. We denote by $\mathcal{P}_P$ (resp. $\mathcal{P}_A \subseteq \mathcal{P}_P$) the set of passively (resp. actively) corrupted parties and denote by $t_P$ (resp. $t_A$) its cardinality. A multi-threshold is a set of pairs of thresholds $(t_1, t_2)$. We say that $(t_P, t_A) \leq T$ for a multi-threshold $T$ if there exists $(t_1, t_2) \in T$ such that $t_P \leq t_1$ and $t_A \leq t_2$. For two multi-thresholds $T_a, T_b$ we say that $T_a \leq T_b$ if for all $(t_{a1}, t_{a2}) \in T_a$, it holds that $(t_{a1}, t_{a2}) \leq T_b$.

### 2.2 Security Properties

Throughout the paper, we study four security properties: *correctness, secrecy, robustness, and fairness*. We denote the corresponding multi-thresholds $T_c$, $T_s$, $T_r$, and $T_f$. Each property is considered guaranteed if $(t_P, t_A)$ is smaller than the corresponding multi-threshold. These properties are standard analytic tools for protocols security. For a protocol $\Pi$:

- *Correctness*: Given the inputs from $P_1, .., P_n$, each party engaged in $\Pi$ either obtains the correct output or obtains a special output $\perp$.
- *Secrecy*: The adversary cannot learn more information on the other parties' inputs and outputs than can be learned from its own inputs and outputs.
- *Robustness*: The adversary cannot deny their output to the honest parties.
- *Fairness*: Either every party obtains its output or nobody does.

We have $T_r \leq T_c$ and $T_f \leq T_s \leq T_c$ since we cannot define secrecy, fairness or robustness without correctness and secrecy is required by fairness. Note that all the protocols in this work are not robust when there are more than a few (generally 1 or 2) active corruptions. Thus, we are not going to study the robustness of our protocols as they will not provide it in most cases. As such, unless explicitly specified, the robustness threshold is $T_r = \{(n, 1)\}$.

### 2.3 (V/P/DP) Secret Sharing Definitions

*Verifiable Secret Sharing (VSS).* A VSS scheme enables an (untrusted) dealer to securely share a secret $s$ among the parties in $\mathcal{P}$, such that a set of honest parties can reconstruct $s$ if they reveal their shares to each other.

**Definition 1 (Verifiable Secret Sharing [28]).** *A $(T_s, T_r)$-secure Verifiable Secret Sharing (VSS) scheme is a pair of protocols* `Share` *and* `Reconstruct`*, where* `Share` *takes inputs $s$ from the dealer and* `Reconstruct` *outputs $s$ to each party, if the following conditions are fulfilled:*

- Secrecy*: if $(t_P, t_A) \leq T_s$, then in* `Share` *the adversary learns no information about $s$;*
- Correctness*: After* `Share`*, the dealer is bound to the values $s'$, where $s' = s$ if the dealer is honest. In* `Reconstruct`*, either each honest party outputs $s'$ or all honest parties abort.*

– Robustness*: the adversary cannot abort* `Share`*, and cannot abort* `Reconstruct` *if* $(t_P, t_A) \leq T_r$.

*Proactive Secret Sharing (PSS).* A PSS scheme is a VSS scheme secure against a proactive adversary. We recall the definition of PSS from [17]. In particular, a PSS scheme is a VSS scheme extended with two additional sub-protocols: `Refresh` and `Recover`. An execution of PSS will be divided in phases. A refresh phase (resp. recovery phase) is the period of time between two consecutive executions of `Refresh` (resp. `Recover`). Furthermore, the period of time between `Share` and the first `Refresh` (resp. `Recover`) is a refresh phase (resp. recovery phase), and similarly for the period of time between the last `Refresh` (resp. `Recover`) and `Reconstruct`.

**Definition 2 (Proactive Secret Sharing [17]).** *A Proactive Secret Sharing (PSS) scheme consists of four protocols* `Share`*,* `Reconstruct`*,* `Refresh`*, and* `Recover`*.* `Share` *takes inputs* $s$ *from the dealer and* `Reconstruct` *outputs* $s'$ *to each party.* `Refresh` *is executed between two consecutive phases* $\sigma$ *and* $\sigma + 1$ *and generates new shares for phase* $\sigma + 1$ *that encode the same secrets as the shares for phase* $\sigma$*.* `Recover` *allows parties that lost their shares to obtain new shares encoding* $s$ *with the help of the other honest parties. A* $(T_s, T_r, T_c)$*-secure PSS scheme fulfills the following conditions:*

– Termination*: all honest parties complete each execution of* `Share`*,* `Refresh`*,* `Recover`*, and* `Reconstruct`*.*
– Secrecy*: if* $(t_P, t_A) \leq T_s$*, then in* `Share` *the adversary learns no information about* $s$*. If* $(t_P, t_A) \leq T_s$ *in both phases* $\sigma$ *and* $\sigma + 1$*, and if* `Refresh` *and* `Recover` *are run between phases* $\sigma$ *and* $\sigma + 1$*, then the adversary learns no information about* $s$*.*
– Correctness*: After* `Share`*, the dealer is bound to the values* $s'$*, where* $s' = s$ *if the dealer is honest. If* $(t_P, t_A) \leq T_c$*, upon completing* `Refresh` *or* `Recover`*, either the shares held by the parties encodes* $s$ *or all honest parties abort. In* `Reconstruct` *either each honest party outputs* $s'$ *or all honest parties abort.*
– Robustness*: the adversary cannot abort* `Share`*, and cannot abort* `Refresh`*,* `Recover`*, and* `Reconstruct` *if* $(t_P, t_A) \leq T_r$*.*

*Dynamic Proactive Secret Sharing (DPSS).* A DPSS scheme is a PSS scheme extended by a `Redistribute` protocol that enables (secure distributed) transfer of the secret $s$ from one group of participants to another. Our DPSS definition is inspired by a previous one in [3]. The only difference is that we do not combine `Refresh`, `Recover` and `Redistribute` into one phase. We define a redistribute phase analogously to the refresh and recover phases. The refresh phases are denoted by $\sigma$, the redistribute phases by $\omega$, $n^{(\omega)}$ is the number of participants at phase $\omega$. The multi-thresholds $T_r, T_c, T_s$ are considered as functions of $n$ (the number of participants). We denote $T_r^{(\omega)}, T_c^{(\omega)}, T_s^{(\omega)}$ the thresholds at phase $\omega$ computed from $n^{(\omega)}$.

**Definition 3 (Dynamic Proactive Secret Sharing).** *A Dynamic Proactive Secret Sharing (DPSS) scheme consists of a PSS constituted of four protocols*

*`Share`, `Reconstruct`, `Refresh`, `Recover` according to Definition 2 completed by a `Redistribute` protocol. `Redistribute` is executed between consecutive redistribute phases $\omega$ and $\omega+1$ and allows a set of $n^{(\omega)}$ participants at phase $\omega$ to transfer its shares to the set of $n^{(\omega+1)}$ participants of phase $\omega+1$ . In the following, when we denote $(t_P, t_A) \leq T_s^{(\omega)}$, it is implicit that this is true during redistribute phase $\omega$. A $(T_s, T_r, T_c)$-secure DPSS scheme fulfills the following conditions:*

- *For any phase $\omega$, `Share`, `Reconstruct`, `Refresh` and `Recover` is a $(T_s^{(\omega)}, T_r^{(\omega)}, T_c^{(\omega)})$-secure PSS under Definition 2.*
- Termination*: all honest parties complete each execution `Redistribute`.*
- Secrecy*: if $(t_P, t_A) \leq T_s^{(\omega)}$ and $(t_P, t_A) \leq T_s^{(\omega+1)}$, the adversary learns no information about $s$ during the execution of `Redistribute` between phases $\omega$ and $\omega+1$.*
- Correctness*: After `Share`, the dealer is bound to the values $s'$, where $s' = s$ if the dealer is honest. If $(t_P, t_A) \leq T_c^{(\omega)}$, upon completing `Redistribute`, either the shares held by the parties encodes $s$ or all honest parties abort.*
- Robustness*: the adversary cannot abort `Redistribute` if $(t_P, t_A) \leq T_r^{(\omega)}$.*

### 2.4 Homomorphic Commitments and VSS

To obtain security against a malicious adversary, we use an homomorphic commitment scheme, e.g., Pedersen commitments [33]. We assume that all the values (secrets, polynomial coefficients) are in $\mathbb{Z}_q$ for a prime $q$ and that a cyclic group $\mathbb{G}$ of order $q$ with two random generators $g, h$ has been distributed to each party. The commitment to a secret $m$ is $C(m, r) = g^m \cdot h^r$ for a random value $r$. Further explanations on Perdersen's commitments are given in Appendix A.1. Due to the use of the Pedersen commitment scheme, our protocols are computationally secure under the Discrete Logarithm Problem (DLP) hardness assumption.

### 2.5 Bivariate Polynomials

We rely on bivariate polynomials as a building block in our design of a batched secret sharing scheme for groups with dishonest majorities. We use polynomials of degree $d$ in both $x$ and $y$ variables. Such a polynomial $g$ is uniquely defined by $(d+1)^2$ points $g(x, y)$ with $(x, y) \in X \times Y$ and $|X| = |Y| = d+1$. Indeed, for any $(x_0, y_0)$, the value $g(x_0, y_0)$ can be found by the interpolation of $g(x, y_0)$ for all $x \in X$. The values $g(x, y_0)$ can be interpolated with $g(x, y)$ for all $y \in Y$. In the following, when we say that $g$ is a bivariate polynomial of degree $d$, it means that $g$ is of degree $d$ in both its variables.

## 3 Batched Proactive Secret Sharing for a Static Group with a Dishonest Majority

In this section, we introduce the definition of (Dynamic) Batched Proactive Secret Sharing (BPSS). In order to understand why the batched setting requires

new definitions, we first explain the issue arising when using batching in PSS against mixed adversaries in Section 3.1. Then, we introduce the definition of a $\ell$-Batch $\ell'$-Gradual Secret Sharing in Section 3.2, and the definition of (Dynamic) Batched Proactive Secret Sharing in Section 3.3.

### 3.1 The Issue of Number of Secrets

Recall the naive version of Shamir's $(t, n)$-secret sharing [35] for $t < n$: a secret $s \in \mathbb{F}$ is stored in the constant coefficient $f(0) \coloneqq s$ of a polynomial $f \in \mathbb{P}_t$. Each party $P_r$ for $r \in [n]$ will receive $f(\alpha_r)$ where the $\alpha_j$'s are (public) distinct nonzero elements and reconstruction is performed by interpolating of the value in 0 using $t + 1$ evaluations of $f$.

The extension the above secret sharing scheme to handle batching is a well-known construction [23]: to share $\ell$ secrets $s_1, \ldots, s_\ell$, sample a polynomial $f \in \mathbb{P}_{t+\ell-1}$ such that $f(i) = s_i$ and set $\alpha_r \notin [\ell]$. However, now one must ensure that $t + \ell - 1 < n$ so that $(s_1, \ldots, s_\ell)$ remains information-theoretically hidden given up to $t$ evaluations of $f$ in the $\alpha_r$'s; i.e., there is a linear dependency between the number of shared batched secrets and the bound on the tolerated corruption threshold (with respect to $n$).

Now, let us recall the core idea from [28] to design a fair secret sharing scheme against mixed adversaries. We consider Shamir's secret sharing extended with homomorphic commitments in order to provide verifiability [32]. Now, during the reconstruction step, all correct parties broadcast their shares, and secrecy is given up against all subsets at one. Therefore, the reconstruction protocol does not achieve fairness (that is, every party obtains its output or nobody does). In order to achieve fairness and handle mixed adversaries, Hirt et al. [28] propose to first split the secret into additive summands, i.e.,

$$s = s^{(1)} + \cdots + s^{(d)} ,$$

with $d = n - 1$ and then use Shamir's $(i, n)$-secret sharing on $s^{(i)} = f_i(0)$ for all $i \in [d]$. Next, $P_r$ for $r \in [n]$ receives as share the tuple

$$(f_1(\alpha_r), \ldots, f_d(\alpha_r)) .$$

Reconstruction then recovers each of the $s^{(i)}$ for $i$ from $d = n-1$ to 1 sequentially. If there is a violation of fairness at any step, i.e., an $s^{(i)}$ cannot be reconstructed, the protocol `aborts`. A mixed adversary cannot abort before the degree $i_0 = t_P$ (for $i < t_P$ the adversary already knows all the values $f_i(0)$). In this case, to preserve fairness the honest parties need to be able to recover all the remaining values $f_i(0)$. Thus we have $i_0+1 \leq n-t_A$. By putting the two constraints together we obtain the bound $(t_P, t_A) \leq (n - k - 1, k)$. Additionally, since $t_A \leq t_P$, we get $k \leq \lceil \frac{n}{2} \rceil - 1$.

Now, assume we want to design a batched secret sharing scheme against mixed adversaries. Combining the above arguments prevents a mixed adversary from aborting before the degree $i_0 = t_P + \ell - 1$ and therefore we obtain the bound

$$(t_P, t_A) \leq (n - k - \ell, k) .$$

In particular, this implies that as soon as one batches $\ell \geq n/2$ secrets, achieving security with a dishonest majority is not attainable.

To overcome this issue, we introduce a notion of $\ell$-Batch $\ell'$-Gradual Secret Sharing against mixed adversaries with bound $(t_P, t_A) \leq (n - k - \ell', k)$ in Section 3.2; and then similarly to [17], it is easy to extend the latter primitive to define a Batched PSS against mixed adversaries. In Section 4, we will instantiate such a primitive for $\ell \leq n - 2$ and $\ell' = \lfloor \sqrt{\ell} \rfloor$ by revisiting the idea of secret sharing using bivariate polynomials (e.g., [36]).

### 3.2 Batched Gradual Secret Sharing Against Mixed Adversaries

We first recall the definition of Gradual Secret Sharing from [28].

**Definition 4 (Gradual VSS [28]).** *A $(T_s, T_r, T_c)$-secure VSS scheme is gradual if the following conditions are fulfilled: If* `Reconstruct` *aborts, each party outputs a non-empty set $B \subset \mathcal{P}_A$ and the adversary cannot obtain information about the secret $s$ if $(t_P, t_A) \leq T_s$ and $t_P \leq n - |B| - 1$.*

In particular, this definition is equivalent to fairness when the adversary is bounded by a multi-threshold $T_f = \{(n - k - 1, k) : \ k \in [0, \lceil \frac{n}{2} \rceil - 1]$ and $(n - k - 1, k) \leq T_s\}$.

*Batched Gradual VSS.* We naturally extend Definition 1 (definition of a VSS) and Definition 4 to batch $\ell$ secrets. A Batch VSS scheme enables a dealer to share $\ell$ secrets $s_1, \ldots, s_\ell$ among the parties in $\mathcal{P}$, such that the parties can reconstruct the secrets.

**Definition 5 ($\ell$-Batch VSS).** *A $(T_s, T_r)$-secure $\ell$-Batch VSS scheme is a pair of protocols* `Share` *and* `Reconstruct`*, where* `Share` *takes inputs $s_1, \ldots, s_\ell$ from the dealer and* `Reconstruct` *outputs $s'_1, \ldots, s'_\ell$ to each party, if the following conditions are fulfilled:*

- Secrecy*: if $(t_P, t_A) \leq T_s$, then in* `Share` *the adversary learns no information about $s_1, \ldots, s_\ell$;*
- Correctness*: After* `Share`*, the dealer is bound to the values $s'_1, \ldots, s'_\ell$, where $s'_i = s_i$ if the dealer is honest. In* `Reconstruct`*, either each honest party outputs $s'_1, \ldots, s'_\ell$ or all honest parties abort.*
- Robustness*: the adversary cannot abort* `Share`*, and cannot abort* `Reconstruct` *if $(t_P, t_A) \leq T_r$.*

**Definition 6 ($\ell$-Batch $\ell'$-Gradual VSS).** *A $(T_s, T_r, T_c)$-secure $\ell$-Batch VSS is $\ell'$-gradual if the following conditions are fulfilled: If* `Reconstruct` *aborts, each party outputs a non-empty set $B \subset \mathcal{P}_A$ and the adversary cannot obtain information about the secret $s$ if $(t_P, t_A) \leq T_s$ and $t_P \leq n - |B| - \ell'$.*

Once again, this definition is equivalent to fairness when the adversary is bounded by a multi-threshold $T_f = \{(n-k-\ell', k) : k \in [0, \lceil \frac{n-\ell'}{2} \rceil - 1]$ and $(n-k-\ell', k) \leq T_s\}$.

### 3.3 Batched Dynamic Proactive Secret Sharing

We directly adapt Definition 3 to the batch setting.

**Definition 7 (Batched Dynamic Proactive Secret Sharing).** *A Batched Dynamic Proactive Secret Sharing (BDPSS) scheme consists of five protocols* `Share`*,* `Reconstruct`*,* `Refresh`*,* `Recover` *and* `Redistribute`*, where* `Share` *takes inputs* $s_1, \ldots, s_\ell$ *from the dealer and* `Reconstruct` *outputs* $s'_1, \ldots, s'_\ell$ *to each party.* `Refresh` *is executed between two consecutive refresh phases* $\sigma$ *and* $\sigma + 1$ *and generates new shares for phase* $\sigma + 1$ *that encode the same secrets as the shares for phase* $\sigma$*.* `Recover` *allows parties that lost their shares to obtain new shares encoding* $s$ *with the help of the other honest parties.* `Redistribute` *is executed between consecutive redistribute phases* $\omega$ *and* $\omega + 1$ *and allows a set of* $n^{(\omega)}$ *participants at phase* $\omega$ *to transfer its shares to the set of* $n^{(\omega)+1}$ *participants of phase* $\omega + 1$ *. In the following, when we denote* $(t_P, t_A) \leq T_s^{(\omega)}$*, it is implicit that this is true during redistribute phase* $\omega$*. A* $(T_s^{(\omega)}, T_r^{(\omega)}, T_c^{(\omega)})$*-secure PSS fulfills the following conditions:*

- Termination*: all honest parties complete each execution of* `Share`*,* `Refresh`*,* `Recover`*,* `Redistribute` *and* `Reconstruct`*.*
- Secrecy*: if* $(t_P, t_A) \leq T_s^{(\omega)}$*, then in* `Share` *the adversary learns no information about* $s_1, \ldots, s_\ell$*. If* $(t_P, t_A) \leq T_s^{(\omega)}$ *in both refresh phases* $\sigma$ *and* $\sigma + 1$*, and if* `Refresh` *and* `Recover` *are run between phases* $\sigma$ *and* $\sigma + 1$*, then the adversary learns no information about* $s_1, \ldots, s_\ell$*. If* $(t_P, t_A) \leq T_s^{(\omega)}$ *and* $(t_P, t_A) \leq T_s^{(\omega+1)}$*, the adversary learns no information about* $s_1, \ldots, s_\ell$ *during the execution of* `Redistribute` *between phases* $\omega$ *and* $\omega + 1$*.*
- Correctness*: After* `Share`*, the dealer is bound to the values* $s'_1, \ldots, s'_\ell$*, where* $s'_i = s_i$ *if the dealer is honest. If* $(t_P, t_A) \leq T_c^{(\omega)}$*, upon completing* `Refresh`*,* `Redistribute` *or* `Recover`*, either the shares held by the parties encodes* $s_1, \ldots, s_\ell$ *or all honest parties abort. In* `Reconstruct` *either each honest party outputs* $s'_1, \ldots, s'_\ell$ *or all honest parties abort.*
- Robustness*: the adversary cannot abort* `Share`*, and cannot abort* `Refresh`*,* `Recover`*,* `Redistribute` *and* `Reconstruct` *if* $(t_P, t_A) \leq T_r^{(\omega)}$*.*

## 4 Efficient Batched PSS using Bivariate Polynomials

We defer the ideal functionality definitions of the `Share`, `Reconstruct`, `Refresh`, and `Recover` subprotocols, and their formal simulator-based security proofs, to Appendix B. In this section, we introduce the protocols and prove in preliminary lemmas the core elements of their security proofs.

