

Honest Majority MPC Achieving GOD with $O(|C|)$ Communication From Pairwise Random OLE Correlations

Zonghang Du¹, Yifan Song², and Xiaxi Ye³

¹ Tsinghua University

duzh23@mails.tsinghua.edu.cn

² Tsinghua University and Shanghai Qi Zhi Institute

yfsong@mail.tsinghua.edu.cn

³ Tsinghua University

yexx23@mails.tsinghua.edu.cn

Abstract. In this work, we study the communication complexity of information-theoretic MPC with guaranteed output delivery (GOD) in the honest majority setting ($t < n/2$).

The recent work by Song and Ye (EUROCRYPT 2025) gives an honest majority MPC in the random oblivious linear evaluation (OLE) preprocessing model that computes an arithmetic circuit of size $|C|$ with malicious security with abort at the cost of $O(|C|)$ of both communication in field elements and total number of random OLE correlations. We explore the possibility of deriving a similar result for the stronger security notion of GOD. Specifically, we ask the question: “Is it possible to construct an information-theoretic honest majority MPC in the random OLE preprocessing model that computes an arithmetic circuit of size $|C|$ and achieves GOD with $O(|C|)$ field elements of communication and $O(|C|)$ amount of random OLE correlations in total?”

We resolve the above question in the affirmative by providing a concrete construction. To achieve our result, we mainly rely on two techniques. First, we propose a new secret sharing scheme which we refer to as *detectable secret sharing*. It is a packed secret sharing scheme with an authentication mechanism. Second, to efficiently verify the computation and locate an error when faults occur, we extensively make use of the virtual transcript technique introduced in the work by Goyal, Song, and Zhu (CRYPTO 2020).

Our construction is in the random OLE preprocessing model where we assume random OLE correlations between each pair of parties. By instantiating the pairwise random OLE correlations from various assumptions, we obtain three honest majority MPC protocols with GOD with the following security and communication: (1) the first one achieves information-theoretic security with offline communication $O(|C|n)$ elements and online communication $O(|C|)$ elements, (2) the second one achieves computational security with $\tilde{O}(|C|)$ elements of communication only from a random oracle, and (3) the third one achieves computational security with $O(|C|)$ elements of communication from pseudorandom correlation generators (PCG).

Table of Contents

Honest Majority MPC Achieving GOD with $O(C)$ Communication From Pairwise Random OLE Correlations	1
<i>Zonghang Du, Yifan Song, and Xiari Ye</i>	
1 Introduction	4
1.1 Our Contribution	4
2 Technical Overview	5
2.1 Call for Verifiability of Packed Sharings - Detectable Secret Sharing Scheme	6
2.2 Achieving GOD from Detectable Secret Sharings	7
2.3 Defining Detectable Secret Sharings	9
2.4 Computing Authentication Tags Assuming OLE Correlations	11
2.5 Checking Authentication Tags with Weak Homomorphism	12
2.6 Conclusion and Organization	13
3 Preliminary	13
3.1 Security Definition	13
3.2 Dispute Control Framework	14
4 Detectable Secret Sharings	14
4.1 Distributing Detectable Secret Sharings	15
4.2 Verifying Detectable Secret Sharings	18
4.3 Overview of Extended Detectable Secret Sharings	19
5 Online Computation	19
5.1 Recursive Verification	20
5.2 Segment Computation for SIMD Circuits	24
Bibliography	26
A Additional Preliminary	27
A.1 UC Security	27
A.2 Dispute Control Framework	27
A.3 Verifiable Secret Sharing Scheme	28
A.4 Packed Triple Generation from Pairwise OLE Correlations	29
B Additional Instantiation of Base DSS	31
B.1 Authenticating Degree- $(k - 1)$ Packed Shamir Sharings	31
B.2 Summary of the Base DSS	33
C Detailed Instantiation of Extended DSS	33
C.1 Functionality for Extended Detectable Secret Sharings	34
C.2 Authenticated Packed Beaver Triples	35
C.3 Authenticated Linear Masks for Transpose	37
C.4 Reconstructing Detectable Secret Sharings	38
C.5 Summary of the Extended DSS	40
D The Protocol	41
D.1 Towards General Circuits	41
D.2 Summary of the Computation Segment	43
D.3 Input Layer	45
D.4 Main Protocol	45
E Instantiating OLE Functionality	48
E.1 Information-theoretic Instantiation	48
E.2 Instantiation in Minicrypt	48
E.3 Instantiation from PCG	49
F Useful Building Blocks	50

F.1	The Protocols	50
F.2	The Functionalities	52
G	Figures	53
H	Security Proofs	53
H.1	Proof of Lemma 1	53
H.2	Proof of Lemma 2	64
H.3	Proof for Lemma 3	68
H.4	Proof for Lemma 4	70
H.5	Proof for Lemma 6	72

1 Introduction

In this work, we focus on the communication complexity of information-theoretic MPC protocols with guaranteed output delivery (GOD) in the honest majority setting ($t < n/2$). The first feasibility results are due to [RB89, Bea89]. After that, a sequence of works [CDD⁺99, BH06] aim at improving the communication complexity. Among them, the works [BFO12, GSZ20] achieve a linear communication of $O(|C| \cdot n)$ elements. (Here we omit the factors that are independent of the circuit size.) However, all known results in this setting require a communication of at least $\Omega(|C| \cdot n)$ elements, which leads us to ask whether we can derive an information-theoretic honest majority MPC achieving GOD with $o(|C| \cdot n)$ elements.

This question has been previously considered for a weaker security notion of malicious security with abort. In this setting, the recent work [SY25a] gives a construction in the random *oblivious linear evaluation* (OLE) preprocessing model with $O(|C|)$ elements of communication and $O(|C|)$ amount of pairwise random OLE correlations in total across all pairs of parties, where a pairwise random OLE correlation is a two-party correlation in the form of $(a, x), (b, y)$ such that $a \cdot b = x + y$, allowing two parties to additively share the product of two random secrets held by each party respectively. However, by extending the lower bound proved in [DLN19], [SY25a] shows a negative result that any information-theoretic MPC protocols that prepare $\Theta(|C|)$ random OLE correlations require communicating at least $\Theta(|C| \cdot n)$ elements. As an interpretation of their work, generating OLE correlations can be regarded as the bottleneck computation task in the honest majority setting which incurs a provably linear amount of communication in the number of parties.

Nonetheless, one may relax to computational security and leverage cryptographic primitives (such as oblivious transfer (OT) extension as proposed in [IKNP03] or pseudorandom correlation generators (PCGs)) to generate $O(|C|)$ pairwise random OLE correlations at a communication cost of $\tilde{O}(|C|)^4$ or even $o(|C|)$ field elements. In particular, [SY25a] presents the first MPC protocol in the honest majority setting achieving overall $\tilde{O}(|C|)$ communication in Minicrypt from a random oracle. Such scalable MPC protocols, in which the total communication does not grow with the number of parties, have numerous potential real-world applications when a large number of parties wish to compute a common task.

Motivated from both theoretical and practical factors, we wonder whether the same asymptotic cost can be achieved for the stronger security notion of GOD. Specifically, we ask the following question:

“Is it possible to construct an information-theoretic honest majority MPC protocol in the random OLE preprocessing model that achieves guaranteed output delivery with $O(|C|)$ elements of communication and $O(|C|)$ pairwise random OLE correlations in total across all pairs of parties?”

1.1 Our Contribution

In this work, we answer the above question affirmatively and show the following theorem, where we omit the circuit-independent cost for simplicity.

Theorem 1 (informal). *Let n be the number of parties and $t = (n - 1)/2$. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2^\kappa$, where κ is the security parameter. For an arithmetic circuit C over $\mathbb{F}$, there exists an information-theoretic MPC protocol that securely computes C with GOD in the random OLE preprocessing model against a fully malicious adversary controlling at most t parties. The cost is $O(|C| + D \cdot n + C_O \cdot n)$ elements of communication, along with $O(|C| + D \cdot n)$ amount of pairwise random OLE correlations in total across all pairs of parties, where $|C|$, D , and C_O denote the circuit size, the circuit depth, and the number of output gates, respectively.*

The formal theorem is stated as Theorem 2, which deferred to Section D. To achieve this result, we mainly rely on two techniques. First, we propose a packed secret sharing scheme, termed *detectable secret sharing* scheme, which provides a verifiable property that enables all parties to detect an incorrect share

⁴ We omit the multiplicative factor depending on the security parameter in the $\tilde{O}$ notation.

even without help from the dealer. Second, to verify the computation and locate an error when a fault occurs without compromising efficiency and privacy, we extensively make use of the virtual transcript technique introduced in [GSZ20] to various protocols relying on different verifiability of input sharings.

By instantiating the OLE correlations from various assumptions, we obtain the following three corollaries.

Corollary 1 (informal). *Let n be the number of parties and $t = (n - 1)/2$. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2^\kappa$, where κ is the security parameter. For an arithmetic circuit C over $\mathbb{F}$, there exists an information-theoretic MPC protocol that securely computes C with GOD against a fully malicious adversary controlling at most t parties. The communication cost of the protocol is $O(|C| + D \cdot n + C_O \cdot n)$ elements for the online phase and $O(|C| \cdot n + D \cdot n^2)$ for the preprocessing phase.*

Corollary 2 (informal). *Let n be the number of parties and $t = (n - 1)/2$. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2^\kappa$, where κ is the security parameter. For an arithmetic circuit C over $\mathbb{F}$, there exists an MPC protocol in the random oracle model that securely computes C with GOD against a fully malicious adversary controlling at most t parties. The overall communication cost of the protocol is $O((|C| + D \cdot n) \cdot (\kappa + \ell) + C_O \cdot n)$ elements, where $\ell = \log |\mathbb{F}|$.*

Corollary 3 (informal). *Let n be the number of parties and $t = (n - 1)/2$. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2^\kappa$, where κ is the security parameter. Assuming the hardness of quasi-abelian syndrome decoding (QA-SD) and a length-doubling PRG, for an arithmetic circuit C over $\mathbb{F}$, there exists an MPC protocol that securely computes C with GOD against a fully malicious adversary controlling at most t parties. The overall communication cost of the protocol is $O(|C| + D \cdot n + C_O \cdot n + \text{poly}(n, \kappa))$ elements.*

The formal corollaries are stated as Corollary 4, 5, and 6, respectively.

2 Technical Overview

The recent work [GPS22] demonstrates that, assuming preprocessing data of size $O(|C|)$ that primarily consists of packed Beaver triples, parties can compute a circuit of size $|C|$ with communication of $O(|C|)$ elements. In the online protocol, each group of $k = \Theta(n)$ multiplication gates are evaluated together using the packed Beaver triple technique [GPS22] and parties compute a packed Shamir sharing for each group of k wire values. We refer to k as the packing parameter. Then, [SY25a] presents a protocol to generate packed Beaver triples of size $O(|C|)$ at the cost of $O(|C|)$ for both communication in field elements and amount of pairwise random OLE correlations. When combining it with the online protocol in [GPS22], [SY25a] obtains an honest majority MPC that achieves malicious security with abort with communication of $O(|C|)$ elements and $O(|C|)$ amount of pairwise random OLE correlations. Our goal is to achieve the stronger security of GOD while maintaining the same asymptotic cost.

Why Previous Techniques Fail to Achieve Our Result. At a first glance, one may attempt to follow the generic approach that converts a secure-with-abort protocol to GOD via dispute control [BH06]: The circuit computation is divided into several small segments, which are evaluated by all parties sequentially. For each segment, all parties run a secure-with-abort MPC protocol.

- If the computation succeeds, all parties move to the next segment.
- Otherwise, all parties run a fault localization protocol that identifies a party who deviates from the protocol description or a pair of parties that do not agree on the message sent from one party to the other. We refer the latter as a dispute pair. Then the identified corrupted party will be eliminated and the identified dispute pair will never talk to each other. In this way, the evaluation will not fail again due to the same corrupted party or dispute pair.

Note that the fault localization can be achieved if all parties open their views in the current segment. To protect the secrecy of honest parties' views, all parties may run a generic MPC protocol with GOD to identify a corrupted party or a dispute pair. Since the number of dispute pairs is bounded by $O(n^2)$, the

number of failures is also bounded by $O(n^2)$. By setting the segment size to be small enough, the cost of the fault localization can be sublinear in the circuit size, and the overall communication is dominated by the secure-with-abort MPC protocol for each segment. When instantiating the above template with the secure-with-abort MPC protocol in [SY25a], it seems like we have already obtained an MPC protocol that achieves GOD with $O(|C|)$ elements of communication and $O(|C|)$ amount of pairwise random OLE correlations.

Unfortunately, this is not the case. Recall that the construction in [SY25a] is in the random OLE preprocessing model. Directly using the above template would require pairwise random OLE correlations to instantiate the construction in [SY25a] for each segment. As a result, in the final protocol, parties would require pairwise random OLE correlations even in the online phase. While one may try to prepare a sufficient number of pairwise random OLE correlations in the preprocessing phase, this does not work either. This is because in [SY25a], each pair of parties require the same amount (i.e. $O(|C|/n^2)$) of random OLE correlations. However, when applying the dispute control framework, after all parties identify a new dispute pair, these two parties should be removed⁵ and we need to prepare random OLE correlations for every two of the remaining $n - 2$ parties. In the worst case, if only a constant number of parties remained, each pair of parties would require $O(|C|)$ random OLE correlations. As a result, the number of random OLE correlations required by each pair of parties can be as large as $O(|C|)$ and we cannot afford the cost of preparing this amount of random OLE correlations for every pair of parties, especially when aiming for information-theoretic security. Moreover, the parties would need to store preprocessing data of size $O(|C| \cdot n^2)$, which fails to achieve our scalability goal.

Besides, the above template requires all parties to hold a *verifiable secret sharing* for each input wire of all segments [IKP⁺16]. Otherwise, if the input secret sharings of some segment are incorrect, even if all parties follow the protocol, the computation of the current segment would still fail and parties cannot locate a new corrupted party or dispute pair. This incurs a large communication overhead depending on the width of the circuit as noted in [GSZ20].

2.1 Call for Verifiability of Packed Sharings - Detectable Secret Sharing Scheme

Our construction follows [GPS22, SY25a] and uses packed Shamir sharings to achieve $O(|C|)$ communication. In particular, we use degree- $(t' + k - 1)$ packed Shamir sharings to store k secrets, where $t' = t + 1$. In addition, we follow the dispute control framework [BH06] to achieve GOD.

To avoid the need of random OLE correlations during the online phase, as discussed in the above template, we introduce a new secret sharing scheme which we call the *detectable secret sharing* scheme (DSS). It provides the verifiable property on top of packed Shamir secret sharings such that after a packed Shamir secret sharing has been authenticated as a detectable secret sharing, any incorrect share can be detected without the help from the dealer. In addition, the detectable secret sharing scheme maintains the linear homomorphism of the packed Shamir secret sharing scheme. We will use the notation $[\mathbf{x}]_{t'+k-1}$ to denote the standard packed Shamir sharing of $\mathbf{x} \in \mathbb{F}^k$ and use $[[\mathbf{x}]]_{t'+k-1}$ to denote the packed Shamir sharing authenticated as a detectable secret sharing. We abstract the (simplified) detectable secret sharing functionality as $\mathcal{F}_{\text{DSS}}$ below.

Comparing our detectable secret sharing scheme with a verifiable secret sharing scheme, we emphasize that the DSS scheme is a weaker primitive in the sense that the DSS functionality only provides detectability of the shares and cannot ensure success of the reconstruction if too many corrupted parties refuse to provide correct shares. This is unavoidable due to the large degree of the underlying packed Shamir sharings. Consequently, DSS cannot serve as a replacement for VSS in applications that require guaranteed reconstruction. In the following, we first show that based on $\mathcal{F}_{\text{DSS}}$, we can construct an information-theoretic MPC protocol that achieves GOD with $O(|C|)$ communication in Section 2.2. Then we demonstrate how to instantiate the detectable secret sharings in Sections 2.3, 2.4 and 2.5.

⁵ The construction in [SY25a] relies on the ratio between the number of pairs of corrupted parties and the total number of pairs of parties. If we only remove the identified dispute pair, this ratio may increase. To maintain the ratio, we have to remove all pairs that include at least one party in the identified dispute pair.

Functionality 1: $\mathcal{F}_{\text{DSS}}$

1. On receiving $[\mathbf{x}]_{t'+k-1}$ from P_d , $\mathcal{F}_{\text{DSS}}$ stores $[\mathbf{x}]_{t'+k-1}$ as $[\![\mathbf{x}]\!]_{t'+k-1}$ and distributes the shares to all parties.
2. On receiving a request for a random Beaver triple, $\mathcal{F}_{\text{DSS}}$ randomly generates $([\![\mathbf{a}]\!]_{t'+k-1}, [\![\mathbf{b}]\!]_{t'+k-1}, [\![\mathbf{c}]\!]_{t'+k-1})$ such that $\mathbf{a} * \mathbf{b} = \mathbf{c}$ and stores them. Then $\mathcal{F}_{\text{DSS}}$ distributes the shares to all parties.
3. On receiving a request of checking $[\mathbf{w}]_{t'+k-1}$ which can be expressed by a linear combination of degree- $(t' + k - 1)$ packed Shamir sharings stored by $\mathcal{F}_{\text{DSS}}$,
 - $\mathcal{F}_{\text{DSS}}$ receives the sharing $[\mathbf{w}]_{t'+k-1}$ from all parties.
 - $\mathcal{F}_{\text{DSS}}$ computes $[\![\mathbf{w}]\!]_{t'+k-1}$ following the same linear combination and using its stored degree- $(t' + k - 1)$ packed Shamir sharings.
 - For all i , if the i -th share is incorrect, output $(P_i, \text{corrupt})$ to all parties. If all shares are correct, output **accept**.

2.2 Achieving GOD from Detectable Secret Sharings

Following [GPS22], in the online phase, the goal is to compute a detectable secret sharing for each group of k wires. For simplicity, we start with a SIMD circuit (or k copies of the same circuit) where we naturally group the same wire in different copies together. In this case, we only need to focus on the computation and verification of multiplication gates as the secrets of input packed secret sharings are naturally aligned and addition gates can be locally computed.

Following the dispute control framework, parties will together maintain the set of all identified dispute pairs. Note that if a party has dispute with $t + 1$ parties, this party must be corrupted. Such a party will be eliminated. We will use n' to denote the number of remaining parties.

Computation and Verification of Multiplication Gates. We first review the multiplication protocol based on packed Beaver triples from [GPS22]. Suppose all parties hold degree- $(t' + k - 1)$ packed Shamir sharings $[\![\mathbf{x}]\!]_{t'+k-1}, [\![\mathbf{y}]\!]_{t'+k-1}$ authenticated by $\mathcal{F}_{\text{DSS}}$, and wish to compute a degree- $(t' + k - 1)$ packed Shamir sharing of $\mathbf{z} = \mathbf{x} * \mathbf{y}$, where $*$ denotes element-wise multiplication. In the multiplication protocol, all parties consume a fresh Beaver triple $([\![\mathbf{a}]\!]_{t'+k-1}, [\![\mathbf{b}]\!]_{t'+k-1}, [\![\mathbf{c}]\!]_{t'+k-1})$ authenticated by $\mathcal{F}_{\text{DSS}}$, together with a fresh random degree- $(t' + k - 1)$ packed Shamir sharing $[\![\mathbf{r}]\!]_{t'+k-1}$ authenticated by $\mathcal{F}_{\text{DSS}}$. In addition, during the online phase, all parties prepare and consume a random degree- $(n' - 1)$ packed Shamir sharing of the zero vector, denoted by $[\![\mathbf{o}]\!]_{n'-1}$. The protocol appears in [MULT](#).

Protocol 1: MULT

Input: $[\![\mathbf{x}]\!]_{t'+k-1}, [\![\mathbf{y}]\!]_{t'+k-1}$ ^a.

Output: $[\![\mathbf{z}]\!]_{t'+k-1}$, where $\mathbf{z} = \mathbf{x} * \mathbf{y}$ and $*$ denotes element-wise multiplication.

Ingredients: $([\![\mathbf{a}]\!]_{t'+k-1}, [\![\mathbf{b}]\!]_{t'+k-1}, [\![\mathbf{c}]\!]_{t'+k-1}), [\![\mathbf{r}]\!]_{t'+k-1}$, and $[\![\mathbf{o}]\!]_{n'-1}$.

All parties have agreed on a special party P_{king} .

1. All parties locally compute $[\![\mathbf{v}_1]\!]_{t'+k-1} \leftarrow [\![\mathbf{x}]\!]_{t'+k-1} + [\![\mathbf{a}]\!]_{t'+k-1}$, $[\![\mathbf{v}_2]\!]_{t'+k-1} \leftarrow [\![\mathbf{y}]\!]_{t'+k-1} + [\![\mathbf{b}]\!]_{t'+k-1}$ and send their shares to P_{king} .
2. P_{king} reconstructs the secrets $\mathbf{v}_1, \mathbf{v}_2$ using the first $t' + k$ shares and distributes $[\mathbf{v}_1]_{k-1}, [\mathbf{v}_2]_{k-1}$ to all parties.
3. All parties locally compute

$$[\mathbf{e}]_{n'-1} \leftarrow [\mathbf{v}_1]_{k-1} \cdot [\mathbf{v}_2]_{k-1} - [\![\mathbf{a}]\!]_{t'+k-1} \cdot [\mathbf{v}_2]_{k-1} - [\![\mathbf{b}]\!]_{t'+k-1} \cdot [\mathbf{v}_1]_{k-1} + [\![\mathbf{c}]\!]_{t'+k-1} + [\![\mathbf{r}]\!]_{t'+k-1} + [\![\mathbf{o}]\!]_{n'-1}$$

and send their shares to P_{king} .

4. P_{king} reconstructs the secret $\mathbf{e}$ and distributes $[\mathbf{e}]_{k-1}$ to all parties.
5. All parties locally compute $[\mathbf{z}]_{t'+k-1} \leftarrow [\mathbf{e}]_{k-1} - [\mathbf{r}]_{t'+k-1}$.
6. The interaction in Steps 1 and 2 gives rise to two degree-reduction transcripts, while the interaction in Steps 3 and 4 gives rise a single transcript.

^a For a group of multiplication gates not in the first layer of the segment, the input sharings $[\mathbf{x}]_{t'+k-1}, [\mathbf{y}]_{t'+k-1}$ may not be authenticated as detectable secret sharings yet.

We now focus on verifying the multiplication process. Suppose all parties have computed many groups of multiplication gates. In the first interaction round, all parties verify that the sharings $[\mathbf{x} + \mathbf{a}]_{k-1}, [\mathbf{y} + \mathbf{b}]_{k-1}$ distributed by P_{king} are consistent and have the correct secrets. Observe that the sharings $[\mathbf{x} + \mathbf{a}]_{t'+k-1}, [\mathbf{y} + \mathbf{b}]_{t'+k-1}$ sent to P_{king} are authenticated by $\mathcal{F}_{\text{DSS}}$. Consequently, the secrets are authenticated as well. The verification is straightforward. The parties first compute random linear combinations of the sharings and broadcast the resulting sharings, denoted by $([\mathbf{x}_0]_{t'+k-1}, [\mathbf{x}_0]_{k-1})$. They then invoke $\mathcal{F}_{\text{DSS}}$ to verify the correctness of $[\mathbf{x}_0]_{t'+k-1}$ and check that $[\mathbf{x}_0]_{k-1}$ is consistent and has the same secrets as $[\mathbf{x}_0]_{t'+k-1}$. If the sharing $[\mathbf{x}_0]_{k-1}$ is inconsistent or has incorrect secrets, all parties let P_{king} identify a corrupted party P_i and take (P_{king}, P_i) as a new dispute pair.

As for the second interaction round, all parties verify that $[\mathbf{e}]_{k-1}$ distributed by P_{king} is consistent and has the correct secrets. To this end, we consider the transcript

$$([\mathbf{x} + \mathbf{a}]_{k-1}, [\mathbf{y} + \mathbf{b}]_{k-1}, [\mathbf{a}]_{t'+k-1}, [\mathbf{b}]_{t'+k-1}, [\mathbf{c}]_{t'+k-1}, \\ [\mathbf{r}]_{t'+k-1}, [\mathbf{o}]_{n'-1}, [\mathbf{e}]_{n'-1}, [\mathbf{e}]_{k-1}).$$

Every degree- $(k-1)$ packed Shamir sharing in the transcript is generated and distributed by P_{king} , and the whole sharings are determined by the honest parties' shares. Moreover, every degree- $(t'+k-1)$ packed Shamir sharing is authenticated via $\mathcal{F}_{\text{DSS}}$, allowing each share to be verified. Thus, if all parties open the entire transcript together with the generation process of $[\mathbf{o}]_{n'-1}$, they can verify the correctness of $[\mathbf{e}]_{k-1}$ and identify a new dispute pair if the verification fails. To preserve communication efficiency and privacy, we follow the virtual transcript technique in [GSZ20].

At a high level, the parties start with transcripts corresponding to a batch of m groups of multiplication gates. The idea in [GSZ20] is to compress the verification of these m groups of multiplication gates into the verification of a single group. Moreover, all parties can generate a virtual transcript for this group as if they actually computed this group of gates using the multiplication protocol described above. In this way, all parties may open the virtual transcript, verify the correctness of the computation, and identify a new dispute pair if the verification fails, as described above. In [GSZ20], the compression requires only a sublinear amount of communication in the number of groups, and the virtual transcript can be made public without compromising security (essentially, the virtual transcript corresponds to an execution of the protocol on random inputs). We extend this technique, originally used to handle the simpler case of the DN multiplication protocol [DN07] over degree- t (non-packed) Shamir sharings, to our setting. We refer the reader to Section 5.1 for more details.

Verifiability of the Shares. We now briefly discuss the source of verifiability for the packed Shamir sharings appearing in the protocol. In the discussion of multiplication above, we assumed that all degree- $(t'+k-1)$ sharings are authenticated by $\mathcal{F}_{\text{DSS}}$. However, this is not always true. For example, in the final step of the multiplication protocol, all parties compute $[\mathbf{z}]_{t'+k-1} \leftarrow [\mathbf{e}]_{k-1} - [\mathbf{r}]_{t'+k-1}$. Since $[\mathbf{e}]_{k-1}$ is distributed by P_{king} , it is not authenticated by $\mathcal{F}_{\text{DSS}}$, and therefore neither is $[\mathbf{z}]_{t'+k-1}$. Consequently, if all parties later use shares of $[\mathbf{z}]_{t'+k-1}$ for further multiplication gates, in the verification phase, they cannot directly invoke $\mathcal{F}_{\text{DSS}}$ to identify a new corrupted party or a dispute pair.

The key observation underlying our solution is that, every degree- $(t'+k-1)$ packed Shamir sharing in the current segment can be decomposed into the sum of a degree- $(k-1)$ packed Shamir sharing distributed by P_{king} and a degree- $(t'+k-1)$ packed Shamir sharing authenticated by $\mathcal{F}_{\text{DSS}}$: This is true for the output packed Shamir sharing in **MULT** and such a decomposition property preserves under linear combinations.

Hence, to verify the correctness of a degree- $(t' + k - 1)$ packed Shamir sharing $[z]_{t'+k-1}$, the parties first decompose it as

$$[z]_{t'+k-1} = [z_0]_{k-1} + \llbracket z_1 \rrbracket_{t'+k-1},$$

and then verify the two components separately. Specifically, they invoke $\mathcal{F}_{\text{DSS}}$ to verify the degree- $(t' + k - 1)$ component $\llbracket z_1 \rrbracket_{t'+k-1}$. For the low-degree part $[z_0]_{k-1}$, the parties consult P_{king} , who originally distributed the sharing and therefore possesses complete information about it. Note that a degree- $(k - 1)$ packed Shamir sharings is fully determined by the shares of honest parties, and hence verifying this component does not require $\mathcal{F}_{\text{DSS}}$.

Note that the above verification for degree- $(k - 1)$ packed Shamir sharings distributed by P_{king} only works in the current segment since P_{king} remains active. If the current segment succeeds, all parties need to authenticate the degree- $(k - 1)$ packed Shamir sharings distributed by P_{king} in this segment. This is because without P_{king} , parties can only detect inconsistency in such a sharing, but cannot identify the corrupt shares. Consequently, if a malicious P_{king} is eliminated later, reconstructing these degree- $(k - 1)$ packed Shamir sharings may fail when they are needed. To address this issue, our $\mathcal{F}_{\text{DSS}}$ additionally supports authentication of degree- $(k - 1)$ packed Shamir sharings. We refer the reader to Section B.1 for further details.

Fallback Procedure. We note that our protocol requires at least $n' \geq t' + 2k - 1$ active parties; otherwise, the parties will not have sufficiently many shares to reconstruct degree- $(t' + 2k - 2)$ packed Shamir sharings. As a result, once $n - (t' + 2k - 2)$ corrupted parties have been eliminated, the computation can no longer proceed. On the other hand, since all eliminated parties are corrupted, the corruption threshold among the remaining active parties decreases. In this case, the parties restart the whole computation with a smaller corruption threshold. In particular, by setting $k = n/12$, the corruption threshold decreases to $1/4$ after removing $n - (t' + 2k - 2)$ corrupted parties. This enables the parties to switch to a perfectly secure MPC protocol with overall $O(|C|)$ communication in the plain model, as constructed in [SY25b]. Moreover, in our construction, the reconstruction of the final output ensures that either all parties obtain the output or none do. This is achieved by computing a verifiable secret sharing of each output value before reconstruction. Thus, the fallback protocol does not require enforcing consistent inputs from corrupted parties. We refer the reader to Sections C.4 and D.4 for more details.

To compute general circuits, we follow [GPS21, GPS22] to additionally handle network routing, including performing sharing transformations and collecting secrets from different packed Shamir sharings. We refer the reader to Section D.1 for further details.

2.3 Defining Detectable Secret Sharings

To realize $\mathcal{F}_{\text{DSS}}$, our idea is to add verifiability on top of packed Shamir sharings by computing pairwise authentication tags following [BFO12, GSZ20].

Concretely, each party P_v maintains its own long-term secret keys $\{\gamma_{v \rightarrow i}\}_{i=1}^n$, where each $\gamma_{v \rightarrow i} \in \mathbb{F}^k$ is used to authenticate the shares held by P_i and is reused across all batches of sharings. Let $[\mathbf{X}]_{t'+k-1} = ([\mathbf{x}_1]_{t'+k-1}, \dots, [\mathbf{x}_k]_{t'+k-1})$ denote a collection of k degree- $(t' + k - 1)$ packed Shamir sharings, and let $\sigma_i \in \mathbb{F}^k$ denote the share vector held by P_i . Then for every pair of parties (P_i, P_v) , an authentication tag of P_i 's share vector under P_v 's key is defined as

$$\eta_{v \rightarrow i} \leftarrow \sigma_i \odot \gamma_{v \rightarrow i} + \delta_{v \rightarrow i}, \quad (1)$$

where $\odot$ denotes the inner product, and $\delta_{v \rightarrow i}$ is a fresh short-term key held by P_v for the corresponding share vector. The purpose of $\delta_{v \rightarrow i}$ is to preserve the secrecy of the long-term key $\gamma_{v \rightarrow i}$.

In the verification phase, after publishing the share vector σ_i , party P_i provides the authentication tag $\eta_{v \rightarrow i}$ to each party P_v , who then acts as a verifier to check whether σ_i and the received tag $\eta_{v \rightarrow i}$ satisfy Equality 1. A majority vote is then used to determine the correctness of σ_i .

Difficulties of Applying [BFO12,GSZ20] in Our Setting. Consider the cost of computing such pairwise authentication tags. For k packed Shamir sharings, a tag is computed for every pair of parties, and each tag requires k multiplications, resulting in $O(n^2)$ multiplications per packed sharing. Consequently, since each packed Shamir sharing stores $k = \Omega(n)$ secrets, computing all authentication tags corresponds to a circuit of size $O(|C| \cdot n)$. If these authentication tags were computed using a generic MPC protocol, known solutions would require at least $O(|C| \cdot n)$ communication, even assuming access to pairwise random OLE correlations such as the protocol in [SY25a].

We note that in [BFO12,GSZ20], they add verifiability on top of standard degree- t Shamir sharings. As a result, the number of multiplication operations involved is even larger, i.e., $O(|C| \cdot n^2)$. However, this large amount of multiplication operations do not translate to the same communication overhead in their works due to a clever use of the DN multiplication protocol [DN07]. Concretely, the DN multiplication protocol allows all parties to compute an inner-product of two vectors, both shared via degree- t Shamir sharings, at the cost of a single multiplication. This is mainly because after multiplying two degree- t Shamir sharings directly, the resulting sharing has degree- $2t$, which can still be reconstructed with honest majority. In [BFO12,GSZ20], σ_i is from a vector of degree- t Shamir sharings, which is naturally shared via degree- t Shamir sharings. To compute the authentication tag, P_v will share $\mu_{v \rightarrow i}$ via degree- t Shamir sharings as well. In this way, parties only need to pay $O(n)$ communication for each tag. Since $\mu_{v \rightarrow i}$ can be reused, by setting the key size to be sufficiently long (which is $O(n^2)$ rather than $O(n)$ in their constructions), they can maintain an overall linear communication.

However, the same trick does not apply in our case since σ_i is from a vector of degree- $(t' + k - 1)$ packed Shamir sharings. In particular, even with $\gamma_{v \rightarrow i}$ shared using degree- t polynomials, the multiplication result of a degree- $(t' + k - 1)$ packed Shamir sharing with a degree- t Shamir sharing would have degree $n + k - 1$ but we do not have sufficient number of shares to reconstruct such a large degree polynomial.

Our solution. To resolve this issue, our idea is to let a trusted party maintain a *single* long-term authentication key $\gamma = (\gamma_1, \dots, \gamma_k) \in \mathbb{F}^k$. Each party's shares are then authenticated under γ . This reduces the number of multiplications by a factor of n , which implies we can potentially achieve $O(|C|)$ communication relying on packed Shamir sharings. In our construction, γ is shared among all parties. To ensure that γ can always be reconstructed and thus the correctness of each share can be verified, we store γ using a verifiable secret sharing scheme.

Formally, our detectable secret sharing scheme for a batch of k^2 secrets, represented by a $k \times k$ matrix $\mathbf{X} = (x_{i,j})_{i,j=1}^k$, is defined as follows.

- Plain packed secret sharings: For $\ell \in [k]$, let $\mathbf{x}_\ell := (x_{\ell,1}, \dots, x_{\ell,k})$ be the ℓ -th row of the secret matrix $\mathbf{X}$. Each row vector $\mathbf{x}_\ell$ is shared via a packed Shamir sharing $[\mathbf{x}_\ell]_{t'+k-1} := (\sigma_{\ell,1}, \dots, \sigma_{\ell,n})$, where each party P_i holds the i -th share $\sigma_{\ell,i}$. Let $\sigma_i := (\sigma_{1,i}, \dots, \sigma_{k,i})$ denote the share vector held by P_i .
- Authentication secret keys: A long-term key $\gamma = (\gamma_1, \dots, \gamma_k) \in \mathbb{F}^k$ is used across all batches of secrets and all share holders. A short-term key $\delta(\sigma_i) \in \mathbb{F}$ is a fresh random element for each share vector σ_i . All the secret keys, including long-term and short-term keys, are shared as verifiable secret sharings to guarantee successful reconstructions.
- Authentication tags: An authentication tag for the share vector σ_i held by P_i is computed as $\delta(\sigma_i)$ as

$$\eta(\sigma_i) \leftarrow \sigma_i \odot \gamma + \delta(\sigma_i), \quad (2)$$

We use $[\mathbf{X}]_{t'+k-1}$ to denote the DSS of $\mathbf{X}$ and use $[\mathbf{x}_\ell]_{t'+k-1}$ to denote the packed sharing of the ℓ -th row $\mathbf{x}_\ell$ authenticated as a DSS for $\ell \in [k]$.

Such a packed secret sharing scheme is linearly homomorphic. In particular, for two secret matrices $\mathbf{X}^{(1)}$ and $\mathbf{X}^{(2)}$, suppose P_i holds the corresponding share vectors $\sigma_i^{(1)}$ and $\sigma_i^{(2)}$. Then P_i can compute the tag for $\sigma_i \leftarrow \sigma_i^{(1)} + \sigma_i^{(2)}$ by adding up the corresponding tags, i.e.,

$$\eta(\sigma_i) \leftarrow \eta(\sigma_i^{(1)}) + \eta(\sigma_i^{(2)}) = \sigma_i \odot \gamma + \delta(\sigma_i^{(1)}) + \delta(\sigma_i^{(2)}),$$

where the short-term key becomes $\delta(\sigma_i) \leftarrow \delta(\sigma_i^{(1)}) + \delta(\sigma_i^{(2)})$.

Considering the cost of secret keys, every batch of k^2 secrets requires $k + n$ secrets stored via verifiable secret sharings. Ignoring the cost of long-term keys, which is amortized across all batches, this results in an amortized cost of $O(1)$ elements per secret when $k = \Omega(n)$ and the VSS scheme is instantiated using linear-cost protocols [BFO12,GSZ20], as detailed in Section A.3.

Now computing each tag still requires k multiplications for k packed sharings. This time, however, only a single tag needs to be computed for each party. Thus, the total cost is reduced to $O(n)$ multiplications per packed sharing and $O(|C|)$ multiplications overall. In Section 2.4, we show how to compute the tag for each party with amortized $O(n)$ elements of communication plus $O(n)$ random OLEs.

We emphasize that, in our DSS scheme, the authentication keys are shared among all parties, whereas in [BFO12,GSZ20] each party maintains its own authentication keys. This distinction introduces additional challenges when verifying a share vector σ_i . We discuss these challenges in Section 2.5.

2.4 Computing Authentication Tags Assuming OLE Correlations

In this section, we show how to compute the authentication tags with pairwise random OLE correlations. Our construction uses a functionality $\mathcal{F}_{\text{VSS}}$ for verifiable secret sharing scheme, which can be instantiated by [BFO12,GSZ20] with $O(n)$ communication per sharing. We will use $\mathcal{F}_{\text{VSS}}$ to prepare long-term authentication key γ and short-term keys, one for each batch of shares. Every party will hold a degree- t Shamir sharing for each value of the long-term and short-term keys. In addition, we also assume that all parties hold a degree- $(t' + k - 1)$ packed Shamir sharing of γ . We refer the reader to Section A.3 for more discussion about $\mathcal{F}_{\text{VSS}}$ and the preparation of authentication keys.

Authentication of Packed Sharings for a Single Dealer. We consider the scenario where there is a single dealer D who wishes to distribute a batch of degree- $(t' + k - 1)$ packed Shamir sharings and all parties want to compute authentication tags of their shares from D under the global authentication key γ . In the beginning, D will distribute shares to all parties. Then for each P_i and each group of k shares $\mathbf{x}$, we want to let P_i learn $\eta = \mathbf{x} \odot \gamma + \delta$, where δ is the associated short-term key. We view the computation of this authentication tag as a circuit and securely compute it in a packed manner.

Recall that all parties hold $[\gamma]_{t'+k-1}$ and $[\delta]_t$. All parties first use the protocol in [SY25a] to prepare random packed Beaver triples in the form of $([a]_{t'+k-1}, [b]_{t'+k-1}, [c]_{t'+2k-2})$ where $a * b = c$. Then all parties do the following.