In the protocols below, we highlight the critical steps using boxes, as the full protocols includes (standard) use of commitments and openings to resist against malicious/mixed adversaries.

### 4.1 The `Share` Protocol

We assume that $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_\ell \in \mathbb{F}$ are distinct public values. The number $\ell$ is assumed to be smaller than $d$, the degree of the bivariate polynomial produced by the sharing. With $d = n-2$ in practice, we have the bound $\ell \leq n-2$ that we mentioned above.

**Protocol 1**. `Share`

INPUT: Secrets $s_1, \ldots, s_\ell$ held by a dealer $P_{\mathcal{D}}$.
OUTPUT: Each party $P_r$ holds shares $\{g(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}$ of the secrets $s_1, \ldots, s_\ell$ (and the corresponding commitments).

1. For $j \in [\ell]$, the dealer samples $\boxed{f_j \leftarrow \mathbb{P}_d \text{ such that } f_j(\beta_j) = s_j}$.
2. For $r \in [d+1]$, the dealer samples $\boxed{g(\alpha_r, \cdot) \leftarrow \mathbb{P}_d}$ such that $\forall j \in [\ell]$, $\boxed{g(\alpha_r, \beta_j) = f_j(\alpha_r)}$.
   *(Note that this implicitly defines a bivariate polynomial $g$ of degree $d$.)*
3. The dealer $\boxed{\text{interpolates } g(x,y)}$ and $\boxed{\text{computes } \{g(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}}$ for all $r \in [n]$.
4. The dealer broadcasts (homomorphic) commitments of the $g(\alpha_r, \alpha_{r'})$ for all $r, r' \in [d+1]$.
5. Each party $P_r$ locally computes commitments for $\{g(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}$ (using the homomorphic property for $r > d+1$), and the dealer sends the corresponding opening informations to party $P_r$. Each party broadcasts a complaining bit indicating if an opening received from the dealer is incorrect.
6. For each element $g(\alpha_r, \alpha_{r'})$ for which a complaint was broadcast, the dealer broadcasts its opening. If the opening is correct, $P_r$ accepts the value, otherwise the dealer is disqualified.

**Lemma 1.** *Let $d \leq n-1$. `Share` is correct and preserves the secrecy of a batch of secrets $s_1, \ldots, s_\ell$ if $(t_P, t_A) \leq \big\{(d,d)\big\}$.*

*Proof.* Correctness follows from the use of homomorphic commitments which allow the parties to verify that the dealer distributed shares for a bivariate polynomial $g$ of degree $d$ in both variables.

For secrecy, we show that the adversary cannot find the values $s_1, \ldots, s_\ell$ when $t_P \leq d$. Without loss of generality, we assume that the adversary controls passively $\{P_1, \ldots, P_{t_P}\}$ and that the dealer is honest. Hence, the adversary knows the values $\{g(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}$ for $r \in [t_P]$. It can interpolate $g(\alpha_r, \beta_j) = f_j(\alpha_r)$ for all $r \in [t_P]$ and $j \in [\ell]$. For every $j$, since $t_P \leq d$, $f_j(\beta_j) = s_j$ is information-theoretically hidden. □

*Remark 1 (Communication Complexity).* In Step 4, the dealer broadcasts $(d+1)^2$ commitments, and in Step 5, $(d+1) \cdot n$ messages are sent. With $d = O(n)$, we obtain an amortized communication complexity of $O(n^2)/\ell$.

*Remark 2 (Corruption Threshold).* Lemma 1 claims security for up to $d$ corruption when we mentioned several time already that our protocol is secure up to $d+1-\sqrt{\ell}$. This is because the `Share` protocol in itself tolerates more corruptions. The threshold $d+1-\sqrt{\ell}$ is a consequence of the `Recover` protocol, as is explained below.

### 4.2 The `Recover` Protocol

The `Recover` protocol enables a set of $d+1$ parties $\{P_1,\ldots,P_{d+1}\}$ to send to a recovering party $P_{r_C}$ its shares $(g\{\alpha_{r_C},\alpha_{r'})\}_{r'\in[d+1]}$. In [17], to perform the recovery of one value $f(\alpha_{r_C})$, each participant $P_r$ generates one blinding polynomial $f_r$ verifying $f(\alpha_{r_C})=0$ and share it among the other participants so that $P_{r_C}$ can receive $f(\alpha_r)+\sum_{u=1}^{n} f_u(\alpha_r)$ for $r\in[n]$ and interpolate $f(\alpha_{r_C})$. This is inefficient as each value $f(\alpha_r)$ requires $O(n)$ communication to be blinded. In our secret sharing, each participant $P_r$ have a polynomial $g(\alpha_r,\cdot)$. Just like in [17], our `Recover` protocol requires each $P_r$ to generate one polynomial $f_r$ verifying $f_r(\alpha_{r_C})=0$ and share it to the other. The number of blinding polynomials remains the same, but the size of the sharing has been multiplied by a factor $O(n)$, it yields an optimal $O(1)$ communication complexity per value. Yet, it will be enough to blind the batch of $\ell$ secrets when the corruption threshold is decreased to $d+1-\sqrt{\ell}$. Indeed, $P_{r_C}$ is going tor receive the values $g(\alpha_r,\alpha_{r'})+f_{r'}(\alpha_r)$ from each of the $P_r$ for $r'\in[d+1]$. When $P_{r'}$ and $P_{r_C}$ are corrupted, the adversary will be able to learn the values $g(\alpha_r,\alpha_{r'})$ for $r\in[d+1]$ that were unknown to the adversary prior to `Recover`. However, when both $P_r$ and $P_{r'}$ are honest, the value $g(\alpha_r,\alpha_{r'})$ is blinded by $f_{r'}(\alpha_r)$. Therefor, the security of the $\ell$ secrets is going to be protected by the $(d+1-t_P)^2$ values corresponding to pairs $(P_r,P_{r'})$ of honest participants in $\mathcal{P}^2$. That yields the bound $t_P\leq d+1-\sqrt{\ell}$. The formal security analysis of `Recover` is given in Appendix B.2.
Overall, our `Recover` protocol consists of the following steps:

(a) First, the set of parties jointly generate random univariate polynomials $f_1,\ldots,f_{d+1}$ of degree $d$ that evaluates to 0 in $\alpha_{r_C}$.
(b) Then, every party uses its shares of $f_{r'}$'s to randomize its shares $g(\alpha_r,\alpha_{r'})$ so that $P_{r_c}$ can interpolate $g(\alpha_{r_C},\alpha_{r'})$ for $r'\in[d+1]$.

**Protocol 2**. `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 $r' \in [d+1]$.

*Remark 3 (Communication Complexity).* In Step 1, $(d+1)^2$ commitments are broadcast. In Step 2, $(d+2)(d+1)$ openings are sent. In Step 5, $(d+1)^2$ openings are sent. With $d = O(n)$, we obtain an amortized communication complexity of $O(n^2)/\ell$.

### 4.3 The `Reconstruct` Protocol

Recall that gradual verifiable secret sharing was introduced in [28] to capture the notion of a mixed adversary by gradually reducing the number of corrupted parties against which secrecy is guaranteed during reconstruction, and at the same time increasing the number of corrupted parties against which robustness is guaranteed. In particular, in [28] a secret $s$ is split into summands $s = s_1 + \cdots + s_d$ and each $s_i$ is secret shared using a polynomial of degree $i$. During reconstruction, the protocol aborts at step $n-k$ only if strictly less than $n-k+1$ parties opened their commitments correctly and therefore the number of active parties is lower bounded by $k$. Now, if the total number of corruptions is less than $n-k$, then the adversary learns nothing, which retains secrecy against adversaries controlling $k$ parties actively out of $n-k$ compromised parties.

Now, let's assume we instead have a sharing of $0 = e_1 + \cdots + e_d$ (as polynomials), where $e_1, \ldots, e_{d-1}$ are bivariate polynomials of degrees $1, \ldots, d-1$ respectively. Then the above protocol can be reproduced with $s_i = e_i(\beta)$ for

$i < d$ and $s_d = s + e_d(\beta)$; this is the core idea in the protocol below. The core novelty of the protocol is in how to construct this ladder. We will show that by using (i) some fixed public values $\lambda_1, \ldots, \lambda_d$ such that $\sum_{i=1}^{d} \lambda_i = 0$ and (ii) the `Recover` above to share freshly generated polynomials, gradually constructing such a ladder is possible. The key idea is the following: at each step from $i = d$ to $i = 2$, the current bivariate polynomial of degree $i$ is blinded by a random bivariate polynomial of degree $i-1$ *generated by a subset of size $i$ of the parties and recovered with a $i+1$-th party using* `Recover`. All the blinding polynomials $e_i$ will be constructed so that

$$e_1 + \cdots + e_d = \Big(\sum_{k=1}^{d} \lambda_k\Big) \cdot Q\,,$$

at the end of the protocol for $Q$ a random bivariate polynomial, so that $\sum_{k=1}^{d} \lambda_k = 0$ can eventually be factored out. Note that it does not harm the security to take public $\lambda_i$ values. Indeed, the security requires that each of the $s_i$ appears uniformly random (up to $s_1$ that depends on $s$ and the previous $s_i$). The way that each $g_i$ is constructed from the $Q_i$ polynomials that are random polynomials ensures this property.

The `Reconstruct` protocol is described in Protocol 3, and its correctness and security proofs can be found in Appendix B.3.

**Protocol 3**. `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 $(\lambda_k)_{1 \le k \le d}$ 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*: While $i \geq 2$:
   (a) Wlog, assume $\mathcal{P} = \{P_1, \ldots, P_N\}$. For $r \in [i+1]$, $P_r$ broadcasts openings to $\{g_i(\alpha_r, \alpha_{r'})\}_{r' \in [i+1]}$, 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) For $r \in [i+1, N]$, $P_r$ 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) All parties interpolate $g_i$ and update $s_j \leftarrow s_j + g_i(\beta_j, \beta_j)$.
   Set $i \leftarrow i - 1$.
   (d) If $i = 1$, sets $Q_0 = 0$ and go to Step (f).
   Else, for $r \in [i]$, $P_r$ 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.

*Remark 4.* We reiterate that we have the invariant

$$\sum_{k=i}^{d} g_k = g + \left(\sum_{k=i}^{d} \lambda_k\right) \cdot Q_{i-1}$$

for all $i \geq 2$ that comes from the fact that $\sum_{k=1}^{d} \lambda_k = 0$. In particular since $Q_0 = 0$, it holds that $\sum_{k=1}^{d} g_k = g$. Hence, Step 3 (b) and Step 4 yield

$$s_j = \sum_{i=1}^{d} g_i(\beta_j, \beta_j) = g(\beta_j, \beta_j)\,.$$

*Remark 5 (Communication Complexity).* The first thing to note is that the `Recover` in Steps 2(b) and 3(e) are ran a maximum of $d + t_A = O(n)$ times total, which yields a communication complexity of $O(n^3/\ell)$. Ignoring the `Recover`, Step 2 requires $O(n^2)$ communication (broadcast of commitments for the new polynomials and new shares). Then, each iteration of the loop is performed in $O(i^2) = O(n^2)$ with $(i+1)^2$ openings in 3(a), $(i-1)^2$ commitments in 3(d) and $i^2$ commitments in 3(f). Overall, the communication complexity of `Reconstruct` is $O(n^3/\ell)$ for $\ell$ secrets.

**Theorem 1.** *The pair of protocol (`Share`,`Reconstruct`) is an $(T_s, T_c)$-secure*[7] *$\ell$-Batch $\sqrt{\ell}$-Gradual VSS, as in Definition 6, for $T_s = \{(n-1-\lfloor\sqrt{\ell}\rfloor, n-1-\lfloor\sqrt{\ell}\rfloor)$ and $T_c = \{(n, n-1)\}$.*

*Proof.* The ideal functionalities defined in Figs. 2 and 7 of Appendices B.1 and B.3 are taking into account the requirement of secrecy for `Share` and of correctness for `Reconstruct`. The batched gradual property of `Reconstruct` is also considered and Theorem 1 follows from Theorems 4 and 6. □

### 4.4 The `Refresh` Protocol

Similarly to **Reconstruct**, the `Refresh` protocol uses a blinding polynomial $Q$ to guarantee privacy of the secrets. This blinding polynomial $Q$ needs to verify $Q(\beta_j, \beta_j) = 0$ for $j \in [\ell]$. The easiest way to achieve this property is to take $Q(x, y) = (x - y)R(x, y)$ where $R$ is a random bivariate polynomial of degree $d - 1$. However, this polynomial $Q$ is equal to zero on the entire diagonal $(x, x)$. To obtain the level of secrecy required for `Refresh` we also need to refresh the shares $g(x, x)$ for any $x \notin \{\beta_1, \ldots, \beta_\ell\}$.To solve this issue, inspired by the univariate blinding factor in `Recover`, we blind the other diagonal values by a univariate polynomial that evaluates to 0 in the $\beta_j$. More precisely, at the end of the protocol, we constructed $g'$ as $g'(x, y) = g(x, y) + Q(x, y) = g(x, y) + (x - y) \cdot R(x, y) + h(x) \cdot \prod_{j\in\ell}(y - \beta_j)$ where $h$ is a random univariate polynomial in $\mathbb{P}_d$ and $R$ is a random bivariate polynomial.
Concretely, the `Recover` protocol is used to build and share the blinding polynomial in the following manner:

(a) First, a set of $d$ participants generates $R$ a random bivariate polynomial of degree $d - 1$ and uses `Recover` to share it with the remaining participants.
(b) Then, every party generates a random univariate polynomial $h_r$ and share it among each other, so that every participant $P_r$ can compute its value $h(\alpha_r) = \sum_{u=1}^{n} h_u(\alpha_r)$
(c) Finally, all parties compute $g'(\alpha_r, \alpha_{r'}) = g(\alpha_r, \alpha_{r'}) + (\alpha_r - \alpha_{r'}) \cdot R(\alpha_r, \alpha_{r'}) + h(\alpha_r) \cdot \prod_{j\in\ell}(\alpha_{r'} - \beta_j)$ from their blinded shares.

`Refresh` is described in Protocol 4 and its correctness and security proofs can be found in Appendix B.4

[7] Computational security under hardness of DLP.

**Protocol 4.** `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 $r' \in [d+1]$, $\boxed{P_r \text{ computes}}$
   $\boxed{g'(\alpha_r, \alpha_{r'}) = g(\alpha_r, \alpha_{r'}) + (\alpha_r - \alpha_{r'}) \cdot R(\alpha_r, \alpha_{r'}) + h(\alpha_r) \cdot \prod_{j \in [\ell]} (\alpha_{r'} - \beta_j)}$.

*Remark 6 (Communication Complexity).* The bottleneck of the communication is during Step 2 when $(n-d)$ `Recover` are performed. In the case of maximum security (when $n - d - 1 = O(1)$), the communication complexity is $O(n^2)/\ell$ for $\ell$ secrets.

**Theorem 2.** *The four protocols `Share`, `Reconstruct`, `Refresh`, `Recover` creates a $T_s, T_c$-secure [8] $\ell$-Batch PSS with multi-threshold $T_c = \{(n, n-1)\}$ and $T_s = \{(n-1-\lfloor\sqrt{\ell}\rfloor, n-1-\lfloor\sqrt{\ell}\rfloor)\}$ and $\ell = n-2$.*

*Proof.* Theorem 2 follows directly from Theorems 4 to 7, proved in Appendix B. □

## 5 Handling Dynamic Groups

In this Section we extend our batched (static) PSS scheme to handle dynamic groups. We introduce three new protocols: `Increase` (Protocol 5), `Decrease`

[8] Computational security under hardness of DLP.

(Protocol 6) and `DecreaseCorrupt` (Protocol 7). These protocols address the case of the Shamir sharing [35] where one secret $s$ is encoded as $f(\beta)$ for $f$ an univariate polynomial of degree $d$. This is a simpler case that can be easily extended to the batched case as is explained in Section 5.4. The goal of these protocols is to distribute the shared secret $s$ to a new set of parties of size $n+k$ ($k \in \mathbb{Z}$). The 3 protocols all follow the same requirements in terms of security. For correctness, we require that the new set of parties obtains a correct sharing of the secret with an univariate polynomial of degree $d+k$. For secrecy, we require that the adversary cannot recover the value of the secret during the execution protocol (provided that the threshold of corrupted parties is respected both before and after the protocol). These three protocols will be used in the `Redistribute` protocol that extends our PSS to a DPSS.

*Mixed Adversaries.* To ease presentation, we present the three sub-protocols in the semi-honest (honest-but-curious) model. As in previous protocols, active corruptions are handled using homomorphic commitments. In the `Increase, Decrease, DecreaseCorrupt` protocols all the operations are linear. The homomorphic property of the commitments can be used (as before) to verify that each linear operation is correctly performed. After that, we implicitly assume that the `Increase`, `Decrease` and `DecreaseCorrupt` protocols are extended against active corruptions in the way we described. The `Redistribute` protocol is given in the malicious setting.

### 5.1 Increase

In this case $k \in \mathbb{N}$. The goal of `Increase` is to generate $f^+$, a polynomial of degree $d+k$ with $f^+(\beta) = f(\beta)$, and to distribute the share $f^+(\alpha_r)$ to each $P_r \in \{P_1, \ldots, P_{n+k}\}$. Without loss of generality we assume that $P_{n+1}, \ldots, P_{n+k}$ are the new parties. For each $P_r \in \{P_1, \ldots, P_n\}$, $P_r$ has the value $f(\alpha_r)$.

*Constructing a polynomial of higher degree.* To efficiently generate the new $f^+$ polynomial, it is important to rely on the information already given to $P_1, \ldots, P_n$. For that we set

$$f^+(x) = Q(x) + \prod_{i=1}^{k} \frac{x - \alpha_{n+i}}{\beta - \alpha_{n+i}} f(x)$$

where $Q$ a polynomial of degree $d+k$ satisfying $Q(\beta) = 0$. This produces a correct sharing of $s$ while requiring low communication. With this expression of $f^+$, $P_1, \ldots, P_n$ do not have to share information about $f$. The only required communication is in the construction of $Q$. This step can be performed by the new parties. `Increase` is explained in Protocol 5, we give the formal proof of security in Appendix D.1.

**Protocol 5**. `Increase` (semi-honest)

INPUT: $P_r$ holds a share $f(\alpha_r)$ of a secret $s$ shared among $n$ parties $P_1, \ldots, P_n$. $k$ new parties $P_{n+1}, \ldots, P_{n+k}$ join the group.
OUTPUT: Each $P_r \in \{P_1, \ldots, P_{n+k}\}$ holds a share $f^+(\alpha_r)$ of $s$ for the $n+k$ parties.

1. For every $P_r \in \{P_{n+1}, \ldots, P_{n+k}\}$, $P_r$ samples one polynomial $\boxed{Q_r \leftarrow \mathbb{P}_{d+k}}$ verifying $\boxed{Q_r(\beta) = 0}$.
2. For $r \in [n+1, n+k]$ and $r' \in [n+k]$, $P_r$ $\boxed{\text{sends}}$ to $P_{r'}$ the values $\boxed{Q_r(\alpha_{r'})}$ and all parties can compute their random share $\boxed{Q(\alpha_{r'}) = \sum_{u=n+1}^{n+k} Q_u(\alpha_{r'})}$.
3. For $P_r \in \{P_1, \ldots, P_n\}$, $P_r$ computes its new share of the secret as $\boxed{f^+(\alpha_r) = \prod_{i=1}^{k} \frac{\alpha_r - \alpha_{n+i}}{\beta - \alpha_{n+i}} f(\alpha_r) + Q(\alpha_r)}$.
4. For all player $P_r \in \{P_{n+1}, \ldots, P_{n+k}\}$, $P_r$ sets its value $\boxed{f^+(\alpha_r) = Q(\alpha_r)}$

*Remark 7 (Communication Complexity).* In `Increase`, all the communication happens during Step 2. The number of message sent is $k(n+k)$.

### 5.2 Decrease

The `Decrease` protocol handles the case of a subset of the parties leaving the exchange. If $k$ is the size of the leaving subset, the goal is to build $f^-$ a polynomial of degree $d-k$ with $s = f^-(\beta) = f(\beta)$. The `Decrease` protocol requires to communicate with the leaving parties. When the parties are leaving due to a corruption or a failure of some sort, this protocol cannot be applied. That will be the role of the `DecreaseCorrupt` protocol introduced in Section 5.3. Without loss of generality, we consider that the set of leaving parties is $\{P_{n-k+1}, \ldots, P_n\}$.

*Constructing a polynomial of lower degree.* Similar to `Increase`, the $f^-$ will be constructed from $f$. The formula is a bit more complicated because for `Decrease` the data structure $(f)$ is compressed, which is more complicated than expending the data structure as is done in `Increase`. The construction of $f^-$ relies on the formula:

$$f(x) = \prod_{i=0}^{k-1}(x - \alpha_{n-i})Q(x) + \sum_{i=0}^{k-1}\Big(f(\alpha_{n-i}) \prod_{m=0, m\neq i}^{k-1} \frac{x - \alpha_{n-m}}{\alpha_{n-i} - \alpha_{n-m}}\Big)$$

where $Q$ is a polynomial of degree $d-k$. From that we obtain that $Q$ is defined by the following equation

$$Q(x) = \frac{f(x)}{\prod_{i=0}^{k-1}(x - \alpha_{n-i})} - \sum_{i=0}^{k-1}\Big(\frac{f(\alpha_{n-i})}{x - \alpha_{n-i}} \prod_{m=0, m\neq i}^{k-1} \frac{1}{\alpha_{n-i} - \alpha_{n-m}}\Big)$$

We just have to modify $Q$ to obtain $h^-$ of degree $d-k$ with $h^-(\beta) = f(\beta)$. Hence, we set

$$h^-(x) = \Big(Q(x) + \sum_{i=0}^{k-1} \frac{f(\alpha_{n-i})}{\beta - \alpha_{n-i}} \prod_{m=0, m\neq i}^{k-1} \frac{1}{\alpha_{n-i} - \alpha_{n-m}}\Big) \prod_{i=1}^{k-1} (\beta - \alpha_{n-i}) =$$

$$f(x) \prod_{i=1}^{k-1} \frac{\beta - \alpha_{n-i}}{x - \alpha_{n-i}} - \sum_{i=0}^{k-1} f(\alpha_{n-i}) \Big( \frac{x-\beta}{x-\alpha_{n-i}} \prod_{m=0, m\neq i}^{k-1} \frac{\beta - \alpha_{n-m}}{\alpha_{n-i} - \alpha_{n-m}} \Big)$$

To simplify notations we write $h^-(x) = f(x) \prod_{i=1}^{k-1} \frac{\beta - \alpha_{n-i}}{x - \alpha_{n-i}} - \sum_{i=0}^{k-1} \Lambda_i(x)$ with $\Lambda_i(x) = f(\alpha_{n-i}) \Big( \frac{x-\beta}{x-\alpha_{n-i}} \prod_{m=0, m\neq i}^{k-1} \frac{\beta - \alpha_{n-m}}{\alpha_{n-i} - \alpha_{n-m}} \Big)$. To distribute the value $h^-(\alpha_r)$ to $P_r$ for all $r \in [n-k]$, $P_{n-i}$ has to send the value $\Lambda_i(\alpha_r)$ to $P_r$ for all $0 \leq i \leq k-1$. However, that would enable the adversary to compute the value $f(\alpha_{n-i})$. To avoid that, $P_{n-i}$ generates $Q_i$ a polynomial of degree $d-k$ with $Q_i(\beta) = 0$ and send $\Lambda_i(\alpha_r) + Q_i(\alpha_r)$ to every party $P \in \{P_1, \ldots, P_{n-k}\}$. Then, we can define $f^- = h^- + \sum_{i=0}^{k-1} Q_i$. The `Decrease` protocol is given in Protocol 6 and the proof of security for this protocol can be found in Appendix D.1.

**Protocol 6**. `Decrease` (semi-honest)

INPUT: Each party $P_r$ has the share $f(\alpha_r)$ of a secret $s$ for $n$ parties. A subset of size $k$, $\mathcal{P}_{LEAVE} = \{P_{n-k-1}, \ldots, P_n\}$ for the leaving parties.
OUTPUT: Each party $P_r$ holds the share $f^-(\alpha_r)$ of $s$ for the parties in $\{P_1, \ldots, P_{n-k}\}$.

1. For $P_r \in \mathcal{P}_{LEAVE}$, $P_r$ samples $\boxed{Q_{n-r} \leftarrow \mathbb{P}_{d-k}}$, with $\boxed{Q_{n-r}(\beta) = 0}$
2. For $P_r \in \mathcal{P}_{LEAVE}$ and $\forall r' \in [n-k]$, $P_r$ $\boxed{\text{send}}$ to $P_{r'}$ the values $\boxed{Q_{n-r}(\alpha_{r'}) + \Lambda_{n-r}(\alpha_{r'}) f(\alpha_r)}$.
3. For $P_r \in \{P_1, \ldots, P_{n-k}\}$, the party $P_r$ compute its new share of the secret as $\boxed{f^-(\alpha_r) = \prod_{i=0}^{k-1} \frac{\beta - \alpha_{n-i}}{\alpha_r - \alpha_{n-i}} f(\alpha_r) + \sum_{i=0}^{k-1} Q_i(\alpha_r) + \Lambda_i(\alpha_r) f(\alpha_{n-i})}$.

*Remark 8 (Communication Complexity).* The number of message sent during step 2 of `Decrease` is $k(n-k)$.

### 5.3 Decrease Corrupt

This section presents the `DecreaseCorrupt` protocol. It handles the case where some parties are leaving without prior agreement and cannot communicate with the rest of the parties. The remaining parties have to ignore the leaving parties. If we take $d = n-2$ we can tolerate only one leaving party. The `DecreaseCorrupt` protocol in Protocol 7 is addressing the case where $P_n$ is leaving.

*Decreasing the number of parties without prior agreement.* The $f^-$ polynomial as defined in Section 5.2 for $k = 1$ remains an efficient way to produce a polynomial of degree $d-1$ with the same evaluation as $f$ in $\beta$. To construct $f^-$, the parties need the value $f(\alpha_n)$, they will reconstruct this value using a process very similar to the `Recover` protocol of [17]. Each party $P_r$ reveals $f(\alpha_r) + f_n(\alpha_r)$ where $f_n$ is a polynomial of degree $d$ with $f_n(\alpha_n) = 0$. With that, each participant is able to construct the value $f(\alpha_n)$. This is fine, as we consider that $P_n$ is leaving due to a heavy corruption and the adversary already knows the share of $P_n$. The `DecreaseCorrupt` Protocol is explained in Protocol 7, the security proof is given in Appendix D.1.

**Protocol 7**. `DecreaseCorrupt` (semi-honest)

INPUT: Each party $P_r \in \mathcal{P} = \{P_1, \ldots, P_{n-1}\}$ holds the value $f(\alpha_r)$ for $f$ a polynomial of degree $d$.
OUTPUT: Each party $P_r$, $1 \leq r \leq n-1$ holds the value $f^-(\alpha_r)$ for $f^-$ a polynomial of degree $d-1$.

1. For $r \in [n-1]$, $P_r$ samples $\boxed{f_r \leftarrow \mathbb{P}_d}$ with $\boxed{f_r(\alpha_n) = 0}$.
2. For all pair of parties $P_r, P_{r'} \in \mathcal{P}^2$, $P_r$ $\boxed{\text{sends}}$ to $P'_r$ the value $\boxed{f_r(\alpha_{r'})}$.
3. Each $P_r$ computes the value $\boxed{f_n(\alpha_r) = \sum_{u=1}^{n-1} f_u(\alpha_r)}$.
4. Each $P_r$ $\boxed{\text{broadcasts}}$ the value $\boxed{f(\alpha_r) + f_n(\alpha_r)}$.
5. Each $P_r$ interpolates the value $\boxed{f(\alpha_n)}$.
6. Each player $P_r$ computes the value $\boxed{f^-(\alpha_r) = f(\alpha_r)\frac{\beta-\alpha_n}{\alpha_r-\alpha_n} + f(\alpha_n)\frac{\beta-\alpha_r}{\alpha_r-\alpha_n}}$.

*Remark 9 (Tolerating more leaving parties).* In the case where $d < n-2$ more than one leaving party can be tolerated. The extension is quite straightforward. If $k$ is the number of leaving parties, we will set $f^-$ as it is done in `Decrease`. Then, the remaining parties have to reconstruct the missing values. That can be done in the same way it is done for $f(\alpha_n)$ in Protocol 7. After that, each participant can construct the correct evaluation of $f^-$ using the formula.

*Remark 10 (Communication Complexity).* $(n-1)^2$ messages are sent in step 2 and $n-1$ values are broadcasted in step 3. Complexity of `DecreaseCorrupt` is $O(n^2)$.

### 5.4 Dynamic Batching

To produce the `Redistribute` protocol, we need to extend the three protocols `Increase`, `Decrease`, `DecreaseCorrupt` to batched secret sharing. This extension is briefly detailed in the next paragraph. The `Redistribute` will be decomposed in consecutive executions of `Increase`, `Decrease` and `DecreaseCorrupt`. If we consider $\mathcal{P}$ as the set of participants before `Redistribute` and $\mathcal{P}'$ the set

of participants after, we can divide $\mathcal{P}$ as $\mathcal{P}_L \cup \mathcal{P}_{L_C} \cup \mathcal{P}_\cap$ and $\mathcal{P}'$ as $\mathcal{P}_N \cup \mathcal{P}_\cap$ with $\mathcal{P}_\cap = \mathcal{P} \cap \mathcal{P}'$, $\mathcal{P}_L$ the set of leaving parties and $\mathcal{P}_{L_C}$ the set of participants leaving due to corruption. `Redistribute` consists of three steps. First, $\mathcal{P}_L \cup \mathcal{P}_\cap$ perform `DecreaseCorrupt`. Then, $\mathcal{P}_L$ and $\mathcal{P}_\cap$ perform `Decrease`. Finally, $\mathcal{P}_N$ and $\mathcal{P}_\cap$ use `Increase` to share the secrets among every participant in $\mathcal{P}'$.

*Obtaining Dynamic protocols for bivariate sharing.* To reshare $\ell$ secrets from $n$ parties to $n+k$ parties ($k \in \mathbb{Z}$), we need to produce $g^{dyn}$ a bivariate polynomial of degree $d+k$ with $g^{dyn}(\beta_j, \beta_j) = g(\beta_j, \beta_j)$ for all $j \in [\ell]$. We require the additional bound $\ell \leq d+k$. In this paragraph, we briefly present a protocol doing that. It uses a `Dynamic` sub-protocol, that will be either `Increase`, `Decrease` or `DecreaseCorrupt` depending on the situation (here we consider the malicious extension of the protocol in Protocols 5 to 7). For all $j \in [\ell]$, we define $f_j(x) = g(x, \beta_j)$ a polynomial of degree $d$. The parties are going to perform the `Dynamic` protocol with $\beta = \beta_j$ to produce $f_j^{dyn}$ for all $j \in [\ell]$. We define the polynomials $g^{dyn}(x, \beta_j) = f_j^{dyn}(x)$ for all $j$. Each $P_r \in \{P_1, \ldots, P_{d+k+1}\}$ complete its $\ell$ values ($g^{dyn}(\alpha_r, \beta_j)$ for all $j \in [\ell]$) to obtain a polynomial $g^{dyn}(\alpha_r, y)$ of degree $d+k$. This defines one unique bivariate polynomial $g^{dyn}$ of degree $d+k$. Then, $P_1, \ldots, P_{d+k+1}$ perform a `Recover` protocol to give shares to each participant $P_{d+k+2}, \ldots, P_{n+k}$. This is correct as `Recover` only requires that $P_1, \ldots, P_{d+1}$ have valid shares.

**Protocol 8**. `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 $\boxed{\texttt{Decrease} \text{ to share } f_j^2}$ 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 $\boxed{\texttt{Increase} \text{ to produce shares of } f_j^3}$ 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. $\boxed{\{P_1, ..., P_{d'+1}\} \text{ uses } \texttt{Recover} \text{ to share } g' \text{ to } \{P_{d'+2}, \ldots, P_{n^{(\omega+1)}}\}}$.

*Remark 11 (Order of Operations).* We perform `DecreaseCorrupt`, `Decrease` and `Increase` in this order because it minimizes the communication complexity. The only constraint is that `DecreaseCorrupt` must be done first. For instance, for threshold reasons, it might be interesting to perform `Increase` before `Decrease`. This does not change anything.

*Remark 12 (Communication Complexity).* The maximum complexity of `Increase`, `Decrease` and `DecreaseCorrupt` is $O(n^2)$. `Refresh`'s complexity is $O(n^2)$ when $d = n - 2$ and the overall complexity of `Redistribute` is $O(\ell n^2)$ and $O(n^2)$ amortized per secret.

**Theorem 3.** *The five protocols* ***`Share`****,* ***`Reconstruct`****,* ***`Refresh`****,* ***`Recover`*** *and* ***`Redistribute`*** *creates a $T_s^{(\omega)}, T_c^{(\omega)}$-secure[9] $\ell$-Batch DPSS, as in Definition 7, with multi-threshold $T_c^{(\omega)} = \{(n^{(\omega)}, n^{(\omega)}-1)\}$ and $T_s^{(\omega)} = \{(n^{(\omega)}-1-\lfloor\sqrt{\ell}\rfloor, n^{(\omega)}-1-\lfloor\sqrt{\ell}\rfloor)\}$ and $\ell \leq n^{(\omega)} - 2$.*

*Proof.* Theorem 3 follows from Theorems 2 and 11 (cf. Appendix D.2). □

## References


1. Backes, M., Cachin, C., Strobl, R.: Proactive secure message transmission in asynchronous networks. In: Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing, PODC 2003, Boston, Massachusetts, USA, July 13-16, 2003. pp. 223–232 (2003). https://doi.org/10.1145/872035.872069, http://doi.acm.org/10.1145/872035.872069
2. Baron, J., Eldefrawy, K., Lampkins, J., Ostrovsky, R.: How to withstand mobile virus attacks, revisited. In: PODC. pp. 293–302. ACM (2014)
3. Baron, J., Eldefrawy, K., Lampkins, J., Ostrovsky, R.: Communication-optimal proactive secret sharing for dynamic groups. In: ACNS. LNCS, vol. 9092, pp. 23–41. Springer (2015)
4. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure mpc with linear communication complexity. In: Proceedings of the 5th Conference on Theory of Cryptography. pp. 213–230. TCC'08, Springer-Verlag, Berlin, Heidelberg (2008), http://dl.acm.org/citation.cfm?id=1802614.1802632
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC. pp. 1–10. ACM (1988)


[9] Computational security under hardness of DLP.