1. P_i distributes $[\mathbf{x}]_{t'+k-1}$ to all parties. Then all parties use $([a]_{t'+k-1}, [b]_{t'+k-1}, [c]_{t'+2k-2})$ to compute a degree- $(t' + 2k - 2)$ packed Shamir sharing of $[\mathbf{x} * \gamma]_{t'+2k-2}$.
2. Let $\mathbf{e}_1$ be a unit vector where the first entry is 1 but all other entries are 0. All parties locally compute $[\mathbf{y}]_{t'+2k-2} := [\mathbf{x} * \gamma]_{t'+2k-2} + [\delta]_t \cdot [\mathbf{e}_1]_{k-1}$, where $[\delta]_t \cdot [\mathbf{e}_1]_{k-1}$ would be a degree- $(t + k - 1)$ packed Shamir sharing with secrets $\delta \cdot \mathbf{e}_1$.
3. Note that the authentication tag $\eta = \mathbf{x} \odot \gamma + \delta = \sum_{i=1}^k y_i$.
4. All parties send their shares to P_i ⁶.
5. P_i reconstructs the secret $\mathbf{y} = (y_1, \dots, y_k)$ to learn its authentication tag $\eta \leftarrow y_1 + \dots + y_k$.

In this way, each tag can be computed with an amortized of $O(n)$ elements of communication together with $O(n)$ pairwise random OLE correlations.

However, the above process can easily go wrong and an honest P_i may obtain an incorrect authentication tag. To verify his tags for many groups of shares, all parties will compute a tag for a random vector chosen by P_i used as the random mask. Then P_i will broadcast a random linear combination of all his shares and the corresponding tag, denoted by $\tilde{\mathbf{x}}, \tilde{\eta}$. P_i will also broadcast the random coefficients so that all parties can locally compute the corresponding short-term key $[\tilde{\delta}]_t$. To verify whether $\tilde{\eta} = \tilde{\mathbf{x}} \odot \gamma + \tilde{\delta}$, all parties can locally

⁶ The current approach is insecure as it leaks information regarding γ . In the precise protocol, a random packed sharing $[\mathbf{o}]_{n-1}$ with secret $\mathbf{o} = (o_1, \dots, o_k) \in \mathbb{F}^k$ satisfying $o_1 + \dots + o_k = 0$ will be used to randomize the sharing to ensure nothing beyond the inner product is revealed to P_i . We omit this in the overview for simplicity.

compute a degree- t Shamir sharing of $[o]_t = [\tilde{\eta} - \tilde{\mathbf{x}} \odot \boldsymbol{\gamma} - \tilde{\delta}]_t$, which should be a degree- t Shamir sharing of 0. All parties open their shares to check whether the secret is 0.

Note that the above authentication process does not require each P_i to use the actual shares distributed by the dealer. This means that a corrupted P_i may authenticate any shares he want. To check this, all parties together generate a random coin and then open a random linear combination of all sharings dealt by the dealer. In addition, each party needs to provide the associated tag for his shares. All parties follow the above approach to verify each party's tag and check whether all shares lie on a valid degree- $(t' + k - 1)$ polynomial.

Fault Localization. If any check fails, all parties need to identify a new dispute pair. In our construction, we will only invoke this step in the preprocessing phase, which is independent of the function inputs. Thus, we can safely open all parties' views and discard all prepared data. In particular, the dealer will provide all his degree- $(t' + k - 1)$ packed Shamir sharings, each pair of parties will provide their pairwise random OLE correlations, and all parties will provide their shares of the authentication keys, which can be verified relying on $\mathcal{F}_{\text{VSS}}$. To achieve the desired efficiency, we apply the dispute control framework in the preprocessing phase as well.

However, a subtle issue of this approach is that each time the computation fails, all parties have to open the long-term authentication key $\boldsymbol{\gamma}$ to locate a dispute pair. This would ruin the authentications under the same key $\boldsymbol{\gamma}$ in the previous segment. In our construction, we will use a fresh temporary long-term authentication key $\boldsymbol{\gamma}'$ for each segment first. If the computation succeeds in the current segment, all parties will reconstruct $\boldsymbol{\gamma} - \boldsymbol{\gamma}'$ using $\mathcal{F}_{\text{VSS}}$ and each P_i can locally update his authentication tag.

Supporting Random Packed Beaver Triples. Recall that our online protocol requires random packed Beaver triples under authentication. We achieve this by following [SY25a] using pairwise random OLE correlations but over detectable secret sharings.

Concretely, in [SY25a], each pair of parties first uses random OLE correlations to prepare random Beaver triples additively shared between them. Then they use packed Shamir sharings to share their Beaver triples to all parties. Finally, all parties perform the triple extraction technique in [CP17] to obtain fresh Beaver triples over packed Shamir sharings. In our construction, we only change to let each pair of parties use $\mathcal{F}_{\text{DSS}}$ to share their Beaver triples. If the triple extraction fails, all parties simply open their views to identify a new dispute pair. Again since all these random packed Beaver triples are prepared in the preprocessing phase, we can open them without losing the privacy. Note that this time we can rely on $\mathcal{F}_{\text{DSS}}$ to check the correctness of parties' shares and there is no need to reveal the authentication keys.

2.5 Checking Authentication Tags with Weak Homomorphism

We now discuss how to realize the verification of each party's shares in $\mathcal{F}_{\text{DSS}}$ by leveraging linear homomorphism of the authentication tags shown in Section 2.3.

Recall that the task is to verify each party P_i 's share w of a degree- $(t' + k - 1)$ packed Shamir sharing authenticated by $\mathcal{F}_{\text{DSS}}$. Here the subtlety is that the authentication tag can only be used to verify a vector of shares rather than a single share, and the linear homomorphism of the authentication tags only supports linear combinations of shares within the same position in different share vectors. To overcome this difficulty, note that the share w to be verified is a linear combination of P_i 's shares with authentication tags. Then by linear homomorphism of authentication tags, for each $j \in [k]$, we can compute a linear combination of all share vectors (with authentication tags) whose j -th entry contributes to the share w , where this combination is denoted by $\mathbf{x}^{(j)}$. As a result, the problem can be summarized as follows.

- P_i holds k share vectors $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(k)} \in \mathbb{F}^k$ and their corresponding tags.
- The share w we want to verify can be computed as $w = x_1^{(1)} + \dots + x_k^{(k)}$.

Now using the tag $\eta(\mathbf{x}^{(i)})$, we may verify $\mathbf{x}^{(i)}$ only if $\mathbf{x}^{(i)}$ is made public. To protect the secrecy, for each verification, we will consume k fresh random share vectors $\mathbf{r}^{(1)}, \dots, \mathbf{r}^{(k)}$ of P_i with authentications. All parties perform the following steps.

1. P_i broadcasts $z = r_1^{(1)} + \dots + r_k^{(k)}$.
2. All parties generate a random challenge α and compute $w + \alpha \cdot z$. In addition, P_i broadcasts its share vectors $\{\mathbf{x}^{(i)} + \alpha \cdot \mathbf{r}^{(i)}\}_{i=1}^k$ and the corresponding tags.
3. All parties verify that $w + \alpha \cdot z = \sum_{i=1}^k (x_i^{(i)} + \alpha \cdot r_i^{(i)})$ and the tag of each share vector matches.

In particular, in the last step, with the public share vector $\mathbf{x}^{(i)} + \alpha \cdot \mathbf{r}^{(i)}$ and the corresponding tag $\eta(\mathbf{x}^{(i)} + \alpha \cdot \mathbf{r}^{(i)})$, all parties may use their degree- t Shamir sharings of the long-term and short-term authentication keys to compute a degree- t Shamir sharing $[\eta(\mathbf{x}^{(i)} + \alpha \cdot \mathbf{r}^{(i)})]_t$, and then compute $[o]_t := \eta(\mathbf{x}^{(i)} + \alpha \cdot \mathbf{r}^{(i)}) - [\eta(\mathbf{x}^{(i)} + \alpha \cdot \mathbf{r}^{(i)})]_t$. Now it is sufficient to test whether $o = 0$.

However, unlike the preprocessing phase where we can directly open $[o]_t$, here we should not reveal the value o unless $o = 0$. This is because if a corrupted P_i provides an incorrect share vector, although $o \neq 0$ with overwhelming probability, the value o would reveal some linear combination of the long-term key. Recall that in the preprocessing phase, we first use a fresh temporary long-term key and switch to γ only when the verification passes. Thus revealing o will not hurt the privacy of γ .

To resolve this issue, we require a secure check of 0-sharing in $\mathcal{F}_{\text{VSS}}$, where all parties provide their shares to $\mathcal{F}_{\text{VSS}}$ and the functionality returns **Accept** if the secret is 0, and **Reject** otherwise. This can be realized by computing a secure multiplication with a random secret, and then opening the multiplication result, which is supported by [BFO12,GSZ20].

2.6 Conclusion and Organization

To summarize our construction at a high level, we upgrade the construction in [SY25a] to achieve GOD security with the same amount of communication and random OLE correlations. To this end, we require an authentication mechanism for packed Shamir sharings, which is the new notion of detectable secret sharings introduced in Section 2.1. With random packed Beaver triples shared over detectable secret sharings, we can maintain $O(|C|)$ online communication while achieving GOD as outlined in Section 2.2. Then, we show how to realize this authentication mechanism from random OLE correlations with constant overhead as described in Sections 2.3, 2.4, and 2.5, where we introduce a new authentication tag construction that authenticates packed Shamir sharings of total size $O(|C|)$ using only $O(|C|)$ multiplications.

Organization. In Section 3, we present some preliminaries, including security definition and dispute control framework. Section 4 presents basic functionality of the detectable secret sharing scheme, the protocols for distributing detectable secret sharings and verifying detectable secret sharings, along with an overview of additional functionalities that are supported by the DSS functionality. This is followed by Section 5, which shows how to use the DSS functionality to compute SIMD circuits with GOD, for the sake of presentation.

Supplementary material. Section A includes some additional preliminaries, including the instantiation of verifiable secret sharing and generation of packed Beaver triples, which are used to realize the DSS functionality. Sections B and C present the full instantiations of the basic and extended DSS functionalities, respectively. Section D shows the full protocols for computing general circuits. Section E describes different approaches for generating pairwise random OLE correlations. Section F details the underlying building blocks. For ease of reference, Figures 2 and 3 summarize the overall framework of our honest majority GOD protocol and the recursive verification procedure. Finally, we include all the security proofs in Section H.

3 Preliminary

3.1 Security Definition

We consider a set of parties $\{P_1, P_2, \dots, P_n\}$. For every pair of parties, there exists a secure (private and authentic) synchronous channel. Beyond that, we also assume the existence of a secure broadcast channel available to all parties. We denote the communication complexity of a protocol by $\text{P2P}(M) + \text{BC}(L)$, which

means the protocol costs M elements in total over the point-to-point private channels and calls the broadcast channel with messages of L bits.

In our work, we apply the UC security framework proposed in [Can01]. We refer the reader to Section A.1 for more details about the definition.

3.2 Dispute Control Framework

To achieve GOD, our protocols run in the dispute control framework introduced in [BH06]. At a high level, we uniformly divide the circuit into n^2 segments, which will be computed sequentially. In each segment, all parties first evaluate the circuit and then verify the computation. Once the verification fails, all parties locate a dispute pair containing at least one corrupted party. Since each honest party can at most conflict with t corrupted parties, any party conflicting with more than t parties will be identified as a corrupted party and then be eliminated. Before re-computing the current segment, for each dispute pair of two active parties, parties assign a relay party who helps to transmit the messages exchanged between these dispute parties. By doing this, any dispute pair can disrupt the computation at most once and each failure must result in a new dispute pair, which implies the number of failures is bounded by n^2 . Since each segment is of size $O(|C|/n^2)$, the communication cost at most doubles due to failures and retries. We present a procedure to update the set of dispute pairs and determine eliminated parties in **DISPUTE-CONTROL**.

In our construction, as mentioned in Section 2.2, we require the number of active parties to be at least $t' + 2k - 1$ to ensure enough shares for reconstructing a degree- $(t' + 2k - 2)$ packed Shamir sharing. Setting $k = n/12$, after $n - (t' + 2k - 2)$ corrupted parties have been eliminated, the corruption threshold would be reduced to $1/4$. At this time, the remaining parties will restart the computation and run a perfectly secure MPC protocol with overall $O(|C| + D \cdot n + C_O \cdot n + n^4)$ communication in the plain model as constructed in [SY25b].

4 Detectable Secret Sharings

Towards achieving GOD, as outlined in Section 2.1, we require a packed secret sharing scheme providing verifiability. In this section, we present our detectable secret sharing scheme. We first formalize the base DSS functionality in $\mathcal{F}_{\text{base-DSS}}$, which initializes a database that stores packed secret sharings and consists of three instructions: 1) (**Share**, P_d) enables a dealer P_d to distribute degree- $(t' + k - 1)$ packed sharings to all parties, after which the functionality stores the sharings in the database; 2) (**Auth**, P_r) allows to authenticate degree- $(k - 1)$ packed sharings distributed by an active party P_r ; and 3) **Check** allows to verify the correctness of a packed sharing that is a linear combination of the packed sharings stored in the database. If an error is detected, the functionality reports the identity of the party providing the incorrect share.

Functionality 1: $\mathcal{F}_{\text{base-DSS}}(M)$

Setup. On input **Setup**,

1. Receive the set of corrupted parties $\mathcal{C}$.
2. Receive a public parameter M^a and initializes $\text{cnt} \leftarrow 0$.
3. Initialize a database that stores the generated secret sharings and support their subsequent retrieval. $\llbracket \cdot \rrbracket_d$ denotes a degree- d packed Shamir sharing that has been stored by $\mathcal{F}_{\text{base-DSS}}$.
4. Invoke **DISPUTE-CONTROL** to initialize $\mathcal{P}_{\text{active}}$, Corr , and Disp .
5. Wait for an instruction from the adversary.
 - Upon receiving **reject** and $\mathcal{C}' \subseteq \mathcal{C}$ such that $|\mathcal{C}'| = t - 2k + 2$, $\mathcal{F}_{\text{DSS}}$ outputs **abort** and $\mathcal{C}'$.
 - Upon receiving **accept**, $\mathcal{F}_{\text{DSS}}$ sets up successfully.

Update. On input a new dispute pair E , invoke **DISPUTE-CONTROL** to update $\mathcal{P}_{\text{active}}$, Corr , and Disp . If $\mathcal{P}_{\text{active}} < t' + 2k - 1$, $\mathcal{F}_{\text{base-DSS}}$ **aborts**.

Instruction set.

1. On input (**Share**, P_d),
 1. $\mathcal{F}_{\text{base-DSS}}$ receives $[x_1]_{t'+k-1}, \dots, [x_N]_{t'+k-1}$ from the dealer P_d and sends shares of active corrupted parties to the adversary.
 2. $\mathcal{F}_{\text{base-DSS}}$ waits for an instruction from the adversary.
 - Upon receiving (**reject**, E) such that $E \notin \mathcal{D}_{\text{isp}}$, output E to active parties.
 - Upon receiving **accept**, store the sharings and distribute them to active parties.
2. On input (**Auth**, P_r),
 1. $\mathcal{F}_{\text{base-DSS}}$ receives from honest parties their shares of $[x_1]_{k-1}, \dots, [x_N]_{k-1}$, reconstructs the whole sharings, and sends shares of corrupted parties to the adversary.
 2. $\mathcal{F}_{\text{base-DSS}}$ waits for an instruction from the adversary.
 - Upon receiving (**reject**, E) such that $E \notin \mathcal{D}_{\text{isp}}$, output E to active parties.
 - Upon receiving **accept**, store the sharings.
3. On receiving a request **Check** to check a packed Shamir sharing $[\![w]\!]_{t'+k-1}$, represented as a linear combination of packed Shamir sharings stored by $\mathcal{F}_{\text{base-DSS}}$,
 1. $\mathcal{F}_{\text{base-DSS}}$ receives the sharing $[\![w]\!]_{t'+k-1}$ from active parties, sets $\text{cnt} \leftarrow \text{cnt} + 1$, and outputs **fail** if $\text{cnt} > M$.
 2. $\mathcal{F}_{\text{base-DSS}}$ computes $[\![w]\!]_{t'+k-1}$ following the same linear combination and using the stored sharings, and checks the received one.
 - If some P_i provides an incorrect share, output $\{P_i\}$ to active parties.
 - Otherwise, output **ok** to active parties.

^a M specifies the maximum number of invocations of **Check**.

To realize this, we instantiate the packed secret sharing scheme with authentication tags introduced in Section 2.3. For the first instruction, **Share**, we present protocols that allow a dealer to distribute authenticated degree- $(t' + k - 1)$ packed sharings assuming packed Beaver triples provided by $\mathcal{F}_{\text{packed-triple}}$ in Section 4.1. As for the second instruction, **Auth**, the instantiation for authenticating degree- $(k - 1)$ packed sharings is similar to the first one, but doesn't require packed Beaver triples due to the lower degree. We refer the readers to Section B.1 for details. Regarding the third instruction, **Check**, we present the protocols for verifying detectable secret sharings in Section 4.2. With the three building blocks above, we obtain the instantiation for the base DSS functionality, whose formal description is deferred to Section B.2.

In Section 4.3, we give an overview of the extended DSS functionality, which is used in the online computation and whose realization combines the base DSS functionality with known techniques.

4.1 Distributing Detectable Secret Sharings

After a dealer distributes packed Shamir sharings, as outlined in Section 2.4, the parties jointly compute authentication tag of each share vector for the corresponding share holder, assuming packed Beaver triples provided by $\mathcal{F}_{\text{packed-triple}}$. The instantiation of $\mathcal{F}_{\text{packed-triple}}$ is discussed in Section A.4.

We present the protocol that allows a dealer to distribute authenticated packed Shamir sharings in **DSS-SHARE**, which invokes **COMPUTE-TAG** to compute the authentication tags. Regarding the correctness, to ensure each honest party P_i obtains correct tags when invoking **COMPUTE-TAG**(P_i), each P_i checks whether a random linear combination of the tags is correctly computed for the share vector computed using the same linear coefficients. To ensure corrupted parties authenticate correct share vectors distributed by dealer P_d in **DSS-SHARE**, P_d checks whether the corrupted party learns the tag for a random linear combination of the correct share vectors. Moreover, all parties check whether P_d distributes valid degree- $(t' + k - 1)$ packed Shamir sharings. To protect the secrecy, we incorporate an extra random share vector for all the verifications above. In case of failure, as parties first use random long-term keys, the computation does not involve private inputs and thus parties can safely open their transcripts to locate a dispute pair. In case of success, all parties locally update their tags to the real long-term keys.

The cost of **DSS-SHARE** to distribute $m \cdot k$ authenticated packed Shamir sharings is $\text{P2P}(O(m \cdot n^2 + n^3 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^2 \cdot \log |\mathbb{F}|))$ plus $O(m \cdot n^2 + n^3)$ random OLEs provided by $\mathcal{F}_{\text{rand-OLE}}$, amounting to amortized $O(1)$ elements plus $O(1)$ OLEs per secret.

Protocol 2: $\text{DSS-SHARE}(P_d, \gamma)$

Input: $\mathbf{X}^{(1)}, \dots, \mathbf{X}^{(m)}$ held by P_d .

Output: For $\ell \in [m]$, each P_i obtains $\sigma_i^{(\ell)}$ and $\eta(\sigma_i^{(\ell)})$ / a dispute pair.

Ingredients: $(\{\langle \gamma_j | j \rangle_t\}_{j=1}^k, \{\langle \gamma_j | 1 \rangle_t\}_{j=2}^k)$, where $\langle \cdot | j \rangle_t$ denotes a degree- t Shamir secret sharing with its secret at the j -th position, and the sharing is stored by the verifiable secret sharing functionality $\mathcal{F}_{\text{VSS}}$.

- 1: **Distributing shares.** P_d distributes $[\mathbf{X}^{(1)}]_{t'+k-1}, \dots, [\mathbf{X}^{(m)}]_{t'+k-1}, [\mathbf{X}^{(m+1)}]_{t'+k-1}$ to all active parties, where $[\mathbf{X}^{(m+1)}]_{t'+k-1}$ will be used as a random mask for verification.
- 2: **Preparing random secret keys.** All parties invoke $\mathcal{F}_{\text{VSS}}$ with **LongTermKey** to prepare $(\{\langle \mu_j | j \rangle_t\}_{j=1}^k, \{\langle \mu_j | 1 \rangle_t\}_{j=2}^k)$ used as random long-term secret keys. Set $\mu \leftarrow (\mu_1, \dots, \mu_k)$.
- 3: **Computing tags.** For each $i \in \mathcal{P}_{\text{active}}$, all parties invoke **COMPUTE-TAG** (P_i, μ) to compute the authentication tags of $\sigma_i^{(1)}, \dots, \sigma_i^{(m+1)}$ for P_i who obtains the results $y_i^{(1)}, \dots, y_i^{(m+1)}$, where $y_i^{(\ell)} = \sigma_i^{(\ell)} \odot \mu + \delta(\sigma_i^{(\ell)})$ and $\delta(\sigma_i^{(\ell)})$ is the short-term key.
- 4: **Verifying authentication.** All parties and the dealer P_d check the correctness of the authenticated messages as follows.
 1. All parties invoke $\mathcal{F}_{\text{coin}}$ to generate a random element λ .
 2. Each active party P_i computes and broadcasts

$$(\sigma_i, y_i) \leftarrow (\sigma_i^{(m+1)}, y_i^{(m+1)}) + \sum_{\ell=1}^m \lambda^\ell \cdot (\sigma_i^{(\ell)}, y_i^{(\ell)}).$$

3. For each active party P_i , all parties locally compute

$$\begin{aligned} \langle \delta_i | 1 \rangle_t &\leftarrow \langle \delta(\sigma_i^{(m+1)}) | 1 \rangle_t + \sum_{\ell=1}^m \lambda^\ell \cdot \langle \delta(\sigma_i^{(\ell)}) | 1 \rangle_t, \\ \langle o_i | 1 \rangle_t &\leftarrow y_i - \sigma_i \odot (\langle \mu_1 | 1 \rangle_t, \dots, \langle \mu_k | 1 \rangle_t) - \langle \delta_i | 1 \rangle_t. \end{aligned}$$

and invoke $\mathcal{F}_{\text{VSS}}$ with **(CheckZero, 1)** to check whether the secret of $\langle o_i | 1 \rangle_t$ is 0.

- If not, all parties output the corrupted party $\{P_i\}$.
4. The dealer P_d checks whether the share vectors $(\sigma_i)_{i \in \mathcal{P}_{\text{active}}}$ broadcast in Step 4.2 correspond to that distributed by itself.
 - If not, P_d is responsible to locate an incorrect share. All parties identify a dispute pair via relay.
 5. All parties check whether $(\sigma_i)_{i \in \mathcal{P}_{\text{active}}}$ form k valid degree- $(t' + k - 1)$ packed Shamir secret sharings.
 - If not, set each $[\mathbf{X}^{(\ell)}]_{t'+k-1}$ to be an all-zero sharing and halt.
- 5: **Update authentication key.**
1. For $j \in [k]$, all parties invoke $\mathcal{F}_{\text{VSS}}$ with **(Open, 1)** to reveal the secret $\gamma_j - \mu_j$ of the sharing $\langle \gamma_j - \mu_j | 1 \rangle_t$.
 2. For $\ell \in [m]$, each party P_i updates its authentication tag as $\eta(\sigma_i^{(\ell)}) \leftarrow y_i^{(\ell)} + \sigma_i^{(\ell)} \odot (\gamma - \mu)$.

Protocol 3: COMPUTE-TAG(P_i, μ)

Input: $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ held by P_i .

Output: P_i obtains $y^{(1)}, \dots, y^{(m)}$, where $y^{(\ell)} = \mathbf{x}^{(\ell)} \odot \mu + \delta(\mathbf{x}^{(\ell)})$ and $\odot$ denotes the inner product / a dispute pair.

Ingredients: $(\{\langle \mu_j | j \rangle_t\}_{j=1}^k, \{\langle \mu_j | 1 \rangle_t\}_{j=2}^k)$.

All parties agree on a special party P_{king} .

1: Preparing random sharings.

1. All parties invoke $\mathcal{F}_{\text{VSS}}$ with $(\text{Rand}, 1, m+1)$ to prepare short-term secret keys $\langle \delta(\mathbf{x}^{(1)}) | 1 \rangle_t, \dots, \langle \delta(\mathbf{x}^{(m+1)}) | 1 \rangle_t$.
2. All parties invoke $\mathcal{F}_{\text{packed-triple}}$ with $(m+1)$ to generate $m+1$ random packed Beaver triples $([\mathbf{a}^{(\ell)}]_{t'+k-1}, [\mathbf{b}^{(\ell)}]_{t'+k-1}, [\mathbf{c}^{(\ell)}]_{t'+2k-2})_{\ell=1}^{m+1}$.
3. All parties invoke $\text{RAND}(\Sigma_3, m+1)$ to prepare $m+1$ random sharings $[\mathbf{o}^{(1)}]_{n'-1}, \dots, [\mathbf{o}^{(m+1)}]_{n'-1}$ with the secret $\mathbf{o}^{(\ell)} = (o_1^{(\ell)}, \dots, o_k^{(\ell)})$ satisfying $\sum_{j=1}^k o_j^{(\ell)} = 0$ for each $\ell \in [m+1]$.

2: Computing tags. All parties locally compute $[\mu]_{t+k-1} \leftarrow \sum_{j=1}^k [\mathbf{e}_j]_{k-1} \cdot \langle \mu_j | j \rangle_t$, where $\mathbf{e}_j \in \mathbb{F}^k$ denotes the j -th unit vector. P_i randomly samples $\mathbf{x}^{(m+1)} \in \mathbb{F}^k$.

For $\ell \in [m+1]$,

1. P_i distributes a random packed Shamir sharing $[\mathbf{x}^{(\ell)}]_{t'+k-1}$ to all parties.
2. All parties use the packed Beaver triple $([\mathbf{a}^{(\ell)}]_{t'+k-1}, [\mathbf{b}^{(\ell)}]_{t'+k-1}, [\mathbf{c}^{(\ell)}]_{t'+2k-2})$ to compute $[\mathbf{x}^{(\ell)} * \mu]_{t'+2k-2}$. All parties locally compute $[\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{t'+k-1}, [\mu + \mathbf{b}^{(\ell)}]_{t'+k-1}$ and send them to P_{king} who reconstructs the secrets $\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}, \mu + \mathbf{b}^{(\ell)}$ and distributes $[\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{k-1}, [\mu + \mathbf{b}^{(\ell)}]_{k-1}$ to all parties.
3. All parties locally compute

$$\begin{aligned} [\mathbf{y}^{(\ell)}]_{n'-1} &\leftarrow [\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{k-1} \cdot [\mu + \mathbf{b}^{(\ell)}]_{k-1} \\ &\quad - [\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{k-1} \cdot [\mathbf{b}^{(\ell)}]_{t'+k-1} \\ &\quad - [\mu + \mathbf{b}^{(\ell)}]_{k-1} \cdot [\mathbf{a}^{(\ell)}]_{t'+k-1} + [\mathbf{c}^{(\ell)}]_{t'+2k-2} \\ &\quad + \langle \delta(\mathbf{x}^{(\ell)}) | 1 \rangle_t \cdot [\mathbf{e}_1]_{k-1} + [\mathbf{o}^{(\ell)}]_{n'-1}, \end{aligned}$$

where $\mathbf{e}_1 = (1, 0, \dots, 0) \in \mathbb{F}^k$, and send $[\mathbf{y}^{(\ell)}]_{n'-1}$ to P_i .

4. P_i reconstructs the secret $\mathbf{y}^{(\ell)} = (y_1^{(\ell)}, \dots, y_k^{(\ell)})$ and computes $y^{(\ell)} \leftarrow \sum_{j=1}^k y_j^{(\ell)}$.

3: Verifying tag computation. P_i verifies the correctness of tag computation as follows.

1. All parties invoke $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_1)$ to check whether P_{king} distributes valid degree- $(k-1)$ packed sharings in Step 2.2.
 - In case of failure, output the dispute pair E .
2. P_i randomly samples α and broadcasts it to all parties.
3. P_i computes and broadcasts

$$(\mathbf{x}, y) \leftarrow (\mathbf{x}^{(m+1)}, y^{(m+1)}) + \sum_{\ell=1}^m \alpha^\ell \cdot (\mathbf{x}^{(\ell)}, y^{(\ell)}).$$

4. All parties locally compute

$$\begin{aligned} \langle \delta(\mathbf{x}) | 1 \rangle_t &\leftarrow \langle \delta(\mathbf{x}^{(m+1)}) | 1 \rangle_t + \sum_{\ell=1}^m \alpha^\ell \cdot \langle \delta(\mathbf{x}^{(\ell)}) | 1 \rangle_t, \\ \langle o | 1 \rangle_t &\leftarrow y - \mathbf{x} \odot (\langle \mu_1 | 1 \rangle_t, \dots, \langle \mu_k | 1 \rangle_t) - \langle \delta(\mathbf{x}) | 1 \rangle_t. \end{aligned}$$

5. All parties invoke $\mathcal{F}_{\text{VSS}}$ with $(\text{CheckZero}, 1)$ to check whether the secret of $\langle o | 1 \rangle_t$ is 0.
 - If not, all parties open their views to identify a dispute pair E and take E as output.
 - Otherwise, P_i accepts the computation.

4.2 Verifying Detectable Secret Sharings

Suppose each party P_i broadcasts its share σ_i of the sharing $\llbracket \mathbf{w} \rrbracket_{t'+k-1}$ and all parties aim to verify the correctness of σ_i , where $\llbracket \mathbf{w} \rrbracket_{t'+k-1}$ is a linear combination of the authenticated packed sharings from different positions of different sharing vectors. As outlined in Section 2.5, we decompose the sharing as

$$\llbracket \mathbf{w} \rrbracket_{t'+k-1} = \llbracket \mathbf{w}_1^{(1)} \rrbracket_{t'+k-1} + \cdots + \llbracket \mathbf{w}_k^{(k)} \rrbracket_{t'+k-1},$$

where $\mathbf{w}_j^{(j)}$ denotes the j -th row of the secret matrix $\mathbf{W}^{(j)}$ for $j \in [k]$. Although revealing the sharing vector $\llbracket \mathbf{W}^{(j)} \rrbracket_{t'+k-1}$ and its tags allows parties to check the correctness of $\llbracket \mathbf{W}^{(j)} \rrbracket_{t'+k-1}$ and thus $\llbracket \mathbf{w}_j^{(j)} \rrbracket_{t'+k-1}$, this would break the privacy. Instead, each party P_i prepares a random vector $\mathbf{z}_i^{(j)}$ with an authentication tag $\eta(\mathbf{z}_i^{(j)})$ for $j \in [k]$ and broadcasts $z_i \leftarrow \sum_{j=1}^k \mathbf{z}_i^{(j)}$. After generating a public coin α , P_i provides the masked share vector $\sigma_i^{(j)} + \alpha \cdot \mathbf{z}_i^{(j)}$ together with its authentication tag, where $\sigma_i^{(j)}$ denotes the i -th share vector of the sharing vector $\llbracket \mathbf{W}^{(j)} \rrbracket_{t'+k-1}$. Due to the randomness of α , the correctness of $\sigma_i^{(j)} + \alpha \cdot \mathbf{z}_i^{(j)}$ implies the correctness of $\sigma_i^{(j)}$ and $\mathbf{z}_i^{(j)}$. Therefore, after ensuring $\sigma_i + \alpha \cdot z_i$ is correctly computed, the correctness of σ_i is again guaranteed by the randomness of α .

We present the protocol for checking the correctness of a detectable secret sharing in **DSS-CHECK**. The cost of **RAND-TAG** to prepare m random messages with authentication tags for each party is $\text{P2P}(O(m \cdot n^2 + n^3 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^2 \cdot \log |\mathbb{F}|))$ plus $O(m \cdot n^2 + n^3)$ random OLEs provided by $\mathcal{F}_{\text{rand-OLE}}$. The cost of **DSS-CHECK** is $\text{P2P}(O(n^3 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^3 \cdot \log |\mathbb{F}|))$ plus k pairs of random messages and tags for each active party prepared in advance.

Protocol 4: DSS-CHECK($\gamma, \llbracket \mathbf{w} \rrbracket_{t'+k-1}$)

Input: $\llbracket \mathbf{w} \rrbracket_{t'+k-1}$.

Output: ok / a dispute pair.

Ingredients: random messages with tags $\{\mathbf{z}_i^{(\ell)}, \eta(\mathbf{z}_i^{(\ell)})\}_{\ell \in [k], i \in \mathcal{P}_{\text{active}}}$.

Assume that all parties have broadcast their shares of $\llbracket \mathbf{w} \rrbracket_{t'+k-1}$. Suppose w_i is P_i 's share of $\llbracket \mathbf{w} \rrbracket_{t'+k-1}$.

- 1: **Decomposition.** Decompose $\llbracket \mathbf{w} \rrbracket_{t'+k-1}$ as $\llbracket \mathbf{w} \rrbracket_{t'+k-1} = \sum_{j=1}^k \llbracket \mathbf{w}_j^{(j)} \rrbracket_{t'+k-1}$. The notations are defined as follows:
 - for $\ell \in [k]$, $\mathbf{W}^{(\ell)} \in \mathbb{F}^{k \times k}$ denotes the secret matrix that has its ℓ -th row contributing to $\mathbf{w}$;
 - $\mathbf{w}_j^{(\ell)} \in \mathbb{F}^k$ denotes the j -th row of $\mathbf{W}^{(\ell)}$;
 - $\sigma_i^{(\ell)} = (\sigma_{j,i}^{(\ell)})_{j=1}^k$ denotes P_i 's share vector of $\llbracket \mathbf{W}^{(\ell)} \rrbracket_{t'+k-1}$. Its authentication tag is $\eta(\sigma_i^{(\ell)}) = \sigma_i^{(\ell)} \odot \gamma + \delta(\sigma_i^{(\ell)})$.
- 2: **Pick random masks.** For each $i \in \mathcal{P}_{\text{active}}$, P_i picks k unused pairs of random messages and authentication tags $(\mathbf{z}_i^{(1)}, \eta(\mathbf{z}_i^{(1)})), \dots, (\mathbf{z}_i^{(k)}, \eta(\mathbf{z}_i^{(k)}))$ prepared in the initialization phase of **BASE-DSS**, where for each $\ell \in [k]$, $\eta(\mathbf{z}_i^{(\ell)}) = \mathbf{z}_i^{(\ell)} \odot \gamma + \delta(\mathbf{z}_i^{(\ell)})$ and $\delta(\mathbf{z}_i^{(\ell)})$ is a fresh short-term key.
- 3: **Reveal masked share vectors and authentication tags.**
 1. Each active party P_i locally computes and broadcasts $z_i \leftarrow z_{i,1}^{(1)} + \cdots + z_{i,k}^{(k)}$.
 2. All parties invoke $\mathcal{F}_{\text{coin}}$ to generate a random element α .
 3. For each $\ell \in [k]$, each active party P_i locally computes and broadcasts

$$(\mathbf{y}_i^{(\ell)}, \eta(\mathbf{y}_i^{(\ell)})) \leftarrow (\mathbf{z}_i^{(\ell)}, \eta(\mathbf{z}_i^{(\ell)})) + \alpha \cdot (\sigma_i^{(\ell)}, \eta(\sigma_i^{(\ell)})).$$

4: Verification.

1. For each $i \in \mathcal{P}_{\text{active}}$, all parties check whether

$$y_{i,1}^{(1)} + \cdots + y_{i,k}^{(k)} = z_i + \alpha \cdot w_i.$$

2. For each $i \in \mathcal{P}_{\text{active}}$ and $\ell \in [k]$, all parties locally compute

$$\begin{aligned}\langle \delta(\mathbf{y}_i^{(\ell)}) | 1 \rangle_t &\leftarrow \langle \delta(\mathbf{z}_i^{(\ell)}) | 1 \rangle_t + \alpha \cdot \langle \delta(\boldsymbol{\sigma}_i^{(\ell)}) | 1 \rangle_t, \\ \langle o_i^{(\ell)} | 1 \rangle_t &\leftarrow \eta(\mathbf{y}_i^{(\ell)}) - \mathbf{y}_i^{(\ell)} \odot (\langle \gamma_1 | 1 \rangle_t, \dots, \langle \gamma_k | 1 \rangle_t) - \langle \delta(\mathbf{y}_i^{(\ell)}) | 1 \rangle_t.\end{aligned}$$

All parties invoke $\mathcal{F}_{\text{VSS}}$ with $(\text{CheckZero}, 1)$ to check whether the secret of $\langle o_i^{(\ell)} | 1 \rangle_t$ is 0. If any check fails, all parties output $\{P_i\}$, where i is the smallest index of a party whose check fails. Otherwise, all parties output ok.

Protocol 5: RAND-TAG(γ, m)

Output: for each P_i : $(\mathbf{z}_i^{(\ell)}, \eta(\mathbf{z}_i^{(\ell)}))$ for $\ell \in [m]$ / a dispute pair.

Ingredients: $(\{\langle \gamma_j | j \rangle_t\}_{j=1}^k, \{\langle \gamma_j | 1 \rangle_t\}_{j=2}^k)$.

1. All parties invoke $\mathcal{F}_{\text{VSS}}$ with LongTermKey to prepare a group of sharings $(\{\langle \mu_j | j \rangle_t\}_{j=1}^k, \{\langle \mu_j | 1 \rangle_t\}_{j=2}^k)$. Set $\boldsymbol{\mu} \leftarrow (\mu_1, \dots, \mu_k)$.
2. **Computing tags.** For each $i \in \mathcal{P}_{\text{active}}$, P_i randomly samples m messages $\mathbf{z}_i^{(1)}, \dots, \mathbf{z}_i^{(m)}$. Then all parties invoke $\text{COMPUTE-TAG}(P_i, \boldsymbol{\mu})$ to compute the authentication tags of $\mathbf{z}_i^{(1)}, \dots, \mathbf{z}_i^{(m)}$ for P_i who obtains the results $y_i^{(1)}, \dots, y_i^{(m)}$, where for each $j \in [m]$, $y_i^{(j)} = \mathbf{z}_i^{(j)} \odot \boldsymbol{\mu} + \delta(\mathbf{z}_i^{(j)})$ and $\delta(\mathbf{z}_i^{(j)})$ is a fresh short-term key.
3. **Update authentication tags.**
 - a. All parties invoke $\mathcal{F}_{\text{VSS}}$ with $(\text{Open}, 1)$ to reveal the secret $\gamma_j - \mu_j$ of the sharing $\langle \gamma_j - \mu_j | 1 \rangle_t$ for $j \in [k]$.
 - b. For each $i \in \mathcal{P}_{\text{active}}$, P_i updates its authentication tag with respect to the long-term key γ by locally computing $\eta(\mathbf{z}_i^{(\ell)}) \leftarrow y_i^{(\ell)} + \mathbf{z}_i^{(\ell)} \odot (\gamma - \boldsymbol{\mu})$ for each $\ell \in [m]$.

4.3 Overview of Extended Detectable Secret Sharings

We extend the base DSS functionality $\mathcal{F}_{\text{base-DSS}}$ to support the generation of various preprocessing data and to design an online protocol in a more modular manner. In this section, we provide an overview of the extended functionality. Detailed protocols for realizing each subfunctionality are presented in Section C.

Compared with the basic functionality, the extended DSS functionality additionally supports the following instructions:

- Authenticated packed triple $(\llbracket \mathbf{a} \rrbracket_{t'+k-1}, \llbracket \mathbf{b} \rrbracket_{t'+k-1}, \llbracket \mathbf{c} \rrbracket_{t'+k-1})$ with $\mathbf{c} = \mathbf{a} * \mathbf{b}$, labeled (Triple, m) , used for evaluating multiplication gates.
- Authenticated linear mask for transpose $(\{\llbracket \mathbf{r}_{\ell*} \rrbracket_{t'+k-1}\}_{\ell=1}^k, \{\llbracket \mathbf{r}_{*\ell} \rrbracket_{t'+k-1}\}_{\ell=1}^k)$, labeled $(\text{Transpose}, m)$, used for performing sharing transformations.
- Verifiable reconstruction of DSS to designated receivers (which may fail), labeled Recon , enabling the output phase.

We formally present this extended functionality in $\mathcal{F}_{\text{DSS}}$ in Section C, whose instantiation is obtained by combining the base DSS with known techniques.

5 Online Computation

In this section, relying on the preprocessing data and verifiability provided by the detectable secret sharing functionality, we present our online protocols for computing SIMD circuits with GOD, for ease of presentation due to space limitations. The full protocols for computing general circuits appear in Section D.

As outlined in Section 2.2, we follow the dispute control framework to achieve GOD, where we uniformly divide the circuit into n^2 segments. When computing each segment, all parties agree on a special party P_{king} , evaluate the circuit, and then verify the computation. As for the evaluation phase, we present the protocol **MULT** for evaluating multiplication gates in Section 2.2. In the following, we first show how to efficiently verify the computation of multiplication gates in Section 5.1, and then summarize the computation for a segment of a SIMD circuit in Section 5.2.

5.1 Recursive Verification

Uniform form of the transcript. As mentioned in Section 2.2, we adapt the virtual transcript technique proposed in [GSZ20] to efficiently verify the computation and locate a dispute pair when faults occur. To illustrate this, we first analyze the transcripts to be verified and transform them into a unified form. A key observation, as already discussed in Section 2.2, is that in each segment, starting from input sharings authenticated as detectable secret sharings, after evaluating the addition and multiplication gates, all parties have computed a degree- $(t' + k - 1)$ packed Shamir sharing $[z]_{t'+k-1}$ for each group of k wires z . Importantly, there is an invariant that $[z]_{t'+k-1}$ can be decomposed into two components, i.e., $[z]_{t'+k-1} = [z_0]_{k-1} + \llbracket z_1 \rrbracket_{t'+k-1}$, where the first low-degree sharing is distributed by the active party P_{king} and the second high-degree sharing is authenticated as a detectable secret sharing. Relying on this invariant, all generated transcripts can be transformed into the form produced by **EXTEND-DEG-REDUCTION**, which communicates $\text{P2P}(O(n))$ elements.

Protocol 6: EXTEND-DEG-REDUCTION

Input: $[\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}, [\vec{d}]_{k-1}, \llbracket \vec{u} \rrbracket_{t'+k-1}$, where $[\cdot]_d$ denotes a vector of degree- d Shamir sharings.

Output: $[e]_{k-1}$, where $e = \vec{v}_1 \odot \vec{v}_2 + \vec{d} \odot \vec{u} + r$ and $\odot$ denotes the inner product.

Ingredients: $\llbracket r \rrbracket_{t'+k-1}, [o]_{n'-1}$.

1. All parties locally compute $[e]_{n'-1} \leftarrow [\vec{v}_1]_{k-1} \odot [\vec{v}_2]_{k-1} + [\vec{d}]_{k-1} \odot \llbracket \vec{u} \rrbracket_{t'+k-1} + \llbracket r \rrbracket_{t'+k-1} + [o]_{n'-1}$ and send their shares to P_{king} .
2. P_{king} reconstructs the secret e and distributes $[e]_{k-1}$ to all parties.
3. The transcript is of the following form

$$([\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}, [\vec{d}]_{k-1}, \llbracket \vec{u} \rrbracket_{t'+k-1}, \llbracket r \rrbracket_{t'+k-1}, [o]_{n'-1}, [e]_{n'-1}, [e]_{k-1}).$$

We call this computation an *extended degree reduction*, and say that it has dimension (ℓ, s) if $[\vec{v}_1]_{k-1}$ and $[\vec{d}]_{k-1}$ have dimensions ℓ and s , respectively. The correctness of such a transcript refers to the secret of $[e]_{k-1}$ being equal to $\vec{v}_1 \odot \vec{v}_2 + \vec{d} \odot \vec{u} + r$, where the sharings $[e]_{k-1}, [\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}$, and $[\vec{d}]_{k-1}$ are determined by honest parties' shares, while $\llbracket \vec{u} \rrbracket_{t'+k-1}$ and $\llbracket r \rrbracket_{t'+k-1}$ are authenticated as detectable secret sharings. If this condition does not hold, then the computation must be incorrect, and a dispute pair should be identified. Based on this unified representation, we can jointly verify the correctness of all generated transcripts. In the following, we first present **VRIFY-SINGLE-TRANSCRIPT**, which verifies the correctness of a single transcript and identifies either a dispute pair or a corrupted party if the check fails. We then show how to compress all transcripts to be verified into a single random virtual transcript, which can subsequently be verified using **VRIFY-SINGLE-TRANSCRIPT**.

The cost of **VRIFY-SINGLE-TRANSCRIPT** is $\text{P2P}(O(n^3 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^3 \cdot \log |\mathbb{F}|))$ and two invocations of $\mathcal{F}_{\text{DSS}}$ with **Check**.

Protocol 7: VRIFY-SINGLE-TRANSCRIPT

Input: A transcript $([\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}, [\vec{d}]_{k-1}, \llbracket \vec{u} \rrbracket_{t'+k-1}, \llbracket r \rrbracket_{t'+k-1}, [o]_{n'-1}, [e]_{n'-1}, [e]_{k-1})$ of dimension $(1, 1)$.

Output: ok / a dispute pair.

1. All parties broadcast their shares of $[\mathbf{v}_1]_{k-1}, [\mathbf{v}_2]_{k-1}, [\mathbf{d}]_{k-1}, \llbracket \mathbf{u} \rrbracket_{t'+k-1}, \llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}, [\mathbf{e}]_{n'-1}, [\mathbf{e}]_{k-1}$. P_{king} broadcasts the sharing $[\mathbf{e}]_{n'-1}$ it received and $[\mathbf{e}]_{k-1}$ it distributed. The relay parties (if any) also broadcast the values they helped pass.
2. All parties follow the protocol to verify message consistency and the correctness of local computations. In case of failure, the parties output either a pair of adjacent parties that disagree on the same message or a corrupted party that performed an incorrect local computation.
3. **Check input sharings.**
 - (a) All parties invoke $\mathcal{F}_{\text{DSS}}$ with **Check** and provide the sharings $\llbracket \mathbf{u} \rrbracket_{t'+k-1}, \llbracket \mathbf{r} \rrbracket_{t'+k-1}$ to check the correctness of them. In case of incorrectness, $\mathcal{F}_{\text{DSS}}$ outputs a dispute pair E .
 - (b) All parties check whether $[\mathbf{o}]_{n'-1}$ is a degree- $(n'-1)$ packed Shamir sharing with secret $\mathbf{0}$. If not, all parties invoke **FL-ZERO SHARING** to locate a dispute pair.
 - (c) All parties check whether $[\mathbf{v}_1]_{k-1}, [\mathbf{v}_2]_{k-1}$ are valid degree- $(k-1)$ packed Shamir sharings. If not, P_{king} is responsible to locate an incorrect share. All parties identify a dispute pair via relay.
4. If all checks above pass, all parties take ok as output.

Compressing Transcripts. Following [GSZ20], the compression procedure proceeds in three steps: 1) De-Linearization, 2) Dimension-Reduction, and 3) Recursion and Randomization.

Step One: De-Linearization. The first step transforms the check of M transcripts for extended degree reduction, where the i -th transcript has dimension (ℓ_i, s_i) , into the check of a single transcript of dimension $(\sum_{i=1}^M \ell_i, \sum_{i=1}^M s_i)$. Importantly, a random challenge is incorporated to ensure that if any of the original transcripts is incorrect, the resulting transcript is incorrect with overwhelming probability. The protocol for de-linearization appears in **DE-LINEARIZATION**. The cost of **DE-LINEARIZATION** is $\text{BC}(O(\log |\mathbb{F}|))$.

Protocol 8: DE-LINEARIZATION

Input: M transcripts $([\vec{\mathbf{v}}_1^{(i)}]_{k-1}, [\vec{\mathbf{v}}_2^{(i)}]_{k-1}, [\vec{\mathbf{d}}^{(i)}]_{k-1}, \llbracket \vec{\mathbf{u}}^{(i)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{k-1})_{i=1}^M$, where the i -th transcript is of dimension (ℓ_i, s_i) .

Output: A single transcript of dimension $(\sum_{i=1}^M \ell_i, \sum_{i=1}^M s_i)$.

1. All parties invoke $\mathcal{F}_{\text{coin}}$ to generate a random element λ .
2. All parties locally set

$$\begin{aligned}
[\vec{\mathbf{v}}_1]_{k-1} &= ([\vec{\mathbf{v}}_1^{(1)}]_{k-1}, \lambda \cdot [\vec{\mathbf{v}}_1^{(2)}]_{k-1}, \dots, \lambda^{M-1} \cdot [\vec{\mathbf{v}}_1^{(M)}]_{k-1}), \\
[\vec{\mathbf{v}}_2]_{k-1} &= ([\vec{\mathbf{v}}_2^{(1)}]_{k-1}, [\vec{\mathbf{v}}_2^{(2)}]_{k-1}, \dots, [\vec{\mathbf{v}}_2^{(M)}]_{k-1}), \\
[\vec{\mathbf{d}}]_{k-1} &= ([\vec{\mathbf{d}}^{(1)}]_{k-1}, \lambda \cdot [\vec{\mathbf{d}}^{(2)}]_{k-1}, \dots, \lambda^{M-1} \cdot [\vec{\mathbf{d}}^{(M)}]_{k-1}), \\
\llbracket \vec{\mathbf{u}} \rrbracket_{t'+k-1} &= (\llbracket \vec{\mathbf{u}}^{(1)} \rrbracket_{t'+k-1}, \llbracket \vec{\mathbf{u}}^{(2)} \rrbracket_{t'+k-1}, \dots, \llbracket \vec{\mathbf{u}}^{(M)} \rrbracket_{t'+k-1}), \\
&(\llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}, [\mathbf{e}]_{n'-1}, [\mathbf{e}]_{k-1}) \\
&= \sum_{i=1}^M \lambda^{i-1} \cdot (\llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{k-1})
\end{aligned}$$

and output $([\vec{\mathbf{v}}_1]_{k-1}, [\vec{\mathbf{v}}_2]_{k-1}, [\vec{\mathbf{d}}]_{k-1}, \llbracket \vec{\mathbf{u}} \rrbracket_{t'+k-1}, \llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}, [\mathbf{e}]_{n'-1}, [\mathbf{e}]_{k-1})$.

Step Two: Dimension-Reduction. The second step reduces the dimension of a transcript for extended degree reduction by a factor of κ . To do this, the transcript is first partitioned into κ equal parts. We then compress the check of κ transcripts into one check of a newly generated transcript with the same dimension by establishing polynomials, as described in **COMPRESS**. The protocol for dimension reduction appears in **DIMENSION-REDUCTION**.

The cost of **COMPRESS**(M) is $\text{P2P}(O(M \cdot n)) + \text{BC}(O(\log |\mathbb{F}|))$. The cost of **DIMENSION-REDUCTION**(κ) is $\text{P2P}(O(\kappa \cdot n)) + \text{BC}(O(\log |\mathbb{F}|))$.

Protocol 9: COMPRESS(M)

Input: M transcripts $([\vec{v}_1^{(i)}]_{k-1}, [\vec{v}_2^{(i)}]_{k-1}, [\vec{d}^{(i)}]_{k-1}, \llbracket \vec{u}^{(i)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{k-1})_{i=1}^M$, each of dimension (ℓ, s) .

Output: A single transcript of dimension (ℓ, s) .

Ingredients: $M - 1$ random sharings of the form $\llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}$.

1. All parties locally compute $[\vec{V}_1(\cdot)]_{k-1}, [\vec{V}_2(\cdot)]_{k-1}, [\vec{D}(\cdot)]_{k-1}$, and $\llbracket \vec{U}(\cdot) \rrbracket_{t'+k-1}$, which are vectors of degree- $(M - 1)$ polynomials defined as for $i \in [M]$,

$$[\vec{V}_1(i)]_{k-1} = [\vec{v}_1^{(i)}]_{k-1}, [\vec{V}_2(i)]_{k-1} = [\vec{v}_2^{(i)}]_{k-1}, \\ [\vec{D}(i)]_{k-1} = [\vec{d}^{(i)}]_{k-1}, \llbracket \vec{U}(i) \rrbracket_{t'+k-1} = \llbracket \vec{u}^{(i)} \rrbracket_{t'+k-1}.$$

2. For $i \in \{M + 1, \dots, 2M - 1\}$, all parties compute $[\vec{v}_1^{(i)}]_{k-1} = [\vec{V}_1(i)]_{k-1}, [\vec{v}_2^{(i)}]_{k-1} = [\vec{V}_2(i)]_{k-1}, [\vec{d}^{(i)}]_{k-1} = [\vec{D}(i)]_{k-1}$, and $\llbracket \vec{u}^{(i)} \rrbracket_{t'+k-1} = \llbracket \vec{U}(i) \rrbracket_{t'+k-1}$.
3. For $i \in \{M + 1, \dots, 2M - 1\}$, all parties invoke **EXTEND-DEG-REDUCTION** on $[\vec{v}_1^{(i)}]_{k-1}, [\vec{v}_2^{(i)}]_{k-1}, [\vec{d}^{(i)}]_{k-1}, \llbracket \vec{u}^{(i)} \rrbracket_{t'+k-1}$ and obtain the transcript

$$([\vec{v}_1^{(i)}]_{k-1}, [\vec{v}_2^{(i)}]_{k-1}, [\vec{d}^{(i)}]_{k-1}, \llbracket \vec{u}^{(i)} \rrbracket_{t'+k-1}, \\ \llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{k-1}).$$

4. All parties locally compute $\llbracket \mathbf{R}(\cdot) \rrbracket_{t'+k-1}, [\mathbf{O}(\cdot)]_{n'-1}, [\mathbf{E}(\cdot)]_{n'-1}, [\mathbf{E}(\cdot)]_{k-1}$, which are vectors of degree- $2(M - 1)$ polynomials defined as for $i \in [2M - 1]$,

$$\llbracket \mathbf{R}(i) \rrbracket_{t'+k-1} = \llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}, [\mathbf{O}(i)]_{n'-1} = [\mathbf{o}^{(i)}]_{n'-1}, \\ [\mathbf{E}(i)]_{n'-1} = [\mathbf{e}^{(i)}]_{n'-1}, [\mathbf{E}(i)]_{k-1} = [\mathbf{e}^{(i)}]_{k-1}.$$

5. All parties invoke $\mathcal{F}_{\text{coin}}$ to generate a random element λ .
6. All parties locally compute and output

$$([\vec{V}_1(\lambda)]_{k-1}, [\vec{V}_2(\lambda)]_{k-1}, [\vec{D}(\lambda)]_{k-1}, \llbracket \vec{U}(\lambda) \rrbracket_{t'+k-1}, \\ \llbracket \mathbf{R}(\lambda) \rrbracket_{t'+k-1}, [\mathbf{O}(\lambda)]_{n'-1}, [\mathbf{E}(\lambda)]_{n'-1}, [\mathbf{E}(\lambda)]_{k-1}).$$

Protocol 10: DIMENSION-REDUCTION(κ)

Input: A transcript $([\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}, [\vec{d}]_{k-1}, \llbracket \vec{u} \rrbracket_{t'+k-1}, \llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}, [\mathbf{e}]_{n'-1}, [\mathbf{e}]_{k-1})$ of dimension (M, N) .

Output: A transcript of dimension $(M/\kappa, N/\kappa)$.

Ingredients: $2\kappa - 2$ random sharings of the form $\llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}$.

1. For each $[\vec{x}]_d \in \{[\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}, [\vec{d}]_{k-1}, \llbracket \vec{u} \rrbracket_{t'+k-1}\}$, all parties partition $[\vec{x}]_d$ into κ equal pieces as

$$[\vec{x}]_d = ([\mathbf{x}^{(1)}]_d, [\mathbf{x}^{(2)}]_d, \dots, [\mathbf{x}^{(\kappa)}]_d),$$

where $[\vec{v}_1^{(i)}]_{k-1}, [\vec{v}_2^{(i)}]_{k-1}$ have dimension M/κ and $[\vec{d}^{(i)}]_{k-1}, \llbracket \vec{u}^{(i)} \rrbracket_{t'+k-1}$ have dimension N/κ .

2. For $i \in [\kappa - 1]$, all parties invoke **EXTEND-DEG-REDUCTION** on $[\vec{v}_1^{(i)}]_{k-1}, [\vec{v}_2^{(i)}]_{k-1}, [\vec{d}^{(i)}]_{k-1}, \llbracket \vec{u}^{(i)} \rrbracket_{t'+k-1}$ and obtain the transcript

$$([\vec{v}_1^{(i)}]_{k-1}, [\vec{v}_2^{(i)}]_{k-1}, [\vec{d}^{(i)}]_{k-1}, \llbracket \vec{u}^{(i)} \rrbracket_{t'+k-1}, \\ \llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{k-1}).$$

3. All parties compute $(\llbracket \mathbf{r}^{(\kappa)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(\kappa)}]_{n'-1}, [\mathbf{e}^{(\kappa)}]_{n'-1}, [\mathbf{e}^{(\kappa)}]_{k-1})$ as

$$(\llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}, [\mathbf{e}]_{n'-1}, [\mathbf{e}]_{k-1}) \\ - \sum_{i=1}^{\kappa-1} (\llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{k-1}).$$

4. All parties invoke **COMPRESS**(κ) on

$$\{([\vec{v}_1^{(i)}]_{k-1}, [\vec{v}_2^{(i)}]_{k-1}, [\vec{d}^{(i)}]_{k-1}, \llbracket \vec{u}^{(i)} \rrbracket_{t'+k-1}, \\ \llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{k-1})\}_{i \in [\kappa]}.$$

The resulting transcript has dimension $(M/\kappa, N/\kappa)$.

Step Three: Recursion and Randomization. The second step is repeated $\log_{\kappa} \ell$ times so that all parties only need to verify the correctness of a single transcript for extended degree reduction in the end, where $\ell = \max\{\sum_{i=1}^M \ell_i, \sum_{i=1}^M s_i\}$. In the final iteration of the second step, an additional random transcript is incorporated to produce the final random virtual transcript. The randomization protocol is presented in **RANDOMIZATION**. The cost of **RANDOMIZATION**(s) is $\text{P2P}(O(s \cdot n)) + \text{BC}(O(\log |\mathbb{F}|))$.

Protocol 11: RANDOMIZATION(s)

Input: A transcript $([\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}, [\vec{d}]_{k-1}, \llbracket \vec{u} \rrbracket_{t'+k-1}, \llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}, [\mathbf{e}]_{n'-1}, [\mathbf{e}]_{k-1})$ of dimension (s, s) .

Output: A random transcript of dimension $(1, 1)$.

Ingredients: $2s + 1$ random sharings of the form $\llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}$.

1. **Preparing a random transcript.** All parties pick one random authenticated sharing $\llbracket \mathbf{u}^{(0)} \rrbracket_{t'+k-1}$, set $[\mathbf{v}_1^{(0)}]_{k-1}, [\mathbf{v}_2^{(0)}]_{k-1}$ as all-zero sharings, and set $[\mathbf{d}^{(0)}]_{k-1}$ as an all-one sharing. They then invoke **EXTEND-DEG-REDUCTION** on $[\mathbf{v}_1^{(0)}]_{k-1}, [\mathbf{v}_2^{(0)}]_{k-1}, [\mathbf{d}^{(0)}]_{k-1}, \llbracket \mathbf{u}^{(0)} \rrbracket_{t'+k-1}$ and obtain the transcript

$$([\mathbf{v}_1^{(0)}]_{k-1}, [\mathbf{v}_2^{(0)}]_{k-1}, [\mathbf{d}^{(0)}]_{k-1}, \llbracket \mathbf{u}^{(0)} \rrbracket_{t'+k-1}, \\ \llbracket \mathbf{r}^{(0)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(0)}]_{n'-1}, [\mathbf{e}^{(0)}]_{n'-1}, [\mathbf{e}^{(0)}]_{k-1}).$$

2. For each $[\vec{x}]_d \in \{[\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}, [\vec{d}]_{k-1}, \llbracket \vec{u} \rrbracket_{t'+k-1}\}$, all parties interpret $[\vec{x}]_d$ as

$$[\vec{x}]_d = ([\mathbf{x}^{(1)}]_d, \dots, [\mathbf{x}^{(s)}]_d).$$

3. For $i \in [s - 1]$, all parties invoke **EXTEND-DEG-REDUCTION** on $[\mathbf{v}_1^{(i)}]_{k-1}, [\mathbf{v}_2^{(i)}]_{k-1}, [\mathbf{d}^{(i)}]_{k-1}, \llbracket \mathbf{u}^{(i)} \rrbracket_{t'+k-1}$ and obtain the transcript

$$([\mathbf{v}_1^{(i)}]_{k-1}, [\mathbf{v}_2^{(i)}]_{k-1}, [\mathbf{d}^{(i)}]_{k-1}, \llbracket \mathbf{u}^{(i)} \rrbracket_{t'+k-1}, \\ \llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{k-1}).$$

4. All parties compute $(\llbracket \mathbf{r}^{(s)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(s)}]_{n'-1}, [\mathbf{e}^{(s)}]_{n'-1}, [\mathbf{e}^{(s)}]_{k-1})$ as

$$(\llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}, [\mathbf{e}]_{n'-1}, [\mathbf{e}]_{k-1}) - \sum_{i=1}^{s-1} (\llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{k-1}).$$

5. All parties invoke **COMPRESS** $(s+1)$ on

$$\{([\mathbf{v}_1^{(i)}]_{k-1}, [\mathbf{v}_2^{(i)}]_{k-1}, [\mathbf{d}^{(i)}]_{k-1}, \llbracket \mathbf{u}^{(i)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}, [\mathbf{o}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{n'-1}, [\mathbf{e}^{(i)}]_{k-1})\}_{i=0}^s.$$

The resulting transcript has dimension $(1, 1)$.

Combining three steps above, the recursive verification procedure is concluded in **VRFY-TRANSCRIPTS**. We also illustrate the structure of this procedure in Fig. 3 at the end of the document. The cost of **VRFY-TRANSCRIPTS** (ℓ) is $\text{P2P}(O(\kappa \cdot \log_{\kappa} \ell \cdot n + n^3 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(\log_{\kappa} \ell \cdot \log |\mathbb{F}| + n^3 \cdot \log |\mathbb{F}|))$ and two invocations of $\mathcal{F}_{\text{DSS}}$ with **Check**.

Protocol 12: VRFY-TRANSCRIPTS (ℓ)

Input: Multiple transcripts, of dimension (ℓ, ℓ) in total.

Output: ok / a dispute pair.

Ingredients: $2\kappa + 1 + (2\kappa - 2) \cdot \log_{\kappa} \ell$ random sharings of the form $\llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}$.

1. All parties invoke **DE-LINEARIZATION** to combine all transcripts.
2. All parties repeatedly invoke **DIMENSION-REDUCTION** (κ) until the dimension of the output transcript is reduced to (κ, κ) .
3. All parties invoke **RANDOMIZATION** (κ) to prepare a randomized transcript of dimension $(1, 1)$.
4. All parties invoke **VRFY-SINGLE-TRANSCRIPT** on the generated transcript to check its correctness.
 - If a new dispute pair E is identified, all parties take E as output.
 - Otherwise, all parties output ok.

5.2 Segment Computation for SIMD Circuits

After the computation has been verified, the parties need to authenticate the output sharings of the segment in order to preserve the invariant that provides verifiability of the sharings used in the later segments, as discussed in Section 2.2. In particular, for each output sharing $[\mathbf{z}]_{t'+k-1}$, the key observation implies that it can be decomposed as $[\mathbf{z}]_{t'+k-1} = [\mathbf{z}_0]_{k-1} + \llbracket \mathbf{z}_1 \rrbracket_{t'+k-1}$, where $[\mathbf{z}_0]_{k-1}$ is distributed by P_{king} . The parties therefore invoke the DSS functionality to authenticate $[\mathbf{z}_0]_{k-1}$. The protocol for computing a segment of a SIMD circuit is summarized in **COMP-SEG-SIMD**.

The cost of **COMP-SEG-SIMD** $(m \cdot k)$ to compute m groups of k multiplication gates is $\text{P2P}(O(m \cdot n + n^3 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^3 \cdot \log |\mathbb{F}|))$ and two invocations of $\mathcal{F}_{\text{DSS}}$ with **Check**.

Protocol 13: COMP-SEG-SIMD $(m \cdot k)$

Input: Input sharings of the form $\llbracket \mathbf{x} \rrbracket_{t'+k-1}$.

Output: Output sharings of the form $\llbracket \mathbf{z} \rrbracket_{t'+k-1}$ / a dispute pair.

- 1: All parties agree on a special party P_{king} .

- 2: All parties invoke **RAND**(Σ_2) to prepare N random packed degree- $(n' - 1)$ zero-sharings of the form $[\mathbf{o}]_{n'-1}$.
- 3: **Evaluation.** All parties evaluate the segment layer by layer.
 For each group of k gates with input sharings $[\mathbf{x}]_{t'+k-1}, [\mathbf{y}]_{t'+k-1}$,
 - for addition, compute $[\mathbf{z}]_{t'+k-1} \leftarrow [\mathbf{x}]_{t'+k-1} + [\mathbf{y}]_{t'+k-1}$.
 - for multiplication, invoke **MULT** to compute $[\mathbf{z}]_{t'+k-1}$ with $\mathbf{z} = \mathbf{x} * \mathbf{y}$.
- 4: **Verification.**
 1. All parties invoke **F_{vrify-linear-sharing}**(Σ_1) to check whether P_{king} distributed valid degree- $(k-1)$ packed Shamir sharings.
 2. All parties invoke **VRIFY-TRANSCRIPTS** with their transcripts generated in **MULT** to verify the correctness of degree reduction.
- 5: **Adding verifiability.** Let $[\mathbf{z}^{(1)}]_{t'+k-1}, \dots, [\mathbf{z}^{(m)}]_{t'+k-1}$ be the output sharings of the segment, where $[\mathbf{z}^{(\ell)}]_{t'+k-1} = [\mathbf{z}_0^{(\ell)}]_{k-1} + \llbracket \mathbf{z}_1^{(\ell)} \rrbracket_{t'+k-1}$ with $[\mathbf{z}_0^{(\ell)}]_{k-1}$ distributed by P_{king} . All parties invoke **F_{DSS}** with **Auth** to obtain $\llbracket \mathbf{z}_0^{(1)} \rrbracket_{k-1}, \dots, \llbracket \mathbf{z}_0^{(m)} \rrbracket_{k-1}$.
 - In case of failure, output the identified dispute pair.
 - Otherwise, output $\llbracket \mathbf{z}^{(1)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{z}^{(m)} \rrbracket_{t'+k-1}$.

To compute a SIMD circuit, in the preprocessing phase, parties invoke **F_{DSS}** with **Triple** to receive authenticated packed Beaver triples and invoke **F_{DSS}** with **(Share, P_i)** to let each P_i distribute random detectable secret sharings used for the input layers. In the online phase, for input layers, each P_i updates their real inputs by distributing low-degree packed Shamir sharings. For each computation segment, all parties invoke **COMP-SEG-SIMD** to evaluate the circuit. Finally, for the output layer, all parties invoke **F_{DSS}** with **Recon** to verifiably reconstruct the outputs to designated receivers. During the computation, if the number of remaining parties falls below $t' + 2k - 1$, all parties invoke the fallback protocol to restart the computation with a smaller corruption threshold, as outlined in Section 2.2. Combining the above construction with our instantiation of **F_{DSS}**, we derive an honest majority MPC protocol that computes SIMD circuits with GOD and consumes $O(|C|)$ elements of communication and $O(|C|)$ random OLE correlations across all pairs of parties per circuit copy, ignoring circuit-independent terms. To support general circuits, we follow [GPS21, GPS22] to handle network routing, which is formally demonstrated in Section D.1.

We further prepare OLE correlations either with an information-theoretic security, through OT extension protocols, or by leveraging efficient PCG constructions. We refer the reader to Section E for details.

References

- BCG⁺19. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round ot extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 291–308, New York, NY, USA, 2019. Association for Computing Machinery.
- BCG⁺20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudo-random correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 387–416, Cham, 2020. Springer International Publishing.
- BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 896–912, New York, NY, USA, 2018. Association for Computing Machinery.
- Bea89. Donald Beaver. Multiparty protocols tolerating half faulty processors. In *Conference on the Theory and Application of Cryptology*, pages 560–572. Springer, 1989.
- BFO12. Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 663–680, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- BH06. Zuzana Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 305–328, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

- Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.
- CDD⁺99. Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT ’99*, pages 311–326, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
- CP17. Ashish Choudhury and Arpita Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Transactions on Information Theory*, 63:428–468, 2017.
- DLN19. Ivan Damgård, Kasper Green Larsen, and Jesper Buus Nielsen. Communication lower bounds for statistically secure MPC, with or without preprocessing. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 61–84, Cham, 2019. Springer International Publishing.
- DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.
- GPS21. Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Unconditional communication-efficient MPC via hall’s marriage theorem. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 275–304, Cham, 2021. Springer International Publishing.
- GPS22. Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 3–32, Cham, 2022. Springer Nature Switzerland.
- GSZ20. Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority MPC. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 618–646, Cham, 2020. Springer International Publishing.
- IKNP03. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 145–161, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- IKP⁺16. Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 430–458, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
- KOS15. Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 724–741, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’16, page 830–842, New York, NY, USA, 2016. Association for Computing Machinery.
- LXY25. Zhe Li, Chaoping Xing, Yizhou Yao, and Chen Yuan. Efficient pseudorandom correlation generators for any finite field. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025*, pages 145–175, Cham, 2025. Springer Nature Switzerland.
- RB89. T. Rabin and Ben-Or, M. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC ’89, page 73–85, New York, NY, USA, 1989. Association for Computing Machinery.
- Roy22. Lawrence Roy. Softspokenot: Quieter OT extension from small-field silent VOLE in the minicrypt model. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part I*, page 657–687, Berlin, Heidelberg, 2022. Springer-Verlag.
- SY24. Yifan Song and Xiaxi Ye. Perfectly-secure MPC with constant online communication complexity. In *Theory of Cryptography Conference*, pages 329–361. Springer, 2024.
- SY25a. Yifan Song and Xiaxi Ye. Honest majority MPC with $\tilde{O}(|C|)$ communication in minicrypt. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025*, pages 453–483, Cham, 2025. Springer Nature Switzerland.
- SY25b. Yifan Song and Xiaxi Ye. Scalable secure multiparty computation with perfect security from preprocessing. Cryptology ePrint Archive, Paper 2025/1406, 2025.

A Additional Preliminary

A.1 UC Security

In UC security framework, there exists two worlds, the ideal world and the real world. A protocol UC-securely realizes an ideal functionality if the distributions of the two worlds are identical. The details are given as follows.

Real World. In real world, there are n parties, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$. $\mathcal{A}$ can corrupt at most $t < \frac{n}{2}$ parties and can fully control the behavior of the corrupted parties. The environment provides input for the honest parties, and communicate with the adversary $\mathcal{A}$. At the end of the execution, $\mathcal{Z}$ also receives the output of honest parties. We use interactive Turing machines with random tapes and possible inputs to model the n parties and the adversary $\mathcal{A}$.

At the end of the execution, $\mathcal{Z}$ outputs a bit 0 or 1 depending on the outputs of the honest parties and the information it received from the adversary $\mathcal{A}$. We define $\text{REAL}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, z, \bar{r})$ to be the random variable that represents the output of $\mathcal{Z}$ when the input of $\mathcal{Z}$ is z , and the random tape for the n parties, adversary $\mathcal{A}$ and environment $\mathcal{Z}$ are $\bar{r} = (r_1, r_2, \dots, r_n, r_{\mathcal{A}}, r_{\mathcal{Z}})$. When $\bar{r}$ is uniformly generated, we use $\text{REAL}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, z)$ to denote the output of $\mathcal{Z}$.

Ideal World. In ideal world, there are n parties, the ideal adversary (simulator) $\mathcal{S}$, the environment $\mathcal{Z}$, and the ideal functionality $\mathcal{F}$. $\mathcal{Z}$ provides input to the honest parties and interact with the ideal adversary $\mathcal{S}$. $\mathcal{S}$ provides corrupted parties' inputs to the ideal functionality $\mathcal{F}$, then all parties simply invoke the ideal functionality $\mathcal{F}$ on their inputs. After the execution of the functionality, $\mathcal{Z}$ obtains the output of honest parties, and $\mathcal{S}$ obtains the output of corrupted parties. Similar to the case in real world, at the end $\mathcal{Z}$ will output a bit, and we use $\text{IDEAL}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, z, \bar{r})$, to denote the output of $\mathcal{Z}$ when the random tape of $\mathcal{S}, \mathcal{Z}$ are $\bar{r} = (r_{\mathcal{S}}, r_{\mathcal{Z}})$ respectively. We use $\text{IDEAL}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, z)$ to denote the output of $\mathcal{Z}$ when $\bar{r}$ is uniformly generated.

Security Definition. We define protocol Π to t -securely realizes $\mathcal{F}$ if for any adversary $\mathcal{A}$ that controls at most t corrupted parties and any environment $\mathcal{Z}$, there exists simulator $\mathcal{S}$ such that

$$\text{REAL}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, z) \equiv \text{IDEAL}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, z).$$

We also define protocol Π to t -securely realizes $\mathcal{F}$ with statistical security if for any adversary $\mathcal{A}$ that controls at most t corrupted parties and any environment $\mathcal{Z}$, there exists simulator $\mathcal{S}$ such that

$$\text{REAL}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, z) \equiv_{\epsilon} \text{IDEAL}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, z).$$

A.2 Dispute Control Framework

We present in [DISPUTE-CONTROL](#) the procedure to track the set of active parties $\mathcal{P}_{\text{active}}$, the set of eliminated parties Corr , and the set of dispute pairs Disp .

Procedure 1:DISPUTE-CONTROL

Initialization. Set $\mathcal{P}_{\text{active}} \leftarrow \{1, \dots, n\}$, $\text{Corr} \leftarrow \emptyset$, and $\text{Disp} \leftarrow \emptyset$.

Update. On input a new dispute pair E , update as follows.

1. Update the set of dispute pairs.
 - If $|E| = 1$, say $E = \{P_i\}$, then set $\text{Disp} \leftarrow \text{Disp} \cup \{(i, j) \mid j \in [n]\}$.
 - If $|E| = 2$, set $\text{Disp} \leftarrow \text{Disp} \cup \{E\}$.
2. Detect corrupted parties.
 - a. Set $\mathcal{C}_{\text{new}} \leftarrow \emptyset$.
 - b. For $i \in \mathcal{P}_{\text{active}}$, if $|\{i' \in [n] \mid (i, i') \in \text{Disp}\}| > t$, set $\mathcal{C}_{\text{new}} \leftarrow \mathcal{C}_{\text{new}} \cup \{i\}$.
 - c. Set $\text{Corr} \leftarrow \text{Corr} \cup \mathcal{C}_{\text{new}}$ and $\mathcal{P}_{\text{active}} \leftarrow \mathcal{P}_{\text{active}} \setminus \mathcal{C}_{\text{new}}$.

A.3 Verifiable Secret Sharing Scheme

Our requirement for verifiable secret sharing is that both the long-term and short-term authentication keys used to authenticate packed Shamir sharings, as well as the circuit outputs, must always be successfully reconstructible. We summarize the verifiable secret sharing functionality as $\mathcal{F}_{\text{VSS}}$, which will be instantiated from $\mathcal{F}_{\text{base-VSS}}$ as introduced later.

Functionality 2: $\mathcal{F}_{\text{VSS}}$

Setup. On input **Setup**,

1. Receive the set of corrupted parties $\mathcal{C}$.
2. Initialize a database that stores the generated secret sharings and supports their subsequent retrieval. $\langle \cdot | j \rangle_t$ denotes a degree- t Shamir sharing stored by $\mathcal{F}_{\text{VSS}}$, with the secret placed at the j -th position. We omit 1 to write $\langle \cdot \rangle_t$ to denote the secret is placed at the first position.
3. Invoke **DISPUTE-CONTROL** to initialize $\mathcal{P}_{\text{active}}$, Corr , and Disp .

Update. On input a new dispute pair E , invoke **DISPUTE-CONTROL** to update $\mathcal{P}_{\text{active}}$, Corr , and Disp .

Instruction set.

1. On input (vShare, j, P_d) ,
 1. $\mathcal{F}_{\text{VSS}}$ receives m degree- t Shamir secret sharings with secrets $s_1, \dots, s_m$ from the dealer P_d .
 2. $\mathcal{F}_{\text{VSS}}$ waits for an instruction from the adversary.
 - Upon receiving **(reject, E)** such that $E \notin \text{Disp}$, output E to active parties.
 - Upon receiving **accept**, store the sharings $\langle s_1 | j \rangle_t, \dots, \langle s_m | j \rangle_t$ and distribute them to active parties.
2. On input (vRand, j, m) ,
 1. $\mathcal{F}_{\text{VSS}}$ receives from the adversary shares of active corrupted parties $\{u_{\ell, i}\}_{\ell \in [m], i \in \mathcal{C} \setminus \text{Corr}}$.
 2. For $\ell \in [m]$, $\mathcal{F}_{\text{VSS}}$ randomly samples $\langle r_\ell | j \rangle_t$ according to shares of active corrupted parties.
 3. $\mathcal{F}_{\text{VSS}}$ waits for an instruction from the adversary.
 - Upon receiving **(reject, E)** such that $E \notin \text{Disp}$, output E to active parties.
 - Upon receiving **accept**, store the sharings $\langle s_1 | j \rangle_t, \dots, \langle s_m | j \rangle_t$ and distribute them to active parties.
3. On input **LongTermKey**,
 1. $\mathcal{F}_{\text{VSS}}$ receives from the adversary shares of active corrupted parties $\{u_{\ell, i}\}_{\ell \in [2k-1], i \in \mathcal{C} \setminus \text{Corr}}$.
 2. $\mathcal{F}_{\text{VSS}}$ randomly samples $\mu_1, \dots, \mu_k$ and randomly samples $(\{\langle \mu_j | j \rangle_t\}_{j=1}^k, \{\langle \mu_j | 1 \rangle_t\}_{j=2}^k)$ according to the secrets and shares of active corrupted parties.
 3. $\mathcal{F}_{\text{VSS}}$ waits for an instruction from the adversary.
 - Upon receiving **(reject, E)** such that $E \notin \text{Disp}$, output E to active parties.
 - Upon receiving **accept**, store the sharings $\langle s_1 | j \rangle_t, \dots, \langle s_m | j \rangle_t$ and distribute them to active parties.
4. On receiving a request for a secret sharing $\langle s | j \rangle_t$, represented as a linear combination of degree- t Shamir sharings stored by $\mathcal{F}_{\text{VSS}}$, $\mathcal{F}_{\text{VSS}}$ computes $\langle s | j \rangle_t$ following the same linear combination and using the stored sharings, and reconstructs the secret s .
 - On input **(Open, j)**, $\mathcal{F}_{\text{VSS}}$ outputs s to all active parties.
 - On input **(PrivRecon, j, P_o)**, $\mathcal{F}_{\text{VSS}}$ outputs s to P_o .
 - On input **(CheckZero, j)**, $\mathcal{F}_{\text{VSS}}$ outputs **accept** if $s = 0$ and **reject** otherwise.

Instantiating Verifiable Secret Sharings. To realize $\mathcal{F}_{\text{VSS}}$, we first formalize the basic verifiable secret sharing functionality as $\mathcal{F}_{\text{base-VSS}}$ and instantiate it using the construction in [GSZ20]. The cost of invoking (vShare, j, P_d) to let the dealer P_d distribute m degree- t Shamir sharings is $\text{P2P}(O(m \cdot n + n^6)) + \text{BC}(O(n^3 \cdot \log |\mathbb{F}|))$, where the terms $\text{P2P}(O(n^6))$ and $\text{BC}(O(n^3 \cdot \log |\mathbb{F}|))$ only appear once for one setup of $\mathcal{F}_{\text{base-VSS}}$. The cost of (Open, j) to publicly reconstruct m degree- t Shamir sharings is $\text{P2P}(O(m \cdot n)) + \text{BC}(O(n \cdot \log |\mathbb{F}|))$. The cost of invoking (Mult, j) to compute m multiplications is $\text{P2P}(O(m \cdot n + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n \cdot \log |\mathbb{F}|))$.