6. Ben-Sasson, E., Fehr, S., Ostrovsky, R.: Near-linear unconditionally-secure multiparty computation with a dishonest minority. In: CRYPTO. LNCS, vol. 7417, pp. 663–680. Springer (2012)
7. Blakley, G.R.: Safeguarding cryptographic keys. Proc. of AFIPS National Computer Conference **48**, 313–317 (1979)
8. Boneh, D., Gennaro, R., Goldfeder, S.: Using level-1 homomorphic encryption to improve threshold DSA signatures for Bitcoin wallet security. In: Latincrypt (2017)
9. Cachin, C., Kursawe, K., Lysyanskaya, A., Strobl, R.: Asynchronous verifiable secret sharing and proactive cryptosystems. In: ACM Conference on Computer and Communications Security. pp. 88–97 (2002)
10. Canetti, R., Herzberg, A.: Maintaining security in the presence of transient faults. In: CRYPTO. pp. 425–438 (1994)
11. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. **20**(4), 398–461 (2002)
12. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: Proceedings of the twentieth annual ACM symposium on Theory of computing. pp. 11–19. STOC '88, ACM, New York, NY, USA (1988). https://doi.org/10.1145/62212.62214, http://doi.acm.org/10.1145/62212.62214
13. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography. In: EUROCRYPT. LNCS, vol. 6110, pp. 445–465. Springer (2010)
14. Damgård, I., Ishai, Y., Krøigaard, M., Nielsen, J.B., Smith, A.: Scalable multiparty computation with nearly optimal work and resilience. In: CRYPTO. pp. 241–261 (2008)
15. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: CRYPTO. LNCS, vol. 4622, pp. 572–590. Springer (2007)
16. Desmedt, Y., Jajodia, S.: Redistributing secret shares to new access structures and its applications (1997), technical Report ISSE TR-97-01, George Mason University
17. Dolev, S., Eldefrawy, K., Lampkins, J., Ostrovsky, R., Yung, M.: Proactive secret sharing with a dishonest majority. In: SCN. LNCS, vol. 9841, pp. 529–548. Springer (2016)
18. Dolev, S., Garay, J., Gilboa, N., Kolesnikov, V.: Swarming secrets. In: Proceedings of the 47th Annual Allerton Conference on Communication, Control, and Computing. pp. 1438–1445. Allerton'09, IEEE Press, Piscataway, NJ, USA (2009), http://dl.acm.org/citation.cfm?id=1793974.1794220
19. Dolev, S., Garay, J.A., Gilboa, N., Kolesnikov, V.: Secret sharing krohn-rhodes: Private and perennial distributed computation. In: ICS (2011)
20. Dolev, S., Garay, J.A., Gilboa, N., andYelena Yuditsky, V.K.: Towards efficient private distributed computation on unbounded input streams. J. Mathematical Cryptology **9**(2), 79–94 (2015). https://doi.org/10.1515/jmc-2013-0039, http://dx.doi.org/10.1515/jmc-2013-0039
21. Eldefrawy, K., Ostrovsky, R., Park, S., Yung, M.: Proactive secure multiparty computation with a dishonest majority. In: SCN. LNCS, vol. 11035, pp. 200–215. Springer (2018)
22. Frankel, Y., Gemmell, P., MacKenzie, P.D., Yung, M.: Optimal resilience proactive public-key cryptosystems. In: 38th Annual Symposium on Foundations of Computer Science, FOCS'97, Miami Beach, Florida, USA, October 19-22, 1997. pp. 384–393. IEEE Computer Society (1997). https://doi.org/10.1109/SFCS.1997.646127, https://doi.org/10.1109/SFCS.1997.646127

23. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: STOC. pp. 699–710 (1992)
24. Gennaro, R., Goldfeder, S.: Fast multiparty threshold ECDSA with fast trustless setup. In: ACM Conference on Computer and Communications Security. pp. 1179–1194. ACM (2018)
25. Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In: ACNS. LNCS, vol. 9696, pp. 156–174. Springer (2016)
26. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A.V. (ed.) STOC. pp. 218–229. ACM (1987)
27. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: How to cope with perpetual leakage. In: CRYPTO. pp. 339–352 (1995)
28. Hirt, M., Lucas, C., Maurer, U.: A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In: CRYPTO (2). LNCS, vol. 8043, pp. 203–219. Springer (2013)
29. Lindell, Y., Nof, A.: Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1837–1854. CCS '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243788, http://doi.acm.org/10.1145/3243734.3243788
30. Lindell, Y., Nof, A.: Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In: ACM Conference on Computer and Communications Security. pp. 1837–1854. ACM (2018)
31. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: PODC. pp. 51–59. ACM (1991)
32. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO. pp. 129–140 (1991)
33. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: STOC. pp. 73–85. ACM (1989)
34. Schultz, D.: Mobile Proactive Secret Sharing. Ph.D. thesis, Massachusetts Institute of Technology (2007)
35. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
36. Tassa, T., Dyn, N.: Multipartite secret sharing by bivariate interpolation. J. Cryptology **22**(2), 227–258 (2009)
37. Wong, T.M., Wang, C., Wing, J.M.: Verifiable secret redistribution for archive system. In: IEEE Security in Storage Workshop. pp. 94–106. IEEE Computer Society (2002)
38. Zhou, L., Schneider, F.B., van Renesse, R.: Apss: proactive secret sharing in asynchronous systems. ACM Trans. Inf. Syst. Secur. **8**(3), 259–286 (2005)

# A Security Definitions

In this Section, we introduce several security definitions.

## A.1 Homomorphic Commitment

The Gradual VSS from [28] is based on Shamir's secret sharing scheme [35]. It uses homomorphic commitments to make the secret sharing verifiable as was

done in [32] with Pedersen Commitment scheme. A commitment scheme is a protocol that allows one player $P$ to commit to a secret message $m$ to some other parties. This is done by sending to the other parties the commitment to $m$ under some randomness $\texttt{rand}$, that will be noted $C(m, \texttt{rand})$. Later, the committing party $P$ can reveal the values $m$, $\texttt{rand}$ and all the witnessing parties can verify that the commitment they had corresponded to the value $C(m, \texttt{rand})$. It is required from a commitment scheme to be hiding and binding. The binding property ensures that $P$ cannot change its mind by finding a second set of values $m', \texttt{rand}'$ that opens correctly the commitment. The hiding property means that the witness parties cannot find the value of $m$ from the view of $C(m, \texttt{rand})$. An Additively Homomorphic commitment $C$ has the additional property that there exists some operation $\star$ over the commitments space such that $C(m_1, \texttt{rand}_1) \star C(m_2, \texttt{rand}_2) = C(m_1 + m_2, \texttt{rand}_1 + \texttt{rand}_2)$.

*Perdersen VSS.* In [32] one of the first VSS scheme was introduced using Pedersen's commitment. It takes advantage of the homomorphic property of the exponentiation to produce an Homomorphic commitment. Assume that we have a cyclic group $\mathbb{G} = \langle g \rangle$ of order $q$. If we take $g, h$ two random generators of $\mathbb{G}$. The commitment to a message $m \in \mathbb{Z}_q$ under randomness $r$ is $C(m, r) = g^m h^r$ is a Homomorphic Commitment to $m$. The homomorphic addition is the multiplication $g^{m_1} h^{r_1} g^{m_2} h^{r_2} = g^{m_1+m_2} h^{r_1+r_2}$ and the constant multiplication is the exponentiation $(g^m h^r)^\lambda = g^{\lambda m} h^{\lambda r}$. Under the difficulty of the discrete logarithm problem, this commitment scheme is perfectly hiding and computationally binding.

### A.2 Round and Corruption Model

In this Section, we outline the corruption model in the proactive setting. The protocol execution is divided into phases $\phi$. There are two kinds of phases: operation phases (the computation of a circuit's layer with additions and multiplications) and refresh phases. Refresh phases are necessary to prevent the adversary from learning all the secrets. For simplicity of the corruption model, phases are regrouped in stages $\sigma$. A stage $\sigma$ is constituted of 3 consecutive phases: a refresh phase, an operation phase, a refresh phase. Two consecutive stages have a refresh phase in common. If we denote $\ell_C$ the number of layer, there is one stage for each layer, we will denote them $\sigma_1, \ldots, \sigma_{\ell_C}$. We can add the first stage $\sigma_0$ that is constituted of the execution of `Share` followed by one refresh phase and the last stage $\sigma_{\ell_C+1}$ that is a refresh phase followed by `Reconstruct`.

If a party $P_i$ is corrupted by the adversary $\mathcal{A}$ during an operation phase of a stage $\sigma_j$, then $\mathcal{A}$ learns the view of $P_i$ (all values known and received by $P_i$) starting from its state at the beginning of stage $\sigma_j$. If the corruption is made during a refresh phase between consecutive stages $\sigma_j$ and $\sigma_{j+1}$, then $\mathcal{A}$ learns $P_i$'s view starting from the beginning of stage $\sigma_j$. Moreover, in the case of a corruption during a refresh phase, $P_i$ is considered to be corrupt in both stages $\sigma_j$ and $\sigma_{j+1}$. If a party becomes "decorrupted" before the resfresh phase of a stage $\sigma_{j+1}$, it is considered not corrupted only after the refresh phase of $\sigma_{j+1}$. A

party that was only passively corrupted immediately joins the set of honest party after a decorruption. A party that was actively corrupted will be restored to a clean default state. The shares will be recovered at that time using a recovery protocol. It is also important that the random tapes are overwritten with fresh randomness.

### A.3 Passive Security

We introduce the notion of security in the case of a passive adversary. The notion of security in the case of MPC is always defined in regard to the security that can be reached in the case of an ideal world, where a trusted ideal party take the inputs from all parties compute the desired functionality and send back the output to all parties.

We consider a secure multiparty computation protocol between $n$ parties as the computation of a function $f$ on inputs $x_1, x_2, \ldots, x_n$. The function outputs $y_r$ to party $P_r$ with $(y_1, y_2, \ldots, y_n) = f(x_1, \ldots, x_n)$. Given a protocol $\pi$ between $n$ parties, we define the view of the r-th party during the execution of $\pi$ as $VIEW_r^\pi(x_1, \ldots, x_n) = (x_r, \texttt{rand}^r, m_1^r, \ldots, m_v^r)$ $\texttt{rand}^r$ is $P_r$'s internal random tape and $m_j^r$ is the j-th message received by $P_r$. We also define $\texttt{output}_r^\pi(x_1, \ldots, x_n)$ as the output of party $P_r$ after the execution of the protocol $\pi$.

Since the proactive adversary is not supposed to control more than $t$ parties during the execution of the given protocol we can use the definition of security against static adversary controlling passively up to $t$ parties. If we note $R$=$\{r_1, \ldots, r_t\}$ the subset corrupted parties, we can define

$$VIEW_R^\pi = (r, VIEW_{r_1}^\pi, \ldots, VIEW_{r_t}^\pi)\,,$$

and using the same notation as the view, $\texttt{output}_R^\pi(x_1, \ldots, x_n)$ is the set of outputs for all parties in $R$. In this case we will note $f_R(x_1, \ldots, x_n) = (y_{r_1}, \ldots, y_{r_t})$

**Definition 8.** *We say that a protocol $\pi$ is* privately computing *$f$ against any adversary controlling a subset of size $t$ of the parties, if there exists a probabilistic polynomial-time algorithm, denoted $S$, such that for every $R$=$\{r_1, \ldots, r_t\}$, it holds that*

$$\left\{m, S(R, (x_{r_1}, \ldots, x_{r_t}), f_R(x_1, .., x_n))\right\}_{m \in \mathbb{N}, (x_1, \ldots, x_n) \in \left(\{0,1\}^*\right)^n}$$

$$\stackrel{\text{C}}{\equiv} \left\{m, VIEW_R^\pi(x_1, \ldots, x_n), \texttt{output}_R^\pi(x_1, \ldots, x_n)\right\}_{m \in \mathbb{N}, (x_1, \ldots, x_n) \in \left(\{0,1\}^*\right)^n}$$

*where $\stackrel{\text{C}}{\equiv}$ denotes the computational indistinguishability of two family of random variables indexed by $m$. This assumes that the inputs are of length bounded by a security parameter $\kappa$.*

### A.4 Active Security in the Mixed Adversaries Model

Here we define a definition of security against a malicious adversary in the mixed model. First we are going to define a "real world" execution of a protocol $\Pi$ in Fig. 1.

**Figure 1**. "Real-world" execution of the protocol $\Pi$

In this execution, the environment $\mathcal{Z}$ will provide the parties and an ideal adversary $\mathcal{A}$ with inputs of its choice.

**The "real-world" execution:**
INITIALIZATION
1. $\mathcal{Z}$ invokes the adversary $\mathcal{A}$ with an auxiliary input $z$
2. $\mathcal{Z}$ invokes the parties $P_r$ and their inputs $x_r$.
3. The set of passively and actively corrupted parties $\mathcal{P}_P$, $\mathcal{P}_A$ are initialized.
COMPUTATION
4. For each round of the protocol $\Pi$ :
   - *4.1* The honest and semi-honest parties prepare a message to send, as dictated by $\Pi$.
   - *4.2* The adversary prepares a message on behalf of the parties in $\mathcal{P}_A$.
   - *4.3* All parties broadcast their prepared message.

**Outputs:** The honest parties outputs $\perp$ if they have not received an output value, and their outputs otherwise. The adversary outputs a value $v_{\mathcal{A}}$ that may be arbitrarily computed from the information he obtained during the execution of the protocol. After observing the outputs of all parties and $\mathcal{A}$ the environment outputs a bit $b_{\mathcal{Z}}$.

Let $\text{EXEC}^{\Pi}_{\mathcal{A},\mathcal{Z}}(z)$ denote the distribution of the output bit $b_{\mathcal{Z}}$, and let $\text{IDEAL}^{\mathcal{Z},\mathcal{S},\mathcal{F}}(z, T_c, T_s)$ denote the output distribution of $\mathcal{Z}$ during the execution of an ideal process defined from an ideal functionality $\mathcal{F}$ (e.g., see Fig. 2). An adversary $\mathcal{A}$ is said to be $T$-bounded for a multi-threshold $T$ if $(t_P, t_A) \leq T$.

**Definition 9.** *We say that the multiparty protocol $\Pi$ is* securely realizing $\text{IDEAL}^{\mathcal{Z},\mathcal{S},\mathcal{F}}$ *with multi-thresholds $(T_c, T_s)$ if for all adversaries $\mathcal{A}$ attacking $\Pi$ there exists an ideal adversary $\mathcal{S}$ such that:*

- *for all $T \in \{T_c, T_s\}$ if $\mathcal{A}$ is $T$-bounded, then $\mathcal{S}$ is also $T$-bounded;*
- *for every environment $\mathcal{Z}$ it holds that*

$$\left\{\text{IDEAL}^{\mathcal{Z},\mathcal{S},\mathcal{F}}_{mixed}(z, T_c, T_s)\right\}_z \overset{\text{C}}{\equiv} \left\{\text{EXEC}^{\Pi}_{\mathcal{A},\mathcal{Z}}(z)\right\}_z$$

*where $\overset{\text{C}}{\equiv}$ denotes the computational indistinguishability, where all inputs' lengths are bounded by a security parameter $\kappa$.*

# B Formal Definitions and Proofs for the Batched Proactive Secret Sharing Scheme

In this Section we introduce the 4 proofs of security for the `Share`, `Reconstruct`, `Recover` and `Refresh` protocols that constitutes our Proactive Secret Sharing.

## B.1 The `Share` protocol

First we define the ideal functionality for the `Share` protocol. The input is a batch of secrets $s_1, \ldots, s_\ell$ held by a dealer $P_\mathcal{D}$. The output is a set of shares $\{g(\alpha_r, \alpha_{r'})\}_{r' \in [d+1]}$ for a set of parties $\{P_1, \ldots, P_n\}$. These shares are evaluations of a bivariate polynomial of degree $d$ verifying $g(\beta_j, \beta_j) = s_j$ for $j \in [\ell]$.

**Figure 2**. Ideal Process for the `Share` 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 $\mathcal{F}_{\texttt{Share}}$ that will play the role of a trusted third party that will compute the sharing. Upon reception of the dealer's input that is the batch of secret $s_1, \ldots, s_\ell$, the functionality will output the shares for all the parties $P_1, \ldots, P_n$.

*Parameters.* Multi-thresholds $T_c, T_s$.

*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 $x_r$.
3. $\mathcal{S}$ may change the input for the dealer if $P_\mathcal{D} \in \mathcal{P}_A$.
INPUTS
4. Each party sends his input to the ideal functionality.
CORRUPTION
5. If $(t_P, t_A) \not\leq T_s$: $\mathcal{F}_{\texttt{Share}}$ sends all inputs to $\mathcal{S}$.
COMPUTATION
6. $\mathcal{F}_{\texttt{Share}}$ generates $g$ a bivariate sharing as the output of the `Share` protocol on inputs $s_1, \ldots, s_\ell$.
OUTPUTS
7. $\mathcal{F}_{\texttt{Share}}$ sends the shares $g(\alpha_r, \alpha_{r'})$ for the corrupted parties $P_r \in \mathcal{P}_P$ to $\mathcal{S}$.
8. $\mathcal{S}$ sends either `abort` to $\mathcal{F}_{\texttt{Share}}$ and $\mathcal{F}_{\texttt{Share}}$ aborts and outputs $\perp$ or `continue` and the functionality proceeds to the next step.
9. If $(t_P, t_A) \not\leq T_c$ and $P_\mathcal{D} \in \mathcal{P}_A$: $\mathcal{S}$ sends new shares to $\mathcal{F}_{\texttt{Share}}$ instead of

the evaluations of $g$. $\mathcal{F}_{\texttt{Share}}$ replaces the output.
10. $\mathcal{F}_{\texttt{Share}}$ sends their shares to the honest parties.

*Outputs.* Each honest party outputs whatever they received from $\mathcal{F}_{\texttt{Share}}$. 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 the `Share` protocol there are two clear different cases: $P_{\mathcal{D}}$ is actively corrupted (Fig. 3) and we need to verify correctness, $P_{\mathcal{D}}$ is honest and we need to verify secrecy (Fig. 4). Given these two cases we give two different simulators corresponding to these cases. The situation where $P_{\mathcal{D}}$ is only passively corrupted has no interest because the adversary knows the secret but cannot try to damage the correctness of the protocol.

**Figure 3**. Simulator for the Share protocol when $P_{\mathcal{D}}$ is honest or semi-honest.

1. If $(t_P, t_A) \leq T_s$ or $P_{\mathcal{D}} \in \mathcal{P}_P$ $\mathcal{S}$ receives the input of the dealer from $\mathcal{F}_{\texttt{Share}}$, if not it generates $s_1, \ldots, s_\ell$ as a batch of random values.
2. The simulator computes the missing evaluations and generates random polynomials $f_j$ of degree $d$ with $f_j(\beta_j) = s_j$ for all $1 \leq j \leq \beta$.
3. $\mathcal{S}$ generates a bivariate polynomial of degree $d$ with $g(\alpha_r, \beta_j) = f_j(\alpha_r)$ for all $j$. All the values $g(\alpha_r, \alpha_{r'})$ for $r, r' \in [n] \times [d+1]$ are interpolated.
4. $\mathcal{S}$ sends to $\mathcal{A}$ the commitments to $g(\alpha_r, \alpha_{r'})$ for $r, r' \in [d+1]^2$.
5. $\mathcal{S}$ sends to the adversary the opening to the commitments for the corrupted parties, and waits to receive back the complaining bit. If one bit indicates that the opening is not correct, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\texttt{share}}$, otherwise it proceeds to the next step.
6. The simulators sends `continue` to the ideal functionality and outputs whatever $\mathcal{A}$ outputs.

**Figure 4**. Simulator for the Share protocol when $P_{\mathcal{D}}$ is actively corrupted.

1. $\mathcal{S}$ receives the input of the dealer from $\mathcal{F}_{\texttt{Share}}$.
2. $\mathcal{S}$ receives from $\mathcal{A}$ the commitments to the $(d+1)^2$ initial evaluations of $g$ and $\mathcal{S}$ computes locally the commitments to the shares of all the honest parties. If not, $\mathcal{S}$ sends `abort` to the ideal functionality.
3. $\mathcal{S}$ receives from $\mathcal{A}$ the opening to these shares. For those that are incorrect, he sends back a complaining bit.
4. $\mathcal{A}$ broadcasts the opening to the non-correct commitments. If at least one is still not correct, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\texttt{share}}$, otherwise it sends `continue.`
5. The simulators outputs whatever $\mathcal{A}$ outputs.

**Theorem 4.** *The* `Share` *protocol securely realizes* $\text{IDEAL}_{mixed}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{\texttt{Share}}}(z, T_c, T_s)$ *with multi-threshold* $T_c = \{(n, n-1)\}$ *and* $T_s = \{(n-1-\lfloor\sqrt{\ell}\rfloor, n-1-\lfloor\sqrt{\ell}\rfloor)\}$.

*Proof.* In the case where $P_{\mathcal{D}}$ is honest or honest-but-curious we show that the Simulator described in Fig. 3 produces an indistinguishable view from the real execution of the protocol. We start by showing that the view of the adversary is the same in both situations. First, when the secrecy threshold is violated or when the dealer is passively corrupted, $\mathcal{A}$ knows the batch of secrets. In this case, $\mathcal{S}$ knows it as well and produce a valid bivariate sharing of $s_1, \ldots, s_\ell$. Otherwise, the adversary obtains $t_P \cdot (d+1)$ evaluations of a bivariate polynomial $g$ of degree $d$ that is defined by $(d+1)^2$ points. If $t_P < d$, he cannot compute directly the values $g(\beta_j, \beta_j)$ for any $j \in [\ell]$ and the whole tuple $\{g(\beta_1, \beta_1), \ldots, g(\beta_\ell, \beta_\ell)\}$ cannot be distinguished from any random tuple $\{r_1, \ldots, r_\ell\}$ when $\ell \leq (d+1)^2 - t_P \cdot (d+1)$ which is obtained when $T_s$ is not violated if $\ell \leq d+1$, $t_P < d$ and $d = n-1$ (which is the maximum value we can take for $d$). This is under the assumption that no information on the secrets is gathered from the view of the commitments of $g$. The commitment is perfectly hiding and so this concludes the proof of indistinguishability of the adversary's view, which shows that the adversary's output will be indistinguishable in the real and ideal executions.
If the adversary decides to abort the protocol, he would do so with the same probability in both executions during step 4 of the `Share` protocol. When $P_{\mathcal{D}}$ is not actively corrupted the honest parties output is always correct when the protocol goes through. In this case, we have proved Computational Indistinguishability of the view.

In the case where $P_{\mathcal{D}}$ is actively corrupted. From the $(d+1)^2$ initial values, it is computationally hard for the adversary to find a correct opening that is not the evaluation of $g$ for the honest parties. This is due to the fact that our commitment is computationally binding. This ensures that the honest parties outputs a correct evaluations of $g$ or $\perp$ when the protocol aborts. $\mathcal{S}$ behaves as honest parties would and so the view of the adversary is similar in the real and ideal executions. □

### B.2 The `Recover` protocol

In the case of `Recover`, the inputs are the shares and commitments of the $d+1$ parties $P_1, \ldots, P_{d+1}$ who still have their shares, and the identity of a recovering party $P_{r_C}$. The output will be the set of new shares and commitments for the set $\{P_1, \ldots, P_{d+1}, P_{r_C}\}$. Before providing the full simulator-based proof, we prove the following lemma:

**Lemma 2.** `Recover` *computes correctly the recovery of the shares of one participant and preserves secrecy on the* $\ell$ *secrets when* $(t_P, t_A) \leq \{(d, d)\}$.

*Proof. Correctness:* To verify the correctness of `Recover`, we need to verify that each participant $P_r$ receives the values $(g(\alpha_r, \alpha_{r'}))_{r' \in [d+1]}$. The first step, guarantees that $P_{r_C}$ obtains the correct commitments to all the evaluations of $g$. Then,

homomorphic commitments guarantees that $f_r(\alpha_{r_C}) = 0$ for all $r \in [d+1]$. The same commitments ensures that $P_{r_C}$ receives the correct values and with that, $P_{r_C}$ is able to compute the value $g(\alpha_{r_C}, \alpha_{r'})$ for all $r'$. That concludes the sketch of proof for correctness of `Recover` in Theorem 5.

*Secrecy:* The following proof justifies the threshold $d+1-\sqrt{\ell}$ for the number of passive corruptions. There are 2 cases depending on whether $P_{r_C}$ is corrupted or not. If $P_{r_C}$ is honest, the adversary will only learn some values $f_r(\alpha_{r'})$ for honest parties $P_r$ and corrupted parties $P_{r'}$. No new information is gained on $g$ and on the secrets $s_1, \ldots, s_\ell$ and secrecy is preserved.
In the case $P_{r_C}$ corrupted, without loss of generality we can assume that $P_1, \ldots, P_{t_P-1}$ are corrupted as well as $P_{r_C}$. After step 2, the adversary knows $f_r(\alpha_{r'})$ when either $r \in [t_P - 1]$ or $r' \in [t_P - 1]$. Then, after step 5, $P_{r_C}$ sees the value $g(\alpha_r, \alpha_{r'}) + f_{r'}(\alpha_r)$ for all $(r, r') \in [d+1]^2$. From that and the knowledge on $(f_{r'})_{r' \in [d+1]}$, the adversary is able to compute the value of $g(\alpha_r, \alpha_{r'})$ when either $r \in [t_P - 1]$ or $r' \in [t_P - 1]$. The adversary still lacks the $(d + 2 - t_P)^2$ values $g(\alpha_r, \alpha_{r'})$ for $r, r' \in [t_P, d+1]^2$ to hope interpolate the values $g(\beta_j, \beta_j)$ for any $j$. Similarly, the adversary lacks the $(d+2-t_P)^2$ random values $f_r(\alpha_{r'})$ for $r, r' \in [t_P, d+1]^2$. From $f_r(\alpha_{r_C}) = 0$, the adversary obtains $(d+2-t_P)$ equations on those $(d+2-t_P)^2$ unknown values. Combining those with $g(\alpha_r, \alpha_{r'}) + f_{r'}(\alpha_r)$ for $r, r' \in [t_P, d+1]^2$, the adversary obtains $(d+2-t_P) + (d+2-t_P)^2$ equations for $2(d+2-t_P)^2$ unknown values. If the adversary wants to learn $g(\alpha_r, \alpha_{r'})$ for $r, r' \in [t_P, d+1]^2$, it has no choice but to solve this system of equations to try and find the missing evaluations of $g$. If we want information theoretic level of security on the $\ell$ secrets we need to guarantee two things. First, the adversary cannot gather information on the secrets with the evaluations of $g$ discovered during `Recover`. This gives a first bound:

$$\ell \leq (d + 2 - t_P)^2$$

Second, we also need to verify that the adversary cannot use the overall information received by $P_{r_C}$ and the rest of corrupted parties. For that, the difference between the total number of equations and unknown values needs to be higher than $\ell$, giving the inequality:

$$\ell \leq 2(d + 2 - t_P)^2 - (d + 2 - t_P)^2 - (d + 2 - t_P) = (d + 2 - t_P)(d + 1 - t_P)$$

Both inequalities are verified and if we set the threshold

$$\ell \leq (d + 1 - t_P)^2 \quad \Rightarrow \quad t_P \leq d + 1 - \sqrt{\ell}$$

For a maximum level of security we take $d = n-2$ and so we obtain the maximum threshold of $n-1-\lfloor\sqrt{\ell}\rfloor$ for the total number of passive corruptions. In this case, we are guaranteed that the adversary cannot find the values $s_1, \ldots, s_\ell$ from the informations seen in `Recover`, and this concludes the proof of Lemma 2. The bound of $t_P \leq n - 1 - \sqrt{\ell}$ is not the tightest possible for `Recover`, but it is asymptotically optimal. Later, it will appear that this bound will be the tightest possible for the overall PSS. □

**Figure 5**. Ideal Process for the `Recover` 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 $\mathcal{F}_{\texttt{Recover}}$ that will play the role of a trusted third party. Upon reception of the parties' inputs that are the bivariate shares for a batch of secrets $s_1, \ldots, s_\ell$, the functionality will output new shares for all the parties $P_1, \ldots, P_{d+1}$ and $P_{r_C}$.

*Parameters.* Multi-thresholds $T_c, T_s$.

*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 $x_r$.
3. $\mathcal{S}$ may change the input for the actively corrupted parties.
INPUTS
4. Each party sends his input to the ideal functionality.
CORRUPTION
5. If $(t_P, t_A) \not\leq T_s$: $\mathcal{F}_{\texttt{Recover}}$ sends all inputs to $\mathcal{S}$.
COMPUTATION
6. $\mathcal{F}_{\texttt{Recover}}$ evaluates $g$ to obtain the values $s_1, \ldots, s_\ell$ and compute a new sharing $g'$ as the output of the `Recover` protocol, shares for $P_{r_C}$ are included.
OUTPUTS
7. $\mathcal{F}_{\texttt{Recover}}$ sends the shares $g'(\alpha_r, \alpha_{r'})$, verifying $g(x, x) = g'(x, x)$ for all $x$, for the corrupted parties $P_r \in \mathcal{P}_P$ to $\mathcal{S}$.
8. $\mathcal{S}$ sends either `abort` to $\mathcal{F}_{\texttt{Recover}}$ and $\mathcal{F}_{\texttt{Recover}}$ aborts and outputs $\perp$ or `continue` and the functionality proceeds to the next step.
9. If $(t_P, t_A) \not\leq T_c$: $\mathcal{S}$ sends new shares to $\mathcal{F}_{\texttt{Recover}}$ instead of the evaluations of $g$. $\mathcal{F}_{\texttt{Recover}}$ replaces the output..
10. $\mathcal{F}_{\texttt{Recover}}$ sends their shares to the honest parties.

*Outputs.* Each honest party outputs whatever they received from $F_{\texttt{Recover}}$. 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 $b_{\mathcal{Z}}$.

**Figure 6**. Simulator for the `Recover` protocol.

1. If $(t_P, t_A) \leq T_s$ $\mathcal{S}$ sets $g_{\mathcal{S}}$ as $g$ the bivariate sharing that is the input of the parties. Otherwise, $\mathcal{S}$ generates a random $g_{\mathcal{S}}$ verifying $g_{\mathcal{S}}(\alpha_r, y) = g(\alpha_r, y)$ for corrupted $P_r$.
2. $\mathcal{S}$ generates a new sets of commitments for the values of $g_{\mathcal{S}}$ and initializes the adversary with these new commitments.
3. $\mathcal{S}$ broadcasts the commitments to the values $g(\alpha_r, \alpha_{r'})$ for honest $P_r$ and $r' \in [d+1]$.
4. If $\mathcal{A}$ broadcasts incorrect commitments or if $\mathcal{A}$ sends a complaining bit indicating that a honest parties has revealed the wrong commitments, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\texttt{Recover}}$, otherwise proceeds to the next steps.
5. On behalf of the honest parties $P_r \in \{P_1, \ldots, P_{d+1}\}$, the simulator generates a random polynomial $f_r$ verifying $f_r(\alpha_{r_C}) = 0$ and sends commitments to its coefficients to the adversary.
6. $\mathcal{S}$ verifies that 0 is a correct opening for the values $f_u(\alpha_{r_C})$ of corrupted parties $P_u$. If one is not correct or if $\mathcal{A}$ indicates that one of the polynomial for honest parties is not correct, $\mathcal{S}$ sends `abort` to the ideal functionality. Otherwise, proceeds to the next steps.
7. $\mathcal{S}$ sends to all corrupted parties $P_{r'}$, the opening to the value $f_r(\alpha_{r'})$ for all honest $P_r$, the simulator receives the corresponding opening from $\mathcal{A}$.
8. If any of the commitments is not correct or if $\mathcal{S}$ receives a complaining bit, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\texttt{Recover}}$.
9. If $P_{r_C}$ is honest, $\mathcal{S}$ receives from $\mathcal{A}$ the opening for the values $g_{\mathcal{S}}(\alpha_r, \alpha_{r'}) + f_{r'}(\alpha_r)$ for all $r' \in [d+1]$ and all corrupted parties $P_r$. If one of the opening is not correct, the simulator sends `abort` to $\mathcal{F}_{\texttt{Recover}}$.
10. If $P_{r_C}$ is corrupted, $\mathcal{S}$ sends to $\mathcal{A}$ the opening for the values $g_{\mathcal{S}}(\alpha_r, \alpha_{r'}) + f_{r'}(\alpha_r)$ for all $r' \in [d+1]$ and all honest parties $P_r$. If $\mathcal{S}$ receives a complaining bit from the adversary, it sends `abort` to the ideal functionality.
11. The simulators outputs whatever $\mathcal{A}$ outputs.

**Theorem 5.** *The* `Recover` *protocol securely realizes* $\text{IDEAL}_{mixed}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{\texttt{Recover}}}(z, T_c, T_s)$ *with multi-threshold* $T_c = \{(n, n-1)\}$ *and* $T_s\{(n-1-\lfloor\sqrt{\ell}\rfloor, n-1-\lfloor\sqrt{\ell}\rfloor)\}$.

*Proof.* First, let us recall that the perfectly hiding property of the commitment guarantees that no information can be learned from the commitments. The computationally binding and homomorphic properties ensures correctness of any linear operation if we assume the hardness of the DLP problem. Let us prove indistinguishability of the view for the adversary in the real and ideal situations.

Due to the commitments, the input distribution is the same in both cases Then, $\mathcal{S}$ will generate the $f_u$ polynomials for honest $P_u$ following the protocol specifications. To any misbehavior of $\mathcal{A}$ it aborts. For the last steps of the Simulator described in Fig. 6, the situations depends on whether $P_{r_C}$ is corrupted or not. If $P_{r_C}$ is honest, the adversary just sends the opening for the values $g_{\mathcal{S}}(\alpha_r, \alpha_{r'}) + f_{r'}(\alpha_r)$ from all corrupted $P_r$ and $r' \in [d+1]$. If any misbehavior is detected by the simulator it aborts, and the adversary do not get more information than that. In this case, the adversary sees exactly what he would have seen in a real execution. The case where $P_{r_C}$ is corrupted is more problematic. In

this situations, the adversary sees the values $g_{\mathcal{S}}(\alpha_r, \alpha_{r'}) + f_{r'}(\alpha_r)$ for the honest parties and $r' \in [d+1]$. We need to show that this is indistinguishable from the real execution when $g_{\mathcal{S}}$ is really $g$ the bivariate sharing for $s_1, \ldots, s_\ell$. Since, $g$ is supposed to be a random bivariate sharing of the batch of secrets, the only way for the adversary to distinguish between the two is to show that at least one value $g_{\mathcal{S}}(\beta_j, \beta_j)$ is different from $g(\beta_j, \beta_j)$ for $1 \leq j \leq \ell$. Due to Lemma 2, for the adversary the $\ell$ values $g_{\mathcal{S}}(\beta_1, \beta_1), \ldots, g_{\mathcal{S}}(\beta_\ell, \beta_\ell)$ are perfectly random and two distributions of evaluations in real and ideal execution cannot be distinguished. This conclude the proof of indistinguishability for the adversary's output. As we mentioned, $\mathcal{S}$ aborts if and only if $\mathcal{A}$ misbehaves and is detected. With the homomorphic commitments, any misbehavior is detected and so the protocol aborts with the same probability in both cases. As the commitments ensure correctness or abortion, the output of the honest parties is also correct as it is in the ideal execution. □

### B.3 Batched Gradual VSS and the `Reconstruct` Protocol

We are going to prove the security of the `Reconstruct` protocol. First, let us prove the following lemma:

**Lemma 3.** *The `Reconstruct` protocol is correct when $(t_P, t_A) \leq \{(n, n-1)\}$ and it provides information theoretic security on the $\ell$ secret when the fairness property of Definition 6 is respected.*

*Proof. Correctness:* With the homomorphic commitments, actively corrupted participant trying to deviate from the protocol are immediately caught. The parties obtains the correct output $s_1, \ldots, s_\ell$ in the end, because the $g_1, \ldots, g_d$ bivariate polynomials are generated so that $g(x, y) = \sum_{i=1}^{d} g_i(x, y)$ for all $x$ and $y$. The correctness of `Recover` guarantees that the parties obtain evaluations of polynomials with the desired degrees, and when some of the parties obtain evaluations of a polynomial of degree $i$ that they did not generated themselves, it is always by a consistent interpolation with the $(i+1)^2$ original values.

*Fairness* To prove Lemma 3, we need to verify the additional fairness property (see the Definition of Batched Gradual VSS in Section 3). Before going for the actual proof, let us recap what informations are seen by the adversary and what amont of information $\mathcal{A}$ lacks before being able to reconstruct the secrets at any point of `Reconstruct`. Let us denote $U_i$ the number of unknown evaluations required by the adversary to reconstruct the secrets $s_1, \ldots, s_\ell$ after the revelation of $g_d, \ldots, g_i$ and $V_i$ the total number of equations obtained on these unknown values during all previous iteration of the loop and the revelation of $g_d, \ldots, g_i$ in step 2.1. We also denote $u_i$ the number of missing values for the adversary to interpolate the values $Q_{i-1}(\beta_j, \beta_j)$ for any $j \in [\ell]$ and $v_i$ the number of equations revealed during step 2.1 on the $u_i + u_{i+1}$ evaluations of $Q_{i-1}$ and $Q_i$ unknown to the adversary. With $u_{d+1}$ the number of missing value about $g$, we have $U_i = \sum_{k=i}^{d} \tau_k u_k + u_{d+1}$ and $V_i = \sum_{k=i}^{d} \tau_k v_k$, where $\tau_k$ is the number of time the loop is performed for a given value $i = k$ due to incorrect opening in step 2.1.