We then sketch the realization of $\mathcal{F}_{\text{VSS}}$ from $\mathcal{F}_{\text{base-VSS}}$ and analyze the cost as below.

- To realize **vRand**, each party invokes **vShare** to distribute random sharings, which is followed by performing an extraction. The cost to prepare m random verifiable secret sharings is $\text{P2P}(O(m \cdot n + n^2)) + \text{BC}(O(n \cdot \log |\mathbb{F}|))$.
- To realize **LongTermKey**, each party invokes **vShare** to distribute random sharings. After checking the relation of the secrets by opening a random linear combination, all parties extract random long-term keys. The cost of preparing m groups of long-term keys is $\text{P2P}(O(m \cdot n^2 + n^3)) + \text{BC}(O(n \cdot \log |\mathbb{F}|))$.
- **PrivRecon** can be realized by first letting the output receiver deliver a random degree- t sharing $\langle r|j \rangle_t$ to mask the secret output and then all parties invoke **Open** to reveal the masked secret. The cost to privately reconstruct m secrets is $\text{P2P}(O(m \cdot n)) + \text{BC}(O(n \cdot \log |\mathbb{F}|))$.
- Finally, to instantiate **CheckZero**, all parties first invoke **Mult** to multiply the target sharing with a random degree- t sharing and open the secret of the result sharing, whose secret will not be zero except with negligible probability if the original secret is not zero. The cost to check m secrets is $\text{P2P}(O(m \cdot n + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n \cdot \log |\mathbb{F}|))$.

Functionality 3: $\mathcal{F}_{\text{base-VSS}}$

Receive the set of corrupted parties $\mathcal{C}$. **Set up** and **update** as $\mathcal{F}_{\text{VSS}}$.

$\langle \cdot | j \rangle_t$ denotes a degree- t Shamir sharing stored by $\mathcal{F}_{\text{base-VSS}}$, with the secret stored at the j -th position.

Instruction set.

1. On input (vShare, j, P_d) ,
 1. $\mathcal{F}_{\text{base-VSS}}$ receives $\langle s_1 | j \rangle_t, \dots, \langle s_m | j \rangle_t$ from the dealer P_d .
 2. $\mathcal{F}_{\text{VSS}}$ waits for an instruction from the adversary.
 - Upon receiving **(reject, E)** such that $E \notin \text{Disp}$, output E to active parties.
 - Upon receiving **accept**, store the sharings $\langle s_1 | j \rangle_t, \dots, \langle s_m | j \rangle_t$ and distribute them to active parties.
2. On receiving a request (Open, j) to publicly reconstruct a secret sharing $\langle s | j \rangle_t$, represented as a linear combination of degree- t Shamir sharings stored by $\mathcal{F}_{\text{base-VSS}}$, $\mathcal{F}_{\text{base-VSS}}$ computes $\langle s | j \rangle_t$ following the same linear combination and using the stored sharings, reconstructs the secret s , and outputs s to all active parties.
3. On receiving a request (Mult, j) to multiply two secret sharings $\langle s_1 | j \rangle_t$ and $\langle s_2 | j \rangle_t$, which are linear combinations of degree- t Shamir sharings stored by $\mathcal{F}_{\text{base-VSS}}$,
 1. $\mathcal{F}_{\text{base-DSS}}$ computes $\langle s_1 | j \rangle_t$ and $\langle s_2 | j \rangle_t$ following the corresponding linear combinations and using the stored sharings, reconstructs the secrets s_1 and s_2 , and computes $s \leftarrow s_1 \cdot s_2$.
 2. $\mathcal{F}_{\text{base-VSS}}$ receives from adversary the shares of active corrupted parties $\{u_i\}_{i \in \mathcal{C} \setminus \text{Corr}}$.
 3. $\mathcal{F}_{\text{base-VSS}}$ randomly samples $\langle s | j \rangle_t$ according to the secret and shares of active corrupted parties and then stores the sharing.

A.4 Packed Triple Generation from Pairwise OLE Correlations

We first present the functionality $\mathcal{F}_{\text{rand-OLE}}$, which provides random OLE correlations and is used for preparing packed triples. We emphasize that the setup between any two honest parties always succeeds, ensur-

ing that sufficiently many honest virtual parties participate in the triple extraction. The instantiation of $\mathcal{F}_{\text{rand-OLE}}$ is discussed in Section E.

Functionality 4: $\mathcal{F}_{\text{rand-OLE}}(M)$

Setup. On input **Setup**, $\mathcal{F}_{\text{rand-OLE}}$ receives the set of corrupted parties $\mathcal{C}$ and a public parameter M indicating the maximum number of required OLE correlations.

For each pair of parties (P_i, P_j) ,

- If both P_i and P_j are honest, initialize $c_{i,j} \leftarrow 0$.
- If P_i and P_j contain at least one corrupted party, $\mathcal{F}_{\text{rand-OLE}}$ waits for an instruction from the adversary.
 - On receiving **reject**, output $\{P_i, P_j\}$.
 - On receiving **accept**, initialize $c_{i,j} \leftarrow 0$.

Extend. On input **rand-OLE** from P_i and P_j ,

1. Set $c_{i,j} \leftarrow c_{i,j} + 1$. If $\sum_{i,j} c_{i,j} > M$, $\mathcal{F}_{\text{rand-OLE}}$ outputs **fail**.
2. $\mathcal{F}_{\text{rand-OLE}}$ provides a random OLE correlation to P_i and P_j as follows.
 - If both P_i and P_j are honest, $\mathcal{F}_{\text{rand-OLE}}$ randomly samples $a, b, x \in \mathbb{F}$ and computes $y \leftarrow a \cdot b - x$.
 - If both P_i and P_j are corrupted, $\mathcal{F}_{\text{rand-OLE}}$ receives $a, b, x \in \mathbb{F}$ from the adversary and computes $y \leftarrow a \cdot b - x$.
 - If exactly one of two parties is corrupted, say P_i , $\mathcal{F}_{\text{rand-OLE}}$ receives $a, x \in \mathbb{F}$ from the adversary, samples b , and computes $y \leftarrow a \cdot b - x$.

$\mathcal{F}_{\text{rand-OLE}}$ sends (a, x) and (b, y) to P_i and P_j , respectively.

We then present the functionality $\mathcal{F}_{\text{packed-triple}}$, which generates packed triples and can be realized following [SY25a] assuming random OLE correlations. In particular, to realize $\mathcal{F}_{\text{packed-triple}}$, we compile the protocol $\Pi_{\text{packed-triple}}$ from [SY25a], which runs among a set of real parties $\mathcal{P}$ and a set of virtual parties $\mathcal{V}$, where each virtual party is emulated by a pair of parties. In the protocol execution of $\Pi_{\text{packed-triple}}(\mathcal{P}, \mathcal{V})$, after each virtual party in $\mathcal{V}$ distributes packed triples, the parties in $\mathcal{P}$ perform triple extraction to extract random packed triples. It is guaranteed that if $\mathcal{P}$ satisfies an honest majority and strictly more than half of the virtual parties in $\mathcal{V}$ are honest, i.e., they are emulated by two parties that contain at least one honest party, then $\Pi_{\text{packed-triple}}$ prepares packed triples with malicious security with abort in the $\mathcal{F}_{\text{rand-OLE}}$ -hybrid model, and achieves an amortized cost of $O(1)$ elements of communication and $O(1)$ OLE correlations per secret. Based on this protocol, our compiler proceeds as follows, which invokes **VIRTUAL-PARTY** to determine the set of active virtual parties.

Procedure 2: **VIRTUAL-PARTY**

On input $(\text{Corr}, \text{Disp})$, determine the set of virtual parties $\mathcal{V}_{\text{active}}$:

1. Set $\mathcal{P}' \leftarrow [n]$.
2. For each $E \in \text{Disp}$, if $E \subseteq \mathcal{P}'$, set $\mathcal{P}' \leftarrow \mathcal{P}' \setminus E$.
3. For each $i \in \text{Corr}$, set $\mathcal{P}' \leftarrow \mathcal{P}' \setminus \{i\}$.
4. Set $\mathcal{V}_{\text{active}} \leftarrow \{(i', j') \mid i', j' \in \mathcal{P}'\}$.

Output $\mathcal{V}_{\text{active}}$.

Compiler: $\text{COMPILER}_{\text{id-abort}}$

On input $(\Pi_{\text{abort}}, \mathcal{P}_{\text{active}}, \text{Corr}, \text{Disp})$, define the protocol $\Pi_{\text{id-abort}}$:

1. Invoke **VIRTUAL-PARTY** with $(\text{Corr}, \text{Disp})$ to determine the set of virtual parties $\mathcal{V}_{\text{active}}$.
2. Assign relay parties: For each pair $(P_i, P_j) \subseteq \mathcal{P}_{\text{active}}$, if $(P_i, P_j) \in \text{Disp}$, assign a relay party $P_{i \leftrightarrow j} \in \mathcal{P}_{\text{active}}$ such that $(P_i, P_{i \leftrightarrow j}), (P_{i \leftrightarrow j}, P_j) \notin \text{Disp}$.
3. Run Π_{abort} among $(\mathcal{P}_{\text{active}}, \mathcal{V}_{\text{active}})$ and the assigned relay parties. In case of success, output the extracted packed triples as output. Otherwise, proceed to the next step.
4. Run **FAULT-LOCALIZATION** (P_s) among $\mathcal{P}_{\text{active}}$ with transcripts generated in Π_{abort} to identify and output a new dispute pair E , where s is the smallest identity of an active party.

Output $\Pi_{\text{id-abort}}$.

Crucially, the compiler ensures that any pair of parties jointly requesting OLE correlations or communicating with each other are not yet in conflict. As a result, applying $\text{COMPILER}_{\text{id-abort}}$ to $\Pi_{\text{packed-triple}}$ yields a protocol $\Pi_{\text{packed-triple-id-abort}}$ that either successfully prepares random packed triples or identifies a new dispute pair. According to the efficiency of $\Pi_{\text{packed-triple}}$, this realization of $\mathcal{F}_{\text{packed-triple}}(m)$ incurs $\text{P2P}(O(m \cdot n + n^3)) + \text{BC}(O(n \cdot \log |\mathbb{F}|))$ of communication plus $O(m \cdot n + n^3)$ random OLE correlations.

Functionality 5: $\mathcal{F}_{\text{packed-triple}}(m)$

1. $\mathcal{F}_{\text{packed-triple}}$ receives from all parties three sets $\mathcal{C}$, Corr and Disp , which denote the set of corrupted parties, the set of eliminated parties, and the set of dispute pairs, respectively.
2. $\mathcal{F}_{\text{packed-triple}}$ receives from the adversary active corrupted parties' shares $(u_i^{(\ell)}, v_i^{(\ell)}, w_i^{(\ell)})_{\ell \in [m], i \in \mathcal{C} \setminus \text{Corr}}$.
3. For $\ell \in [m]$, $\mathcal{F}_{\text{packed-triple}}$ randomly samples $\mathbf{a}^{(\ell)}, \mathbf{b}^{(\ell)} \in \mathbb{F}^k$, computes $\mathbf{c}^{(\ell)} \leftarrow \mathbf{a}^{(\ell)} * \mathbf{b}^{(\ell)}$, and randomly samples $([\mathbf{a}^{(\ell)}]_{t'+k-1}, [\mathbf{b}^{(\ell)}]_{t'+k-1}, [\mathbf{c}^{(\ell)}]_{t'+2k-2})$ according to the secrets and shares of active corrupted parties.
4. $\mathcal{F}_{\text{packed-triple}}$ waits for an instruction from the adversary.
 - On receiving **(reject, E)** such that $E \notin \text{Disp}$, output E to all active parties.
 - On receiving **accept**, distribute the packed Beaver triples to all active parties.

B Additional Instantiation of Base DSS

B.1 Authenticating Degree- $(k - 1)$ Packed Shamir Sharings

When authenticating degree- $(k - 1)$ packed sharings distributed by an active party, note that, unlike the sharing phase, where the share vector is distributed via a high-degree packed sharing to preserve secrecy, the share vector here is distributed via a low-degree packed sharing. Consequently, computing the authentication tags for these low-degree shares does not require packed Beaver triples, and thus OLE correlations.

We present the protocol for authenticating degree- $(k - 1)$ packed sharings in **AUTH**. The cost of **AUTH** to authenticate $m \cdot k$ degree- $(k - 1)$ packed sharings is $\text{P2P}(O(m \cdot n^2 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^2 \cdot \log |\mathbb{F}|))$, amounting to amortized $O(1)$ elements per secret when $k = \Omega(n)$.

Protocol 14: $\text{AUTH}(P_r, \gamma)$

Input. $[\mathbf{X}^{(1)}]_{k-1}, \dots, [\mathbf{X}^{(m)}]_{k-1}$, where $[\mathbf{X}^{(\ell)}]_{k-1} = (\sigma_1^{(\ell)}, \dots, \sigma_n^{(\ell)})$.

Output. For $\ell \in [m]$, $\eta(\sigma_i^{(\ell)})$ for each P_i / a dispute pair.

Ingredients. $(\{\langle \gamma_j | j \rangle_t\}_{j=1}^k, \{\langle \gamma_j | 1 \rangle_t\}_{j=2}^k)$.

- 1: **Distributing shares.** For $\ell \in [m]$ and $i \in \mathcal{P}_{\text{active}}$, P_r distributes $[\sigma_i^{(\ell)}]_{k-1}$ to all parties, where $\sigma_i^{(\ell)} \in \mathbb{F}^k$ denotes P_i 's share vector of $[\mathbf{X}^{(\ell)}]_{k-1}$.
- 2: **Preparing random secret keys and masks.**
 1. All parties invoke $\mathcal{F}_{\text{VSS}}$ with `LongTermKey` to prepare random long-term secret keys $(\{\langle \mu_j | j \rangle_t\}_{j=1}^k, \{\langle \mu_j | 1 \rangle_t\}_{j=2}^k)$.
 2. All parties invoke $\mathcal{F}_{\text{VSS}}$ with `(Rand, 1,  $m \cdot n'$ )` to prepare $m \cdot n'$ verifiable secret sharings with random secrets, used as $m \cdot n'$ secret short-term keys $\{\langle \delta(\sigma_i^{(\ell)}) | 1 \rangle_t\}_{\ell \in [m], i \in \mathcal{P}_{\text{active}}}$.
 3. All parties invoke $\text{RAND}(\Sigma_3, m \cdot n')$ to prepare $m \cdot n'$ degree- $(n' - 1)$ packed sharings of the form $[\mathbf{o}]_{n'-1}$ with secret $\mathbf{o} = (o_1, \dots, o_k)$ satisfying $o_1 + \dots + o_k = 0$, denoted as $\{[\mathbf{o}_i^{(\ell)}]_{n'-1}\}_{\ell \in [m], i \in \mathcal{P}_{\text{active}}}$.
- 3: **Computing tags.** All parties locally compute $[\mu]_{t+k-1} \leftarrow \sum_{j=1}^k [\mathbf{e}_j]_{k-1} \cdot \langle \mu_j | j \rangle_t$, where $\mathbf{e}_j \in \mathbb{F}^k$ denotes the j -th unit vector.
For each $i \in \mathcal{P}_{\text{active}}$,
 1. For $\ell \in [m]$, all parties locally compute

$$[\mathbf{y}_i^{(\ell)}]_{n'-1} \leftarrow [\sigma_i^{(\ell)}]_{k-1} \cdot [\mu]_{t+k-1} + \langle \delta(\sigma_i^{(\ell)}) | 1 \rangle_t \cdot [\mathbf{e}_1]_{k-1} + [\mathbf{o}_i^{(\ell)}]_{n'-1}$$

and send their shares to P_i .

2. P_i reconstructs the secret $\mathbf{y}_i^{(\ell)} = (y_{i,1}^{(\ell)}, \dots, y_{i,k}^{(\ell)})$ and computes $y_i^{(\ell)} \leftarrow y_{i,1}^{(\ell)} + \dots + y_{i,k}^{(\ell)}$.

4: **Verification.**

1. All parties invoke $\mathcal{F}_{\text{coin}}$ to generate a random field element λ .
2. **Check P_r .**
 - a. All parties compute and broadcast $[\sigma_i]_{k-1} \leftarrow \sum_{\ell=1}^m \lambda^{\ell-1} \cdot [\sigma_i^{(\ell)}]_{k-1}$ for each $i \in \mathcal{P}_{\text{active}}$ and $[\mathbf{X}]_{k-1} \leftarrow \sum_{\ell=1}^m \lambda^{\ell-1} \cdot [\mathbf{X}^{(\ell)}]_{k-1}$.
 - b. All parties check whether 1) The sharings $\{[\sigma_i]_{k-1}\}_{i \in \mathcal{P}_{\text{active}}}$ and $[\mathbf{X}]_{k-1}$ are valid degree- $(k-1)$ packed Shamir sharings and 2) σ_i corresponds to the i -th share vector of $[\mathbf{X}]_{k-1}$.
– If not, P_r locates a corrupt party P_i . All parties locate a dispute pair E via relay and take E as output.
3. **Verification of tag computation.**
 - a. Each active party P_i computes and broadcasts

$$y_i \leftarrow \sum_{\ell=1}^m \lambda^{\ell-1} \cdot y_i^{(\ell)}.$$

- b. For each active party P_i , all parties locally compute

$$\begin{aligned} \langle \delta_i | 1 \rangle_t &\leftarrow \sum_{\ell=1}^m \lambda^{\ell-1} \cdot \langle \delta(\sigma_i^{(\ell)}) | 1 \rangle_t, \\ \langle \mathbf{o}_i | 1 \rangle_t &\leftarrow y_i - \sigma_i \odot (\langle \mu_1 | 1 \rangle_t, \dots, \langle \mu_k | 1 \rangle_t) - \langle \delta_i | 1 \rangle_t. \end{aligned}$$

and invoke $\mathcal{F}_{\text{VSS}}$ with `(CheckZero, 1)` to check whether the secret of $\langle \mathbf{o}_i | 1 \rangle_t$ is 0.

– If not, all parties open their view to locate a dispute pair.

5: **Update authentication key.**

1. For $j \in [k]$, all parties invoke $\mathcal{F}_{\text{VSS}}$ with `(Open, 1)` to reveal the secret of $\langle \gamma_j - \mu_j | 1 \rangle_t$.
2. For $\ell \in [m]$, each party P_i locally computes $\eta(\sigma_i^{(\ell)}) \leftarrow y_i^{(\ell)} + \sigma_i^{(\ell)} \odot (\gamma - \mu)$.

B.2 Summary of the Base DSS

Now we summarize the instantiation of the basic functionality of detectable secret sharings. The base DSS functionality is parameterized by M , which indicates the maximum number of invocations of **Check**. To allow M invocations of **DSS-CHECK**, the parties need to invoke **RAND-TAG** to prepare $M \cdot k$ pairs of random messages with authentication tags for each party in advance.

We summarize the protocol that realizes the basic DSS functionality in **BASE-DSS** and state its security in Lemma 1.

Protocol 15: BASE-DSS(M)

Setup. On input **Setup**,

1. Invoke **DISPUTE-CONTROL** to initialize $\mathcal{P}_{\text{active}}$, Corr , and Disp .
2. Invoke $\mathcal{F}_{\text{rand-OLE}}$ and $\mathcal{F}_{\text{VSS}}$ with **Setup**.
3. All parties invoke $\mathcal{F}_{\text{VSS}}$ with **LongTermKey** to prepare long-term authentication keys $(\{\langle \gamma_j | j \rangle_t\}_{j=1}^k, \{\langle \gamma_j | 1 \rangle_t\}_{j=2}^k)$.
4. Initialize a database to store the generated sharings.
5. All parties prepare random masks used for M invocations of **DSS-CHECK**.
 - (a) All parties invoke **RAND-TAG**($\gamma, M \cdot k$) to prepare $M \cdot k$ pairs of random messages and the corresponding authentication tags of the form $(z, \eta(z))$, for each active party.
 - (b) In case of failure, all parties have identified a dispute pair E . After finishing **update**, go to Step 5.a.

Update. On input a new dispute pair E , invoke **DISPUTE-CONTROL** to update $\mathcal{P}_{\text{active}}$, Corr , and Disp .

- If $|\mathcal{P}_{\text{active}}| < t' + 2k - 1$, all parties **abort**.
- Otherwise, agree on a special party P_{king} and assign a relay party for each dispute pair: for each pair of active parties (P_i, P_j) , if $(P_i, P_j) \in \text{Disp}$, arrange a relay party $P_{i \leftrightarrow j}$ such that $P_{i \leftrightarrow j}$ does not conflict with any of P_i and P_j .

Instruction set.

1. On input **(Share, P_d)**, all parties invoke **DSS-SHARE**(P_d, γ) to let the dealer P_d distribute $[x_1]_{t'+k-1}, \dots, [x_N]_{t'+k-1}$ and store the sharings. In case of failure, go to **update**.
2. On input **(Auth, P_r)**, all parties invoke **AUTH**(P_r, γ) to authenticate their shares of $[x_1]_{k-1}, \dots, [x_N]_{k-1}$, which are received from P_r . In case of failure, go to **update**.
3. On input **Check**, each active party pick k unused pairs of a random message and its authentication tag prepared in Step 5 of **setup** and use them as random masks. All parties then invoke **DSS-CHECK**($\gamma, \llbracket w \rrbracket_{t'+k-1}$) to check the correctness of $\llbracket w \rrbracket_{t'+k-1}$. In case of failure, go to **update**.

Lemma 1. *BASE-DSS computes $\mathcal{F}_{\text{base-DSS}}$ with statistical security against any fully malicious adversary controlling up to t parties in the $\mathcal{F}_{\text{VSS}}$, $\mathcal{F}_{\text{packed-triple}}$, $\mathcal{F}_{\text{coin-hybrid}}$ model.*

For the proof of Lemma 1, we refer the reader to Section H.1.

C Detailed Instantiation of Extended DSS

As outlined in Section 4.3, we further extend the basic DSS functionality in order to present the online computation in a structural manner. In this section, we demonstrate the detailed realization for this extended functionality.

C.1 Functionality for Extended Detectable Secret Sharings

We formally show the extended functionality in $\mathcal{F}_{\text{DSS}}$.

Functionality 6: $\mathcal{F}_{\text{DSS}}(M_1, M_2)$

Setup. On input **Setup**,

1. Receive the set of corrupted parties $\mathcal{C}$.
2. Receive public parameters M_1 and M_2^a , and initialize $\text{cnt} \leftarrow 0$.
3. Initialize a database that stores the generated secret sharings and support their subsequent retrieval. $\llbracket \cdot \rrbracket_d$ denotes a degree- d packed Shamir sharing that has been stored by $\mathcal{F}_{\text{DSS}}$.
4. Invoke **DISPUTE-CONTROL** to initialize $\mathcal{P}_{\text{active}}$, Corr , and Disp .
5. $\mathcal{F}_{\text{DSS}}$ waits for an instruction from the adversary.
 - Upon receiving **reject** and $\mathcal{C}' \subseteq \mathcal{C}$ such that $|\mathcal{C}'| = t - 2k + 2$, $\mathcal{F}_{\text{DSS}}$ outputs **abort** and $\mathcal{C}'$.
 - Upon receiving **accept**, $\mathcal{F}_{\text{DSS}}$ sets up successfully.

Update. On input a new dispute pair E , invoke **DISPUTE-CONTROL** to update $\mathcal{P}_{\text{active}}$, Corr , and Disp . If $\mathcal{P}_{\text{active}} < t' + 2k - 1$, $\mathcal{F}_{\text{DSS}}$ **aborts**.

Instruction Set.^b

1. On input **(Triple, m)**,
 1. $\mathcal{F}_{\text{DSS}}$ receives active corrupted parties' shares $\{u_{\ell,i}, v_{\ell,i}, w_{\ell,i}\}_{\ell \in [m], i \in \mathcal{C} \setminus \mathcal{C}_{\text{corr}}}$ from the adversary.
 2. For $\ell \in [m]$, $\mathcal{F}_{\text{DSS}}$ randomly samples $\mathbf{a}_\ell, \mathbf{b}_\ell \in \mathbb{F}^k$, computes $\mathbf{c}_\ell = \mathbf{a}_\ell * \mathbf{b}_\ell$, and randomly samples $(\llbracket \mathbf{a}_\ell \rrbracket_{t'+k-1}, \llbracket \mathbf{b}_\ell \rrbracket_{t'+k-1}, \llbracket \mathbf{c}_\ell \rrbracket_{t'+k-1})$ according to the secrets and shares of active corrupted parties.
 3. $\mathcal{F}_{\text{DSS}}$ waits for an instruction from the adversary.
 - Upon receiving **(reject, E)** such that $E \notin \text{Disp}$, output E to active parties.
 - Upon receiving **accept**, store the packed Beaver triples and distribute them to active parties.
2. On input **(Transpose, m)**,
 1. $\mathcal{F}_{\text{DSS}}$ receives active corrupted parties' shares $\{u_{j,i}^{(\ell)}, v_{j,i}^{(\ell)}\}_{\ell \in [m], j \in [k], i \in \mathcal{C} \setminus \mathcal{C}_{\text{corr}}}$ from the adversary.
 2. For $\ell \in [m], j \in [k]$, $\mathcal{F}_{\text{DSS}}$ randomly samples $\mathbf{r}_{*j}^{(\ell)} = (r_{j1}^{(\ell)}, \dots, r_{jk}^{(\ell)}) \in \mathbb{F}^k$. Let $\mathbf{R}^{(\ell)} = (r_{ij}^{(\ell)})_{ij} \in \mathbb{F}^{k \times k}$ and $\mathbf{r}_{*j}^{(\ell)} = (r_{1j}^{(\ell)}, \dots, r_{kj}^{(\ell)}) \in \mathbb{F}^k$ be the j -th column of $\mathbf{R}^{(\ell)}$ for $j \in [k]$. Then $\mathcal{F}_{\text{DSS}}$ randomly samples $(\{\llbracket \mathbf{r}_{*j}^{(\ell)} \rrbracket_{t'+k-1} \}_{j=1}^k, \{\llbracket \mathbf{r}_{*j}^{(\ell)} \rrbracket_{t'+k-1} \}_{j=1}^k)$ according to the secrets and shares of active corrupted parties.
 3. $\mathcal{F}_{\text{DSS}}$ waits for an instruction from the adversary.
 - Upon receiving **(reject, E)** such that $E \notin \text{Disp}$, output E to active parties.
 - Upon receiving **accept**, store the random linear masks and distribute them to active parties.
3. On receiving a request **Recon** to privately reconstruct M_2 packed Shamir sharings $\llbracket \mathbf{x}_1 \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{x}_{M_2} \rrbracket_{t'+k-1}$, which are linear combinations of packed Shamir sharings stored by $\mathcal{F}_{\text{DSS}}$,
 1. $\mathcal{F}_{\text{DSS}}$ computes $\llbracket \mathbf{x}_1 \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{x}_{M_2} \rrbracket_{t'+k-1}$ following the same linear combinations and using the stored sharings, reconstructs the secrets $\mathbf{x}_1, \dots, \mathbf{x}_{M_2}$, and sends shares of corrupted parties to the adversary.
 2. $\mathcal{F}_{\text{DSS}}$ waits for an instruction from the adversary.
 - Upon receiving **reject** and $\mathcal{C}' \subseteq \mathcal{C}$ such that $|\mathcal{C}'| = t - 2k + 2$, $\mathcal{F}_{\text{DSS}}$ outputs **abort** and $\mathcal{C}'$.

– Upon receiving **confirm**, output each of the secrets $x_{\ell,j}$ to the designated receiver.

^a M_1 and M_2 specify the maximum number of invocations of **Check** and the maximum number of detectable secret sharings to be reconstructed, respectively.

^b $\mathcal{F}_{\text{DSS}}$ also provides interfaces corresponding to the instructions **Share**, **Auth**, and **Check**, as defined in $\mathcal{F}_{\text{base-DSS}}$. We omit their description here due to space limitation.

C.2 Authenticated Packed Beaver Triples

To generate authenticated packed Beaver triples, we adapt the packed triple extraction protocol in [SY25a]. At a high level, virtual parties emulated by different pairs of active parties first invoke the OLE functionality to obtain random OLE correlations, and then invoke $\mathcal{F}_{\text{base-DSS}}$ to distribute packed Beaver triples authenticated as detectable secret sharings. All parties then perform packed triple extraction to extract a 1/4 fraction of random authenticated packed triples, where triple verification is conducted to ensure the output packed triples satisfy the multiplication relations.

As mentioned on Section A.4, we need to make the dispute control framework and the triple extraction machinery compatible. For this reason, parties invoke **VIRTUAL-PARTY** to determine a set of virtual parties $\mathcal{V}_{\text{active}}$, satisfying that

- every virtual party is not emulated by a dispute pair;
- there are a strictly greater than 1/2 fraction of virtual parties are honest, i.e., emulated by two parties containing at least one honest party.

We present the protocol for preparing authenticated random packed Beaver triple in **AUTH-PACKED-TRIPLES**, which invokes **AUTH-DEG-REDUCTION** to perform degree reduction, and state its security property in Lemma 2.

The cost of invoking **AUTH-DEG-REDUCTION** to perform m degree reductions is $\text{P2P}(O(m \cdot n + n^3 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^3 \cdot \log |\mathbb{F}|))$ plus $O(m \cdot n + n^3)$ OLEs. The cost of **AUTH-PACKED-TRIPLES** to prepare m authenticated packed Beaver triples is $\text{P2P}(O(m \cdot n + n^3 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^3 \cdot \log |\mathbb{F}|))$, $O(m \cdot n + n^3)$ OLEs, and 3 invocations of $\mathcal{F}_{\text{base-DSS}}$ with **Check**, which amounts to amortized $O(1)$ elements plus $O(1)$ OLEs per secret.

Protocol 16: AUTH-PACKED-TRIPLES($\mathcal{V}_{\text{active}}, m$)

Output. $\{\llbracket \mathbf{a}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{b}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{c}^{(\ell)} \rrbracket_{t'+k-1}\}_{\ell=1}^m$ / a dispute pair.

Let N denote the number of active virtual parties, i.e., $N = |\mathcal{V}_{\text{active}}|$. Let $\alpha_1, \dots, \alpha_{4m+3}$ be $4m+3$ distinct field elements such that $\alpha_\ell \notin \{1, 2, \dots, m\}$ for $\ell \in [4m+3]$.

1: **Packed triple distribution.** N virtual parties distribute $4m+3$ packed Beaver triples.

1. For $\ell \in [4m+3]$, supposing the ℓ -th packed triple is distributed by the virtual party emulated by P_{ℓ_1} and P_{ℓ_2} , P_{ℓ_1} and P_{ℓ_2} invoke $\mathcal{F}_{\text{rand-OLE}}$ to receive $2k$ OLE correlations, $\mathbf{a}_\ell^1 * \mathbf{b}_\ell^2 = \mathbf{s}_\ell^1 + \mathbf{s}_\ell^2$ and $\mathbf{b}_\ell^1 * \mathbf{a}_\ell^2 = \mathbf{t}_\ell^1 + \mathbf{t}_\ell^2$, where $\mathbf{a}_\ell^1, \mathbf{b}_\ell^1, \mathbf{s}_\ell^1, \mathbf{t}_\ell^1 \in \mathbb{F}^k$ are maintained by P_{ℓ_1} while $\mathbf{a}_\ell^2, \mathbf{b}_\ell^2, \mathbf{s}_\ell^2, \mathbf{t}_\ell^2 \in \mathbb{F}^k$ are maintained by P_{ℓ_2} .
2. For each P_i , let $I_i := \{(\ell, \sigma) \mid \ell_\sigma = i, \ell \in [4m+3], \sigma = 1, 2\}$. For each $(\ell, \sigma) \in I_i$, P_i randomly samples $[\mathbf{a}_\ell^\sigma]_{t'+k-1}, [\mathbf{b}_\ell^\sigma]_{t'+k-1}, [\mathbf{w}_\ell^\sigma]_{t'+k-1} := [\mathbf{a}_\ell^\sigma * \mathbf{b}_\ell^\sigma + \mathbf{s}_\ell^\sigma + \mathbf{t}_\ell^\sigma]_{t'+k-1}$. Then all parties invoke $\mathcal{F}_{\text{base-DSS}}$ with (**Share**, P_i) to let P_i distribute $\{[\mathbf{a}_\ell^\sigma]_{t'+k-1}, [\mathbf{b}_\ell^\sigma]_{t'+k-1}, [\mathbf{w}_\ell^\sigma]_{t'+k-1}\}_{\ell \in I_i}$.
3. For each $\ell \in [4m+3]$, all parties locally compute $[\mathbf{a}_\ell]_{t'+k-1} \leftarrow [\mathbf{a}_\ell^1]_{t'+k-1} + [\mathbf{a}_\ell^2]_{t'+k-1}$, $[\mathbf{b}_\ell]_{t'+k-1} \leftarrow [\mathbf{b}_\ell^1]_{t'+k-1} + [\mathbf{b}_\ell^2]_{t'+k-1}$, and $[\mathbf{c}_\ell]_{t'+k-1} \leftarrow [\mathbf{w}_\ell^1]_{t'+k-1} + [\mathbf{w}_\ell^2]_{t'+k-1}$.

2: **Packed triple extraction.** All parties establish polynomials and perform packed triple extraction.

1. All parties set two vectors $\mathbf{A}, \mathbf{B}$ of k polynomials of degree $2m+1$ such that $\llbracket \mathbf{A}(\alpha_\ell) \rrbracket_{t'+k-1} = \llbracket \mathbf{a}_\ell \rrbracket_{t'+k-1}$ and $\llbracket \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1} = \llbracket \mathbf{b}_\ell \rrbracket_{t'+k-1}$ for $\ell \in [2m+2]$. All parties locally compute $\llbracket \mathbf{A}(\alpha_\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1}$ for all $\ell \in \{2m+3, \dots, 4m+3\}$.
2. For $\ell \in \{2m+3, \dots, 4m+3\}$, all parties invoke **BEAVER** with $\llbracket \mathbf{A}(\alpha_\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1}$ and the packed Beaver triple $(\llbracket \mathbf{a}_\ell \rrbracket_{t'+k-1}, \llbracket \mathbf{b}_\ell \rrbracket_{t'+k-1}, \llbracket \mathbf{c}_\ell \rrbracket_{t'+k-1})$ to compute $\llbracket \mathbf{A}(\alpha_\ell) * \mathbf{B}(\alpha_\ell) \rrbracket_{t'+2k-2}$.
3. For $\ell \in \{2m+3, \dots, 4m+2\}$, all parties invoke **AUTH-DEG-REDUCTION** with $\llbracket \mathbf{A}(\alpha_\ell) * \mathbf{B}(\alpha_\ell) \rrbracket_{t'+2k-2}$ to compute $\llbracket \mathbf{A}(\alpha_\ell) * \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1}$.
4. All parties set a vector $\mathbf{C}$ of k polynomials of degree $4m+2$ such that $\llbracket \mathbf{C}(\alpha_\ell) \rrbracket_{t'+k-1} = \llbracket \mathbf{c}_\ell \rrbracket_{t'+k-1}$ for $\ell \in [2m+2]$ and $\llbracket \mathbf{C}(\alpha_\ell) \rrbracket_{t'+k-1} = \llbracket \mathbf{A}(\alpha_\ell) * \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1}$ for $\ell \in \{2m+3, \dots, 4m+3\}$.
- 3: **Triple Verification.** All parties verify whether the constructed vectors of polynomials $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ satisfy $\mathbf{A} * \mathbf{B} = \mathbf{C}$.
 1. All parties invoke $\mathcal{F}_{\text{coin}}$ to generate a random field element r . If $r \in [m]$, all parties abort.
 2. All parties compute and broadcast $\llbracket \mathbf{A}(r) \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(r) \rrbracket_{t'+k-1}, \llbracket \mathbf{C}(r) \rrbracket_{t'+k-1}$.
 3. All parties verify as follows.
 - All parties invoke $\mathcal{F}_{\text{base-DSS}}$ with **Check** to check the consistency of $\llbracket \mathbf{A}(r) \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(r) \rrbracket_{t'+k-1}, \llbracket \mathbf{C}(r) \rrbracket_{t'+k-1}$.
 - All parties reconstruct the secrets $\mathbf{A}(r), \mathbf{B}(r), \mathbf{C}(r)$ and check whether $\mathbf{A}(r) * \mathbf{B}(r) = \mathbf{C}(r)$.

If any check fails, P_i broadcasts $(i, \text{complain})$. Otherwise, P_i broadcasts (i, accept) .

- 4. If there are at least $t+1$ parties accepting the verification, all parties accept. Otherwise, all parties invoke **FAULT-LOCALIZATION**(P_i) to locate a new dispute pair, where i is the smallest identity of the complaining parties.
- 4: **Output.** For $\ell \in [m]$, all parties compute

$$(\llbracket \mathbf{a}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{b}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{c}^{(\ell)} \rrbracket_{t'+k-1}) \\ \leftarrow (\llbracket \mathbf{A}(\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{C}(\ell) \rrbracket_{t'+k-1}).$$

^a We let each virtual party distribute an equal amount of $\lceil \frac{4m+3}{N} \rceil$ packed triples. For example, $(\ell_1, \ell_2) = (i_{\lceil \frac{\ell}{n''} \rceil}, i_{(\ell \bmod n'') + 1})$, where $N := (n'')^2$ virtual parties consist of all different pairs of n'' parties, $P_{i_1}, \dots, P_{i_{n''}}$.

Protocol 17: AUTH-DEG-REDUCTION

Inputs. $[z_1]_{t'+2k-2}, \dots, [z_m]_{t'+2k-2}$.

Outputs. $\llbracket z_1 \rrbracket_{t'+k-1}, \dots, \llbracket z_m \rrbracket_{t'+k-1}$ / a dispute pair.

1. All parties invoke **RAND**(Σ_4) to prepare $\llbracket \mathbf{u}_1 \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{u}_m \rrbracket_{t'+k-1}$. All parties invoke **RAND**(Σ_2) to prepare $\llbracket \mathbf{o}_1 \rrbracket_{n'-1}, \dots, \llbracket \mathbf{o}_m \rrbracket_{n'-1}$.
2. For $\ell \in [m]$, all parties locally compute $[z_\ell + \mathbf{u}_\ell]_{n'-1} \leftarrow [z_\ell]_{t'+2k-2} + \llbracket \mathbf{u}_\ell \rrbracket_{t'+k-1} + \llbracket \mathbf{o}_\ell \rrbracket_{n'-1}$ and send it to P_{king} .
3. For $\ell \in [m]$, P_{king} reconstructs the secret $z_\ell + \mathbf{u}_\ell$ and distributes $[z_\ell + \mathbf{u}_\ell]_{k-1}$.
4. All parties invoke $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_1)$ to check whether P_{king} distributed valid degree- $(k-1)$ packed Shamir sharings.
5. All parties invoke $\mathcal{F}_{\text{base-DSS}}$ with **Auth** to obtain $\llbracket z_1 + \mathbf{u}_1 \rrbracket_{k-1}, \dots, \llbracket z_m + \mathbf{u}_m \rrbracket_{k-1}$.
6. For $\ell \in [m]$, all parties locally compute $\llbracket z_\ell \rrbracket_{t'+k-1} \leftarrow \llbracket z_\ell + \mathbf{u}_\ell \rrbracket_{k-1} - \llbracket \mathbf{u}_\ell \rrbracket_{t'+k-1}$.

Lemma 2. ***AUTH-PACKED-TRIPLES** computes Subfunctionality 1. of $\mathcal{F}_{\text{DSS}}$, labeled (Triple, m) , with statistical security against any fully malicious adversary controlling up to t parties in the $\mathcal{F}_{\text{base-DSS}}, \mathcal{F}_{\text{rand-OLE}}, \mathcal{F}_{\text{vrfy-linear-sharing}}, \mathcal{F}_{\text{coin-hybrid}}$ model.*

For the proof of Lemma 2, we refer the reader to Section H.2.

C.3 Authenticated Linear Masks for Transpose

The authenticated linear masks for transpose operation is a tuple of $2k$ authenticated packed Shamir sharings $(\{\llbracket \mathbf{r}_{\ell*} \rrbracket_{t'+k-1}\}_{\ell \in [k]}, \{\llbracket \mathbf{r}_{*\ell} \rrbracket_{t'+k-1}\}_{\ell \in [k]})$, where $\mathbf{r}_{\ell*}$ and $\mathbf{r}_{*\ell}$ denote the ℓ -th row and the ℓ -th column of the secret matrix $\mathbf{R}$, respectively, i.e.,

$$\mathbf{R} = \begin{pmatrix} - & \mathbf{r}_{1*} & - \\ & \vdots & \\ - & \mathbf{r}_{k*} & - \end{pmatrix} = \begin{pmatrix} | & & | \\ \mathbf{r}_{*1} & \dots & \mathbf{r}_{*k} \\ | & & | \end{pmatrix}.$$

Such an authenticated linear mask enables the parties to perform a linear transformation $\mathbf{L}$ to the secrets, allowing them to compute shares of $[\mathbf{L}(\mathbf{x})]_{t'+k-1}$ from $[\mathbf{x}]_{t'+k-1}$, as detailed in Section D.1.

We present the protocol for preparing authenticated linear masks for transpose in **AUTH-TRANSPOSE** and state its security in Lemma 3. The cost of **AUTH-TRANSPOSE** to prepare m authenticated linear masks for transpose is $\text{P2P}(O(m \cdot n^2 + n^4 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^4 \cdot \log |\mathbb{F}|))$, $O(m \cdot n^2 + n^3)$ OLEs, and $2k$ invocations of $\mathcal{F}_{\text{base-DSS}}$ with **Check**, which amounts to amortized $O(1)$ elements plus $O(1)$ OLEs per secret.

Protocol 18: AUTH-TRANSPOSE(m)

Output: m tuples of random sharings of the form $(\{\llbracket \mathbf{r}_{\ell*} \rrbracket_{t'+k-1}\}_{\ell \in [k]}, \{\llbracket \mathbf{r}_{*\ell} \rrbracket_{t'+k-1}\}_{\ell \in [k]})$ / a dispute pair.

- 1: **Extraction.** Let $N = \lceil \frac{m+1}{n-t} \rceil$. All parties agree on a $(n-t) \times n$ Vandermonde matrix $\mathbf{M}$.
 1. For each $i \in \mathcal{P}_{\text{active}}$, all parties invoke $\mathcal{F}_{\text{base-DSS}}$ with (Share, P_i) to let P_i distribute $\{\llbracket \mathbf{s}_{i,1*}^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{s}_{i,k*}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{s}_{i,*1}^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{s}_{i,*k}^{(\ell)} \rrbracket_{t'+k-1}\}_{\ell \in [N]}$. For each $i \in \text{Corr}$, all parties take an all-zero sharing.
 2. For $\ell \in [N], j \in [k]$, all parties locally compute

$$\begin{aligned} & (\llbracket \mathbf{r}_{1,j*}^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_{n-t,j*}^{(\ell)} \rrbracket_{t'+k-1})^\top \\ & \leftarrow \mathbf{M} \cdot (\llbracket \mathbf{s}_{1,j*}^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{s}_{n,j*}^{(\ell)} \rrbracket_{t'+k-1})^\top, \\ & (\llbracket \mathbf{r}_{1,*j}^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_{n-t,*j}^{(\ell)} \rrbracket_{t'+k-1})^\top \\ & \leftarrow \mathbf{M} \cdot (\llbracket \mathbf{s}_{1,*j}^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{s}_{n,*j}^{(\ell)} \rrbracket_{t'+k-1})^\top. \end{aligned}$$

2: Verification.

1. All parties invoke $\mathcal{F}_{\text{coin}}$ to generate a random element λ .
2. For $j \in [k]$, all parties locally compute and broadcast

$$\begin{aligned} \llbracket \mathbf{r}_{j*} \rrbracket_{t'+k-1} & \leftarrow \sum_{\ell=1}^N \sum_{i=1}^{n-t} \lambda^{(\ell-1) \cdot (n-t) + i - 1} \cdot \llbracket \mathbf{r}_{i,j*}^{(\ell)} \rrbracket_{t'+k-1}, \\ \llbracket \mathbf{r}_{*j} \rrbracket_{t'+k-1} & \leftarrow \sum_{\ell=1}^N \sum_{i=1}^{n-t} \lambda^{(\ell-1) \cdot (n-t) + i - 1} \cdot \llbracket \mathbf{r}_{i,*j}^{(\ell)} \rrbracket_{t'+k-1}. \end{aligned}$$

3. All parties invoke $\mathcal{F}_{\text{base-DSS}}$ with **Check** to check the correctness of $\llbracket \mathbf{r}_{1*} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_{k*} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{*1} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_{*k} \rrbracket_{t'+k-1}$.
 - If $\mathcal{F}_{\text{base-DSS}}$ outputs $(i, \text{corrupt})$, output $E \leftarrow \{P_i\}$.

4. All parties reconstruct the secrets $\mathbf{r}_{1*}, \dots, \mathbf{r}_{k*}$ and $\mathbf{r}_{*1}, \dots, \mathbf{r}_{*k}$ and check whether they satisfy the transpose relation, i.e.,

$$\begin{pmatrix} -\mathbf{r}_{1*} & - \\ \vdots & \\ -\mathbf{r}_{k*} & - \end{pmatrix} = \begin{pmatrix} | & & | \\ \mathbf{r}_{*1} & \cdots & \mathbf{r}_{*k} \\ | & & | \end{pmatrix}.$$

If not, P_i broadcasts $(i, \text{complain})$. Otherwise, P_i broadcasts **accept**.

5. If there are at least $t + 1$ parties accepting the verification, all parties accept. Otherwise, all parties invoke **FAULT-LOCALIZATION**(P_i) to locate a new dispute pair, where i is the smallest identity of the complaining parties.

3: Output. For $\ell \in [N], i \in [n - t]$, all parties output $(\llbracket \mathbf{r}_{j,1*}^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_{j,k*}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{j,*1}^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_{j,*k}^{(\ell)} \rrbracket_{t'+k-1})$ except for the one labeled with $\ell = 1, i = 1$.

Lemma 3. ***Auth-Transpose** computes Subfunctionality 2. of $\mathcal{F}_{DSS}$, labeled $(\text{Transpose}, m)$, with statistical security against any fully malicious adversary controlling up to t parties in the $\mathcal{F}_{\text{base-DSS}}, \mathcal{F}_{\text{coin}}$ -hybrid model.*

For the proof of Lemma 3, we refer the reader to Section H.3.

C.4 Reconstructing Detectable Secret Sharings

To support the output phase, the extended functionality provides an interface for reconstructing secrets to designated receivers, with a guarantee that once it receives a **confirm** instruction from the adversary, the reconstruction must succeed. To achieve this, parties first depack vectors of secrets shared via detectable secret sharings into verifiable secret sharings, each storing a single secret. After all detectable secret sharing have been depacked, successful reconstruction follows from the property of verifiable secret sharings.

We present the protocol for reconstructing detectable secret sharings in **DSS-RECON**, which invokes **RAND-DEPACK** to prepare random masks for depack, and state its security in Lemma 4. The cost of **RAND-DEPACK** to prepare m random masks for depack is $\text{P2P}(O(m \cdot n^2 + n^4 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^3 \cdot \log |\mathbb{F}|))$, $O(m \cdot n + n^3)$ OLEs, and one invocation of $\mathcal{F}_{\text{base-DSS}}$ with **Check**. The cost of **DSS-RECON** to reconstruct M detectable secret sharings is $\text{P2P}(O(M \cdot n^2 + n^5 + n^3 \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^5 \cdot \log |\mathbb{F}|))$, $O(M \cdot n + n^5)$ OLEs, and one invocation of $\mathcal{F}_{\text{base-DSS}}$ with **Check**, which amounts to amortized $O(n)$ elements plus $O(n)$ OLEs per secret. Notice that **DSS-RECON** will only be invoked once for computing a circuit.

Protocol 19: DSS-RECON(M)

Input. $(\llbracket \mathbf{x}_1 \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{x}_M \rrbracket_{t'+k-1})$.

Output. $x_{1,1}, \dots, x_{1,k}, \dots, x_{M,1}, \dots, x_{M,k}$ to designated receivers / a dispute pair.

Ingredients. random sharings for depack $\{\llbracket \mathbf{r}_\ell \rrbracket_{t'+k-1}, \{\langle r_{\ell,j} \rangle_t\}_{j=1}^k\}_{\ell \in [M]}$, where we omit 1 to write $\langle \cdot \rangle_t$ to denote the secret is placed at the first position.

Segment division. Let $m = \lceil \frac{M}{n^2} \rceil$. Uniformly divide the computation of depacking M detectable packed sharings into n^2 segments, $\{\text{seg}_1, \dots, \text{seg}_{n^2}\}$, and initialize $i \leftarrow 1$.

- 1: **Pick random masks.** For seg_i , all parties aim to depack $\llbracket \mathbf{x}_1 \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{x}_m \rrbracket_{t'+k-1}$. They pick m unused random masks $\{\llbracket \mathbf{r}_\ell \rrbracket_{t'+k-1}, \langle r_{\ell,1} \rangle_t, \dots, \langle r_{\ell,k} \rangle_t\}_{\ell \in [m]}$, where $\mathbf{r}_\ell = (r_{\ell,1}, \dots, r_{\ell,k})$ for each $\ell \in [m]$.

- 2: **Depack.**

1. For $\ell \in [m]$, all parties locally compute $\llbracket \mathbf{u}_\ell \rrbracket_{t'+k-1} \leftarrow \llbracket \mathbf{x}_\ell \rrbracket_{t'+k-1} + \llbracket \mathbf{r}_\ell \rrbracket_{t'+k-1}$ and send it to P_{king} .

2. For $\ell \in [m]$, P_{king} reconstructs the secret $\mathbf{u}_\ell = (u_{\ell,1}, \dots, u_{\ell,k})$ and distributes $u_{\ell,1}, \dots, u_{\ell,k}$ to all parties.
3. For $\ell \in [m]$, $j \in [k]$, all parties compute $\langle x_{\ell,j} \rangle_t \leftarrow u_{\ell,j} - \langle r_{\ell,j} \rangle_t$.

3: Verification.

1. All parties invoke $\mathcal{F}_{\text{coin}}$ to generate a random element λ .
2. All parties locally compute

$$(\llbracket \mathbf{u} \rrbracket_{t'+k-1}, \mathbf{u}) \leftarrow \sum_{\ell=1}^m \lambda^{\ell-1} \cdot (\llbracket \mathbf{u}_\ell \rrbracket_{t'+k-1}, \mathbf{u}_\ell)$$

All parties broadcast $\llbracket \mathbf{u} \rrbracket_{t'+k-1}$ and invoke $\mathcal{F}_{\text{base-DSS}}$ with **Check** to check the correctness of $\llbracket \mathbf{u} \rrbracket_{t'+k-1}$.

- If $\mathcal{F}_{\text{base-DSS}}$ outputs $(i, \text{corrupt})$, go to Step 3.6.
3. All parties reconstruct the secret of $\llbracket \mathbf{u} \rrbracket_{t'+k-1}$ and check whether it is equal to $\mathbf{u}$. If not, P_i broadcasts $(i, \text{complain})$. Otherwise, P_i broadcasts **accept**.
 4. If there are at least $t+1$ parties accepting the verification, if $i < n^2$, set $i \leftarrow i+1$ and go to Step 1; otherwise, go to Step 4.
 5. Otherwise, P_{king} broadcasts (ℓ, i, v, v') to indicate the ℓ -th message sent by P_i is v' which should be v . The relay party $P_{\text{king} \leftrightarrow i}$ also broadcasts the value it passed. Two adjacent parties broadcasting incorrect values is regarded as a dispute pair.
 6. All parties invoke **DISPUTE-CONTROL** to update $\mathcal{P}_{\text{active}}$, Corr , and Disp .
 - If $|\mathcal{P}_{\text{active}}| \geq t' + 2k - 1$, assign relays and go to Step 1.
 - Otherwise, all parties **abort**.
- 4: Reconstruction.** For each P_o , for each output $x_{\ell,j}$ belonging to P_o , all parties invoke $\mathcal{F}_{\text{vss}}$ with $(\text{PrivRecon}, 1, P_o)$ to reconstruct the secret of $\langle x_{\ell,j} \rangle_t$ to P_o .

Protocol 20: RAND-DEPACK(m)

Output. m tuples of random sharings of the form $(\llbracket \mathbf{r} \rrbracket_{t'+k-1}, \langle r_1 \rangle_t, \dots, \langle r_k \rangle_t)$ with $\mathbf{r} = (r_1, \dots, r_k)$ / a dispute pair.

- 1: **Extraction.** Let $N = \lceil \frac{m+1}{n-t} \rceil$. All parties agree on a $(n-t) \times n$ Vandermonde matrix $\mathbf{M}$.

1. For each $i \in \mathcal{P}_{\text{active}}$, P_i randomly samples $([s_i^{(\ell)}]_{t'+k-1}, [s_{i,1}^{(\ell)}]_t, \dots, [s_{i,k}^{(\ell)}]_t)_{\ell=1}^N$, where $\mathbf{s}_i^{(\ell)} = (s_{i,1}^{(\ell)}, \dots, s_{i,k}^{(\ell)})$ for $\ell \in [N]$. All parties invoke $\mathcal{F}_{\text{base-DSS}}$ with (Share, P_i) and $\mathcal{F}_{\text{vss}}$ with $(\text{vShare}, 1, P_i)$ to let P_i distribute $\llbracket \mathbf{s}_i^{(1)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{s}_i^{(N)} \rrbracket_{t'+k-1}$ and $\langle s_{i,1}^{(1)} \rangle_t, \dots, \langle s_{i,k}^{(1)} \rangle_t, \dots, \langle s_{i,1}^{(N)} \rangle_t, \dots, \langle s_{i,k}^{(N)} \rangle_t$. For each $i \in \text{Corr}$, all parties take an all-zero sharing.
2. For $\ell \in [N]$, all parties locally compute

$$\begin{aligned} & (\llbracket \mathbf{r}_1^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_{n-t}^{(\ell)} \rrbracket_{t'+k-1})^\top \\ & \leftarrow \mathbf{M} \cdot (\llbracket \mathbf{s}_1^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{s}_n^{(\ell)} \rrbracket_{t'+k-1}) \\ & (\langle r_{1,j}^{(\ell)} \rangle_t, \dots, \langle r_{n-t,j}^{(\ell)} \rangle_t)^\top \\ & \leftarrow \mathbf{M} \cdot (\langle s_{1,j}^{(\ell)} \rangle_t, \dots, \langle s_{n,j}^{(\ell)} \rangle_t)^\top, \text{ for } j \in [k]. \end{aligned}$$

2: Verification.

1. All parties invoke $\mathcal{F}_{\text{coin}}$ to generate a random element λ .

2. All parties locally compute

$$\begin{aligned} \llbracket \mathbf{r} \rrbracket_{t'+k-1} &\leftarrow \sum_{\ell=1}^N \sum_{i=1}^{n-t} \lambda^{(\ell-1) \cdot (n-t) + i - 1} \cdot \llbracket \mathbf{r}_i^{(\ell)} \rrbracket_{t'+k-1}, \\ \langle r_j \rangle_t &\leftarrow \sum_{\ell=1}^N \sum_{i=1}^{n-t} \lambda^{(\ell-1) \cdot (n-t) + i - 1} \cdot \langle r_{i,j}^{(\ell)} \rangle_t, \text{ for } j \in [k]. \end{aligned}$$

and broadcast their shares of $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$.

3. All parties invoke $\mathcal{F}_{\text{base-DSS}}$ with **Check** to check the correctness of $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$. If $\mathcal{F}_{\text{base-DSS}}$ outputs $(i, \text{corrupt})$, output $(\text{abort}, \{P_i\})$. Otherwise, reconstruct the secret $\mathbf{r}$.
 4. All parties invoke $\mathcal{F}_{\text{VSS}}$ with **Open** to reconstruct the secrets of $\langle r_1 \rangle_t, \dots, \langle r_k \rangle_t$, namely $r_1, \dots, r_k$.
 5. All parties check whether the secrets satisfy $\mathbf{r} = (r_1, \dots, r_k)$. If not, P_i broadcasts $(i, \text{complain})$. Otherwise, P_i broadcasts **accept**.
 6. If there are at least $t + 1$ parties accepting the verification, all parties accept. Otherwise, all parties invoke $\text{FAULT-LOCALIZATION}(P_i)$ to locate a dispute pair, where i is the smallest identity of the complaining parties.
- 3: Output.** For $\ell \in [N], i \in [n - t]$, all parties output $(\llbracket \mathbf{r}_i^{(\ell)} \rrbracket_{t'+k-1}, \langle r_{i,1}^{(\ell)} \rangle_t, \dots, \langle r_{i,k}^{(\ell)} \rangle_t)$ except for the one labeled with $\ell = 1, i = 1$.

Lemma 4. *DSS-RECON computes Subfunctionality 3. of $\mathcal{F}_{\text{DSS}}$, labeled **Recon**, with statistical security against any fully malicious adversary controlling up to t parties in the $\mathcal{F}_{\text{base-DSS}}, \mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{coin}}$ -hybrid model.*

For the proof of Lemma 4, we refer the reader to Section H.4.

C.5 Summary of the Extended DSS

We now summarize the realization of the extended functionality of detectable secret sharings in **DSS**. The protocol maintains a database that stores the authenticated sharings, updates according to newly detected dispute pairs, and invokes either the appropriate protocol or the basic DSS functionality for each subfunctionality. Combining Lemma 2, 3, and 4, we conclude the security property of **DSS** in Lemma 5.

Protocol 21: $\text{DSS}(M_1, M_2)$

Setup. On input **Setup**,

1. Invoke **DISPUTE-CONTROL** to initialize $\mathcal{P}_{\text{active}}, \text{Corr}$, and Disp .
2. Invoke $\mathcal{F}_{\text{rand-OLE}}(M)$, $\mathcal{F}_{\text{VSS}}$, and $\mathcal{F}_{\text{base-DSS}}(M_1)$ with **Setup**.
3. All parties prepare random masks used for reconstructing M_2 detectable secret sharings following dispute control framework.

Let $m = \lceil \frac{M_2}{n^2} \rceil$. Initialize $i \leftarrow 1$.

- a. All parties invoke **RAND-DEPACK**(m) to prepare m tuples of random sharings of the form $(\llbracket \mathbf{r} \rrbracket_{t'+k-1}, \{\langle r_j \rangle_t\}_{j=1}^k)$.
- b. In case of failure, all parties have identified a dispute pair E . After finishing **update**, go to Step 3.a.
- c. In case of success, if $i < n^2$, set $i \leftarrow i + 1$ and go to Step 3.a.

Update. On input a new dispute pair E , invoke **DISPUTE-CONTROL** to update $\mathcal{P}_{\text{active}}, \text{Corr}$, and Disp .

- If $|\mathcal{P}_{\text{active}}| < t' + 2k - 1$, all parties **abort**.
- Otherwise, agree on a special party P_{king} and assign a relay party for each dispute pair: for each pair of active parties (P_i, P_j) , if $(P_i, P_j) \in \text{Disp}$, arrange a relay party $P_{i \leftrightarrow j}$ such that $P_{i \leftrightarrow j}$ does not conflict with any of P_i and P_j .

Instruction set.

1. On input (Triple, m) , all parties invoke **VIRTUAL-PARTY** $(\text{Corr}, \text{Disp})$ to determine the set of virtual parties $\mathcal{V}_{\text{active}}$, invoke **AUTH-PACKED-TRIPLES** $(\mathcal{V}_{\text{active}}, m)$ to prepare authenticated packed triples of the form $(\llbracket \mathbf{a} \rrbracket_{t'+k-1}, \llbracket \mathbf{b} \rrbracket_{t'+k-1}, \llbracket \mathbf{c} \rrbracket_{t'+k-1})$ and store their shares. In case of failure, go to **update**.
2. On input $(\text{Transpose}, m)$, all parties invoke **AUTH-TRANSPOSE** to prepare authenticated linear masks for transpose of the form $(\{\llbracket \mathbf{r}_{\ell*} \rrbracket_{t'+k-1}\}_{\ell \in [k]}, \{\llbracket \mathbf{r}_{*\ell} \rrbracket_{t'+k-1}\}_{\ell \in [k]})$ and store their shares. In case of failure, go to **update**.
3. On input **Recon**, all parties invoke **DSS-RECON** to reconstruct the secrets to designated receivers.
4. On input (Share, P_d) , **Check** or **Auth**, all parties directly invoke $\mathcal{F}_{\text{base-DSS}}$ with the same instruction. In case of failure, go to **update**.

Lemma 5. *DSS computes $\mathcal{F}_{\text{DSS}}$ with statistical security against any fully malicious adversary controlling up to t corrupted parties in the $\mathcal{F}_{\text{VSS}}$, $\mathcal{F}_{\text{base-DSS}}$, $\mathcal{F}_{\text{rand-OLE}}$, $\mathcal{F}_{\text{vrfy-linear-sharing}}$, $\mathcal{F}_{\text{coin-hybrid}}$ model.*

D The Protocol

To compute general circuits, parties need to handle network routing, as discussed in Section D.1. Combining this with the protocols for computing and verifying the multiplication gates presented in Section 5, we then summarize the segment computation for general circuits in Section D.2. In Section D.3, we describe how the parties process the input layers. Finally, putting everything together, we obtain an honest majority MPC protocol that computes any circuit with GOD in Section D.4.

D.1 Towards General Circuits

Handling Network Routing. Following [GPS21, GPS22], handling network routing can be reduced to two tasks. The first task is to perform linear transformations on the secrets: all parties hold $\llbracket \mathbf{x} \rrbracket_{t'+k-1}$ and aim to compute $\llbracket \mathbf{L}(\mathbf{x}) \rrbracket_{t'+k-1}$, where $\mathbf{L} : \mathbb{F}^k \rightarrow \mathbb{F}^k$ is a linear transformation. The second task is to collect secrets stored at different positions across multiple packed sharings.

As for the first task of performing linear transformations, all parties first prepare random linear masks of the form $(\llbracket \mathbf{r} \rrbracket_{t'+k-1}, \llbracket \mathbf{L}(\mathbf{r}) \rrbracket_{t'+k-1})$ and then follow the *mask-and-open* recipe to compute $\llbracket \mathbf{L}(\mathbf{x}) \rrbracket_{t'+k-1}$ from $\llbracket \mathbf{x} \rrbracket_{t'+k-1}$. A crucial observation in [SY24] is that, if parties hold packed sharings that store k secrets at the same position, then they can locally compute a packed sharing of a linear transformation of these k secrets. Based on this observation, all parties prepare random linear masks $(\llbracket \mathbf{r}_{\ell} \rrbracket_{t'+k-1}, \llbracket \mathbf{L}_{\ell}(\mathbf{r}_{\ell}) \rrbracket_{t'+k-1})_{\ell=1}^k$ for different linear transformations $\mathbf{L}_1, \dots, \mathbf{L}_k$ as follows. For the sake of simplicity, let $\mathbf{v}_{\ell} := \mathbf{L}_{\ell}(\mathbf{r}_{\ell})$.

- Start with $\llbracket \mathbf{r}_1 \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_k \rrbracket_{t'+k-1}$.
- Apply a transpose on secret matrix $(\mathbf{r}_{\ell})_{\ell=1}^k$ to obtain $\llbracket \mathbf{r}_{*1} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_{*k} \rrbracket_{t'+k-1}$.
- Use the observation to compute $\llbracket \mathbf{v}_{*1} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{v}_{*k} \rrbracket_{t'+k-1}$.
- Apply another transpose on secret matrix $(\mathbf{v}_{*\ell})_{\ell=1}^k$ to obtain $\llbracket \mathbf{v}_1 \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{v}_k \rrbracket_{t'+k-1}$.

This effectively reduces preparing random linear masks for different linear transformations to performing transpose operations on secret matrices. Recall all parties have prepared authenticated random linear masks

for transpose of the form $(\{\llbracket \mathbf{r}_{\ell*} \rrbracket_{t'+k-1}\}_{\ell \in [k]}, \{\llbracket \mathbf{r}_{*\ell} \rrbracket_{t'+k-1}\}_{\ell \in [k]})$. Using this preprocessing data, all parties again follow the *mask-and-open* recipe to carry out the transpose operation on secret matrices.

We present the protocols for preparing random linear masks and performing linear transformations in **RAND-LINEAR-MASK** and **SHARING-TRANS**. The cost of **RAND-LINEAR-MASK** to prepare N random linear masks is $\text{P2P}(O(N \cdot n + n^3)) + \text{BC}(O(n + \log |\mathbb{F}|))$. The cost of **SHARING-TRANS** is $\text{P2P}(O(n))$.

Protocol 22: RAND-LINEAR-MASK(N)

Output: N pairs of random linear masks for N different linear transformations, each of the form $(\llbracket \mathbf{r} \rrbracket_{t'+k-1}, \llbracket \mathbf{L}(\mathbf{r}) \rrbracket_{t'+k-1})$, where $\mathbf{L} : \mathbb{F}^k \rightarrow \mathbb{F}^k$ is a linear transformation / a dispute pair.

Ingredients: $2 \lceil \frac{N}{k} \rceil$ tuples of random linear masks of the form $(\llbracket \mathbf{r}_{\ell*} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{*\ell} \rrbracket_{t'+k-1})_{\ell \in [k]} \cdot \lceil \frac{N}{k} \rceil$ random zero sharings of the form $[\mathbf{o}]_{n'-1}$.

1. Preparing linear masks for linear transformations.

For each k linear transformations $\mathbf{L}_1, \dots, \mathbf{L}_k$, all parties use two tuples of linear masks, $(\llbracket \mathbf{r}_{\ell*} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{*\ell} \rrbracket_{t'+k-1})_{\ell \in [k]}$, $(\llbracket \mathbf{u}_{\ell*} \rrbracket_{t'+k-1}, \llbracket \mathbf{u}_{*\ell} \rrbracket_{t'+k-1})_{\ell \in [k]}$, to compute $[\mathbf{L}_1(\mathbf{r}_{1*})]_{t'+k-1}, \dots, [\mathbf{L}_k(\mathbf{r}_{k*})]_{t'+k-1}$ as follows.^a

(a) For each $\ell \in [k]$, all parties locally compute

$$\begin{aligned} [\mathbf{v}_{*\ell} + \mathbf{u}_{*\ell}]_{n'-1} &\leftarrow [\mathbf{L}_{*,\ell 1}]_{k-1} \cdot \llbracket \mathbf{r}_{*1} \rrbracket_{t'+k-1} + \dots \\ &+ [\mathbf{L}_{*,\ell k}]_{k-1} \cdot \llbracket \mathbf{r}_{*k} \rrbracket_{t'+k-1} + \llbracket \mathbf{u}_{*\ell} \rrbracket_{t'+k-1} + [\mathbf{o}_\ell]_{n'-1} \end{aligned}$$

and send their shares to P_{king} , where $\mathbf{L}_{*,\ell j} = (L_{1,\ell j}, \dots, L_{k,\ell j})$ for $j \in [k]$ and $\mathbf{v}_{*\ell} = (L_{1,\ell}(\mathbf{r}_{1*}), L_{2,\ell}(\mathbf{r}_{2*}), \dots, L_{k,\ell}(\mathbf{r}_{k*})) \in \mathbb{F}^k$.

(b) For each $\ell \in [k]$, P_{king} reconstructs the secret $\mathbf{v}_{*\ell} + \mathbf{u}_{*\ell}$. Then for each $\ell \in [k]$, P_{king} computes $\mathbf{v}_{\ell*} + \mathbf{u}_{\ell*}$ and distributes $[\mathbf{v}_{*\ell} + \mathbf{u}_{*\ell}]_{k-1}, [\mathbf{v}_{\ell*} + \mathbf{u}_{\ell*}]_{k-1}$ to all parties. Notice that $\mathbf{v}_{\ell*} = \mathbf{L}_\ell(\mathbf{r}_{\ell*})$ for $\ell \in [k]$.

(c) For each $\ell \in [k]$, all parties locally compute $[\mathbf{L}_\ell(\mathbf{r}_{\ell*})]_{t'+k-1} \leftarrow [\mathbf{L}_\ell(\mathbf{r}_{\ell*}) + \mathbf{u}_{\ell*}]_{k-1} - \llbracket \mathbf{u}_{\ell*} \rrbracket_{t'+k-1}$.

2. **Verification.** All parties invoke $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_3)$ with $(\{[\mathbf{v}_{\ell*} + \mathbf{u}_{\ell*}]_{k-1}\}_{\ell \in [k]}, \{[\mathbf{v}_{*\ell} + \mathbf{u}_{*\ell}]_{k-1}\}_{\ell \in [k]})$ received from P_{king} in Step 1.b to check whether P_{king} honestly applies the transpose operations.

The interaction in Steps 1.a and 1.b gives rise to $k \cdot \lceil \frac{N}{k} \rceil$ degree-reduction transcripts.

^a Each linear transformation $\mathbf{L}_\ell = (L_{\ell,1}, \dots, L_{\ell,k})$ can be represented by a $k \times k$ matrix $(L_{\ell,ij})_{ij} \in \mathbb{F}^{k \times k}$ and the linear transformation is defined as $(x_i)_{i=1}^k \mapsto (\sum_{j=1}^k L_{\ell,ij} \cdot x_j)_{i=1}^k$.

Protocol 23: SHARING-TRANS

Input: $[\mathbf{x}]_{t'+k-1}$.

Output: $[\mathbf{L}(\mathbf{x})]_{t'+k-1}$, where $\mathbf{L} : \mathbb{F}^k \rightarrow \mathbb{F}^k$ is a linear transformation.

Ingredients: $(\llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{L}(\mathbf{r})]_{t'+k-1})$.

1. All parties locally compute $[\mathbf{x} + \mathbf{r}]_{t'+k-1} \leftarrow [\mathbf{x}]_{t'+k-1} + \llbracket \mathbf{r} \rrbracket_{t'+k-1}$ and send their shares to P_{king} .
2. P_{king} reconstructs the secret $\mathbf{x} + \mathbf{r}$ using the first $t' + k$ shares, computes $\mathbf{L}(\mathbf{x} + \mathbf{r})$, and distributes $[\mathbf{x} + \mathbf{r}]_{k-1}, [\mathbf{L}(\mathbf{x} + \mathbf{r})]_{k-1}$ to all parties.
3. All parties compute $[\mathbf{L}(\mathbf{x})]_{t'+k-1} \leftarrow [\mathbf{L}(\mathbf{x} + \mathbf{r})]_{k-1} - [\mathbf{L}(\mathbf{r})]_{t'+k-1}$.

The interaction in Steps 1 and 2 gives rise to a single degree-reduction transcript.

As for the second task of collecting secrets, all parties first use unit vectors to select the target secrets and then perform degree reduction by consuming an authenticated random packed sharing of the form $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$

together with a degree- $(n' - 1)$ random packed zero-sharing of the form $[\mathbf{o}]_{n'-1}$. We present the protocol for secret collection in **SELECT**. The cost of **SELECT** is $\text{P2P}(O(n))$.

Protocol 24: SELECT

Input: $[\mathbf{x}_1]_{t'+k-1}, \dots, [\mathbf{x}_k]_{t'+k-1}$.

Output: $[\mathbf{x}]_{t'+k-1}$, where $\mathbf{x} = (x_1, \dots, x_k)$ and x_ℓ is the ℓ -th element of $\mathbf{x}_\ell$ for $\ell \in [k]$.

Ingredients: $[\mathbf{r}]_{t'+k-1}, [\mathbf{o}]_{n'-1}$.

1. All parties locally compute $[\mathbf{e}]_{n'-1} \leftarrow [\mathbf{e}_1]_{k-1} \cdot [\mathbf{x}_1]_{t'+k-1} + \dots + [\mathbf{e}_k]_{k-1} \cdot [\mathbf{x}_k]_{t'+k-1} + [\mathbf{r}]_{t'+k-1} + [\mathbf{o}]_{n'-1}$ and send their shares to P_{king} , where $\mathbf{e}_\ell \in \mathbb{F}^k$ denotes the ℓ -th unit vector.
2. P_{king} reconstructs the secret $\mathbf{e}$ and distributes $[\mathbf{e}]_{k-1}$ to all parties.
3. All parties locally compute $[\mathbf{x}]_{t'+k-1} \leftarrow [\mathbf{e}]_{k-1} - [\mathbf{r}]_{t'+k-1}$.

The interaction in Steps 1 and 2 gives rise to a single degree-reduction transcript.

Verifying the Computation of Network Routing. To verify the computation of **RAND-LINEAR-MASK**, **SHARING-TRANS**, and **SELECT**, we investigate each round of interaction. Notice that except for Step 2 in **SHARING-TRANS**, where P_{king} distributes two packed Shamir sharings $([\mathbf{x} + \mathbf{r}]_{k-1}, [\mathbf{L}(\mathbf{x} + \mathbf{r})]_{k-1})$, all transcripts generated by interactions in the protocols can be transformed into the form produced by **EXTEND-DEG-REDUCTION**, and therefore can be jointly verified by the recursive verification illustrated in Section 5.1. It remains to design a verification protocol that verifies whether the secrets of $[\mathbf{x} + \mathbf{r}]_{k-1}, [\mathbf{L}(\mathbf{x} + \mathbf{r})]_{k-1}$ distributed by P_{king} correctly satisfy the linear transformation relation.

We use $\mathbf{L}_i : \mathbb{F}^k \rightarrow \mathbb{F}$ to denote the i -th linear function in $\mathbf{L}$, and we abuse the notation to use $\mathbf{L}_i$ to denote the coefficient vector such that $\mathbf{L}_i(\mathbf{x}) = \mathbf{L}_i \odot \mathbf{x}$, where $\odot$ denotes the inner product. Then we need to verify that the i -th secret of $[\mathbf{L}(\mathbf{x} + \mathbf{r})]_{k-1}$ equals $\mathbf{L}_i \odot (\mathbf{x} + \mathbf{r})$ for each $i \in [k]$. Equivalently, it suffices to check that the secret vector of the sharing $[\mathbf{e}]_{k-1} \cdot [\mathbf{L}(\mathbf{x} + \mathbf{r})]_{k-1} - [\mathbf{L}_i]_{k-1} \cdot [\mathbf{x} + \mathbf{r}]_{k-1}$ has entries summing up to zero. This can be efficiently checked in a batch-wise manner.

The protocol that verifies the linear transformations is presented in **VRFY-SHARING-TRANS**. The communication cost of **VRFY-SHARING-TRANS** to check m pairs is $\text{P2P}(O(n^2)) + \text{BC}(O(n + \log |\mathbb{F}|))$.

Protocol 25: VRFY-SHARING-TRANS(m)

Input: $([\mathbf{x}_\ell + \mathbf{r}_\ell]_{k-1}, [\mathbf{L}_\ell(\mathbf{x}_\ell + \mathbf{r}_\ell)]_{k-1})_{\ell \in [m]}$.

Output: ok / a dispute pair.

1. For each $\ell \in [m]$ and $i \in [k]$, supposing $\mathbf{L}_\ell = (\mathbf{L}_{\ell,1}, \dots, \mathbf{L}_{\ell,k})$ with each $\mathbf{L}_{\ell,i} : \mathbb{F}^k \rightarrow \mathbb{F}$, all parties locally compute

$$[\mathbf{o}_{\ell,i}]_{2k-2} \leftarrow [\mathbf{e}_i]_{k-1} \cdot [\mathbf{L}_\ell(\mathbf{x}_\ell + \mathbf{r}_\ell)]_{k-1} - [\mathbf{L}_{\ell,i}]_{k-1} \cdot [\mathbf{x}_\ell + \mathbf{r}_\ell]_{k-1}.$$

2. All parties invoke $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_2)$ to check the consistency.
 - If the check fails, all parties output the identified dispute pair E .
 - Otherwise, all parties accept the computation and output ok.

D.2 Summary of the Computation Segment

We now summarize the protocol for computing a segment of a general circuit. In the beginning, all parties hold the input sharings that are authenticated by $\mathcal{F}_{\text{DSS}}$. All parties first evaluate the circuit layer by layer and then verify the computation. For each layer, all parties invoke **SELECT**, **SHARING-TRANS** to prepare input sharings of each group of k gates in this layer. They then locally evaluate the addition gates and invoke **MULT** to evaluate the multiplication gates. All parties then invoke **SHARING-TRANS** to handle the fan-out gates and achieve non-collision property for each input sharing of the next layer.

During the verification phase, all parties invoke $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_1)$ to check whether P_{king} always distributes valid degree- $(k-1)$ packed Shamir sharings, invoke **VRIFY-SHARING-TRANS** to check whether P_{king} performs the linear transformations correctly, and invoke **VRIFY-TRANSCRIPTS** to verify the correctness of the uniform transcripts.

Finally, after verifying the computation, before proceeding to the next segment, all parties need to authenticate the output sharings of this segment to maintain the verifiability of the sharings used in the subsequent computation.

We conclude the protocol for computing a segment of a general circuit in **COMP-SEG**. The cost of invoking **COMP-SEG** to compute a segment containing N multiplication gates and N_f fan-out gates is $\text{P2P}(O(N + N_f + n^3 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^3 \cdot \log |\mathbb{F}|))$, and two invocations of $\mathcal{F}_{\text{DSS}}$ with **Check**.

Protocol 26: COMP-SEG

Input: Input sharings of the form $[\![\mathbf{x}]\!]_{t'+k-1}$.

Output: Output sharings of the form $[\![\mathbf{z}]\!]_{t'+k-1}$ / a dispute pair.

Ingredients: N/k authenticated triples of the form $([\![\mathbf{a}]\!]_{t'+k-1}, [\![\mathbf{b}]\!]_{t'+k-1}, [\![\mathbf{c}]\!]_{t'+k-1})$, $4N_f/k^2$ authenticated random linear masks of the form $(\{[\![\mathbf{r}_{\ell*}]\!]_{t'+k-1}\}_{\ell \in [k]}, \{[\![\mathbf{r}_{*\ell}]\!]_{t'+k-1}\}_{\ell \in [k]})$, and $N/k + N_f/k + 2\kappa \cdot \log_{\kappa}(5N/k + N_f + 4N_f/k)$ random authenticated packed sharings of the form $[\![\mathbf{r}]\!]_{t'+k-1}$.