Using these notations, let us consider that the secrets are protected by $u_{d+1} = (d+1-t_P)^2$ values $g(\alpha_r, \alpha_{r'})$ for honest parties $P_r, P_{r'}$ in a subset of size $d+1-t_P$ among the honest parties in $\{P_1, \ldots, P_n\} \setminus \mathcal{P}_P$. Let us denote $\mathcal{P}_P^i = \{P_1, \ldots, P_{i+1}\} \cap \mathcal{P}_P$, $t_P^i = |\mathcal{P}_P^i|$ and $\delta_i = t_P^i - t_P^{i-1}$ indicates if $P_{i+1}$ is corrupted for all $i \in [d]$. During the execution of `Recover` in step 1.3 for $i = d$ or step 2.6, the adversary will have $t_P^i$ corrupted parties in the exchange. If $\delta_i = 0$ and $P_{i+1}$ is not corrupted, there are at least $u_i = i(i - t_P^i)$ unknown evaluations of $Q_{i-1}$ (see Appendix B.2 for more details about the amount of information exchanged during `Recover`). During step 2.1 of the next iteration in the while loop, the broadcast of $g_i$'s evaluations from $P_1, \ldots, P_{i+1}$ will reveal $v_i = (i+1-t_P^i)^2$ new equations on the unknown evaluations of $Q_{i-1}$ and $Q_i$ (or $Q_{d-1}$ and $g$ if $i = d$). If $\delta_i = 1$, there are $u_i = 3(i+1-t_P^i)^2$ new unknown values, and the adversary obtains $(i+1-t_P^i)^2 + 2(i+1-t_P^i)$ equations on them during `Recover`. Then, during step 2.1, the adversary will obtain $(i+1-t_P^i)^2$ new equations on the unknown evaluations, with that we have $v_i = 2(i+1-t_P^i)^2 + 2(i+1-t_P^i)$.

If `Reconstruct` aborts it is either during the execution `Recover` in step 1.3 or 2.6, or during the step 2.3. We are not going to consider the case of abortion during `Recover`. Indeed, the recoveries are made on polynomial $Q_{i-1}^*$ for $2 \leq i \leq d$ that are perfectly random and do not carry any information on the secrets at the time of their generation. If the adversary aborts during the recovery, it could have done so in the previous iteration of step 2.1 and obtain the same amount of information on the secrets. We consider that the abortion occurred after the revelation of the evaluations of $g_{i_0}$ for some $1 \leq i_0 \leq d$. At this point, $g_d, \ldots, g_{i_0}$ have been revealed and the parties have been able to compute $\sum_{k=i_0}^{d} g_k(\beta_j, \beta_j) = s_j - \Lambda_{i_0-1} Q_{i_0-1}(\beta_j, \beta_j)$ for any $j \in [\ell]$. If the adversary is able to compute the value $Q_{i_0-1}(\beta_j, \beta_j)$, $\mathcal{A}$ can reconstruct the value of $s_j$ for any $j$. If the protocol aborts and the parties output the set $B$ of corrupted parties, we have $n - |B| < i_0$. We need to show that the adversary cannot recover the secrets if $t_P \leq n - |B| - \lfloor\sqrt{\ell}\rfloor < i_0 - \lfloor\sqrt{\ell}\rfloor$ (see Definition 6 for Batched Gradual Secret Sharing). If we have the bound $t_P < i_0 - \lfloor\sqrt{\ell}\rfloor$ the adversary cannot use the $t_P * (i_0 + 1)$ evaluations of $Q_{i_0-1}$ to directly interpolate the missing values. The inequality $t_P^{i_0} < i_0 - \lfloor\sqrt{\ell}\rfloor$ is also verified. We have showed in the proof of secrecy for the `Recover` protocol, that this was sufficient to grant information theoretic security on the $\ell$ values $Q_{i_0-1}(\beta_j, \beta_j)$ for all $j$. This shows that the adversary cannot find the random secrets stored in $Q_{i_0-1}$ using only the information gained during the execution of `Recover`. If the adversary attempts to combine all the equations obtained during the reconstruction, we need to have $\ell \leq U_{i_0} - E_{i_0}$ to guarantee the information theoretic security of the $\ell$ secrets. $U_{i_0} - V_{i_0} = u_{d+1} + \sum_{k=i_0} \tau_k(u_k - v_k)$. Using the expressions of $u_i$ and $v_i$ given previously it is easy to verify that $u_k - v_k > 0$ for all $k \in [i_0, d]$ if $t_P < i_0 - \lfloor\sqrt{\ell}\rfloor$. We have also $\ell \leq (i+1-t_P)^2(d+1-t_p)^2 \leq u_{d+1}$ and so the inequality $\ell \leq U_{i_0} - V_{i_0}$ is verified. Very similarly we can verify that the adversary was not able to reconstruct the secrets in any previous iterations with $i \leq i_0$. This concludes the proof that the adversary cannot recover the secrets $s_1, \ldots, s_\ell$ if $t_P \leq n - |B| - \lfloor\sqrt{\ell}\rfloor$ and the proof of Lemma 3. Thus, our Secret

Sharing verifies Definition 6. The ideal functionality will contain the properties we want to reach with that definition. □

**Figure 7**. Ideal Process for the `Reconstruct` 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 $\mathcal{F}_{\texttt{Reconstruct}}$ that will play the role of a trusted third party. Upon reception of the input that is a sharing of a batch of secret $s_1, \ldots, s_\ell$, the functionality will output the secrets $s_1, \ldots, s_\ell$.

*Parameters.* Multi-thresholds $T_c, T_s$.

*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 $x_r$.
3. $\mathcal{S}$ may change the input for the corrupted parties.
INPUTS
4. Each party sends his input to the ideal functionality.
CORRUPTION
5. If $(t_P, t_A) \not\leq T_s$: $\mathcal{F}_{\texttt{Reconstruct}}$ sends all inputs to $\mathcal{S}$.
COMPUTATION
6. $\mathcal{F}_{\texttt{Reconstruct}}$ evaluates $g$ to obtain the values $s_1, \ldots, s_\ell$.
OUTPUTS
7. $\mathcal{S}$ sends either `abort` and a set $B$ of the parties to $\mathcal{F}_{\texttt{Reconstruct}}$ and $\mathcal{F}_{\texttt{Reconstruct}}$ aborts and outputs $\perp$ or `continue` and the functionality proceeds to the next step.
8. If $(t_P, t_A) \not\leq T_s$ and $t_P \not\leq n - |B| - \lfloor\sqrt{\ell}\rfloor$: $\mathcal{F}_{\texttt{Reconstruct}}$ sends $s_1, \ldots, s\ell$ to $\mathcal{S}$.
9. If $(t_P, t_A) \not\leq T_c$: $\mathcal{S}$ sends $s_1 + \delta_1, \ldots, s_\ell + \delta_\ell$ to $\mathcal{F}_{\texttt{Reconstruct}}$ and the functionality replace the batch of secrets by the new values.
10. $\mathcal{F}_{\texttt{Reconstruct}}$ sends their outputs to the honest parties.

*Outputs.* Each honest party outputs whatever they received from $F_{\texttt{Reconstruct}}$. 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 $b_{\mathcal{Z}}$.

In the simulator in Fig. 8 when we say that $\mathcal{S}$ performs the `Reconstruct` protocol with the adversary given some input for all the parties, it means that $\mathcal{S}$ follows exactly the protocol in Protocol 3 on behalf of the honest parties and interact with $\mathcal{A}$ who performs the protocol on behalf of the corrupted parties.

**Figure 8**. Simulator for the `Reconstruct` protocol.

1. If $(t_P, t_A) \not\leq T_s$ $\mathcal{S}$ sets $g_{\mathcal{S}}$ as $g$ the bivariate sharing that is the input of the parties. Otherwise, $\mathcal{S}$ generates a random $g_{\mathcal{S}}$ verifying $g_{\mathcal{S}}(\alpha_r, y) = g(\alpha_r, y)$ for corrupted $P_r$.
2. $\mathcal{S}$ generates a new sets of commitments for the values of $g_{\mathcal{S}}$ and initializes the adversary with these new commitments.
3. $\mathcal{S}$ performs the protocol `Reconstruct` with $\mathcal{A}$.
4. If the protocol does not abort, $\mathcal{S}$ sends `continue` to the ideal functionality and proceed to the next steps. If the protocol aborts with a set $B$ of corrupted parties. $\mathcal{S}$ sends `abort` to the ideal functionality along with $B$ the set of parties. If $t_P \not\leq n - |B| - \lfloor\sqrt{\ell}\rfloor$ $\mathcal{S}$ proceeds to the next step. Otherwise, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.
5. $\mathcal{S}$ receives from $\mathcal{F}_{\texttt{Reconstruct}}$ the values $s_1, \ldots, s_\ell$.
6. $\mathcal{S}$ generates a random $g'_{\mathcal{S}}$ with $g'_{\mathcal{S}}(\beta_j, \beta_j) = s_j$ for $j \in [\ell]$ verifying $g_{\mathcal{S}}(\alpha_r, y) = g(\alpha_r, y)$ for corrupted $P_r$.
7. $\mathcal{S}$ generates a new sets of commitments for the values of $g'_{\mathcal{S}}$ and initializes the adversary with these new commitments.
8. $\mathcal{S}$ performs the protocol `Reconstruct` with $\mathcal{A}$.
9. If $(t_P, T_A) \not\leq T_C$, and the obtained values from the `Reconstruct` protocol are different from $s_1, \ldots, s_\ell$, $\mathcal{S}$ sends the new outputs to $\mathcal{F}_{\texttt{Reconstruct}}$.
10. The simulators outputs whatever $\mathcal{A}$ outputs.

**Theorem 6.** *The* `Reconstruct` *protocol securely realizes*

$$\text{IDEAL}_{mixed}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{\texttt{Reconstruct}}}(z, T_c, T_s)$$

*with multi-threshold* $T_c = \{(n, n-1)\}$ *and* $T_s\{(n-1-\lfloor\sqrt{\ell}\rfloor, n-1-\lfloor\sqrt{\ell}\rfloor)\}$.

*Proof.* The reconstruction protocol aims at revealing the values $s_1, \ldots, s_\ell$, the simulator cannot hope to generate an indistinguishable execution if it does not know correct shares for honest parties. To construct those, $\mathcal{S}$ has no choice but to obtain these values from the ideal functionality. However, due to the fairness property that we want to verify, the simulator does not obtain the values for the batch of secrets before indicating if the adversary aborts the protocol. Thus, the simulator has to perform a first execution of the protocol with $\mathcal{A}$ to know if it intends to abort. Even if this first simulation is not made on correct inputs, we will show that it is indistinguishable from the real execution if the adversary decides to abort. If not, the simulator has to run the protocol a second time with the correct input in order to produce an indistinguishable output from the real execution.

We look at the execution of the `Reconstruct` protocol when the inputs have been modified in order to share a set of random values generated by $\mathcal{S}$. We will show that when the protocol aborts, the adversary's view was indistinguishable from the real execution. Let us consider that $\mathcal{A}$ aborts during step $i_0$ of the `Reconstruct` protocol, the set of misbehaving parties is $B$. Since the protocol

aborted we have that $n - |B| < i_0 + 1$, otherwise the parties would have been able to continue without the actively corrupted parties. In the case where $t_P \leq n - t_A - \lfloor\sqrt{\ell}\rfloor \leq n - |B| - \lfloor\sqrt{\ell}\rfloor$ it was shown in Lemma 3 that the adversary was not able to recover the values hidden in the sharing. For the same reason, it is not possible to distinguish when the secrets $s_1, \ldots, s_\ell$ have been replaced by completely random values and $\mathcal{A}$'s output is indistinguishable from the real adversary's output. If we are not in this special case, either the adversary will not abort or the gradual threshold is violated.In both cases, the adversary will know that $s_1, \ldots, s_\ell$ have been replaced by random values. That is why the simulator must do a second execution of `Reconstruct` wit $\mathcal{A}$.

This second execution is necessarily indistinguishable from the real one in the adversary's point of view as the inputs are correct and $\mathcal{S}$ merely follows the protocols. This ensures that the simulator outputs something that is always indistinguishable from the adversary's output in the real execution. If the correctness is ensured when $(t_P, t_A) \leq T_c$, this is also true for the honest parties output. The correctness of the `Reconstruct` follows from Lemma 3. □

### B.4 The `Refresh` protocol

In this Section, we present the simulator-based proof for the `Refresh` protocol. Contrary to all the other protocols we cannot assume that the corruption is static during the execution of the `Refresh` protocol. Thus, we assume that there are three distinct sets of corrupted parties. Each of the sets are labelled $\mathcal{P}_A^i$ and $\mathcal{P}_P^i$ for $i \in \{1, 2, 3\}$. This division is made according to our corruption model for the Proactive adversary (see Appendix A.2). The first set is the set of parties corrupted before the `Refresh` protocol but that are not corrupted during the execution of `Refresh`, the adversary knows these participant's input when they enter the protocol. The second set refers to parties corrupted before, during and after `Refresh`, the adversary knows everything they see before, during and after the protocol. Finally, the third set corresponds to parties corrupted only after the execution of the `Refresh`, the adversary knows the output of the `Refresh` protocol for these parties. To ensure the security of the `Refresh` protocol, the security threshold will need to be respected before, during and after the protocol. We denote $t_P^1 = |\mathcal{P}_P^1| + |\mathcal{P}_P^2|$, $t_P^2 = |\mathcal{P}_P^2|$, $t_P^3 = |\mathcal{P}_P^2| + |\mathcal{P}_P^3|$. In our model $\mathcal{P}_P^1 \cap \mathcal{P}_P^2 = \emptyset$, $\mathcal{P}_P^2 \cap \mathcal{P}_P^3 = \emptyset$ and we have the bounds $|\mathcal{P}_P^1 \cup \mathcal{P}_P^2| \leq d + 1 - \lfloor\sqrt{\ell}\rfloor$, $|\mathcal{P}_P^3 \cup \mathcal{P}_P^2| \leq d + 1 - \lfloor\sqrt{\ell}\rfloor$. We start by showing the following lemma:

**Lemma 4.** *`Refresh` operates correctly to refresh the shares of a set $\mathcal{P}$ of parties without revealing the $\ell$ secrets when the bound $t_P \leq d + 1 - \sqrt{\ell}$ at all times.*

*Proof. Correctness:* We just need to verify that `Refresh` operates correctly. $R$ is shared correctly because `Recover` is correct. $h$ is generated properly because of the homomorphic commitments. If $R$ and $h$ are computed correctly, it is easy to see that $g'(\beta_j, \beta_j) = g(\beta_j, \beta_j)$.

*Secrecy:* In the case of dishonest majority, we require from the `Refresh` protocol to provide secrecy against an adversary controlling a part of the parties

before the execution, and a part of the parties after the execution (the number of corruption both before and after is limited with $t_P \leq d+1-\lfloor\sqrt{\ell}\rfloor$, each $P_r$ is corrupted either before or after, some of the parties might be corrupted during the execution. We also denote $\mathcal{P}_P^R$ the set $\mathcal{P}_P^2 \cap \{P_{d+1}, \ldots, P_n\}$ and $t_P^R = |\mathcal{P}_P^R|$

We assume that the adversary knows the values $g(\alpha_r, \alpha_{r'})$ when $r' \in [d+1]$ and $P_r \in \mathcal{P}_P^1 \cup \mathcal{P}_P^2$. There are $(d+1-t_P^1)^2$ unknown values $g(\alpha_r, \alpha_{r'})$ for $P_r, P_{r'} \in \{P_1, \ldots, P_n\} \setminus (\mathcal{P}_P^1 \cup \mathcal{P}_P^2)$ about the bivariate sharing $g$. Next, we will try to assess how much information can be gathered by the adversary on $R$ during the execution of `Recover`. We refer to Section B.2 for a more detailed analysis of `Recover`. The adversary knows $R(\alpha_r, \alpha_{r'})$ for $P_r \in \mathcal{P}_P^2 \setminus \mathcal{P}_P^R$ and $r' \in [d]$ and is missing $(d+t_P^R-t_P^2)^2$ evaluations to recover it entirely. During the execution of `Recover`, $\mathcal{A}$ is able to gather (via participants $P_i \in \mathcal{P}_P^R$) $t_P^R \cdot \Big((d+t_P^R-t_P^2)^2+(d+t_P^R-t_P^2)\Big)$ equations on the $(d+t_P^R-t_P^2)^2+t_P^R\cdot(d+t_P^R-t_P^2)^2$ unknown values. The $t_P^R \cdot (d+t_P^R-t_P^2)^2$ additional unknowns are the $f_{r'}(\alpha_r)$ values for honest participants $P_r, P_{r'} \in \{P_1, \ldots P_n\} \setminus \mathcal{P}_P^2$ generated during the executions of `Recover` between $\{P_1, \ldots, P_d\}$ and corrupted participants of $\mathcal{P}_P^R$. Additionally, the adversary is missing $d+1-t_P^2$ values $h(\alpha_r)$ for honest $P_r$. Finally, when $\mathcal{A}$ corrupts the participants in $\mathcal{P}_P^3$, it sees the values $g'(\alpha_r, \alpha'_r)$ for $r' \in [d+1]$ and $P_r \in \mathcal{P}_P^3$. The adversary already knew the value $g'(\alpha_r, \alpha_{r'})$ with $P_{r'} \in \mathcal{P}_P^2$ and so the adversary obtains $|\mathcal{P}_P^3|(d+1-t_P^2) = (t_P^3-t_P^2)(d+1-t_P^2)$ new equations. The total number of equations obtained by the adversary is $(t_P^3-t_P^2)(d+1-t_P^2) + t_P^R \cdot \Big((d+t_P^R-t_P^2)^2+(d+t_P^R-t_P^2)\Big)$. The number of unknowns is $(d+1-t_P^1)^2+(d+t_P^R-t_P^2)^2+t_P^R\cdot(d+t_P^R-t_P^2)^2+d+1-t_P^2$. The difference between the two is

$$\Delta = (d+1-t_P^1)^2+(d+t_P^R-t_P^2)^2+t_P^R\cdot(d+t_P^R-t_P^2)^2+(d+1-t_P^2)-$$
$$(t_P^3-t_P^2)\cdot(d+1-t_P^2) - t_P^R\cdot\Big((d+t_P^R-t_P^2)^2+(d+t_P^R-t_P^2)\Big)$$
$$\Delta = (d+1-t_P^1)^2+(d+t_P^R-t_P^2)(d-t_P^2)+d+1-t_P^2-(t_P^3-t_P^2)(d+1-t_P)$$
$$\Delta = (d+1-t_P^1)^2+(d+1-t_P^2)(d+t_P^R-t_P^3)-(t_P^R-1)$$

Thus, the bound $t_P^i \leq d+1-\sqrt{\ell}$ for all $i \in \{1,2,3\}$ gives the inequality $\ell \leq \Delta$ and this proves that we have information theoretic security on the secrets $s_1, \ldots, s_\ell$. □

In the next proof, unless stated otherwise, the set of corrupted participant is $\mathcal{P}_P^2$, the set of parties corrupted before, during and after the `Refresh` protocol. The simulator used in the proof of Theorem 7 relies on another simulator $\mathcal{S}_{\texttt{Recover}}$. This Simulator is the one described in Figure 6.