- 1: All parties agree on a special party P_{king} .
- 2: **Preparing random sharings.**
 1. All parties invoke **RAND**(Σ_2) to prepare $N/k + N_f/k + 2N_f/k^2 + 2\kappa \cdot \log_{\kappa}(5N/k + N_f + 4N_f/k)$ random degree- $(n' - 1)$ packed zero-sharings in the form of $[\![\mathbf{o}]\!]_{n'-1}$, which will be used for invoking **RAND-LINEAR-MASK**, **SELECT**, **MULT**, and **VRIFY-TRANSCRIPTS** subsequently.
 2. All parties invoke **RAND-LINEAR-MASK**($2N_f/k$) to prepare $2N_f/k$ random linear masks in the form of $([\![\mathbf{r}]\!]_{t'+k-1}, [\![\mathbf{L}(\mathbf{r})]\!]_{t'+k-1})$.
- 3: **Segment evaluation.** All parties evaluate the segment layer by layer.
 1. Permute input sharings from previous layers: For each input sharing $[\![\mathbf{x}]\!]_{t'+k-1}$ needed in the current layer, let $[\![\mathbf{x}^{(j)}]\!]_{t'+k-1}$ be an output sharing in previous layers with its j -th position being the q_j -th component of $\mathbf{x}$ for each $j \in [k]$. Let $\mathbf{L} : \mathbb{F}^k \rightarrow \mathbb{F}^k$ denote the permutation defined by $j \mapsto q_j$. All parties invoke **SELECT** on $[\![\mathbf{x}^{(1)}]\!]_{t'+k-1}, \dots, [\![\mathbf{x}^{(k)}]\!]_{t'+k-1}$ to collect the secrets and compute $[\![\mathbf{x}']]\!]_{t'+k-1}$. All parties then invoke **SHARING-TRANS** on $[\![\mathbf{x}']]\!]_{t'+k-1}$ and $\mathbf{L}$ to compute $[\![\mathbf{x}]\!]_{t'+k-1}$.
 2. All parties evaluate the current layer as follows.
 - For each group of k gates with input sharings $[\![\mathbf{x}]\!]_{t'+k-1}, [\![\mathbf{y}]\!]_{t'+k-1}$,
 - for addition gates, all parties locally compute $[\![\mathbf{z}]\!]_{t'+k-1} \leftarrow [\![\mathbf{x}]\!]_{t'+k-1} + [\![\mathbf{y}]\!]_{t'+k-1}$.
 - for multiplication gates, all parties invoke **MULT** to compute $[\![\mathbf{z}]\!]_{t'+k-1}$ with $\mathbf{z} = \mathbf{x} * \mathbf{y}$.
 3. Handling fan-out gates and achieving non-collision property: For the output sharing $[\![\mathbf{z}]\!]_{t'+k-1}$ of each group of multiplication gates or addition gates, where the linear transformation $\mathbf{L} : \mathbb{F}^k \rightarrow \mathbb{F}^k$ is needed to perform on $\mathbf{z}$, all parties invoke **SHARING-TRANS** on $[\![\mathbf{z}]\!]_{t'+k-1}$ to compute $[\![\mathbf{L}(\mathbf{z})]\!]_{t'+k-1}$.
- 4: **Segment verification.**
 1. All parties invoke $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_1)$ to check whether P_{king} always distributes consistent packed Shamir sharings of degree $k-1$ in this segment.
 2. All parties invoke **VRIFY-SHARING-TRANS**($2N_f/k$) to verify the correctness of performing sharing transformations.
 3. All parties invoke **VRIFY-TRANSCRIPTS**(ℓ) to verify the correctness of degree reduction when invoking **RAND-LINEAR-MASK**, **SHARING-TRANS**, **SELECT**, and **MULT**, where the initial transcripts are of dimension $\ell = 5N/k + N_f + 4N_f/k$ in total.

If one of the checks fails, all parties output the identified dispute pair.

- 5: **Adding verifiability.** Let $[z^{(1)}]_{t'+k-1}, \dots, [z^{(m)}]_{t'+k-1}$ be the output sharings of the current segment, where $[z^{(\ell)}]_{t'+k-1} = [z_0^{(\ell)}]_{k-1} + \llbracket z_1^{(\ell)} \rrbracket_{t'+k-1}$ with $[z_0^{(\ell)}]_{k-1}$ distributed by P_{king} . All parties invoke $\mathcal{F}_{\text{DSS}}$ with **Auth** to obtain $\llbracket z_0^{(1)} \rrbracket_{k-1}, \dots, \llbracket z_0^{(m)} \rrbracket_{k-1}$.
- In case of failure, all parties receive and output the identified dispute pair.
 - Otherwise, all parties output $\llbracket z^{(1)} \rrbracket_{t'+k-1}, \dots, \llbracket z^{(m)} \rrbracket_{t'+k-1}$.

D.3 Input Layer

In the input layer, if the inputs belong to party P_i , P_i needs to distribute its inputs to all parties, which are required to be authenticated. To avoid the use of OLE correlations in the online phase, each party is expected to distribute random authenticated packed sharings in the preprocessing phase. Then in the online phase, party P_i can compute a degree- $(k-1)$ packed Shamir sharing whose secret vector is the difference between the real input and the secret vector of the random authenticated packed sharings delivered by P_i in the preprocessing phase. Moreover, similar to the computation segment, all parties need to handle network routing and authenticate the output sharings.

The protocol for input layer appears in **INPUT**. The cost of invoking **INPUT** to handle an input segment containing N input gates and N_f fan-out gates is $\text{P2P}(O(N + N_f + n^3 + n \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^3 \cdot \log |\mathbb{F}|))$, and two invocations of $\mathcal{F}_{\text{DSS}}$ with **Check**.

Protocol 27: INPUT(P_i)

Input. Circuit inputs belonging to $P_i : \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$.

Output. output sharings of the input segment $\llbracket \mathbf{z}^{(1)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{z}^{(m')} \rrbracket_{t'+k-1}$ / a dispute pair.

Ingredients. $\llbracket \mathbf{r}^{(1)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}^{(m)} \rrbracket_{t'+k-1}$ distributed by P_i in the preprocessing phase, $2N_f/k^2$ authenticated random linear masks of the form $(\{\llbracket \mathbf{r}_{\ell*} \rrbracket_{t'+k-1} \}_{\ell \in [k]}, \{\llbracket \mathbf{r}_{*\ell} \rrbracket_{t'+k-1} \}_{\ell \in [k]})$, and $2\kappa \cdot \log_{\kappa} \frac{N_f}{k}$ random authenticated packed sharings of the form $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$.

P_i plays the role of the special party P_{king} .

1. All parties invoke **RAND**(Σ_2) to prepare $N_f/k^2 + 2\kappa \cdot \log_{\kappa} \frac{N_f}{k}$ random degree- $(n'-1)$ packed zero-sharings of the form $[\mathbf{o}]_{n'-1}$ used for invoking **RAND-LINEAR-MASK** and **VRFY-TRANSCRIPTS**.
2. All parties invoke **RAND-LINEAR-MASK**(N_f/k) to prepare random linear masks of the form $(\llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{L}(\mathbf{r})]_{t'+k-1})$.
3. **Sharing input.**
 - (a) If $i \in \text{Corr}$, all parties set each $\llbracket \mathbf{x}^{(\ell)} \rrbracket_{t'+k-1}$ to be an all-zero sharing and halt.
 - (b) For $\ell \in [m]$, P_i distributes $[\mathbf{x}^{(\ell)} + \mathbf{r}^{(\ell)}]_{k-1}$ to all parties.
 - (c) For $\ell \in [m]$, all parties locally compute $[\mathbf{x}^{(\ell)}]_{t'+k-1} \leftarrow [\mathbf{x}^{(\ell)} + \mathbf{r}^{(\ell)}]_{k-1} - \llbracket \mathbf{r}^{(\ell)} \rrbracket_{t'+k-1}$.
 - (d) All parties invoke $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_1)$ to check whether P_i distributed valid degree- $(k-1)$ packed Shamir sharings in Step 3.b.
4. All parties handle fan-out gates and perform sharing transformation to achieve non-collision property as Step 3.3 of **COMP-SEG**.
5. All parties verify the computation of the segment as Step 4 of **COMP-SEG**.
6. All parties add verifiability as Step 5 of **COMP-SEG**.

D.4 Main Protocol

We now summarize the whole protocol assuming $\mathcal{F}_{\text{DSS}}$ functionality. In the preprocessing phase, all parties prepare authenticated packed Beaver triples and authenticated linear masks for transpose that will be consumed in the online phase. Also, each party distributes random authenticated packed sharings used for the input layer. In the online phase, parties invoke **INPUT** for an input segment and **COMP-SEG** for a computation

segment. Finally, if all parties succeed in computing the entire circuit before the number of remaining parties drops below $t' + 2k - 1$, they invoke $\mathcal{F}_{\text{DSS}}$ with **Recon** to reconstruct the outputs to the corresponding parties. Otherwise, with a $1/4$ corruption threshold, the parties restart the computation by invoking $\mathcal{F}_{\text{main-client-server}}^*$. We instantiate $\mathcal{F}_{\text{main-client-server}}^*$ using the construction in [SY25b], which achieves perfect security against a $1/4$ -corruption adversary with $O(|C| + D \cdot n + C_O \cdot n + n^4)$ elements of communication, where D and C_O denote the circuit depth and the size of output gates, respectively.

The whole protocol is presented in **MAIN**, whose security property is stated in Lemma 6. We now consider the communication complexity of our construction. The online communication of **MAIN** is $\text{P2P}(O(|C| + D \cdot n + n^5 + n^3 \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^5 \cdot \log |\mathbb{F}|))$. Now we calculate the amount of preprocessing data consumed by **MAIN** as follows. Let C_I, C_M, C_A and C_O denote the number of input gates, multiplication gates, addition gates, and output gates, respectively. Let N_f denote the number of fan-out gates. Then $N_f \leq 2C_M + 2C_A + C_O$.

- Let N_1 denote the number of required authenticated triples of the form $(\llbracket \mathbf{a} \rrbracket_{t'+k-1}, \llbracket \mathbf{b} \rrbracket_{t'+k-1}, \llbracket \mathbf{c} \rrbracket_{t'+k-1})$. Then $N_1 = \frac{2C_M}{k} = O(|C|/n + D + n^2)$.
- Let N_2 denote the number of required authenticated linear masks of the form $(\{\llbracket \mathbf{r}_{\ell*} \rrbracket_{t'+k-1}\}_{\ell \in [k]}, \{\llbracket \mathbf{r}_{*\ell} \rrbracket_{t'+k-1}\}_{\ell \in [k]})$. Then $N_2 = \frac{8N_f}{k^2} = O(|C|/n^2 + n^2)$.
- Let N_3 denote the number of required random authenticated packed sharings of the form $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$. Then $N_3 = 2(\frac{C_M}{k} + \frac{N_f}{k} + n^2 \cdot 2\kappa \cdot \log_\kappa(\frac{5C_M}{n^2 \cdot k} + \frac{N_f}{n^2} + \frac{4N_f}{n^2 \cdot k})) = O(|C|/n + D + n^2 \cdot \kappa)$.
- Let N_4 denote the number of required random sharings of the form $(\llbracket \mathbf{r} \rrbracket_{t'+k-1}, \langle r_1 | 1 \rangle_t, \dots, \langle r_k | 1 \rangle_t)$ with $\mathbf{r} = (r_1, \dots, r_k)$. Then $N_4 = 2C_O/k$.
- Let N_5 denote the number of required random messages with tags for each party of the form $(\mathbf{z}, \eta(\mathbf{z}))$, which are used for invoking $\mathcal{F}_{\text{DSS}}$ with **Check**. Then $N_5 = (9k + 2k^2) \cdot n^2$.

Considering our instantiation of detectable secret sharing functionality assuming random OLE correlations, the cost of the preprocessing phase is $\text{P2P}(O(|C| + D \cdot n + C_O \cdot n + n^6 + n^3 \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^6 \cdot \log |\mathbb{F}|))$ and $O(|C| + D \cdot n + n^5)$ random OLEs.

Putting things altogether, we derive an information-theoretic MPC protocol for any arithmetic circuit C achieving GOD in the OLE preprocessing model with communication $\text{P2P}(O(|C| + D \cdot n + C_O \cdot n + n^6 + n^3 \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^6 \cdot \log |\mathbb{F}|))$ and $O(|C| + D \cdot n + n^5)$ random OLE correlations. We conclude this result in Theorem 2.

Protocol 28: MAIN

Input. Circuit inputs of size C_I .

Output. Circuit outputs of size C_O .

Let N_1, N_2 and N_3 denote the number of required authenticated packed triples, authenticated linear masks for transpose, and random authenticated packed sharings, respectively.

Invoke **DISPUTE-CONTROL** to initialize $\mathcal{P}_{\text{active}}, \text{Corr}, \text{Disp}$.

Invoke $\mathcal{F}_{\text{rand-OLE}}(M_1)$ and $\mathcal{F}_{\text{DSS}}(M_2, C_O)$ with **Setup**, where $M_1 = (224n + 8k) \cdot (N_1 + N_3/2) + 32nk \cdot N_2 + 8n \cdot \frac{2C_I}{k} + 16n \cdot \frac{2C_O}{k} + 8k^2n \cdot M_2$ and $M_2 = (9 + 2k) \cdot n^2$.

- 1: **Preprocessing phase.** Uniformly divide the preprocessing phase into n^2 segments, $\{\text{seg}_1, \dots, \text{seg}_{n^2}\}$, and initialize $i \leftarrow 1$.
 1. All parties compute seg_i as follows.
 - a. Invoke $\mathcal{F}_{\text{DSS}}$ with **(Triple, $\lceil \frac{N_1 + N_3/2}{n^2} \rceil$)**^a to prepare all the needed authenticated packed Beaver triples in the form of $(\llbracket \mathbf{a} \rrbracket_{t'+k-1}, \llbracket \mathbf{b} \rrbracket_{t'+k-1}, \llbracket \mathbf{c} \rrbracket_{t'+k-1})$.
 - b. Invoke $\mathcal{F}_{\text{DSS}}$ with **(Transpose, $\lceil \frac{N_2}{n^2} \rceil$)** to prepare all the needed authenticated linear masks for transpose in the form of $(\{\llbracket \mathbf{r}_{\ell*} \rrbracket_{t'+k-1}\}_{\ell \in [k]}, \{\llbracket \mathbf{r}_{*\ell} \rrbracket_{t'+k-1}\}_{\ell \in [k]})$.
 - c. For each active party P_j , all parties invoke $\mathcal{F}_{\text{DSS}}$ with **(Share, P_j)** to let P_j distribute m_j random authenticated packed sharings, where $m_j = \lceil \frac{2C_{I_j}}{n^2} \rceil$ and C_{I_j} denotes the number of input gates of P_j .
 2. In case of success, if $i < n^2$, set $i \leftarrow i + 1$ and go to Step 1.1; otherwise, go to Step 2 to compute the online phase.

3. In case of failure, if $\mathcal{F}_{\text{DSS}}$ outputs **abort** and $\mathcal{C}'$, set $\mathcal{P}_{\text{active}} \leftarrow \mathcal{P}_{\text{active}} \setminus \mathcal{C}'$ go to Step 4 to restart; otherwise go to Step 1.1 to re-compute the current segment.
- 2: **Online phase.** Uniformly divide the circuit into $2n^2$ segments, $\{\text{seg}_1, \dots, \text{seg}_{2n^2}\}$, and initialize $i \leftarrow 1$.
 1. Depending on the type of seg_i , there are 2 cases.
 - If seg_i is an input segment and the inputs belong to P_j , all parties invoke $\text{INPUT}(P_j)$.
 - If seg_i is a computation segment, all parties invoke COMP-SEG .
 2. In case of success, if $i < 2n^2$, set $i \leftarrow i + 1$ and go to Step 2.1; otherwise, go to Step 3 to reconstruct the outputs.
 3. In case of failure, all parties identify a new dispute pair E and invoke DISPUTE-CONTROL to update $\mathcal{P}_{\text{active}}$, Corr , and Disp .
 - If $|\mathcal{P}_{\text{active}}| \geq t' + 2k - 1$, assign relay parties^b and go to Step 2.1 to re-compute the current segment.
 - Otherwise, go to Step 4 to restart.
- 3: **Reconstruction.** Let $N = \lceil C_O/k \rceil$. For all the output wires with secrets $\mathbf{x}_1, \dots, \mathbf{x}_N$, all parties invoke $\mathcal{F}_{\text{DSS}}$ with **Recon** to reconstruct $\llbracket \mathbf{x}_1 \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{x}_N \rrbracket_{t'+k-1}$.
 - In case of success, each output $x_{\ell,j}$ has been distributed to the designated party.
 - If $\mathcal{F}_{\text{DSS}}$ outputs **abort** and $\mathcal{C}'$, set $\mathcal{P}_{\text{active}} \leftarrow \mathcal{P}_{\text{active}} \setminus \mathcal{C}'$ go to Step 4 to restart.
- 4: **Restart.** All parties invoke $\mathcal{F}_{\text{main-client-server}}^*$. Every party serves as a client, providing its input and receiving its output, while the parties in $\mathcal{P}_{\text{active}}$ act as servers.

^a We will use the sharings $\llbracket \mathbf{a} \rrbracket_{t'+k-1}, \llbracket \mathbf{b} \rrbracket_{t'+k-1}$ inside an authenticated packed triple as random authenticated packed sharings of the form $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$.

^b For each pair of two active parties (P_i, P_j) , if $(P_i, P_j) \in \text{Disp}$, arrange a relay party $P_{i \leftrightarrow j}$ such that $P_{i \leftrightarrow j}$ does not conflict with any of P_i and P_j .

Functionality 7: $\mathcal{F}_{\text{main-client-server}}^*$

1. $\mathcal{F}_{\text{main-client-server}}^*$ ^a receives the input from all clients. Let x denote the input.
2. $\mathcal{F}_{\text{main-client-server}}^*$ computes $C(x)$ and distributes the output to all clients.

^a We write $\mathcal{F}_{\text{main-client-server}}^*$ to emphasize that $\mathcal{F}_{\text{main-client-server}}$ can be invoked only by an adversary corrupting at most a $1/4$ fraction of the servers.

Functionality 8: $\mathcal{F}_{\text{main}}$

1. $\mathcal{F}_{\text{main}}$ receives the input from all parties. Let x denote the input and C denote the circuit.
2. $\mathcal{F}_{\text{main}}$ computes $C(x)$ and distributes the output to all parties.

Lemma 6. Let $k = (n+5)/12$ and $t = (n-1)/2$. MAIN computes $\mathcal{F}_{\text{main}}$ in $\mathcal{F}_{\text{DSS}}, \mathcal{F}_{\text{verify-linear-sharing}}, \mathcal{F}_{\text{coin}}, \mathcal{F}_{\text{main-client-server}}^*$ -hybrid model with statistical security against a fully malicious adversary corrupting at most t parties.

For the proof of Lemma 6, we refer the reader to Section H.5.

Theorem 2. Let n be the number of parties. Let $k = (n+5)/12$ and $t = (n-1)/2$. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2^\kappa$, where κ is the security parameter. For an arithmetic circuit C over $\mathbb{F}$, there exists a statistically secure MPC protocol in the OLE preprocessing model which computes C achieving GOD in the presence of a fully malicious adversary controlling at most t parties. The cost is $\text{P2P}(O(|C| + D \cdot n + C_O \cdot n + n^6 + n^3 \cdot \log |\mathbb{F}|)) + \text{BC}(O(n^6 \cdot \log |\mathbb{F}|))$, along with $O(|C| + D \cdot n + n^5)$ random OLE correlations in total across all pairs of parties, where D and C_O denote the circuit depth and the number of output gates, respectively.

E Instantiating OLE Functionality

Till now, we construct an honest majority MPC achieving GOD at a cost of $O(|C|)$ communication and $O(|C|)$ OLE correlations. To obtain a protocol in the plain model, it remains to instantiate the OLE functionality which appears in $\mathcal{F}_{\text{rand-OLE}}$. To do this, we provide three approaches, including the information-theoretic instantiation with linear cost, instantiation from OT extension protocols with sublinear cost, instantiation from PCGs with sublinear cost, which will be elaborated as follows.

E.1 Information-theoretic Instantiation

To prepare M OLE correlations with information-theoretic security, in the setup phase, all parties first compute M OLE correlations and store them as verifiable secret sharings. To do this, all parties invoke the protocol achieving GOD in honest majority setting [GSZ20] to compute M OLE correlations, which can be viewed as a circuit C_{OLE} of size $O(M)$ defined as follows. All parties stop the computation before the output reconstruction phase of [GSZ20] and hold verifiable secret sharings of M OLE correlations. The cost for instantiating the setup phase is $O(M \cdot n + \text{poly}(n, \kappa))$ elements.

Circuit C_{OLE} : For each $\ell \in [M]$, sample a random OLE correlation by first randomly sampling three elements a_ℓ, b_ℓ, s_ℓ and computing $t_\ell \leftarrow a_\ell \cdot b_\ell - s_\ell$.

Then in the extension phase, once a virtual party emulated by P_i and P_j invokes the functionality for a fresh OLE correlation, parties reconstruct the verifiable secret sharings that store an unused OLE correlation to P_i and P_j . By the property of verifiable secret sharing, the success and correctness of the reconstructions are guaranteed. The amortized cost for providing each OLE correlation for two parties is $O(n)$ elements. Combining this with our MPC protocol in the OLE-hybrid model and observing that OLE correlations are only used in the circuit-independent preprocessing phase, we get the following result.

Corollary 4. *Let n be the number of parties and $t = (n - 1)/2$ denote the number of corrupted parties. Let κ be the security parameter and $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2^\kappa$. For an arithmetic circuit C over $\mathbb{F}$, there exists an information-theoretic MPC protocol which securely computes $\mathcal{F}_{\text{main}}$ in the presence of a fully malicious adversary controlling up to t parties, where $\mathcal{F}_{\text{main}}$ denotes the target functionality computing C with GOD. The communication cost of the protocol is $\text{P2P}(O(|C| + D \cdot n + C_O \cdot n + \text{poly}(n, \kappa))) + \text{BC}(O(n^5 \cdot \log |\mathbb{F}|))$ for the online phase and $\text{P2P}(O(|C| \cdot n + D \cdot n^2 + C_O \cdot n + \text{poly}(n, \kappa))) + \text{BC}(O(n^6 \cdot \log |\mathbb{F}|))$ for the preprocessing phase, where D and C_O denote the circuit depth and the number of output gates, respectively.*

E.2 Instantiation in Minicrypt

Assuming random oracle access, we follow [SY25a] to prepare OLE correlations by leveraging OT extension protocols first proposed in [IKNP03]. At first, parties invoke the protocol achieving GOD in honest majority setting [GSZ20] to compute the base OT instances for every pair of parties which will later be used as seed materials for OT extensions. We present the functionality for base-OT correlations in $\mathcal{F}_{\text{OT}}$. The communication cost is $O(\kappa^2 \cdot n + \text{poly}(n, \kappa))$ elements for every pair of parties.

Functionality 9: $\mathcal{F}_{\text{OT}}(\kappa, s)$

$\mathcal{F}_{\text{OT}}$ running with S as sender, R as receiver, and an adversary, aims to perform a number of s OT instances between S and R with message length of κ bits and proceeds as follows:

1. For each $i \in [s]$, $\mathcal{F}_{\text{OT}}$ waits for input $(\mathbf{v}_{0,i}, \mathbf{v}_{1,i}) \in \mathbb{F}_2^\kappa \times \mathbb{F}_2^\kappa$ from S and $x_i \in \mathbb{F}_2$ from R .
2. For each $i \in [s]$, $\mathcal{F}_{\text{OT}}$ outputs $\mathbf{v}_{x_i,i}$ to R .

With base-OT correlations at hand, every pair of two parties invoke the OT extension protocol with malicious security with abort, such as protocols in [KOS15, Roy22], which extends κ OT correlations with

message length of κ bits to s OT correlations with message length of κ bits to instantiate $\mathcal{F}_{\text{ROT}}(\kappa, s)$ in the $\mathcal{F}_{\text{OT}}(\kappa, \kappa)$ -hybrid model. The communication for extending base OT correlations is $O(s \cdot \kappa)$ bits plus one invocation of $\mathcal{F}_{\text{OT}}(\kappa, \kappa)$.

Functionality 10: $\mathcal{F}_{\text{ROT}}(\kappa, s)$

$\mathcal{F}_{\text{ROT}}$ is parametrized with the length of OT strings κ and the number of s of resulting OTs. Running with S as sender, R as receiver, and an adversary, $\mathcal{F}_{\text{ROT}}(\kappa, s)$ proceeds as follows.

1. $\mathcal{F}_{\text{ROT}}$ waits for input from R , where $x_j \in \mathbb{F}_2$ for all $j \in [s]$, samples random $(\mathbf{v}_{0,j}, \mathbf{v}_{1,j}) \in \mathbb{F}_2^\kappa \times \mathbb{F}_2^\kappa$ for all $j \in [s]$. Then it sends $(\mathbf{v}_{0,j}, \mathbf{v}_{1,j})$ to S and $\mathbf{v}_{x_j,j}$ to R .
2. If R is corrupt: if the adversary inputs **abort**, $\mathcal{F}_{\text{ROT}}$ sends **abort** to S and halts. Otherwise, $\mathcal{F}_{\text{ROT}}$ waits for the adversary to input x_j for all $j \in [s]$. Then $\mathcal{F}_{\text{ROT}}$ samples random $(\mathbf{v}_{0,j}, \mathbf{v}_{1,j})$ for all $j \in [s]$, outputs them to S , and outputs $\mathbf{v}_{x_j,j}$ to the adversary for all $j \in [s]$.
3. If S is corrupt, $\mathcal{F}_{\text{ROT}}$ waits for the adversary to input $(\mathbf{v}_{0,j}, \mathbf{v}_{1,j})$ for all $j \in [s]$, and then outputs as above using these values.

From the extended random OT correlations, every pair of parties invokes the triple generation protocol in [KOS16] which achieves malicious security with abort to compute OLE correlations from OT correlations. The communication cost is $O(M \cdot \ell^2)$ bits plus one invocation of $\mathcal{F}_{\text{ROT}}(\kappa, O(M \cdot \ell))$, where ℓ denotes the length of a field element. Although the protocols for OT extension and generating OLE correlations from OT correlations only achieve security with abort. Notice that when a pair of two honest parties aim to require OLE correlations, the computation is guaranteed to succeed, which ensures that there are enough honest virtual parties participating in packed triple extraction. When the two parties contain at least one corrupted party and the computation fails, all parties will update the set of active virtual parties by considering these two parties as a newly detected dispute pair.

In conclusion, the cost to prepare M OLE correlations is $O(M \cdot \ell \cdot \kappa + M \cdot \ell^2 + \text{poly}(n, \kappa))$ bits, which leads to the following theorem.

Corollary 5. *Let n be the number of parties and $t = (n - 1)/2$ denote the number of corrupted parties. Let κ be the security parameter and $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2^\kappa$ with each element of ℓ bits length. For an arithmetic circuit C over $\mathbb{F}$, there exists an MPC protocol in the random oracle model which securely computes $\mathcal{F}_{\text{main}}$ in the presence of a fully malicious adversary controlling up to t parties, where $\mathcal{F}_{\text{main}}$ denotes the target functionality computing C with GOD. The overall communication cost of the protocol is $\text{P2P}(O((|C| + D \cdot n) \cdot (\kappa + \ell) + C_O \cdot n + \text{poly}(n, \kappa))) + \text{BC}(O(n^6 \cdot \log |\mathbb{F}|))$, where D and C_O denote the circuit depth and the number of outputs, respectively.*

E.3 Instantiation from PCG

Alternatively, we can leverage cryptographic assumptions to generate random OLE correlations. The concept of pseudorandom correlation generators (PCGs) was introduced in [BCG18, BCG⁺19, BCG⁺20], which allows compressing various types of correlated randomness into short PCG seeds, including useful OLE correlations. Applying PCGs to MPC protocols in the correlated randomness model could significantly enhance the efficiency. Specifically, after the PCG seeds are distributed, parties locally expand their own seeds to obtain the target long correlations without interaction, and use these generated correlations for secure computation.

To prepare OLE correlations for our protocol, we apply an efficient PCG for OLE correlations over any finite field proposed in [LXY25]. The authors construct the PCGs based on the quasi-abelian syndrome decoding (QA-SD) assumption and manage to compress M OLE correlations into short seeds of length $O(\kappa^3 \cdot \log M)$, where κ is the security parameter. In their construction, the circuit that generates the PCG seeds is dominated by $O(\kappa \cdot \lambda^2 \cdot \log M)$ calls of a length-doubling PRG, where λ denotes the noise weight. Then to instantiate $\mathcal{F}_{\text{rand-OLE}}$, we invoke the honest majority MPC protocol achieving GOD [GSZ20] to distributively compute the PCG seeds for every pair of parties, which costs communication of $O(\kappa \cdot \lambda^2 \cdot$

$\log M \cdot c \cdot n^3 + \text{poly}(n, \kappa)$ elements and c denotes the circuit size of the PRG. This leads to the following corollary.

Corollary 6. *Let n be the number of parties and $t = (n - 1)/2$ denote the number of corrupted parties. Let κ be the security parameter and $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2^\kappa$. Assuming the hardness of QA-SD and a length-doubling PRG computable by a circuit of size c , for an arithmetic circuit C over $\mathbb{F}$, there exists an MPC protocol which securely computes $\mathcal{F}_{\text{main}}$ in the presence of a fully malicious adversary controlling up to t parties, where $\mathcal{F}_{\text{main}}$ denotes the target functionality computing C with GOD. The overall communication cost of the protocol is $\text{P2P}(O(|C| + D \cdot n + C_O \cdot n + \text{poly}(c, \lambda, n, \kappa))) + \text{BC}(O(n^6 \cdot \log |\mathbb{F}|))$, where λ , D , and C_O denote the noise weight of QA-SD, the circuit depth, and the number of outputs, respectively.*

F Useful Building Blocks

F.1 The Protocols

Protocol 29: RAND(Σ)

Output: $n - t$ random Σ -sharings with $\Sigma \in \{\Sigma_1, \dots, \Sigma_4\}$ defined as follows.

- Σ_1 denotes a pair of random double sharings of the form $([r]_{t'+k-1}, [r]_{t'+2k-2})$.
- Σ_2 denotes a random packed zero-sharing of the form $[o]_{n'-1}$.
- Σ_3 denotes a random sharing $[o]_{n'-1}$ with its secret $\mathbf{o} = (o_1, \dots, o_k)$ satisfying that $\sum_{i=1}^k o_i = 0$.
- Σ_4 denotes a random sharing of the form $[r]_{t'+k-1}$.

1. Each party $P_i \in \mathcal{P}_{\text{active}}$ samples a random Σ -sharing $R^{(i)}$ and distributes it to other parties. For each $P_i \in \text{Corr}$, all parties take an all-zero sharing as $R^{(i)}$.
2. All parties agree on a $(n - t) \times n$ Vandermonde matrix $\mathbf{M}$ and locally compute

$$(R_{(1)}, R_{(2)}, \dots, R_{(n-t)})^\top \leftarrow \mathbf{M} \cdot (R^{(1)}, R^{(2)}, \dots, R^{(n)})^\top$$

and output $R_{(1)}, R_{(2)}, \dots, R_{(n-t)}$.

Protocol 30: BEAVER

Input. $[x]_{t'+k-1}, [y]_{t'+k-1}$.

Output. $[z]_{t'+2k-2}$ with $z = x * y$.

Ingredients. $([a]_{t'+k-1}, [b]_{t'+k-1}, [c]_{t'+2k-2})$.

1. All parties compute $[x + a]_{t'+k-1} \leftarrow [x]_{t'+k-1} + [a]_{t'+k-1}$, $[y + b]_{t'+k-1} \leftarrow [y]_{t'+k-1} + [b]_{t'+k-1}$ and send them to P_{king} .
2. P_{king} reconstructs the secrets $x + a, y + b$ by using the first $t' + k$ shares and distributes $[x + a]_{k-1}, [y + b]_{k-1}$ to all parties.
3. All parties locally compute

$$\begin{aligned} [z]_{t'+2k-2} &\leftarrow [x + a]_{k-1} \cdot [y + b]_{k-1} - [a]_{t'+k-1} \cdot [y + b]_{k-1} \\ &\quad - [x + a]_{k-1} \cdot [b]_{t'+k-1} + [c]_{t'+2k-2}. \end{aligned}$$

Protocol 31: FL-ZEROSHARING

Input: $[o]_{n'-1}$.

Output: a dispute pair.

1. Let P_v be the first party complaining the inconsistency of the zero-sharing $[\mathbf{o}]_{n'-1}$. P_v will act as a referee to locate a dispute pair.
2. For each $i \in \mathcal{P}_{\text{active}}$, P_i distributes a random zero-sharing $[\mathbf{o}'_i]_{n'-1}$. For each $i \in \text{Corr}$, all parties take an all-zero sharing as $[\mathbf{o}'_i]_{n'-1}$.
3. All parties compute and broadcast $[\mathbf{o}']_{n'-1} \leftarrow \sum_{i \in [n]} [\mathbf{o}'_i]_{n'-1}$.
4. If $[\mathbf{o}']_{n'-1}$ is inconsistent, set $[\tilde{\mathbf{o}}]_{n'-1} \leftarrow [\mathbf{o}']_{n'-1}$; otherwise, set $[\tilde{\mathbf{o}}]_{n'-1} \leftarrow [\mathbf{o}]_{n'-1} + [\mathbf{o}']_{n'-1}$.
5. All parties decompose $[\tilde{\mathbf{o}}]_{n'-1}$ as $[\tilde{\mathbf{o}}]_{n'-1} = \sum_{i=1}^n [\tilde{\mathbf{o}}_i]_{n'-1}$, where each $[\tilde{\mathbf{o}}_i]_{n'-1}$ is distributed by P_i . All parties send their shares of $\{[\tilde{\mathbf{o}}_i]_{n'-1}\}_{i \in [n]}$ to P_v (via relay if necessary).
6. P_v first checks whether each P_j 's shares correspond to that has been broadcast by P_j .
 - If not, P_v broadcasts $(j, \text{corrupt}, o_j, o'_j)$ to indicate the broadcast value o_j does not agree with the value o'_j computed from shares sent privately by P_j . The relay party $P_{j \leftrightarrow v}$ also publishes the value it passed. All parties take two adjacent parties broadcasting different values as a new dispute pair.

P_v then locates an inconsistent sharing $[\tilde{\mathbf{o}}_i]_{n'-1}$ and broadcasts $(i, \text{incorrect})$. P_v then sends the inconsistent sharing $[\tilde{\mathbf{o}}_i]_{n'-1}$ to P_i (via relay if necessary).
7. P_i checks the received sharing $[\tilde{\mathbf{o}}_i]_{n'-1}$.
 - If the sharing is consistent, P_i broadcasts $(v, \text{corrupt})$. The relay party $P_{v \leftrightarrow i}$ also publishes its idea.
 - Otherwise, P_i broadcasts $(j, \text{incorrect}, o_j, o'_j)$ to indicate the j -th share o_j distributed by itself to P_j does not agree with o'_j received from P_v . Then $P_{i \leftrightarrow j}, P_j, P_{j \leftrightarrow v}, P_v$ publish the values they believe.

All parties take two adjacent parties broadcasting different ideas as a new dispute pair.

Protocol 32: FAULT-LOCALIZATION(P_s)

Input. The transcripts.

Output. A dispute pair.

1. All parties send their transcripts to P_s (via relay if necessary).
 - If the protocol invokes $\mathcal{F}_{\text{base-DSS}}$ (or $\mathcal{F}_{\text{DSS}}$) with (Share, P_d) , the dealer P_d sends the shares it sent to $\mathcal{F}_{\text{base-DSS}}$ (or $\mathcal{F}_{\text{DSS}}$) and all active parties send shares they received from $\mathcal{F}_{\text{base-DSS}}$ (or $\mathcal{F}_{\text{DSS}}$). By functionality of $\mathcal{F}_{\text{base-DSS}}$, the sharings sent from P_d to $\mathcal{F}_{\text{base-DSS}}$ must agree with that are sent from $\mathcal{F}_{\text{base-DSS}}$ to all active parties.
2. **Check message consistency.** P_s checks whether a message provided by the sender agrees with that provided by the receiver. If not, P_s broadcasts (ℓ, i, j, v, v') to indicate that the ℓ -th message sent by P_i was v while P_j claims v' was received. The relay parties also broadcast the values they passed. All parties take two adjacent parties broadcasting different values are regarded as a dispute pair and halt.
3. **Check local computation.** P_s checks whether each party follows the protocol honestly when performing local computation. If not, P_s broadcasts $(\ell, i, \text{transcript})$ to indicate the ℓ -message conducted by P_i is incorrect, where the computation is shown in **transcript**. The relay party also broadcasts the value it passed. All parties take two adjacent parties broadcasting different values are regarded as a dispute pair and halt.
4. **Check input sharings.** All input sharings are generated by active parties. The active dealer helps P_s to locate a dispute pair as in **FL-ZERO SHARING**.
5. If the protocol invokes $\mathcal{F}_{\text{rand-OLE}}$, extra checks will be conducted as follows.
 - (a) P_s reconstructs the secrets of each packed triple and locates inconsistent OLE correlations. In particular, P_s detects a virtual party containing P_i and P_j distributed an incorrect packed triple with secrets $(\mathbf{a}, \mathbf{b}, \mathbf{c})$. Specifically, P_i distributed $(\mathbf{a}^i, \mathbf{b}^i, \mathbf{w}^i)$ and P_j distributed

- $(\mathbf{a}^j, \mathbf{b}^j, \mathbf{w}^j)$, where the multiplication relation does not hold, i.e., $(\mathbf{a}^i + \mathbf{a}^j) * (\mathbf{b}^i + \mathbf{b}^j) \neq \mathbf{w}^i + \mathbf{w}^j$. Then P_s broadcasts $(\ell, r, a_r^i, b_r^i, w_r^i, a_r^j, b_r^j, w_r^j)$ to indicate the r -th entry of the ℓ -th packed triple is incorrect.
- (b) If $(a_r^i + a_r^j) \cdot (b_r^i + b_r^j) = w_r^i + w_r^j$, all parties take $E \leftarrow \{P_s\}$ as output. Otherwise, all parties identify a dispute pair as follows.
- If both P_i and P_j agree with P_s , all parties take $E \leftarrow \{P_i, P_j\}$ as output.
 - Otherwise, if one of P_i and P_j , say P_i , disagrees with P_s . The relay party $P_{s \leftrightarrow i}$ (if exists) broadcasts its opinion. All parties take two adjacent parties broadcasting different values as output.

F.2 The Functionalities

We present the functionality $\mathcal{F}_{\text{coin}}$, which generates random coins and is instantiated following [BFO12]. The cost of sampling a random element is $\text{BC}(O(\log |\mathbb{F}|))$.

Functionality 11: $\mathcal{F}_{\text{coin}}$

$\mathcal{F}_{\text{coin}}$ receives the set of corrupted parties $\mathcal{C}$ and the set of eliminated parties Corr . $\mathcal{F}_{\text{coin}}$ samples a random field element r and outputs it to all active parties.

We present the functionality $\mathcal{F}_{\text{vrfy-linear-sharing}}$, which checks the consistency of linear secret sharings distributed by an active party and satisfying that the whole sharings are determined by shares of honest parties. This functionality can be realized by checking a random linear combination and letting the dealer point out a corrupted party in case of inconsistency. The communication to check m Σ -sharings is $\text{P2P}(O(n^2 \cdot |\Sigma|)) + \text{BC}(O(n + \log |\Sigma| + \log |\mathbb{F}|))$, where $|\Sigma|$ denotes the share size of a Σ -sharing.

Functionality 12: $\mathcal{F}_{\text{vrfy-linear-sharing}}$

$\mathcal{F}_{\text{vrfy-linear-sharing}}$ verifies the consistency of the following types of linear secret sharings.

- Σ_1 : degree- $(k-1)$ packed Shamir sharing of the form $[\mathbf{u}]_{k-1}$.
 - Σ_2 : degree- $(2k-2)$ packed Shamir sharing of the form $[\mathbf{o}]_{2k-2}$ with the secret $\mathbf{o} = (o_1, \dots, o_k)$ satisfying that $o_1 + \dots + o_k = 0$.
 - Σ_3 : a group of $2k$ degree- $(k-1)$ packed Shamir sharings in the form of $(\{[\mathbf{u}_{\ell*}]_{k-1}\}_{\ell=1}^k, \{[\mathbf{u}_{*1}]_{k-1}\}_{\ell=1}^k)$.
1. $\mathcal{F}_{\text{vrfy-linear-sharing}}$ receives the set of corrupted parties $\mathcal{C}$, the set of eliminated parties Corr , and the set of dispute pairs Disp .
 2. $\mathcal{F}_{\text{vrfy-linear-sharing}}$ receives from honest parties their shares and sends them to the adversary.
 3. $\mathcal{F}_{\text{vrfy-linear-sharing}}$ checks the consistency of the sharings. In case of inconsistency, $\mathcal{F}_{\text{vrfy-linear-sharing}}$ receives a dispute pair E such that $E \notin \text{Disp}$ from the adversary and output it to all active parties. Otherwise, $\mathcal{F}_{\text{vrfy-linear-sharing}}$ waits for an instruction from the adversary.
 - Upon receiving (reject, E) such that $E \notin \text{Disp}$, output E to all active parties.
 - Upon receiving accept , output accept to all active parties.

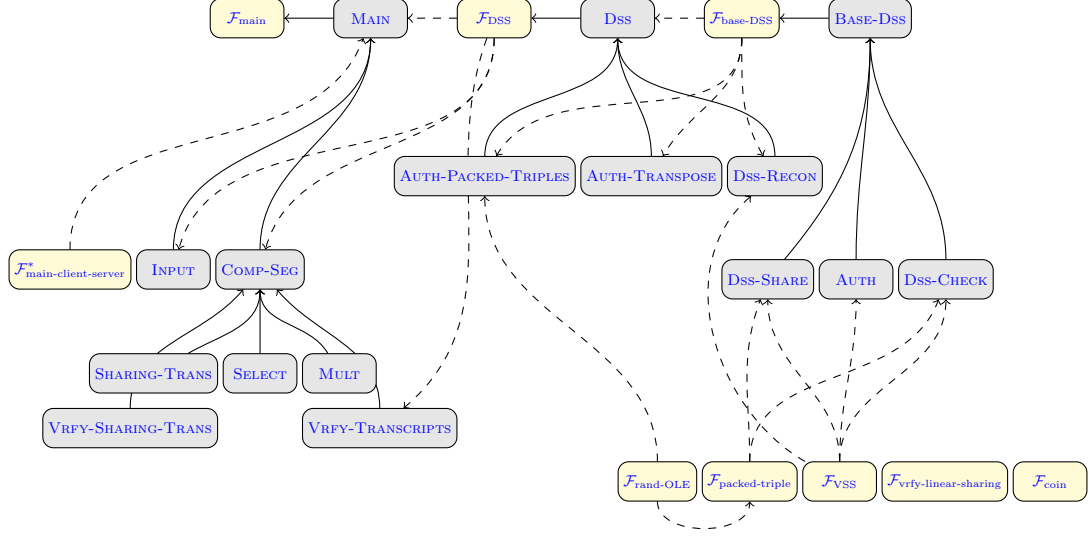


Fig. 2: Dependency graph of our honest majority MPC protocol.

G Figures

Table 1: **Table of notations.**

Notation	Description	Reference
$\mathcal{P}_{\text{active}}$	Set of active parties	DISPUTE-CONTROL
$\mathcal{C}_{\text{corr}}$	Set of eliminated parties	DISPUTE-CONTROL
$\mathcal{D}_{\text{disp}}$	Set of dispute pairs	DISPUTE-CONTROL
$*$	Element-wise multiplication	MULT
$\odot$	Inner product	COMPUTE-TAG
$\langle \cdot j \rangle_t$	Degree- t Shamir secret sharing with the secret placed at the j -th position and maintained by $\mathcal{F}_{\text{VSS}}$	$\mathcal{F}_{\text{VSS}}$
$\llbracket \cdot \rrbracket_d$	Degree- d packed Shamir sharing maintained by $\mathcal{F}_{\text{DSS}}$	$\mathcal{F}_{\text{base-DSS}}, \mathcal{F}_{\text{DSS}}$

H Security Proofs

H.1 Proof of Lemma 1

Lemma 7 (Restatement of Lemma 1). *BASE-DSS computes $\mathcal{F}_{\text{base-DSS}}$ with statistical security against any fully malicious adversary controlling up to t parties in the $\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{packed-triple}}, \mathcal{F}_{\text{coin}}$ -hybrid model.*

Proof. We construct a simulator, $\text{Sim}_{\text{base-dss}}$, that interacts with $\mathcal{F}_{\text{base-DSS}}$, such that no environment $\mathcal{Z}$ can distinguish between an interaction with $\text{Sim}_{\text{base-dss}}$ and an interaction with the adversary and BASE-DSS.

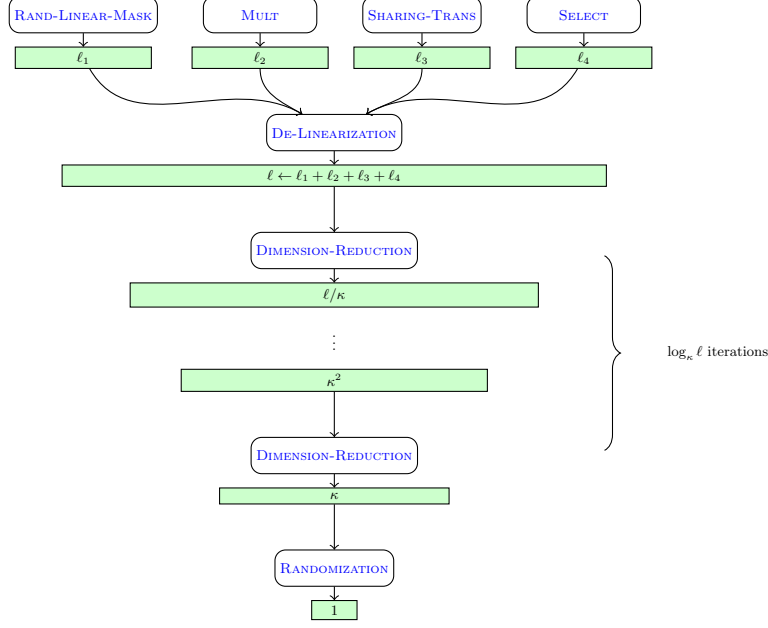


Fig. 3: Framework of recursive verification procedure.

Simulator 1: $\text{Sim}_{\text{base-DSS}}$ for **BASE-DSS**

1. In **setup** phase, in Step 3, Sim emulates $\mathcal{F}_{\text{VSS}}$ and records shares of corrupted parties. In Step 5, Sim invokes $\text{Sim}_{\text{rand-tag}}(\gamma)$. In case of failure, Sim invokes **DISPUTE-CONTROL** to update $\mathcal{P}_{\text{active}}$, Corr , and Disp .
 - If $|\mathcal{P}_{\text{active}}| < t' + 2k - 1$, Sim sends **reject** and $\mathcal{C}' \leftarrow \text{Corr}$ to $\mathcal{F}_{\text{DSS}}$.
2. If **DSS-SHARE**(P_d, γ) is invoked, Sim invokes $\text{Sim}_{\text{DSS-share}}(P_d, \gamma)$.
3. If **AUTH**(P_r, γ) is invoked, Sim invokes $\text{Sim}_{\text{auth}}(P_r, \gamma)$.
4. If **DSS-CHECK**($\gamma, \llbracket \mathbf{w} \rrbracket_{t'+k-1}$) is invoked, Sim invokes $\text{Sim}_{\text{DSS-check}}(\gamma, \llbracket \mathbf{w} \rrbracket_{t'+k-1})$.
5. Sim outputs what the adversary outputs and honest parties output what they receive from $\mathcal{F}_{\text{base-DSS}}$.

Simulator 2: $\text{Sim}_{\text{DSS-share}}(P_d, \gamma)$ for **DSS-SHARE**

- If P_d is honest,
1. In Step 1, Sim receives shares of active corrupted parties from $\mathcal{F}_{\text{base-DSS}}$, denoted by $\{\sigma_i^{(\ell)}\}_{i \in \mathcal{C} \setminus \text{Corr}, \ell \in [m]}$, randomly samples $\{\sigma_i^{(m+1)}\}_{i \in \mathcal{C} \setminus \text{Corr}}$, and sends them to the adversary on behalf of honest P_d .
 2. In Step 2, Sim emulates $\mathcal{F}_{\text{VSS}}$ and records shares of active corrupted parties.
 3. In Step 3, Sim invokes $\text{Sim}_{\text{compute-tag}}(P_i)$ for each P_i . If P_i is corrupted, suppose P_i authenticates the message $\overline{\sigma_i^{(\ell)}}$ and obtains its tag $y_i^{(\ell)}$ for $\ell \in [m]$.
 4. In Step 4.1, Sim emulates $\mathcal{F}_{\text{coin}}$ by sampling a random element λ .
 5. In Step 4.2, for each honest party P_i , Sim samples random elements as (σ_i, y_i) and broadcasts them on behalf of honest parties.
 6. In Step 4.3, for each honest party P_i , Sim emulates $\mathcal{F}_{\text{VSS}}$ by outputting **accept**. For each active corrupted party P_i , Sim emulates $\mathcal{F}_{\text{VSS}}$ by checking the following requirements.
 - σ_i broadcast by P_i corresponds to $\overline{\sigma_i^{(m+1)}} + \sum_{\ell=1}^m \lambda^\ell \cdot \overline{\sigma_i^{(\ell)}}$.

– y_i broadcast by P_i corresponds to $y_i^{(m+1)} + \sum_{\ell=1}^m \lambda^\ell \cdot y_i^{(\ell)}$.

If both requirements are satisfied, **Sim** emulates $\mathcal{F}_{\text{VSS}}$ by outputting **accept**. Otherwise, **Sim** samples honest parties' shares of $\{\langle \mu_j | j \rangle_t\}_{j \in [k]}$ and $\{\langle \delta(\sigma_i^{(\ell)}) | 1 \rangle_t\}_{\ell \in [m+1]}$ and honestly follows the protocol. However, if the verification passes, **Sim** outputs $\perp$ and halts.

7. In Step 4.4, **Sim** checks active corrupted parties' share vectors.
8. In Step 4.5, **Sim** honestly follows the protocol.
9. In Step 5.1, **Sim** samples random elements as $\{\gamma_j - \mu_j\}_{j \in [k]}$ and emulates $\mathcal{F}_{\text{VSS}}$.

If P_d is corrupted,

10. In Step 1, **Sim** receives each honest party P_i 's shares from the adversary, denoted as $\sigma_i^{(1)}, \dots, \sigma_i^{(m+1)}$.
11. In Steps 2, 3, and 4.1, **Sim** simulates as Steps 2, 3, and 4, respectively.
12. In Step 4.2, **Sim** honestly follows the protocol.
13. In Step 4.3, same as Step 6.
14. In Steps 4.4 and 4.5, **Sim** honestly follows the protocol.
15. In Step 5.1, same as Step 9.
16. At the end of the simulation, if P_i is corrupted, **Sim** learns its share vectors with tags, $(\sigma_i^{(\ell)}, \eta(\sigma_i^{(\ell)}))_{\ell=1}^m$.

Simulator 3: $\text{Sim}_{\text{compute-tag}}(P_i, \mu)$ for **COMPUTE-TAG**

1. In Step 1.1, **Sim** emulates $\mathcal{F}_{\text{VSS}}$ and records shares of $\{\langle \delta(\mathbf{x}^{(\ell)}) | 1 \rangle_t\}_{\ell \in [m+1]}$ of active corrupted parties.
2. In Step 1.2, **Sim** emulates $\mathcal{F}_{\text{packed-triple}}$ and records shares of $\{[\mathbf{a}^{(\ell)}]_{t'+k-1}, [\mathbf{b}^{(\ell)}]_{t'+k-1}, [\mathbf{c}^{(\ell)}]_{t'+2k-2}\}_{\ell \in [m+1]}$ of active corrupted parties.
3. In Step 1.3, **Sim** invokes $\text{Sim}_{\text{rand}}(\Sigma_3)$.

If P_i is honest,

4. In Step 2.1, **Sim** samples random elements as active corrupted parties' shares of $[\mathbf{x}^{(\ell)}]_{t'+k-1}$ and sends them to the adversary on behalf of honest P_i .
5. In Step 2.2, **Sim** computes corrupted parties' shares of $[\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{t'+k-1}, [\mu + \mathbf{b}^{(\ell)}]_{t'+k-1}$, randomly samples the whole sharings based on shares of corrupted parties, and sends shares of honest parties to P_{king} on behalf of honest parties. If P_{king} is honest, **Sim** follows the protocol honestly. **Sim** receives shares of $[\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{k-1}, [\mu + \mathbf{b}^{(\ell)}]_{k-1}$ of honest parties from P_{king} .
6. In Step 2.3, **Sim** receives active corrupted parties' shares of $[\mathbf{y}^{(\ell)}]_{n'-1}$ from the adversary, denoted by $[\widetilde{\mathbf{y}^{(\ell)}}]_{n'-1}$.
7. **Sim** first computes the expected corrupted parties' shares of $[\mathbf{y}^{(\ell)}]_{n'-1}$, denoted by $[\overline{\mathbf{y}^{(\ell)}}]_{n'-1}$, and then computes $\mathbf{d}^{(\ell)}, \mathbf{e}^{(\ell)}$, and $\epsilon^{(\ell)}$ as

$$\begin{aligned} \mathbf{d}^{(\ell)} &\leftarrow \overline{\mu + \mathbf{b}^{(\ell)}} - (\mu + \mathbf{b}^{(\ell)}), \\ \mathbf{e}^{(\ell)} &\leftarrow \overline{\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}} - (\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}), \\ \epsilon^{(\ell)} &\leftarrow (\widetilde{\mathbf{y}^{(\ell)}} - \overline{\mathbf{y}^{(\ell)}}) \odot \mathbf{1}. \end{aligned}$$

8. In Step 3.1, **Sim** emulates $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_1)$ by checking whether shares of honest parties form valid degree- $(k-1)$ packed Shamir sharings. If not, **Sim** receives a new dispute pair from the adversary, outputs it, and aborts on behalf of honest parties. Otherwise, **Sim** waits for an instruction from the adversary. On receiving **accept**, output **accept**; on receiving **(reject, E)**, output E and abort on behalf of honest parties.

9. In Step 3.2, Sim follows the protocol honestly.
10. In Step 3.3, Sim first decides whether the verification passes by checking whether the following requirements are satisfied.

For $\ell \in [m+1]$,

- If $\mathbf{x}^{(\ell)}$ is unknown to the adversary, $\mathbf{d}^{(\ell)} = \mathbf{e}^{(\ell)} = \mathbf{0}$ and $\epsilon^{(\ell)} = 0$.
- If $\mathbf{x}^{(\ell)}$ is known to the adversary, $\mathbf{e}^{(\ell)} = \mathbf{0}$ and $\mathbf{x}^{(\ell)} \odot \mathbf{d}^{(\ell)} + \epsilon^{(\ell)} = 0$.

Sim simulates $(\mathbf{x}, y)$ depending on whether the verification succeeds.

- If all requirements are satisfied, Sim samples random elements as $(\mathbf{x}, y)$. In Step 3.5, Sim emulates $\mathcal{F}_{\text{VSS}}$ by outputting **accept**.
- Otherwise, Sim samples honest parties' shares of $\{\langle \mu_j | j \rangle_t\}_{j \in [k]}$ and $\{\mathbf{x}^{(\ell)}\}_{\ell \in [m+1]}$. Sim then honestly follows the protocol and emulates $\mathcal{F}_{\text{VSS}}$ in Step 3.5. However, if the verification passes, Sim outputs $\perp$ and halts.

If P_i is corrupted,

11. In Step 2.1, Sim receives honest parties' shares of $[\mathbf{x}^{(\ell)}]_{t'+k-1}$ from the adversary.
12. In Step 2.2, Sim samples the whole sharing $[\mathbf{a}^{(\ell)}]_{t'+k-1}$ based on shares of corrupted parties and computes honest parties' shares of $[\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{t'+k-1}$. Sim computes corrupted parties' shares of $[\boldsymbol{\mu} + \mathbf{b}^{(\ell)}]_{t'+k-1}$ and samples the whole sharing based on shares of corrupted parties. Sim then sends honest parties' shares of $[\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{t'+k-1}$, $[\boldsymbol{\mu} + \mathbf{b}^{(\ell)}]_{t'+k-1}$ to P_{king} . If P_{king} is honest, Sim honestly follows the protocol. Sim receives shares of $[\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{k-1}$, $[\boldsymbol{\mu} + \mathbf{b}^{(\ell)}]_{k-1}$ of honest parties from P_{king} and computes $\mathbf{x}^{(\ell)} \leftarrow (\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}) - \mathbf{a}^{(\ell)}$.
13. In Step 2.3, Sim samples random elements as honest parties' shares of $[\mathbf{y}^{(\ell)}]_{n'-1}$ and sends them to the adversary.
14. Sim first computes $\overline{\mathbf{y}^{(\ell)}}$ and then computes $\mathbf{d}^{(\ell)}, \epsilon^{(\ell)}$ as

$$\begin{aligned} \mathbf{d}^{(\ell)} &\leftarrow \overline{\boldsymbol{\mu} + \mathbf{b}^{(\ell)}} - (\boldsymbol{\mu} + \mathbf{b}^{(\ell)}), \\ \epsilon^{(\ell)} &\leftarrow \mathbf{x}^{(\ell)} \odot \mathbf{d}^{(\ell)}. \end{aligned}$$

15. In Step 3.1, Sim emulates $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_1)$ as in Step 8.
16. In Step 3.2, Sim receives $(\mathbf{x}, y)$ broadcast by the adversary.
17. In Step 3.5, Sim emulates $\mathcal{F}_{\text{VSS}}$ depending on whether the following requirements are satisfied.
 - $\mathbf{x}$ corresponds to $\mathbf{x}^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot \mathbf{x}^{(\ell)}$.
 - y corresponds to $\overline{y^{(m+1)}} - \epsilon^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot (\overline{y^{(\ell)}} - \epsilon^{(\ell)})$.

If all requirements are satisfied, Sim outputs **accept**. Otherwise, Sim samples honest parties' shares of $\{\langle \mu_j | j \rangle_t\}_{j \in [k]}$ and follows the protocol honestly. However, if the verification passes, Sim outputs $\perp$ and halts.

18. At the end of the simulation, if P_i is corrupted, Sim learns the authenticated messages and the tags, $(\mathbf{x}^{(\ell)}, y^{(\ell)})_{\ell=1}^m$.

Simulator 4: $\text{Sim}_{\text{auth}}(P_r, \gamma)$ for **AUTH**

Sim receives shares of corrupted parties from $\mathcal{F}_{\text{base-DSS}}$ and computes the whole sharings $[\mathbf{X}^{(1)}]_{k-1}, \dots, [\mathbf{X}^{(m)}]_{k-1}$.

1. In Step 1, if P_r is honest, Sim honestly follows the protocol. If P_r is corrupted, Sim receives honest parties' shares of $[\sigma_i^{(\ell)}]_{k-1}$ for $i \in \mathcal{P}_{\text{active}}, \ell \in [m]$.
2. In Steps 2.1 and 2.2, Sim emulates $\mathcal{F}_{\text{VSS}}$ and records shares of active corrupted parties.
3. In Step 2.3, Sim invokes $\text{Sim}_{\text{rand}}(\Sigma_3)$.

4. In Step 3.1, if P_i is corrupted, Sim samples random elements as honest parties' shares of $[\overline{\mathbf{y}_i^{(\ell)}}]_{n'-1}$ and sends them to the adversary. If P_i is honest, Sim receives active corrupted parties' shares of $[\mathbf{y}_i^{(\ell)}]_{n'-1}$, denoted by $[\widetilde{\mathbf{y}_i^{(\ell)}}]_{n'-1}$.
5. For each corrupted P_i , Sim computes $\overline{y_i^{(\ell)}}$. For each honest P_i , Sim computes the expected corrupted parties' shares of $[\mathbf{y}_i^{(\ell)}]_{n'-1}$, denoted by $[\widetilde{\mathbf{y}_i^{(\ell)}}]_{n'-1}$, and computes $\epsilon_i^{(\ell)} \leftarrow (\mathbf{y}_i^{(\ell)} - \overline{\mathbf{y}_i^{(\ell)}}) \odot \mathbf{1}$.
6. In Step 4.1, Sim emulates $\mathcal{F}_{\text{VSS}}$ by sampling a random element λ .
7. In Step 4.2, Sim honestly follows the protocol.
8. In Step 4.3.a, if P_i is honest, Sim samples a random element as y_i .
9. In Step 4.3.b, for each active party P_i ,
 - if P_i is honest, Sim checks whether $\epsilon_i^{(\ell)} = 0$ for each $\ell \in [m]$;
 - if P_i is corrupted, Sim checks whether $y_i^{(\ell)}$ broadcast by P_i corresponds to $\sum_{\ell=1}^m \lambda^{\ell-1} \cdot \overline{y_i^{(\ell)}}$ for each $\ell \in [m]$.

If yes, Sim emulates $\mathcal{F}_{\text{VSS}}$ by outputting **accept**. Otherwise, Sim samples honest parties' shares of $(\{\langle \mu_j | j \rangle_t\}_{j=1}^k, \{\langle \mu_j | 1 \rangle_t\}_{j=2}^k)$ and $\{\langle \delta(\sigma_i^{(\ell)}) | 1 \rangle_t\}_{\ell \in [m]}$ and follows the protocol honestly. However, if the verification passes, Sim outputs $\perp$ and halts.
10. In Step 5.1, Sim emulates $\mathcal{F}_{\text{VSS}}$ by sampling random elements as $\{\gamma_j - \mu_j\}_{j \in [k]}$.

Simulator 5: $\text{Sim}_{\text{DSS-check}}(\gamma, \llbracket \mathbf{w} \rrbracket_{t'+k-1})$ for **DSS-CHECK**

Sim computes active corrupted parties' shares of $\llbracket \mathbf{w} \rrbracket_{t'+k-1}$.

1. In Step 3.1, if P_i is honest, Sim samples a random element as z_i .
2. In Step 3.2, Sim emulates $\mathcal{F}_{\text{coin}}$ by sampling a random element α . If $\alpha = 0$, Sim outputs $\perp$ and halts.
3. In Step 3.3, if P_i is honest, Sim randomly samples elements as $(\mathbf{y}_i^{(\ell)}, \eta(\mathbf{y}_i^{(\ell)}))_{\ell=1}^k$.
4. In Step 4.1, Sim honestly follows the protocol.
5. In Step 4.2, if P_i is honest, Sim emulates $\mathcal{F}_{\text{VSS}}$ by outputting **accept**. If P_i is corrupted, Sim checks whether
 - $\mathbf{y}_i^{(\ell)}$ broadcast by P_i corresponds to $\mathbf{z}_i^{(\ell)} + \alpha \cdot \sigma_i^{(\ell)}$
 - $\eta(\mathbf{y}_i^{(\ell)})$ corresponds to $\eta(\mathbf{z}_i^{(\ell)}) + \alpha \cdot \eta(\sigma_i^{(\ell)})$.

If yes, Sim emulates $\mathcal{F}_{\text{VSS}}$ by outputting **accept**. Otherwise, Sim outputs **reject**.
6. Sim interacts with $\mathcal{F}_{\text{base-DSS}}$. For each active corrupted party P_i ,
 - if P_i provides correct authenticated messages with tags $(\mathbf{y}_i^{(\ell)}, \eta(\mathbf{y}_i^{(\ell)}))_{\ell=1}^k$, Sim sends the correct share to $\mathcal{F}_{\text{base-DSS}}$.
 - Otherwise, Sim sends an incorrect value to $\mathcal{F}_{\text{base-DSS}}$.

Simulator 6: $\text{Sim}_{\text{rand-tag}}(\gamma)$ for **RAND-TAG**

1. In Step 1, Sim emulates $\mathcal{F}_{\text{VSS}}$ and records shares of active corrupted parties.
2. In Step 2, Sim invokes $\text{Sim}_{\text{compute-tag}}(P_i)$ for each active party P_i . If P_i is corrupted, suppose P_i authenticates the message $\mathbf{z}_i^{(\ell)}$ and obtains its tag.
3. In Step 3.a, Sim samples random elements as $\{\gamma_j - \mu_j\}_{j \in [k]}$ and emulates $\mathcal{F}_{\text{VSS}}$.
4. At the end of the simulation, if P_i is corrupted, Sim learns its share vectors with tags, $(\mathbf{z}_i^{(\ell)}, \eta(\mathbf{z}_i^{(\ell)}))_{\ell=1}^m$.

Simulator 7: $\text{Sim}_{\text{rand}}(\Sigma)$ for $\text{RAND}(\Sigma)$

1. For each honest party P_i , Sim samples random elements as active corrupted parties' shares and sends them to the adversary on behalf of honest P_i . For each corrupted party P_i , Sim receives shares of honest parties from the adversary.

Lemma 8. *If P_i invokes $\text{COMPUTE-TAG}(P_i, \mu)$ with messages $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ and all parties pass the **verification of tag computation**, then P_i obtains correctly computed authentication tags for each $\mathbf{x}_\ell$ under the long-term key μ . If this is the case, we say P_i has authenticated the message $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$.*

Proof. We continue the notations defined in $\text{Sim}_{\text{compute-tag}}(P_i, \mu)$. If P_i is an honest party, P_i will provide $\tilde{y}$ in **verification of tag computation**, where $\tilde{y}$ can be computed as

$$\begin{aligned} & \widetilde{y^{(m+1)}} + \sum_{\ell=1}^m \alpha^\ell \cdot \widetilde{y^{(\ell)}} \\ &= \mathbf{x} \odot \mu + (\mathbf{x}^{(m+1)} \odot \mathbf{d}^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot \mathbf{x}^{(\ell)} \odot \mathbf{d}^{(\ell)}) \\ & \quad + (\mathbf{e}^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot \mathbf{e}^{(\ell)}) \odot \mu + (w^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot w^{(\ell)}) + \delta, \end{aligned}$$

where $\mathbf{x} = \mathbf{x}^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot \mathbf{x}^{(\ell)}$, $w^{(\ell)} = \mathbf{d}^{(\ell)} \odot \mathbf{e}^{(\ell)} + \epsilon^{(\ell)}$, and $\delta = \delta^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot \delta^{(\ell)}$. If the verification passes, by verifiability of μ and δ , it holds that

$$\begin{aligned} 0 &= (\mathbf{x}^{(m+1)} \odot \mathbf{d}^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot \mathbf{x}^{(\ell)} \odot \mathbf{d}^{(\ell)}) + (\mathbf{e}^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot \mathbf{e}^{(\ell)}) \odot \mu \\ & \quad + (w^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot w^{(\ell)}). \end{aligned}$$

There are two cases depending on whether the authenticated messages are private.

(i) If the messages $\{\mathbf{x}^{(\ell)}\}_{\ell=1}^m$ are unknown to the adversary, by randomness of $\{\mathbf{x}^{(\ell)}\}_{\ell \in [m+1]}$ and μ , we know

- $\mathbf{d}^{(\ell)} = \mathbf{0}$, for $\ell \in [m+1]$.
- $\mathbf{e}^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot \mathbf{e}^{(\ell)} = \mathbf{0}$. By randomness of α , $\mathbf{e}^{(\ell)} = \mathbf{0}$ for $\ell \in [m+1]$.
- $w^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot w^{(\ell)} = 0$. By randomness of α , $w^{(\ell)} = 0$ and thus $\epsilon^{(\ell)} = 0$ for $\ell \in [m+1]$.

(ii) If the messages $\{\mathbf{x}^{(\ell)}\}_{\ell=1}^m$ are known to the adversary, by randomness of $\mathbf{x}^{(m+1)}$ and μ , we know

- $\mathbf{d}^{(m+1)} = \mathbf{0}$.
- $\mathbf{e}^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot \mathbf{e}^{(\ell)} = \mathbf{0}$. By randomness of α , $\mathbf{e}^{(\ell)} = \mathbf{0}$ for $\ell \in [m+1]$.
- $w^{(m+1)} + \sum_{\ell=1}^m \alpha^\ell \cdot (\mathbf{x}^{(\ell)} \odot \mathbf{d}^{(\ell)} + w^{(\ell)}) = 0$. By randomness of α , $w^{(m+1)} = 0$ and $\mathbf{x}^{(\ell)} \odot \mathbf{d}^{(\ell)} + w^{(\ell)} = 0$ for $\ell \in [m]$.

For both cases, for $\ell \in [m+1]$, $\widetilde{y^{(\ell)}} = \mathbf{x}^{(\ell)} \odot \mu + \delta^{(\ell)}$, which implies the authentication tag for each message $\mathbf{x}^{(\ell)}$ is correctly computed.

If P_i is corrupted, for each $\ell \in [m+1]$, P_i learnt that

$$\overline{y^{(\ell)}} = \mathbf{x}^{(\ell)} \odot \mu + \delta^{(\ell)} + \mathbf{x}^{(\ell)} \odot \mathbf{d}^{(\ell)},$$

where $\mathbf{x}^{(\ell)}$ and $\mathbf{d}^{(\ell)}$ are known to the adversary. This implies P_i has learnt the correct authentication tag for $\mathbf{x}^{(\ell)}$ under the long-term key μ , which is $\overline{y^{(\ell)}} - \mathbf{x}^{(\ell)} \odot \mathbf{d}^{(\ell)}$. $\square$

Lemma 9. *If dealer P_d successfully distributes packed sharings through $\text{DSS-SHARE}(P_d, \gamma)$, where all parties pass the **verifying authentication**, then each party has authenticated their shares, except with negligible probability. Furthermore, the following requirements regarding these values are satisfied.*

- *If the dealer is honest, then each authenticated share corresponds to the one distributed by the honest dealer.*
- *If the dealer is corrupted, then the authenticated shares of all parties form valid degree- $(t' + k - 1)$ packed Shamir sharings.*

Proof. If the dealer P_d is honest, for each active corrupted party P_i , in **verifying authentication**, to pass the verification in Step 4.3 of DSS-SHARE , by randomness of μ plus verifiability of μ and δ_i , P_i must provide $\bar{\sigma}_i$ such that $\bar{\sigma}_i = \sigma_i^{(m+1)} + \sum_{\ell=1}^m \lambda^\ell \cdot \sigma_i^{(\ell)}$.

To pass the verification in Step 4.4 of DSS-SHARE , it should hold that $\bar{\sigma}_i = \sigma_i^{(m+1)} + \sum_{\ell=1}^m \lambda^\ell \cdot \sigma_i^{(\ell)}$. By randomness of λ and Schwartz-Zippel Lemma, $\bar{\sigma}_i^{(\ell)} = \sigma_i^{(\ell)}$ for each $\ell \in [m+1]$. Hence, P_i must authenticate the message $\sigma_i^{(\ell)}$ for each $\ell \in [m]$, which is the share vector distributed by the honest dealer P_d .

If the dealer P_d is corrupted, for each active corrupted party P_i , in **verifying authentication**, to pass the verification in Step 4.3 of DSS-SHARE , P_i must provide $\bar{\sigma}_i$ such that $\bar{\sigma}_i = \sigma_i^{(m+1)} + \sum_{\ell=1}^m \lambda^\ell \cdot \sigma_i^{(\ell)}$. Meanwhile, each honest party P_i will provide $\sigma_i = \sigma_i^{(m+1)} + \sum_{\ell=1}^m \lambda^\ell \cdot \sigma_i^{(\ell)}$.

By randomness of λ and Schwartz-Zippel Lemma, we know if $(\{\sigma_i^{(\ell)}\}_{i \in \mathcal{C} \setminus \text{Corr}}, \{\sigma_i^{(\ell)}\}_{i \in \mathcal{H}})$ does not form a valid degree- $(t' + k - 1)$ packed Shamir sharing for some ℓ , $(\{\bar{\sigma}_i\}_{i \in \mathcal{C} \setminus \text{Corr}}, \{\sigma_i\}_{i \in \mathcal{H}})$ will not form a valid degree- $(t' + k - 1)$ packed Shamir sharing, except with negligible probability.

Then if all verifications in DSS-SHARE have been passed, each party learns their authentication tags with respect to the long-term key γ . $\square$

Lemma 10. *If all parties successfully compute authentication tags for degree- $(k - 1)$ packed Shamir sharing through $\text{AUTH}(P_r, \gamma)$, where all parties pass the **verification of tag computation**, then each party has authenticated its shares, except with negligible probability. Furthermore, the authenticated shares correspond to the shares of degree- $(k - 1)$ packed Shamir sharings distributed by P_r .*

Proof. We continue the notations defined in Sim_{auth} . By randomness of λ and Schwartz-Zippel Lemma, we know if $\{[\sigma_i^{(\ell)}]_{k-1}\}_{i \in \mathcal{P}_{\text{active}}}$ and $[\mathbf{X}^{(\ell)}]_{k-1}$ do not satisfy the following requirements for some $\ell \in [m]$

- for $i \in \mathcal{P}_{\text{active}}$, all parties' shares of $[\sigma_i^{(\ell)}]_{k-1}$ form a valid degree- $(k - 1)$ packed Shamir sharing;
- for $i \in \mathcal{P}_{\text{active}}$, the secret of $[\sigma_i^{(\ell)}]_{k-1}$ corresponds to the i -th share vector of $[\mathbf{X}^{(\ell)}]_{k-1}$,

then except with negligible probability, one of the following requirements will not be satisfied by $\{[\bar{\sigma}_i]_{k-1}\}_{i \in \mathcal{P}_{\text{active}}}$ and $[\mathbf{X}]_{k-1}$,

- for $i \in \mathcal{P}_{\text{active}}$, all parties' shares of $[\bar{\sigma}_i]_{k-1}$ form a valid degree- $(k - 1)$ packed Shamir sharing;
- for $i \in \mathcal{P}_{\text{active}}$, the secret of $[\bar{\sigma}_i]_{k-1}$ corresponds to the i -th share vector of $[\mathbf{X}]_{k-1}$.

Hence, if the verification in Step 4.2 of AUTH has been passed, then for each $i \in \mathcal{P}_{\text{active}}$ and $\ell \in [m]$, we have $\bar{\sigma}_i^{(\ell)} = \sigma_i^{(\ell)}$.

For each honest party P_i , P_i provides $\tilde{y}_i$ in Step 4.3.a, where $\tilde{y}_i$ can be computed as

$$\tilde{y}_i = \sum_{\ell=1}^m \lambda^{\ell-1} \cdot \widetilde{y_i^{(\ell)}} = \sigma_i \odot \mu + \delta_i + \sum_{\ell=1}^m \lambda^{\ell-1} \cdot \epsilon_i^{(\ell)},$$

where $\sigma_i = \sum_{\ell=1}^m \lambda^{\ell-1} \cdot \sigma_i^{(\ell)}$ and $\delta_i = \sum_{\ell=1}^m \lambda^{\ell-1} \cdot \delta_i^{(\ell)}$. Then to pass the verification in Step 4.3.b, it should be satisfied that $\sum_{\ell=1}^m \lambda^{\ell-1} \cdot \epsilon_i^{(\ell)} = 0$. By randomness of λ , $\epsilon_i^{(\ell)} = 0$ for each $\ell \in [m]$. Hence, the authentication tag for P_i 's share vector $\sigma_i^{(\ell)}$ is correctly computed.

For each active corrupted party P_i , during the computation and for each $\ell \in [m]$, P_i learnt that

$$\mathbf{y}_i^{(\ell)} = \boldsymbol{\sigma}_i^{(\ell)} \odot \boldsymbol{\mu} + \delta_i^{(\ell)}.$$

Then if all verifications in **Auth** have been passed, after revealing the difference $\boldsymbol{\gamma} - \boldsymbol{\mu}$, each party learns their authentication tags with respect to the long-term key $\boldsymbol{\gamma}$. $\square$

Lemma 11. *If P_i passes **DSS-CHECK**($\boldsymbol{\gamma}, \llbracket \mathbf{w} \rrbracket_{t'+k-1}$), then P_i must provide a correct share w_i , which is a linear combination of authenticated shares. Otherwise, P_i already learns some information regarding the long-term secret key $\boldsymbol{\gamma}$, except with negligible probability.*

Proof. Notice P_i has authenticated $2k$ messages $\mathbf{w}_i^{(1)}, \dots, \mathbf{w}_i^{(k)}, \mathbf{z}_i^{(1)}, \dots, \mathbf{z}_i^{(k)}$ in **DSS-SHARE**, **Auth**, and **RAND-TAG**. First we prove that to pass the verification in Step 4.b, P_i must provide $\mathbf{y}_i^{(\ell)} \leftarrow \mathbf{z}_i^{(\ell)} + \alpha \cdot \boldsymbol{\sigma}_i^{(\ell)}$ and $\eta(\mathbf{y}_i^{(\ell)}) \leftarrow \eta(\mathbf{z}_i^{(\ell)}) + \alpha \cdot \eta(\boldsymbol{\sigma}_i^{(\ell)})$ for each $\ell \in [k]$. Otherwise, suppose P_i uses an incorrect authenticated message $\mathbf{y}_i'^{(\ell)}$ such that $\mathbf{y}_i'^{(\ell)} \neq \mathbf{y}_i^{(\ell)}$ together with a fake tag $\eta(\mathbf{y}_i'^{(\ell)})$ and passes the verification in Step 4.b for some $\ell \in [k]$, which means

$$\begin{aligned} \eta(\mathbf{y}_i^{(\ell)}) &= \mathbf{y}_i^{(\ell)} \odot \boldsymbol{\gamma} + \delta(\mathbf{y}_i^{(\ell)}) \\ \eta(\mathbf{y}_i'^{(\ell)}) &= \mathbf{y}_i'^{(\ell)} \odot \boldsymbol{\gamma} + \delta(\mathbf{y}_i'^{(\ell)}). \end{aligned}$$

Hence, P_i knows the information regarding the long-term key $\eta(\mathbf{y}_i'^{(\ell)}) - \eta(\mathbf{y}_i^{(\ell)}) = (\mathbf{y}_i'^{(\ell)} - \mathbf{y}_i^{(\ell)}) \odot \boldsymbol{\gamma}$.

Next, we prove to pass the verification in Step 4.1, P_i must provide a correct share $w_i = \sigma_{i,1}^{(1)} + \dots + \sigma_{i,k}^{(k)}$. Suppose P_i provides $\overline{w}_i$ and $\overline{z}_i$. Let $\epsilon_i = \overline{w}_i - w_i$ and $e_i = \overline{z}_i - (z_{i,1}^{(1)} + \dots + z_{i,k}^{(k)})$. Then passing the verification in Step 4.1 amounts to $\overline{z}_i + \alpha \cdot \overline{w}_i = y_{i,1}^{(1)} + \dots + y_{i,k}^{(k)} = (z_{i,1}^{(1)} + \dots + z_{i,k}^{(k)}) + \alpha \cdot (\sigma_{i,1}^{(1)} + \dots + \sigma_{i,k}^{(k)})$, which implies $\alpha \cdot \epsilon_i + e_i = 0$. By randomness of α , we get $\epsilon_i = e_i = 0$ and $\overline{w}_i = w_i$. $\square$

Hybrid arguments. Now we use hybrid arguments and make use of the lemmas above to show the validity of **Sim_{base-dss}**.

Hyb₀. In this hybrid, this corresponds to real-world execution. In the end, the adversary outputs its outputs and honest parties output their outputs.

Hyb_{0,1}. In this hybrid, in Step 3 of **setup**, instead of invoking $\mathcal{F}_{\text{VSS}}$ to generate long-term keys, **Sim** emulates the interaction between $\mathcal{F}_{\text{VSS}}$ and the adversary by itself. The distributions in **Hyb_{0,1}** and **Hyb₀** are identical.

Hyb₁. In this hybrid, we simulate **DSS-SHARE**($P_d, \boldsymbol{\gamma}$).

Hyb_{1,1}. In this hybrid, in Step 1, if the dealer is honest, instead of letting the dealer directly generate the whole sharings, we first sample random elements as shares of active corrupted parties and sample rest of the shares. The distributions in **Hyb_{1,1}** and **Hyb_{0,1}** are identical.

Hyb_{1,2}. In this hybrid, in Step 2, instead of invoking $\mathcal{F}_{\text{VSS}}$, **Sim** emulates the interaction between the adversary and $\mathcal{F}_{\text{VSS}}$ by itself as described in the simulation, which makes no difference. The distributions in **Hyb_{1,2}** and **Hyb_{1,1}** are identical.