**Figure 9**. Ideal Process for the `Refresh` 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 $\mathcal{F}_{\texttt{Refresh}}$ that will play the role of a trusted third party that will compute the refreshing of the shares. Upon reception of the input that is a sharing of a batch of secret $s_1, \ldots, s_\ell$, the functionality will output a new sets of shares for the same batch of secrets.

*Parameters.* Multi-thresholds $T_c, T_s$.

*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 divided in three subgroups $\mathcal{P}_A^i$ and $\mathcal{P}_P^i$ for $i \in \{1, 2, 3\}$.
2. $\mathcal{Z}$ invokes the parties $P_r$ and their inputs $x_r$.
3. $\mathcal{S}$ receives the inputs for the parties in $\mathcal{P}_P^1 \cup \mathcal{P}_P^2$ and may change the input for the corrupted parties among this set.
INPUTS
4. Each party sends his input to the ideal functionality.
CORRUPTION
5. If $(t_P^1, t_A^1) \not\leq T_s$: $\mathcal{F}_{\texttt{Refresh}}$ sends all inputs to $\mathcal{S}$.
COMPUTATION
6. $\mathcal{F}_{\texttt{Refresh}}$ evaluates $g$ to obtain the values $s_1, \ldots, s_\ell$. And generate a new bivariate sharing $g'$ for $s_1, \ldots, s_\ell$.
OUTPUTS
7. $\mathcal{F}_{\texttt{Refresh}}$ sends the shares $g'(\alpha_r, \alpha_{r'})$,for the corrupted parties $P_r \in \mathcal{P}_P^2 \cup \mathcal{P}_P^3$ to $\mathcal{S}$.
8. $\mathcal{S}$ sends either `abort` to $\mathcal{F}_{\texttt{Refresh}}$ and $\mathcal{F}_{\texttt{Refresh}}$ aborts and outputs $\perp$ or `continue` and the functionality proceeds to the next step.
9. If $(t_P^3, t_A^3) \not\leq T_s$ $\mathcal{F}_{\texttt{Refresh}}$ sends $s_1, \ldots, s_\ell$ to $\mathcal{S}$.
10. If $(t_P^2, t_A^2) \not\leq T_c$: $\mathcal{S}$ sends new shares to $\mathcal{F}_{\texttt{Refresh}}$ instead of the evaluations of $g'$. $\mathcal{F}_{\texttt{Refresh}}$ replaces the output.
11. $\mathcal{F}_{\texttt{Refresht}}$ sends their outputs to the honest parties.

*Outputs.* Each honest party outputs whatever they received from $F_{\texttt{Refresh}}$. 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 $b_{\mathcal{Z}}$.

**Figure 10**. Simulator for the `Refresh` protocol.

1. If $(t_P^1, t_A^1) \not\leq T_s$ $\mathcal{S}$ sets $g_{\mathcal{S}}$ as $g$ the bivariate sharing that is the input of the parties. Otherwise, $\mathcal{S}$ generates a random $g_{\mathcal{S}}$ verifying $g_{\mathcal{S}}(\alpha_r, y) = g(\alpha_r, y)$ for corrupted $P_r$ in $\mathcal{P}_P^1 \cup \mathcal{P}_P^2$.

2. $\mathcal{S}$ generates a new sets of commitments for the values of $g_{\mathcal{S}}$ and initializes the adversary with these new commitments. After this step, the sets of corrupted parties is $\mathcal{P}_P^2$ and $\mathcal{P}_A^2$.
3. $\mathcal{S}$ generates for all the honest parties $P_r$ in $\{P_1, \ldots, P_d\}$ a random polynomial $R_r$ of degree $d-1$, and sends to $\mathcal{A}$ the commitments to the coefficients of $R_r$.
4. $\mathcal{S}$ uses the simulator $\mathcal{S}_{\texttt{Recover}}$ to simulate the successive interaction of $P_i \in \{P_{d+1}, \ldots, P_n\}$ with $\{P_1, \ldots, P_d\}$ during `Recover`.
5. $\mathcal{S}$ generates $h_r$ a random univariate polynomial of degree $d$ for all honest $P_r$ and sends to $\mathcal{A}$ the commitments to the coefficients. The simulator receives the corresponding commitments from the adversary.
6. $\mathcal{S}$ exchanges with $\mathcal{A}$ the opening for the evaluations of all the $h_r$ polynomials. If one error is detected on the opening of the adversary's values or if $\mathcal{A}$ sends a complaining bit, $\mathcal{S}$ sends `abort` to the ideal functionality. Otherwise, proceeds to the next step.
7. $\mathcal{S}$ compute the evaluations of $g'$ for all the honest parties. If $t_P^2, t_A^2 \not\leq T_C$, $\mathcal{S}$ sends to $\mathcal{F}_{\texttt{Refresh}}$ the new evaluations
8. $\mathcal{S}$ sends to $\mathcal{A}$ the evaluations of the polynomial $g'$ for the newly corrupted parties in $\mathcal{P}_P^3$.
9. The simulators outputs whatever $\mathcal{A}$ outputs.

**Theorem 7.** *The* `Refresh` *protocol securely realizes* $\text{IDEAL}_{mixed}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{Refresh}}(z, T_c, T_s)$ *with multi-threshold* $T_c = \{(n, n-1)\}$ *and* $T_s\{(n-1-\lfloor\sqrt{\ell}\rfloor, n-1-\lfloor\sqrt{\ell}\rfloor)\}$.

*Proof.* To prove the indistinguishability of output in the two situations we start by showing indistinguishability of the view for the adversary. If $t_P^1, t_A^1 \not\leq T_s$, the simulator obtains the input values of the honest parties. If not, $\mathcal{S}$ generates his own random polynomial $g_{\mathcal{S}}$ that coincides with $g$ on the evaluations of corrupted parties. Due to the perfectly hiding property of the commitment scheme, in both cases the input distribution is identical to the one in the real execution. The simulator $\mathcal{S}_{\texttt{Recover}}$ produces an indistinguishable view from the real execution for all the interactions of step 2. Apart from that, it is easy to see that $\mathcal{S}$ behaves exactly as the set of honest parties would during the execution of `Refresh` with the evaluations of $g_{\mathcal{S}}$ as inputs. In the next-to-the-last step of the Simulator, $\mathcal{S}$ sends to $\mathcal{A}$ the shares for the newly corrupted parties in $\mathcal{P}_P^3$ to emulate the corruption phase happening after the `Refresh` phase. We showed in Lemma 4 that when the threshold for secrecy was respected by the 3 pairs $(t_P^i, t_A^i)$ for $i \in \{1, 2, 3\}$, there was information theoretic security on the values $s_1, \ldots s_\ell$. For the same reasons, $\mathcal{A}$ cannot distinguish between a real execution with a bivariate sharing of the batch of secrets and an execution with a random polynomial $g_{\mathcal{S}}$. Whenever $\mathcal{A}$ attempts to abort the protocol, $\mathcal{S}$ sends `abort` to the ideal functionality. When the correctness threshold is violated, $\mathcal{S}$ forward the values obtainds from the adversary to $\mathcal{F}_{\texttt{Refresh}}$, in this case the honest parties is the same that they would have obtained in the real execution. Otherwise, the protocol is correct as was proved for Lemma 4 (due to the homomorphic commitments) and the honest parties obtain a correct sharing for $s_1, \ldots, s_\ell$. □

## C An Efficient PSS with Full Threshold

In Section 3, we introduced an Batched Gradual PSS scheme. We obtained an overall improvement in communication complexity from $O(n^4)$ to $O(n^2)$ compared to the work of [17]. This improvement was made with the batching of $\ell = O(n)$ secrets instead of one, and the idea to transform the classical Secret Sharing into a Gradual one only before the reconstruction instead of performing every protocol with the gradual sharing. The improvement with batching comes at a cost, the maximum thresholds for corruptions are all reduced by a term $\lfloor\sqrt{\ell}\rfloor$. However, the other $O(n)$ improvement does not cost anything in terms of threshold. In this section, we briefly outline a Gradual PSS with an overall communication of $O(n^3)$ (instead of $O(n^4)$ in [17]) which avoids the threshold loss.

In this PSS, the `Share`, `Refresh` and `Recover` are very similar to the protocols presented in [17]. The only difference is that, each operation is made with one polynomial $f$ of degree $d$ with $f(0) = s$ instead of $d$ polynomials $f_1, .., f_d$. The major difference is in in the reconstruction protocol. To obtain the Gradual property we need to transform the [35]'s sharing into the one from [28] so that we can perform the reconstruction exactly as described in the `Reconstruct` protocol in [28,17]. This is done in $O(n^3)$ with a `ReshareGradual` protocol that is detailed in Protocol 9. This `Reshare` is directly inspired from the `Refresh` in [17]. The aim is to generate d polynomials $e_1, \ldots, e_d$ of degree $1, \ldots, d$ verifying $\sum_{i=1}^{d} e_i(0) = 0$. Then, we can set $f_i = e_i$ for all $1 \leq i \leq d-1$ and $f_d = f + e_d$.

**Protocol 9**. `Reshare`

INPUT: Each party $P_r \in \mathcal{P}$ has the value $f(\alpha_r)$ with $f$ a polynomial of degree $d$ and $f(0) = s$ for a secret $s$.

OUTPUT: Each party has the values $(f_i(\alpha_r))_{i\in[d]}$ such that $\sum_{i=1}^{d} f_i(\alpha_r) = s$.

1. For all $r \in [n]$, $P_r$ generates random polynomials $(e_i^r)_{i\in[d]}$ of degree $d$ with $\sum_{i=1}^{d} e_i^r(0) = 0$. And broadcasts commitments to the coefficients of $e_i^r$ for all $i$.
2. For all $r, r' \in [n]^2$ and $i$, $P_r$ sends to $P_{r'}$ an opening to the commitment of $e_i^r(\alpha_{r'})$. $P_{r'}$ is able to compute locally a commitment to this value. $P_r$ broadcasts complaining bits indicating if the openings received from $P_r$ is a correct opening.
3. For each share $e_r^i(\alpha_{r'})$, 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 $B$ of corrupted parties.
4. Each participant $P_r$ sets its value $f_i(\alpha_r) = \sum_{u=1}^{n} e_u^i(\alpha_r)$ for all $i \in [d-1]$, and $f_d(\alpha_r) = f(\alpha_r) + \sum_{u=1}^{n} e_u^d(\alpha_r)$.

The Secret Sharing that we briefly described above can easily be extended to a DPSS using a `Redistribute` protocol very similar to Protocol 8. `DecreaseCorrupt`,

`Decrease` and `Increase` can be sequentially performed on the univariate sharing $f$ and this will perform the secret redistribution from one group of party to another.

# D Proofs for the Dynamic Groups Protocols

## D.1 Proofs for the Dynamic groups protocol in the Semi Honest model

In this appendix section, we prove the security of the three sub-protocols `Increase`, `Decrease` and `DecreaseCorrupt` presented in Protocols 5 to 7. We only introduced these three protocols in the Semi-Honest model as the extension to the malicious adversary model is straight-forward with the homomorphic commitments. We use the Passive security definition (Definition 8). We introduce three functionalities $\mathcal{F}_{\texttt{Increase}}$, $\mathcal{F}_{\texttt{Decrease}}$, $\mathcal{F}_{\texttt{DecreaseCorrupt}}$ that are to be realized by our protocols. We will describe three simulators $\mathcal{S}_{\texttt{Increase}}$, $\mathcal{S}_{\texttt{Decrease}}$ and $\mathcal{S}_{\texttt{DecreaseCorrupt}}$ that will produce indistinguishable views from the adversary's one using only the inputs and outputs of the corrupted parties. In the three cases, the inputs are the shares for a univariate polynomial $f$ and the outputs are the shares for a new polynomial that will be denoted either $f^+$ or $f^-$ such that $f(\beta) = f^{\pm}(\beta)$ for an either extended or reduced set of corrupted parties. The initial state of corrupted participant is $\mathcal{P}_P$, and the new set is $\mathcal{P}_P^{\pm}$. We write $t_P = |\mathcal{P}_P|$ and $t_P^{\pm} = |\mathcal{P}_P^{\pm}|$. The corruption threshold proposed in the next Theorems are not always the tightest possible, but they grant secrecy in an unified manner.

*Increase.* The functionality $\mathcal{F}_{\texttt{Increase}}$, takes the shares of the initial set of parties $P_1, \ldots, P_n\}$, and distributes new shares to the extended set of parties $\{P_1, \ldots, P_{n+k}\}$.

**Theorem 8.** *The protocol* `Increase` *privately computes the functionality* $\mathcal{F}_{\texttt{Increase}}$ *against an adversary that is bounded by* $t_P \leq n-3$, $t_P^+ \leq n+k-3$.

*Proof.* During this protocol, the adversary's view includes the $Q_r$ polynomials for all corrupted parties in $\mathcal{P}_P^{\pm} \setminus \mathcal{P}_P$, the values $Q_r(\alpha_{r'})$ for honest $P_r$ and $P_{r'} \in \mathcal{P}_P^{\pm}$. Thus, the simulator $\mathcal{S}_{\texttt{Increase}}$ works as follows. From the output $f^+(\alpha_r)$ and the input $f(\alpha_r)$ for $P_r \in \mathcal{P}_P$, the simulator computes $Q(\alpha_r) = f^+(\alpha_r) - \prod_{i=1}^{k} \frac{\alpha_r - \alpha_{n+i}}{\beta - \alpha_{n+i}} f(\alpha_r)$. For corrupted parties in $\{P_{n+1}, \ldots, P_{n+k}\}$, $Q(\alpha_r)$ is just $f^+(\alpha_r)$. For all $P_r \in \mathcal{P}_P^{\pm}$, $\mathcal{S}_{\texttt{Increase}}$ can generate the values $Q_u(\alpha_r)$ $\forall u \in [n+k]$ as random values verifying $\sum_{u=1}^{n+k} Q_u(\alpha_r) = Q(\alpha_r)$. When $P_r \in \mathcal{P}_P^{\pm} \setminus \mathcal{P}_P$, the simulator generates the missing evaluations of the polynomial $Q_r$ so that $Q_r(\beta) = 0$.

In this case, the Simulator generates a view that has the same distribution as the adversary's view during a real execution. When the adversary is limited by the thresholds in Theorem 8 for $d = n-2$, the adversary has less than $d+1$ evaluations of $f$ and less than $d+k+1$ evaluations of $f^+$ and $Q$ (even when counting the additional information $Q(\beta) = 0$). Thus, $\mathcal{A}$ cannot recover the evaluations $f(\beta)$ or $f^+(\beta)$. □

*Decrease.* The functionality $\mathcal{F}_{\texttt{Decrease}}$, takes the shares of the initial set of parties $P_1, \ldots, P_n\}$, and distributes new shares to the reduced set of parties $\{P_1, \ldots, P_{n-k}\}$.

**Theorem 9.** *The protocol* `Decrease` *privately computes the functionality* $\mathcal{F}_{\texttt{Decrease}}$ *against an adversary that is bounded by* $t_P \leq n-3$, $t_P^- \leq n-k-3$.

*Proof.* In this protocol, apart from the inputs and outputs, the adversary sees the polynomials $Q_{n-r}$ when $P_r \in \mathcal{P}_P^- \setminus \mathcal{P}_P$, and receives the values $Q_{n-r}(\alpha'_r) + \Pi_{n-r}(\alpha_{r'})f(\alpha_r)$ when $P_r$ is a honest leaving party and $P_{r'} \in \mathcal{P}_P^-$. Now we can specify the action of $\mathcal{S}_{\texttt{Decrease}}$.

First, the simulator computes the sum $\sum_{i=0}^{k-1} Q_i(\alpha_r) + \Pi_i(\alpha_r)f(\alpha_{n-i}) = f^-(\alpha_r) - \prod_{i=0}^{k-1} \frac{\beta-\alpha_{n-i}}{\alpha_r-\alpha_{n-i}} f(\alpha_r)$ for all $P_r \in \mathcal{P}_P^-$. Then, $\mathcal{S}_{\texttt{Decrease}}$ generates $f_{\mathcal{S}}$ a random polynomial verifying $f(\alpha_r) = f_{\mathcal{S}}(\alpha_r)$ for all corrupted $P_r$. After that, the simulator can generate values $Q_i(\alpha_r)$ for all $P_r \in \mathcal{P}_P^-$ and $i \in [0, k-1]$ as random number that sum up to the desired value. Then, for $P_u \in \mathcal{P}_P^- \setminus \mathcal{P}_P$, $\mathcal{S}_{\texttt{Decrease}}$ generate the remaining part of the polynomials $Q_{n-r}$ so that $Q_{n-r}(\beta) = 0$.

All the $Q_{n-r}$ polynomials for corrupted parties $P_r$ are constructed from $d$ random values and the data $Q_{n-r}(\beta) = 0$. Thus, the $Q_{n-r}$ polynomials are distributed as random polynomials of degree $d$ that have $\beta$ as a root. We need to verify that the adversary has no way to distinguish between the $Q_{n-r}(\alpha_{r'}) + \Pi_{n-r}(\alpha_{r'})f(\alpha_r)$ and $Q_{n-r}(\alpha_{r'}) + \Pi_{n-r}(\alpha_{r'})f_{\mathcal{S}}(\alpha_r)$ when $P_r$ is honest and $P_{r'}$ corrupted. If there is a distinguisher between $f$ and $f_{\mathcal{S}}$, then there is a distinguisher between $f(\beta)$ and $f_{\mathcal{S}}(\beta)$, as it is the only information conveyed by the sharing. As such, it suffices to prove that there is information theoretic security on the value $f(\beta)$. Let us denote $h$ the number of honest parties in $\mathcal{P}_{LEAVE}$. We have the equation $k = (t_P - t_P^-) + h$. As the evaluations of $f^-$ are just linear combination of informations on $f$ we are going to focus on how the adversary might try to recover $f(\beta)$ from his knowledge of the polynomial $f$. Before the exchange, the adversary was missing $d+1-t_P$ values to interpolate $f(\beta)$. The exchange with honest leaving parties brings $ht_P^-$ new equations on the $d+1-t_P$ unknown values $f(\alpha_r)$ for honest parties $P_r$. These new equations introduces $h * t_P^-$ new unknown values $Q_{n-r}(\alpha_{r'})$ for honest leaving parties $P_r$ and $P_{r'} \in \mathcal{P}_P^-$. To that, we can add $h$ equations from the knowledge $Q_{n-r}(\beta) = 0$ for all leaving $P_r$. The difference between the number of unknown values in the system and the number of equations is $\Delta d + 1 - t_P + h * t_P^- - h * (t_P + 1) = d + 1 - t_P - h$. To provide information theoretic security on $f(\beta)$ we need to have $1 \leq d+1-t_P-h$. From the equation on $k, h, t_P$ and $t_P^-$, this bound can be rewritten $1 \leq (d-k)+1-t_P^-$. This is exactly the assumption we had on the value $t_P^-$. □

*DecreaseCorrupt.* The functionality $\mathcal{F}_{\texttt{DecreaseCorrupt}}$, takes the shares of the reduced initial set of parties $\{P_1, \ldots, P_{n-1}\}$, and distributes new shares to the reduced set of parties $\{P_1, \ldots, P_{n-1}\}$.

**Theorem 10.** *The protocol* `DecreaseCorrupt` *privately computes the functionality* $\mathcal{F}_{\texttt{DecreaseCorrupt}}$ *against an adversary that is bounded by* $t_P \leq n-3$, $t_P^- \leq n-4$.

*Proof.* In the decrease corrupt protocol, only one participant is leaving the group. This participant is assumed to be permanently corrupted and cannot interact with the other parties. We assume that $P_n$ is the leaving participant. The value $f(\alpha_n)$ is part of the adversary's view. Apart from that, the adversary knows the polynomials $f_r$ and the values $f_{r'}(\alpha_r)$ for all corrupted $P_r$ and honest $P_{r'}$. Then, the adversary sees $f(\alpha_r) + f_n(\alpha_r)$ for all the parties $P_r$.
To simulate that view, the simulator $\mathcal{S}_{\texttt{DecreaseCorrupt}}$ can use the inputs and outputs to compute the value $f(\alpha_n)$. Then, $\mathcal{S}_{\texttt{DecreaseCorrupt}}$ generates a polynomial $f_{\mathcal{S}}$ of degree $d$ with $f_{\mathcal{S}}(\alpha_r) = f(\alpha_r)$ for all corrupted $P_r$ including $P_n$. After that, the simulator can generate $n-1$ polynomials $f_u$ verifying $f_u(\alpha_n) = 0$ and compute the values $f_{\mathcal{S}}(\alpha_r) + \sum_{u=1}^{n-1} f_u(\alpha_r)$ for all $1 \leq r \leq n-1$.

The values for the $f_u$ polynomials are clearly generated genuinely. Just as we did for the `Decrease` protocol, we just have to prove that there is information theoretic security on the value $f(\beta)$. The adversary misses $d+1-t_P$ evaluation of $f$. The view of $f(\alpha_r) + f_n(\alpha_r)$ for the $n-1-t_P$ honest parties provides the adversary with $n-1-t$ new equations. From the construction, it is clear that we can assume that $f_n(\alpha_r)$ is an unknown random value for the adversary when $P_r$ is honest. Each of the $n-1-t_P$ evaluation of $f$ are hidden by the same amount of unknown random values. The additional property $f_n(\alpha_n) = 0$ does not profit the adversary. Indeed, since $P_n$ is corrupted, $n-1-t_P = n-t_P^+$. With $d = n-2$, the bound $t_P^- \leq n-4$ ensures information theoretic security on the secret. Hence, no information is learned by the adversary on the secret that has information theoretic security before the protocol, it remains this way. □

*Dynamic Batching.* We give below an informal result of security for the procedure outlined in Section 5.4 that we call `DynamicBatching`. Lemma 5 grasps the important elements behind the full security of the `Redistribute` protocol. The full proof of `Redistribute` is given below in Appendix D.2.

**Lemma 5.** *Let us denote* $\texttt{sign}(k)$ *the sign of* $k$, $T_s^i = \{(d+\texttt{sign}(k).i-\lfloor\sqrt{\ell}\rfloor, d-\texttt{sign}(k).i-\lfloor\sqrt{\ell}\rfloor)\}$ *and* $t_P^i = |\mathcal{P}_P \cap \{P_1, \ldots, P_{n+\texttt{sign}(k).i}\}|$ *(same for* $t_A^i$*) for* $0 \leq i \leq k-1$. *The* `DynamicBatching` *protocol is correct and keeps the secrets hidden when* $(t_P^0, t_A^0) \leq T_s^0$ *and* $(t_P^k, t_A^k) \leq T_s^k$.

*Proof.* The correctness follow directly from the correctness of the four protocols `Refresh`, `Increase`, `Decrease`, and `DecreaseCorrupt`.

We propose a proof for the secrecy in the case where $d = n-2$. The secrecy threshold for the `Dynamic` sub-protocol (no matter which one is used) is a lot higher than the threshold in Lemma 5, this guarantees that no information is learned on any of the secrets by the adversary during the $\ell$ executions of `Dynamic`. Then, the proof of `Recover` almost gives the result stated in Lemma 5. The only difference is that $P_n$ knows the $\ell$ values $g(\alpha_n, \beta_j)$ for all $j \in [\ell]$. In the case where $P_n$ is corrupted we can see that as $\ell$ new equations on the missing evaluations. Thus, the bound given by $\ell \leq \Delta$ with $\Delta$ the difference between the number of unknown values and the number of equations becomes

$$\ell \leq 2(d+k+1-t_P)(d+k+2-t_P) - \ell$$

Fortunately, with $t_P^k \leq d+k-\lfloor\sqrt{\ell}\rfloor$, the previous inequality is verified. We have information theoretic security on the $\ell$ secrets and this concludes the proof. □

### D.2 Proofs for the Dynamic PSS

The simulator used in the proof of Theorem 11 relies on another simulator $\mathcal{S}_{\texttt{Refresh}}$. This Simulator is the one described in Figure 10. It also relies on simulators $\mathcal{S}_{\texttt{Increase}}$,$\int_{\texttt{Decrease}}$ and $\mathcal{S}_{\texttt{DecreaseCorrupt}}$ that are extensions to the malicious setting fo the simulator presented in Appendix D.1 We adopt the additional following notation : for $k \in \mathbb{Z}$ and a multi-threshold $T_x^{(\omega)}$, $T_{x,k}^{(\omega)} = \{(y+k, z+k),\ (y,z) \in T_x^{(\omega)}\}$. In Fig. 11, the number of passive (resp. active) corruptions in a set $\mathcal{P}_x$ is denoted $t_P^x$ (resp. $t_A^x$).

**Figure 11**. Ideal Process for the `Redistibute` 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 $\mathcal{F}_{\texttt{Redistribute}}$ that will play the role of a trusted third party that will compute the redistribution of the shares. Upon reception of the input that is a sharing of a batch of secret $s_1, \ldots, s_\ell$ for a set of size $n^{(\omega)}$, the functionality will output a new sets of shares for the same batch of secrets to the new set of participants of size $n^{(\omega+1)}$.

*Parameters.* Multi-thresholds $T_c^{(\omega)}, T_s^{(\omega)}$.

*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 among the sets $\mathcal{P}^{(\omega)}$, $\mathcal{P}^{(\omega+1)}$ and subsets $\mathcal{P}_\cap$, $\mathcal{P}_L$, $\mathcal{P}_{L_C}$ and $\mathcal{P}_N$.
2. $\mathcal{Z}$ invokes the parties $P_r \in \mathcal{P}^{(\omega)}$ and their inputs $x_r$.
3. $\mathcal{S}$ receives the inputs for the corrupted parties in $\mathcal{P}^{(\omega)}$ and may change the input for the corrupted parties among this set.
INPUTS
4. Each party sends his input to the ideal functionality.
CORRUPTION
5. If $(t_P^{(\omega)}, t_A^{(\omega)}) \not\leq T_s^{(\omega)}$: $\mathcal{F}_{\texttt{Redistribute}}$ sends all inputs to $\mathcal{S}$.
COMPUTATION
6. $\mathcal{F}_{\texttt{Redistribute}}$ evaluates $g$ to obtain the values $s_1, \ldots, s_\ell$. And generate a new bivariate sharing $g'$ for $s_1, \ldots, s_\ell$ for the participants in $\mathcal{P}^{(\omega+1)}$.
OUTPUTS
7. $\mathcal{F}_{\texttt{Redistribute}}$ sends the shares $g'(\alpha_r, \alpha_{r'})$,for the corrupted parties $P_r \in \mathcal{P}^{(\omega)}$ to $\mathcal{S}$.

8. $\mathcal{S}$ sends either `abort` to $\mathcal{F}_{\texttt{Redistribute}}$ and $\mathcal{F}_{\texttt{Redistribute}}$ aborts and outputs $\perp$ or `continue` and the functionality proceeds to the next step.
9. If $(t_P^{(\omega+1)}, t_A^{(\omega+1)}) \not\leq T_s^{(\omega+1)}$ `or` $(t_P^{\cap}, t_A^{\cap}) \not\leq T_{s,-|\mathcal{P}_L \cup \mathcal{P}_{L_C}|}^{(\omega)}$: $\mathcal{F}_{\texttt{Redistribute}}$ sends $s_1, \ldots, s_\ell$ to $\mathcal{S}$.
10. If $(t_P^{(\omega)}, t_A^{(\omega)}) \not\leq T_c^{(\omega)}$ `or` $(t_P^{\cap}, t_A^{\cap}) \not\leq T_{c,-|\mathcal{P}_L \cup \mathcal{P}_{L_C}|}^{(\omega)}$ `or` $(t_P^{(\omega+1)}, t_A^{(\omega+1)}) \not\leq T_c^{(\omega+1)}$ : $\mathcal{S}$ sends new shares to $\mathcal{F}_{\texttt{Redistribute}}$ instead of the evaluations of $g'$. $\mathcal{F}_{\texttt{Redistribute}}$ replaces the output.
11. $\mathcal{F}_{\texttt{Redistribute}}$ sends their outputs to the honest parties.

*Outputs.* Each honest party outputs whatever they received from $F_{\texttt{Redistribute}}$. 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 $b_{\mathcal{Z}}$.

**Figure 12**. Simulator for the `Redistibute` protocol.

1. If $(t_P^{(\omega)}, t_A^{(\omega)}) \not\leq T_s^{(\omega)}$ $\mathcal{S}$ sets $g_{\mathcal{S}}$ as $g$ the bivariate sharing that is the input of the parties. Otherwise, $\mathcal{S}$ generates a random $g_{\mathcal{S}}$ verifying $g_{\mathcal{S}}(\alpha_r, y) = g(\alpha_r, y)$ for corrupted $P_r$ in $\mathcal{P}^{(\omega)}$.
2. $\mathcal{S}$ generates a new sets of commitments for the values of $g_{\mathcal{S}}$ and initializes the adversary with these new commitments.
3. $\mathcal{S}$ sets $f_j(\alpha_r) = g(\alpha_r, \beta_j)$ for all the honest parties $P_r$ in $\mathcal{P}^{(\omega)}$.
4. $\mathcal{S}$ simulates the $\ell$ executions of `DecreaseCorrupt` with $\mathcal{S}_{\texttt{DecreaseCorrupt}}$.
5. $\mathcal{S}$ simulates the $\ell$ executions of `Decrease` with $\mathcal{S}_{\texttt{Decrease}}$.
6. $\mathcal{S}$ simulates the $\ell$ executions of `Increase` with $\mathcal{S}_{\texttt{Increase}}$.
7. On behalf of honest participants among the $d^{(\omega+1)} + 1$ first participants of $\mathcal{P}^{(\omega+1)}$, $\mathcal{S}$ generates a random $g'(\alpha_r, .)$ verifying $g'(\alpha_r, \beta_j) = f_j^3(\alpha_r)$ for all $j \in [\ell]$.
8. $\mathcal{S}$ exchanges the commitments to $g'$ with $\mathcal{A}$. If one error is detected on the opening of the adversary's values or if $\mathcal{A}$ sends a complaining bit, $\mathcal{S}$ sends `abort` to the ideal functionality. Otherwise, proceeds to the next step.
9. $\mathcal{S}$ uses the simulator $\mathcal{S}_{\texttt{Refresh}}$ to simulate the successive executions of `Recover`.
10. $\mathcal{S}$ compute the evaluations of $g'$ for all the honest parties. If $(t_P^{(\omega)}, t_A^{(\omega)}) \not\leq T_c^{(\omega)}$ `or` $(t_P^{\cap}, t_A^{\cap}) \not\leq T_{c,-|\mathcal{P}_L \cup \mathcal{P}_{L_C}|}^{(\omega)}$ `or` $(t_P^{(\omega+1)}, t_A^{(\omega+1)}) \not\leq T_c^{(\omega+1)}$, $\mathcal{S}$ sends to $\mathcal{F}_{\texttt{Redistribute}}$ the new evaluations
11. The simulators outputs whatever $\mathcal{A}$ outputs.

**Theorem 11.** *The `Redistribute` protocol securely realizes*

$$\text{IDEAL}_{mixed}^{\mathcal{Z},\mathcal{S},\mathcal{F}_{\texttt{Redistribute}}}(z, T_c^{(\omega)}, T_s^{(\omega)})$$

*with multi-threshold* $T_c^{(\omega)} = \{(n^{(\omega)}, n^{(\omega)} - 1)\}$ *and* $T_s\{(n^{(\omega)} - 1 - \lfloor\sqrt{\ell}\rfloor, n^{(\omega)} - 1 - \lfloor\sqrt{\ell}\rfloor)\}$.

*Proof.* The Theorem follows from the various results on `Recover`, `Increase`, `Decrease`, and `DecreaseCorrupt`. Indeed, due to the simulators associated to these protocols, $\mathcal{S}$ is able to produce a view indistinguishable from the real execution. Furthermore, the input distribution is identical in any cases (due to the perfectly hiding property of the commitments). Every step not involved in a sub-protocol is done exactly as would the set of honest participant would. The homomorphic commitments allows to ensure correctness when the thresholds are respected. Lemma 5 justifies that the adversary cannot distinguish between the real and ideal executions. □

## E Related Work

*Proactive Secret Sharing (PSS).* There are several PSS schemes in the literature and they can serve as a building block for proactive MPC (PMPC) protocols. Most PSS schemes (see Table 3) are insecure when a majority of the parties are compromised, even if the compromise is only passive. Such schemes [31,27,37,38,34,2,3] typically store the secret as the free term of a polynomial of degree $d < \frac{n}{2}$, thus only an adversary compromising $d + 1$ parties can recover the secret (by interpolation). More Recently [17] developed the first PSS scheme secure against a dishonest majority, it is based on the standard secret sharing scheme from [28] which introduced the notion of gradual secret sharing. The main purpose of gradual secret sharing scheme is to ensure *fairness* (if the corrupted parties can deny the output of a protocol to the set of honest parties, then they cannot learn the secret) in a mixed adversary model that allows simultaneously $k$ active corruption and $n-k-1$ passive one with $k \in \{0, \ldots, \lceil\frac{n}{2}\rceil - 1\}$. This scheme is used in [17] to build a PSS scheme that is secure against a dishonest majority and also retains this fairness property. The [17] PSS scheme is secure against an adversary bounded by $n - 3$ passive only corruptions, $\frac{n}{2} - 1$ active only corruptions, and a mix of $k$ active and $\min(n - 3, n - k - 1)$ passive corruptions for $k \in \{0, \ldots, \lceil\frac{n}{2}\rceil - 1\}$.

*Dynamic groups* As we mentioned earlier, Dynamic Secret Sharing (DSS) handles the redistribution of the secrets when the number of parties is evolving (either increasing or decreasing). The authors in [3] extended the PSS introduced in [2] with ideas from [15,14,13] to produce a Dynamic PSS scheme for honest majorities; we are not aware of other DPSS schemes for dishonest majority.

| Protocol | Thresh. (Cor+Sec) Passive (Active) | Thresh.(Cor+Sec+Fair) Passive (Active) **Mixed** | Security | Network Type | Communication Complexity | Dynamic |
|---|---|---|---|---|---|---|
| [37] | $\frac{n}{2}$ $(\frac{n}{2})$ | | Computational | Synchronized | $exp(n)$ | ✓ |
| [38] | $\frac{n}{3}$ $(\frac{n}{3})$ | | Computational | Asynchronized | $exp(n)$ | ✓ |
| [9] | $\frac{n}{3}$ $(\frac{n}{3})$ | | Computational | Asynchronized | $O(n^4)$ | ✗ |
| [34] | $\frac{n}{3}$ $(\frac{n}{3})$ | | Computational | Asynchronized | $O(n^4)$ | ✓ |
| [27] | $\frac{n}{2}$ $(\frac{n}{2})$ | | Computational | Synchronized | $O(n^2)$ | ✗ |
| [2] | $\frac{n}{3} - \epsilon$ $(\frac{n}{3} - \epsilon)$ | | Perfect | Synchronized | $O(1)$ (Amortized) | ✗ |
| [2] | $\frac{n}{2} - \epsilon$ $(\frac{n}{2} - \epsilon)$ | | Statistical | Synchronized | $O(1)$ (Amortized) | ✗ |
| [3] | $\frac{n}{3} - \epsilon$ $(\frac{n}{3} - \epsilon)$ | | Perfect | Synchronized | $O(1)$ (Amortized) | ✓ |
| [3] | $\frac{n}{2} - \epsilon$ $(\frac{n}{2} - \epsilon)$ | | Statistical | Synchronized | $O(1)$ (Amortized) | ✓ |
| [17] | $n-3$ $(n-3)$ | $n-3$ $(\frac{n}{2}-1)$ $\boldsymbol{(n-k-1, k)}$ | Computational | Synchronized | $O(n^4)$ | ✗ |
| This work ($\ell = n-2$ secrets) | $n-1-\lfloor\sqrt{\ell}\rfloor$ $(n-1-\lfloor\sqrt{\ell}\rfloor)$ | $n-1-\lfloor\sqrt{\ell}\rfloor$ $(\frac{n-1-\lfloor\sqrt{\ell}\rfloor}{2}-1)$ $\boldsymbol{(n-k-\lfloor\sqrt{\ell}\rfloor, k)}$ | Computational | Synchronized | $O(n^2)$ (Amortized) | ✓ |
| This work (1 secret) | $n-3$ $(n-3)$ | $n-3$ $(\frac{n}{2}-1)$ $\boldsymbol{(n-k-1, k)}$ | Computational | Synchronized | $O(n^3)$ | ✓ |

**Table 3.** Comparison of existing PSS schemes. Cor=Correctness, Sec=Secrecy, Fair=Fairness.