Hyb_{1,3}. In this hybrid, we simulate **COMPUTE-TAG**($P_i, \boldsymbol{\mu}$).

Hyb_{1,3,1}. In this hybrid, in Steps 1.1 and 1.2, instead of invoking $\mathcal{F}_{\text{VSS}}$ and $\mathcal{F}_{\text{packed-triple}}$, **Sim** emulates the interaction between the adversary and $\mathcal{F}_{\text{VSS}}$, $\mathcal{F}_{\text{packed-triple}}$ by itself as described in the simulation. We then simulate **RAND** in Step 1.3. When an honest party distributes a random sharing of the form $[\mathbf{o}]_{n'-1}$ with secret $\mathbf{o} = (o_1, \dots, o_k)$, we first randomly sample shares of corrupted parties and then sample the whole sharing subject to shares of corrupted parties and $o_1 + \dots + o_k = 0$. This makes no difference on the output distribution. The distributions in **Hyb_{1,3,1}** and **Hyb_{1,2}** are identical.

Hyb_{1,3,2}. In this hybrid, we simulate Step 2 of **COMPUTE-TAG**.

Hyb_{1,3,2,1}. If P_i is honest, in Step 2.1, for each $\ell \in [m+1]$, we first randomly sample shares of $[\mathbf{x}^{(\ell)}]_{t'+k-1}$ of corrupted parties and then generate the whole sharing based on the secrets and shares of corrupted parties. This makes no difference on the distribution. The distributions in **Hyb_{1,3,2,1}** and **Hyb_{1,3,1}** are identical.

Hyb_{1,3,2,2}. If P_i is honest, in Step 2.2, we compute corrupted parties' shares of $[\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{t'+k-1}, [\boldsymbol{\mu} + \mathbf{b}^{(\ell)}]_{t'+k-1}$ and randomly sample shares of honest parties based on shares of corrupted parties. Then we compute shares of honest parties of $[\mathbf{a}^{(\ell)}]_{t'+k-1}$ and $[\mathbf{b}^{(\ell)}]_{t'+k-1}$ as $[\mathbf{a}^{(\ell)}]_{t'+k-1} \leftarrow [\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{t'+k-1} - [\mathbf{x}^{(\ell)}]_{t'+k-1}$ and $[\mathbf{b}^{(\ell)}]_{t'+k-1} \leftarrow [\boldsymbol{\mu} + \mathbf{b}^{(\ell)}]_{t'+k-1} - [\boldsymbol{\mu}]_{t'+k-1}$, respectively. This does not change the distribution due to randomness of honest parties' shares of $[\mathbf{a}^{(\ell)}]_{t'+k-1}$ and $[\mathbf{b}^{(\ell)}]_{t'+k-1}$. The distributions in **Hyb_{1,3,2,2}** and **Hyb_{1,3,2,1}** are identical.

Hyb_{1,3,2,3}. If P_i is corrupted, in Step 2.3, we change the computation of honest parties' shares of $[\overline{\mathbf{y}}^{(\ell)}]_{n'-1}$. We first sample random elements as honest parties' shares of $[\overline{\mathbf{y}}^{(\ell)}]_{n'-1}$ and compute the secret $\overline{\mathbf{y}}^{(\ell)}$ based on shares of corrupted parties. Then we compute the short-term key $\delta(\mathbf{x}^{(\ell)}) \leftarrow \overline{\mathbf{y}}^{(\ell)} \odot \mathbf{1} - \mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)} \odot \boldsymbol{\mu} + \mathbf{b}^{(\ell)} + \mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)} \odot \mathbf{b}^{(\ell)} + \boldsymbol{\mu} + \mathbf{b}^{(\ell)} \odot \mathbf{a}^{(\ell)} - \mathbf{c}^{(\ell)} \odot \mathbf{1}$. Then we can compute honest parties' shares of $\langle \delta(\mathbf{x}^{(\ell)}) | 1 \rangle_t$ according to the secret and shares of corrupted parties and compute honest parties' shares of $[\mathbf{o}^{(\ell)}]_{n'-1}$ as $[\mathbf{o}^{(\ell)}]_{n'-1} \leftarrow [\overline{\mathbf{y}}^{(\ell)}]_{n'-1} - [\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{k-1} \cdot [\boldsymbol{\mu} + \mathbf{b}^{(\ell)}]_{k-1} + [\mathbf{x}^{(\ell)} + \mathbf{a}^{(\ell)}]_{k-1} \cdot [\mathbf{b}^{(\ell)}]_{t'+k-1} + [\boldsymbol{\mu} + \mathbf{b}^{(\ell)}]_{k-1} \cdot [\mathbf{a}^{(\ell)}]_{t'+k-1} - [\mathbf{c}^{(\ell)}]_{t'+2k-2} - \langle \delta(\mathbf{x}^{(\ell)}) | 1 \rangle_t \cdot [\mathbf{e}_1]_{k-1}$. This does not change the distribution due to randomness of honest parties' shares of $\langle \delta(\mathbf{x}^{(\ell)}) | 1 \rangle_t$ and $[\mathbf{o}^{(\ell)}]_{n'-1}$. The distributions in **Hyb_{1,3,2,3}** and **Hyb_{1,3,2,2}** are identical.

Hyb_{1,3,3}. In this hybrid, we simulate Step 3 of **COMPUTE-TAG**.

Hyb_{1,3,3,1}. In this hybrid, in Step 3.1, instead of invoking $\mathcal{F}_{\text{vrfy-linear-sharing}}$, Sim emulates the interaction between the adversary and $\mathcal{F}_{\text{vrfy-linear-sharing}}$ by itself as described in the simulation. This does not change the distribution. The distributions in **Hyb_{1,3,3,1}** and **Hyb_{1,3,2,3}** are identical.

Hyb_{1,3,3,2}. In this hybrid,

- if P_d is honest, if P_i is honest, we don't compute $\{[\mathbf{a}^{(\ell)}]_{t'+k-1}, [\mathbf{b}^{(\ell)}]_{t'+k-1}\}_{\ell \in [m+1]}$ and delay the sampling of honest parties' shares $\{[\mathbf{x}^{(\ell)}]_{t'+k-1}\}_{\ell \in [m+1]}$ and $(\{\langle \mu_j | j \rangle_t\}_{j=1}^k, \{\langle \mu_j | 1 \rangle_t\}_{j=2}^k)$.
- If P_d is corrupted and P_i is honest, we don't compute $\{[\mathbf{b}^{(\ell)}]_{t'+k-1}\}_{\ell \in [m+1]}, [\mathbf{a}^{(m+1)}]_{t'+k-1}$ and delay the sampling of honest parties' shares of $[\mathbf{x}^{(m+1)}]_{t'+k-1}$ and $(\{\langle \mu_j | j \rangle_t\}_{j=1}^k, \{\langle \mu_j | 1 \rangle_t\}_{j=2}^k)$.
- If P_i is corrupted (no matter P_d is honest or corrupted), we don't compute $\{[\mathbf{b}^{(\ell)}]_{t'+k-1}, \langle \delta(\mathbf{x}^{(\ell)}) | 1 \rangle_t, [\mathbf{o}^{(\ell)}]_{n'-1}\}_{\ell \in [m+1]}$ and delay the sampling of honest parties' shares of $(\{\langle \mu_j | j \rangle_t\}_{j=1}^k, \{\langle \mu_j | 1 \rangle_t\}_{j=2}^k)$.

This is feasible as they are not used in the current simulation until **verification of tag computation**. The distributions in **Hyb_{1,3,3,2}** and **Hyb_{1,3,3,1}** are identical.

Hyb_{1,3,3,3}. In this hybrid, in Step 3.5, instead of invoking $\mathcal{F}_{\text{vss}}$ to check whether the secret of $\langle \mathbf{o} | 1 \rangle_t$ is 0, Sim emulates by itself as follows.

- If P_i is honest, we check for $\ell \in [m+1]$,
 - if $\mathbf{x}^{(\ell)}$ is unknown to the adversary, $\mathbf{d}^{(\ell)} = \mathbf{e}^{(\ell)} = \mathbf{0}$ and $\epsilon^{(\ell)} = 0$;
 - if $\mathbf{x}^{(\ell)}$ is known to the adversary, $\mathbf{e}^{(\ell)} = \mathbf{0}$ and $\mathbf{x}^{(\ell)} \odot \mathbf{d}^{(\ell)} + \epsilon^{(\ell)} = 0$.

If these requirements are satisfied, in Step 3.3, we change the computation of $\mathbf{x}$ and $\mathbf{y}$. We first sample random elements as $\mathbf{x}$ and $\mathbf{y}$ and then compute $\mathbf{x}^{(m+1)} \leftarrow \mathbf{x} - \sum_{\ell=1}^m \alpha^\ell \cdot \mathbf{x}^{(\ell)}$, $\mathbf{y}^{(m+1)} \leftarrow \mathbf{y} - \sum_{\ell=1}^m \alpha^\ell \cdot \mathbf{y}^{(\ell)}$, and $\delta(\mathbf{x}^{(m+1)}) \leftarrow \mathbf{y}^{(m+1)} - \mathbf{x}^{(m+1)} \odot \boldsymbol{\mu}$. Then in Step 3.4, Sim emulates $\mathcal{F}_{\text{vss}}$ by outputting **accept**. This does not change the distribution by randomness of $\mathbf{x}^{(m+1)}$ and $\delta(\mathbf{x}^{(m+1)})$, the relation $\mathbf{y} = \mathbf{x} \odot \boldsymbol{\mu} + \delta$, and verifiability of $\boldsymbol{\mu}, \delta$.

If the requirement is not satisfied but the verification has been passed, Sim outputs $\perp$ and halts, which only happens with negligible probability due to Lemma 8.

- If P_i is corrupted, Sim emulates $\mathcal{F}_{\text{vss}}$ as in Step 17. This does not change the distribution due to the relation $\overline{\mathbf{y}}^{(\ell)} = \mathbf{x}^{(\ell)} \odot \boldsymbol{\mu} + \delta^{(\ell)} + \epsilon^{(\ell)}$ for $\ell \in [m+1]$ and the verifiability of $\boldsymbol{\mu}, \delta$.

The distribution in **Hyb_{1,3,3,3}** is statistically closed to that in **Hyb_{1,3,3,2}**.

Hyb_{1,4}. In this hybrid, we simulate Step 4 of **DSS-SHARE** as follows.

Hyb_{1,4,1}. In this hybrid, in Step 4.1, instead of invoking $\mathcal{F}_{\text{coin}}$, Sim samples a random element by itself. The distributions in **Hyb_{1,4,1}** and **Hyb_{1,3,3,3}** are identical.

Hyb_{1,4,2}. In this hybrid, we simulate Steps 4.2 and 4.3.

If P_d is honest,

- for each honest P_i , in Step 4.2, we change the computation of σ_i and y_i . We first sample random elements as σ_i, y_i and compute $\sigma_i^{(m+1)} \leftarrow \sigma_i - \sum_{\ell=1}^m \lambda^\ell \cdot \sigma_i^{(\ell)}$, $y_i^{(m+1)} \leftarrow y_i - \sum_{\ell=1}^m \lambda^\ell \cdot y_i^{(\ell)}$, and $\delta(\sigma_i^{(m+1)}) \leftarrow y_i^{(m+1)} - \sigma_i^{(m+1)} \odot \mu$. In Step 4.3, Sim emulates $\mathcal{F}_{\text{VSS}}$ by outputting **accept**. This does not change the distribution due to the randomness of $\sigma_i^{(m+1)}$ and $\delta(\sigma_i^{(m+1)})$ and the relation $y_i = \sigma_i \odot \mu + \delta_i$ and the verifiability of μ, δ_i .
- For each active corrupted party P_i , Sim emulates $\mathcal{F}_{\text{VSS}}$ as in Step 6. This does not change the output distribution if Sim does not halt and halts except with negligible probability by randomness of μ , the relation $\bar{y}_i = \bar{\sigma}_i \odot \mu + \delta_i$, and verifiability of μ, δ_i .

If P_d is corrupted,

- for each honest party P_i , in Step 4.2, we change the computation of y_i . We first sample a random element as y_i and compute $y_i^{(m+1)} \leftarrow y_i - \sum_{\ell=1}^m \lambda^\ell \cdot y_i^{(\ell)}$ and $\delta(\sigma_i^{(m+1)}) \leftarrow y_i^{(m+1)} - \sigma_i^{(m+1)} \odot \mu$. In Step 4.3, Sim then emulates $\mathcal{F}_{\text{VSS}}$ by outputting **accept**. This does not change the distribution due to the randomness of $\delta(\sigma_i^{(m+1)})$, the relation $y_i = \sigma_i \odot \mu + \delta_i$, and the verifiability of μ, δ_i .
- For each active corrupted party P_i , Sim does the same as the case when P_d is honest.

The distribution in $\mathbf{Hyb}_{1,4,2}$ is statistically closed to that in $\mathbf{Hyb}_{1,4,1}$.

Hyb_{1,5}. In this hybrid, in Step 5.1, we change the computation of μ . Sim first samples random elements as $\gamma_j - \mu_j$ and computes $\mu_j \leftarrow \gamma_j - (\gamma_j - \mu_j)$ for $j \in [k]$. Then Sim emulates $\mathcal{F}_{\text{VSS}}$ by outputting $\{\gamma_j - \mu_j\}_{j \in [k]}$. This does not change the distribution by randomness of μ . The distributions in $\mathbf{Hyb}_{1,5}$ and $\mathbf{Hyb}_{1,4,2}$ are identical.

Hyb_{1,6}. In this hybrid, we don't compute μ and delay the sampling of the long-term key γ as it is not used until the end of the current invocation of **DSS-SHARE**.

- If P_d is honest, Sim delays the sampling of shares of $\{[x^{(\ell)}]_{t'+k-1}\}_{\ell \in [m]}$ held by honest parties and uses the active corrupted parties' shares received from $\mathcal{F}_{\text{base-DSS}}$.
- If P_d is corrupted, Sim sends $\{[x^{(\ell)}]_{t'+k-1}\}_{\ell \in [m]}$ to $\mathcal{F}_{\text{base-DSS}}$ and lets honest parties output what they received from $\mathcal{F}_{\text{base-DSS}}$.

If one of checks fails, Sim follows the protocol, identifies a dispute pair E , and sends (**reject**, E) to $\mathcal{F}_{\text{base-DSS}}$. Otherwise, Sim sends **accept** to $\mathcal{F}_{\text{base-DSS}}$. By Lemma 9, the distributions in $\mathbf{Hyb}_{1,6}$ and $\mathbf{Hyb}_{1,5}$ are identical.

Hyb₂. In this hybrid, we simulate **AUTH**.

Hyb_{2,1}. In this hybrid, in Steps 2.1 and 2.2, instead of invoking $\mathcal{F}_{\text{VSS}}$, $\mathcal{S}$ emulates the interaction between $\mathcal{F}_{\text{VSS}}$ and the adversary by itself. The distributions in $\mathbf{Hyb}_{2,1}$ and $\mathbf{Hyb}_{1,6}$ are identical.

Hyb_{2,2}. In this hybrid, we simulate **RAND**(Σ_3) in Step 2.3. When an honest party distributes a random sharing of the form $[\mathbf{o}]_{n'-1}$ with secret $\mathbf{o} = (o_1, \dots, o_k)$, we first sample random elements as shares of active corrupted parties and sample the whole sharing subject to the shares of corrupted parties and the constraint of the secret $o_1 + \dots + o_k = 0$, which makes no difference on the output distribution. The distributions in $\mathbf{Hyb}_{2,2}$ and $\mathbf{Hyb}_{2,1}$ are identical.

Hyb_{2,3}. In this hybrid, in Step 2.3, when invoking **RAND**(Σ_3), we change the computation of $\{[\mathbf{o}_{(i)}]_{n'-1}\}_{i=1}^{n-t}$. Sim first randomly samples honest parties' shares of $\{[\mathbf{o}_{(i)}]_{n'-1}\}_{i=1}^{n-t}$ based on shares of corrupted parties and computes honest parties' shares of $\{[\mathbf{o}^{(i)}]_{n'-1}\}_{i \in \mathcal{H}}$ from their shares of $\{[\mathbf{o}_{(i)}]_{n'-1}\}_{i=1}^{n-t}$ and $\{[\mathbf{o}^{(i)}]_{n'-1}\}_{i \in \mathcal{C}}$ using the one-to-one correspondence defined by the submatrix $\mathbf{M}_{\mathcal{H}}$. Due to the randomness of $\{[\mathbf{o}^{(i)}]_{n'-1}\}_{i \in \mathcal{H}}$ and the one-to-one correspondence between it and $\{[\mathbf{o}_{(i)}]_{n'-1}\}_{i=1}^{n-t}$, the distributions in $\mathbf{Hyb}_{2,3}$ and $\mathbf{Hyb}_{2,2}$ are identical.

Hyb_{2,4}. In this hybrid, in Step 3.1, if P_i is corrupted, we change the computation of honest parties' shares of $[\bar{y}_i^{(\ell)}]_{n'-1}$. We first randomly sample honest parties' shares of $[\bar{y}_i^{(\ell)}]_{n'-1}$ and compute $\bar{y}_i^{(\ell)}$ further based on shares of corrupted parties. Then we compute $\delta(\sigma_i^{(\ell)}) \leftarrow \bar{y}_i^{(\ell)} - \sigma_i^{(\ell)} \odot \mu$ and $\langle \delta(\sigma_i^{(\ell)}) | 1 \rangle_t$ according to the secret and shares of corrupted parties. We then compute $[\mathbf{o}_i^{(\ell)}]_{n'-1} \leftarrow [\bar{y}_i^{(\ell)}]_{n'-1} - [\sigma_i^{(\ell)}]_{k-1} \cdot [\mu]_{t+k-1} -$

$\langle \delta(\sigma_i^{(\ell)}) | 1 \rangle_t \cdot [e_1]_{k-1}$. This does not change the distribution due to the randomness of honest parties' shares of $\langle \delta(\sigma_i^{(\ell)}) | 1 \rangle_t$ and $[o_i^{(\ell)}]_{n'-1}$. The distributions in **Hyb**_{2,4} and **Hyb**_{2,3} are identical.

Hyb_{2,5}. In this hybrid, we delay the sampling of honest parties' shares of $(\{\langle \mu_j | j \rangle_t\}_{j=1}^k, \{\langle \mu_j | 1 \rangle_t\}_{j=2}^k)$ until **verification**. The distributions in **Hyb**_{2,5} and **Hyb**_{2,4} are identical.

Hyb_{2,6}. In this hybrid, in Step 4.1, instead of invoking $\mathcal{F}_{\text{coin}}$, Sim samples a random element by itself. The distributions in **Hyb**_{2,6} and **Hyb**_{2,5} are identical.

Hyb_{2,7}. In this hybrid, we simulate Step 4.3.

- If P_i is honest, Sim checks whether $\epsilon_i^{(\ell)} = 0$ for each $\ell \in [m]$. If this requirement is satisfied, in Step 4.3.a, we change the computation of $\overline{y_i}$. We first sample a random element as $\overline{y_i}$ and compute $y_i^{(1)} = \overline{y_i} - \sum_{\ell=2}^m \lambda^{\ell-1} \cdot \overline{y_i^{(\ell)}}$ and $\delta(\sigma_i^{(1)}) \leftarrow y_i^{(1)} - x_i^{(1)} \odot \mu$. In Step 4.3.b, Sim emulates $\mathcal{F}_{\text{vss}}$ by outputting **accept**. This does not change the distribution due to the randomness of $\delta(\sigma_i^{(1)})$, the relation $\overline{y_i} = \sigma_i \odot \mu + \delta_i$ and the verifiability of μ, δ_i .
If the requirement is not satisfied but the verification has been passed, Sim outputs $\perp$ and halts, which only happens with negligible probability due to Lemma 10.
- If P_i is corrupted, Sim simulates Step 4.3.b as Step 9 of **Sim**_{auth}. The output distribution is statistically closed to that in **Hyb**_{2,6} by randomness of μ and verifiability of μ, δ_i .

The distributions in **Hyb**_{2,7} and **Hyb**_{2,6} are statistically closed.

Hyb_{2,8}. In this hybrid, in Step 5.1, we change the computation of μ . We first randomly sample $\gamma_j - \mu_j$ and compute $\mu_j \leftarrow \gamma_j - (\gamma_j - \mu_j)$ for $j \in [k]$. Then Sim emulates $\mathcal{F}_{\text{vss}}$ by outputting $\{\gamma_j - \mu_j\}_{j \in [k]}$. This does not change the distribution due to the randomness of μ . The distributions in **Hyb**_{2,8} and **Hyb**_{2,7} are identical.

Hyb_{2,9}. In this hybrid, for each active corrupted party P_i , we don't compute $\{\langle \delta(\sigma_i^{(\ell)}) | 1 \rangle_t, [o_i^{(\ell)}]_{n'-1}\}_{\ell \in [m]}$; for each honest P_i , we don't compute $y_i^{(1)}, \delta(\sigma_i^{(1)})$. We delay the sampling of the long-term key γ as it is not used until the end of the current invocation of **AUTH**. The distributions in **Hyb**_{2,9} and **Hyb**_{2,8} are identical.

Hyb₃. In this hybrid, we simulate **RAND-TAG**.

Hyb_{3,1}. In this hybrid, in Step 1, instead of invoking $\mathcal{F}_{\text{vss}}$ to prepare a random long-term key, Sim emulates the interaction between the adversary and $\mathcal{F}_{\text{vss}}$ by itself. The distributions in **Hyb**_{3,1} and **Hyb**_{2,7} are identical.

Hyb_{3,2}. In this hybrid, we simulate **COMPUTE-TAG** as **Sim**_{compute-tag}. Similar to **Hyb**_{1,3}. For each active corrupted party P_i , Sim learns the authenticated messages and tags $\{z_i^{(\ell)}, y_i^{(\ell)}\}_{\ell \in [m]}$. The distribution in **Hyb**_{3,2} is statistically close to that in **Hyb**_{3,1}.

Hyb_{3,3}. In this hybrid, in Step 3.a, we change the computation of μ . We first randomly sample $\gamma_j - \mu_j$ and compute $\mu_j \leftarrow \gamma_j - (\gamma_j - \mu_j)$ for $j \in [k]$. Then Sim emulates $\mathcal{F}_{\text{vss}}$ by outputting $\{\gamma_j - \mu_j\}_{j \in [k]}$. This does not change the distribution by randomness of μ . The distributions in **Hyb**_{3,3} and **Hyb**_{3,2} are identical.

Hyb_{3,4}. In this hybrid, we delay the sampling of the long-term key γ until the end of **RAND-TAG**. For every honest P_i , we delay the sampling of the honest parties' shares of $\{[z_i^{(\ell)}]_{t'+k-1}\}_{\ell \in [m]}$ until the end of **RAND-TAG**. This is feasible as they are not used in the current simulation. The distributions in **Hyb**_{3,4} and **Hyb**_{3,3} are identical.

Hyb₄. In this hybrid, we simulate **DSS-CHECK**.

Hyb_{4,1}. In this hybrid, in Step 3.1, for every honest party P_i , we change the computation of z_i . We first randomly sample $z_i, z_{1,i}^{(1)}, \dots, z_{i,k-1}^{(k-1)}$ and compute $z_{i,k}^{(k)} = z_i - \sum_{\ell=1}^{k-1} z_{i,\ell}^{(\ell)}$. This does not change the distribution due to the randomness of $z_{i,k}^{(k)}$. The distributions in **Hyb**_{4,1} and **Hyb**_{3,4} are identical.

Hyb_{4,2}. In this hybrid, in Step 3.2, instead of invoking $\mathcal{F}_{\text{coin}}$, $\mathcal{S}$ samples a random element by itself. The distributions in **Hyb**_{4,2} and **Hyb**_{4,1} are identical.

Hyb_{4,3}. In this hybrid, in Step 3.3, for every honest party P_i , we change the computation of $y_i^{(\ell)}$ for each $\ell \in [k]$. In particular, we first randomly sample $y_i^{(1)}, \dots, y_i^{(k)}$ subject to $\sum_{\ell=1}^k y_{i,j}^{(\ell)} = z_i + \alpha \cdot w_i$ and

sample random elements as $\eta(\mathbf{y}_i^{(1)}), \dots, \eta(\mathbf{y}_i^{(k)})$. Then for each $\ell \in [k]$, we compute $\mathbf{z}_i^{(\ell)} = \mathbf{y}_i^{(\ell)} - \alpha \cdot \sigma_i^{(\ell)}$, $\eta(\mathbf{z}_i^{(\ell)}) \leftarrow \eta(\mathbf{y}_i^{(\ell)}) - \alpha \cdot \eta(\sigma_i^{(\ell)})$, and $\delta(\mathbf{z}_i^{(\ell)}) \leftarrow \eta(\mathbf{z}_i^{(\ell)}) - \mathbf{z}_i^{(\ell)} \odot \gamma$. This does not change the distribution due to the randomness of $\mathbf{z}_i^{(1)}, \dots, \mathbf{z}_i^{(k)}$ and $\delta(\mathbf{z}_i^{(1)}), \dots, \delta(\mathbf{z}_i^{(k)})$. The distributions in **Hyb**_{4,3} and **Hyb**_{4,2} are identical.

Hyb_{4,4}. In this hybrid, we simulate Step 4.2.

- If P_i is honest, Sim emulates $\mathcal{F}_{\text{VSS}}$ by outputting **accept**. This does not change the distribution as honest parties always use correct shares and their corresponding authentication tags.
- If P_i is corrupted, Sim checks whether the broadcast values of $\mathbf{y}_i^{(\ell)}$ and $\eta(\mathbf{y}_i^{(\ell)})$ correspond to the computed values. If this requirement is satisfied, Sim emulates $\mathcal{F}_{\text{VSS}}$ by outputting **accept**. Otherwise, Sim emulates $\mathcal{F}_{\text{VSS}}$ by outputting **reject**. By randomness of μ and verifiability of μ and $\{\delta(\mathbf{y}_i^{(\ell)})\}_{\ell \in [k]}$, the output distribution is statistically closed to that in **Hyb**_{4,3}.

The distributions in **Hyb**_{4,4} and **Hyb**_{4,3} are statistically closed.

Hyb_{4,5}. In this hybrid, for each honest P_i , we don't compute $\{\mathbf{z}_i^{(\ell)}, \delta(\mathbf{z}_i^{(\ell)})\}_{\ell \in [k]}$ and delay the sampling of $(\{\langle \mu_j | j \rangle_t\}_{j=1}^k, \{\langle \mu_j | 1 \rangle_t\}_{j=2}^k)$. For each active corrupted party P_i , if P_i passes the verification, Sim sends the correct share w_i to $\mathcal{F}_{\text{base-DSS}}$ and sends an incorrect share otherwise. Let honest parties output what they received from $\mathcal{F}_{\text{base-DSS}}$. By Lemma 11, the distributions in **Hyb**_{4,5} and **Hyb**_{4,4} are statistically closed.

Noticing that **Hyb**_{4,5} does not use honest parties' inputs. As **Hyb**_{4,5} already corresponds to the distribution in the ideal world, we conclude that **BASE-DSS** securely computes $\mathcal{F}_{\text{base-DSS}}$ in $\mathcal{F}_{\text{VSS}}$, $\mathcal{F}_{\text{packed-triple}}$, $\mathcal{F}_{\text{coin-hybrid}}$ model. $\square$

H.2 Proof of Lemma 2

Lemma 12 (Restatement of Lemma 2). ***AUTH-PACKED-TRIPLES** computes Subfunctionality 1. of $\mathcal{F}_{\text{DSS}}$, labeled with (Triple, m), with statistical security against any fully malicious adversary controlling up to t parties in the $\mathcal{F}_{\text{base-DSS}}$, $\mathcal{F}_{\text{rand-OLE}}$, $\mathcal{F}_{\text{vrfy-linear-sharing}}$, $\mathcal{F}_{\text{coin-hybrid}}$ model.*

Proof (Proof of Lemma 2).

We construct a simulator, **Sim**_{auth-packed-triple}, that interacts with $\mathcal{F}_{\text{DSS}}$, such that no environment $\mathcal{Z}$ can distinguish between an interaction with **Sim**_{auth-packed-triple} and an interaction with the adversary and **AUTH-PACKED-TRIPLES**.

Simulator 8: **Sim**_{auth-packed-triple} for **AUTH-PACKED-TRIPLES**

1. In Step 1.1, Sim emulates the interaction between $\mathcal{F}_{\text{rand-OLE}}$ and the adversary and records the values.
2. In Step 1.2, if P_i is corrupted, Sim emulates the interaction between $\mathcal{F}_{\text{base-DSS}}$ and the adversary and records the whole packed Shamir sharings; if P_i is honest, Sim randomly samples corrupted parties' shares and sends them to the adversary on behalf of $\mathcal{F}_{\text{base-DSS}}$. For each packed triple, Sim computes the errors $\Delta_a^{(\ell)}, \Delta_b^{(\ell)}, \Delta_c^{(\ell)}$ such that $\mathbf{c}_\ell = \mathbf{a}_\ell * \mathbf{b}_\ell + \Delta_a^{(\ell)} * \mathbf{a}_\ell + \Delta_b^{(\ell)} * \mathbf{b}_\ell + \Delta_c^{(\ell)}$.
 - For a packed triple distributed by two honest parties, $\Delta_a^{(\ell)} = \Delta_b^{(\ell)} = \Delta_c^{(\ell)} = \mathbf{0}$.
 - For a packed triple distributed by two corrupted parties, $\Delta_a^{(\ell)} = \Delta_b^{(\ell)} = \mathbf{0}$, and $\Delta_c^{(\ell)} = \mathbf{a}_\ell * \mathbf{b}_\ell - \mathbf{c}_\ell$.
 - For a packed triple distributed by one honest party and one corrupted party, $\Delta_a^{(\ell)}, \Delta_b^{(\ell)}, \Delta_c^{(\ell)}$ are computed according to values recorded when emulating $\mathcal{F}_{\text{rand-OLE}}$. Concretely, suppose the virtual party consists of the corrupt P_i and honest P_j . In Step 1.1, OLE correlations $\mathbf{a}^1 * \mathbf{b}^2 = \mathbf{s}^1 + \mathbf{s}^2$ and $\mathbf{a}^2 * \mathbf{b}^1 = \mathbf{t}^1 + \mathbf{t}^2$ are generated, where P_i holds $\mathbf{a}^1, \mathbf{s}^1, \mathbf{b}^1, \mathbf{t}^1$ and P_j holds $\mathbf{b}^2, \mathbf{s}^2, \mathbf{a}^2, \mathbf{t}^2$. In Step 1.2, when emulating $\mathcal{F}_{\text{base-DSS}}$, P_i distributes $\llbracket \mathbf{a}^1 \rrbracket_{t'+k-1}$, $\llbracket \mathbf{b}^1 \rrbracket_{t'+k-1}$, and

$\llbracket \overline{w^1} \rrbracket_{t'+k-1}$. Then Sim computes

$$\begin{aligned}\Delta_a^{(\ell)} &\leftarrow b^1 - \overline{b^1}, \quad \Delta_b^{(\ell)} \leftarrow a^1 - \overline{a^1}, \\ \Delta_c^{(\ell)} &\leftarrow \overline{w^1} - (a^1 * b^1 + s^1 + t^1) + \Delta_a^{(\ell)} * \Delta_b^{(\ell)}.\end{aligned}$$

3. In Step 2.2, Sim computes corrupted parties' shares of $\llbracket A(\alpha_\ell) + a_\ell \rrbracket_{t'+k-1}$ and $\llbracket B(\alpha_\ell) + b_\ell \rrbracket_{t'+k-1}$, samples the honest parties' shares based on shares of corrupted parties, and sends them to P_{king} . If P_{king} is honest, Sim honestly follows the protocol. Sim then receives honest parties' shares of $\llbracket A(\alpha_\ell) + a_\ell \rrbracket_{k-1}$ and $\llbracket B(\alpha_\ell) + b_\ell \rrbracket_{k-1}$. If they are valid degree- $(k-1)$ packed Shamir sharings, Sim computes

$$\begin{aligned}\Delta_1^{(\ell)} &\leftarrow \overline{A(\alpha_\ell) + a_\ell} - (A(\alpha_\ell) + a_\ell), \\ \Delta_2^{(\ell)} &\leftarrow \overline{B(\alpha_\ell) + b_\ell} - (B(\alpha_\ell) + b_\ell).\end{aligned}$$

4. Sim invokes $\text{Sim}_{\text{auth-deg-reduction}}$ and learns corrupted parties' shares of $\llbracket A(\alpha_\ell) * B(\alpha_\ell) \rrbracket_{t'+k-1}$ and the additive error d_ℓ .
5. In Step 3.1, Sim emulates $\mathcal{F}_{\text{coin}}$ by generating a random element r . If $r \in [m]$, Sim aborts on behalf of honest parties.
6. Sim checks the following requirements and simulates honest parties depending on whether the verification passes.
 - For $\ell \in [2m+2]$, $\Delta_a^{(\ell)} = \Delta_b^{(\ell)} = \Delta_c^{(\ell)} = 0$.
 - For $\ell \in \{2m+3, \dots, 4m+3\}$,
 - if the used triple is distributed by an honest virtual party, $\Delta_2^{(\ell)} - \Delta_a^{(\ell)} = \Delta_1^{(\ell)} - \Delta_b^{(\ell)} = 0$ and $\Delta_a^{(\ell)} * (\overline{A(\alpha_\ell) + a_\ell} - \Delta_1^{(\ell)}) + \Delta_b^{(\ell)} * (\overline{B(\alpha_\ell) + b_\ell} - \Delta_2^{(\ell)}) + \Delta_1^{(\ell)} * \Delta_2^{(\ell)} + \Delta_c^{(\ell)} + d_\ell = 0$;
 - if the used triple is distributed by a corrupt virtual party, $\Delta_1^{(\ell)} * B(\alpha_\ell) + \Delta_2^{(\ell)} * A(\alpha_\ell) + \Delta_1^{(\ell)} * \Delta_2^{(\ell)} + \Delta_c^{(\ell)} + d_\ell = 0$.

If all requirements are satisfied, Sim computes corrupted parties' shares of $\llbracket A(r) \rrbracket_{t'+k-1}$, $\llbracket B(r) \rrbracket_{t'+k-1}$, and $\llbracket C(r) \rrbracket_{t'+k-1}$. Sim then randomly samples $A(r), B(r)$, computes $C(r) \leftarrow A(r) * B(r)$, and computes honest parties' shares of $\llbracket A(r) \rrbracket_{t'+k-1}, \llbracket B(r) \rrbracket_{t'+k-1}, \llbracket C(r) \rrbracket_{t'+k-1}$ according to shares of corrupted parties and the secrets. In Step 3.2, Sim broadcasts shares of honest parties. In Step 3.3, Sim emulates $\mathcal{F}_{\text{base-DSS}}$ to check whether the adversary provides correct shares.

Otherwise, Sim samples honest parties' shares of $\{\llbracket a_\ell \rrbracket_{t'+k-1}, \llbracket b_\ell \rrbracket_{t'+k-1}, \llbracket c_\ell \rrbracket_{t'+k-1}\}_{\ell \in [4m+3]}$ and $\{\llbracket u_\ell \rrbracket_{t'+k-1}\}_{\ell \in [2m+1]}$, and honestly follows the protocol in Step 3.2. In Step 3.3, Sim emulates $\mathcal{F}_{\text{base-DSS}}$. However, if the verifications in Steps 3.2 and 3.3 have been passed, Sim outputs $\perp$ and halts.

7. Sim interacts with $\mathcal{F}_{\text{DSS}}$.
 - If some corrupted party P_i provides incorrect shares in Step 3.2, Sim sends $(\text{reject}, \{i\})$ to $\mathcal{F}_{\text{DSS}}$.
 - Otherwise, if the adversary provides correct shares in Step 3.2,
 - if the requirements are satisfied, Sim computes active corrupted parties' shares of $\{\llbracket a^{(\ell)} \rrbracket_{t'+k-1}, \llbracket b^{(\ell)} \rrbracket_{t'+k-1}, \llbracket c^{(\ell)} \rrbracket_{t'+k-1}\}_{\ell=1}^m$ and sends them together with accept to $\mathcal{F}_{\text{DSS}}$;
 - otherwise, Sim follows the protocol in Step 3.4 to locate a dispute pair E and sends (reject, E) to $\mathcal{F}_{\text{DSS}}$.
8. Sim outputs what the adversary outputs and honest parties output what they receive from $\mathcal{F}_{\text{DSS}}$.

Simulator 9: $\text{Sim}_{\text{auth-deg-reduction}}$ for **AUTH-DEG-REDUCTION**

1. Sim invokes Sim_{rand} .
2. In Step 2, Sim randomly samples honest parties' shares of $[\mathbf{z}_\ell + \mathbf{u}_\ell]_{n'-1}$ and sends them to P_{king} .
3. In Step 3, if P_{king} is honest, Sim honestly follows the protocol. Sim receives honest parties' shares of $[\mathbf{z}_\ell + \mathbf{u}_\ell]_{k-1}$ from P_{king} . If it is a valid degree- $(k-1)$ packed Shamir sharing, Sim computes $\mathbf{d}_\ell \leftarrow \overline{\mathbf{z}_\ell + \mathbf{u}_\ell} - (\mathbf{z}_\ell + \mathbf{u}_\ell)$.
4. In Step 4, Sim checks whether P_{king} distributes valid degree- $(k-1)$ packed Shamir sharings and emulates the interaction between $\mathcal{F}_{\text{vrfy-linear-sharing}}$ and the adversary.
5. In Step 5, Sim emulates $\mathcal{F}_{\text{base-DSS}}$ and records the sharings $[\mathbf{z}_1 + \mathbf{u}_1]_{k-1}, \dots, [\mathbf{z}_m + \mathbf{u}_m]_{k-1}$.

Lemma 13. *Following the notations defined in $\text{Sim}_{\text{auth-packed-triple}}$, if all parties pass the **triple verification**, then the following requirements are satisfied except with negligible probability.*

- For $\ell \in [2m+2]$, $\Delta_a^{(\ell)} = \Delta_b^{(\ell)} = \Delta_c^{(\ell)} = \mathbf{0}$.
- For $\ell \in \{2m+3, \dots, 4m+3\}$,
 - if the used triple is distributed by an honest virtual party, $\Delta_2^{(\ell)} - \Delta_a^{(\ell)} = \Delta_1^{(\ell)} - \Delta_b^{(\ell)} = \mathbf{0}$ and $\Delta_a^{(\ell)} * (\overline{\mathbf{A}(\alpha_\ell)} + \mathbf{a}_\ell - \Delta_1^{(\ell)}) + \Delta_b^{(\ell)} * (\overline{\mathbf{B}(\alpha_\ell)} + \mathbf{b}_\ell - \Delta_2^{(\ell)}) + \Delta_1^{(\ell)} * \Delta_2^{(\ell)} + \Delta_c^{(\ell)} + \mathbf{d}_\ell = \mathbf{0}$;
 - if the used triple is distributed by a corrupt virtual party, $\Delta_1^{(\ell)} * \mathbf{B}(\alpha_\ell) + \Delta_2^{(\ell)} * \mathbf{A}(\alpha_\ell) + \Delta_1^{(\ell)} * \Delta_2^{(\ell)} + \Delta_c^{(\ell)} + \mathbf{d}_\ell = \mathbf{0}$.

Proof. For $\ell \in [2m+2]$, we have that

$$\mathbf{c}_\ell = \mathbf{a}_\ell * \mathbf{b}_\ell + \Delta_a^{(\ell)} * \mathbf{a}_\ell + \Delta_b^{(\ell)} * \mathbf{b}_\ell + \Delta_c^{(\ell)}.$$

For $\ell \in \{2m+3, \dots, 4m+3\}$, we have that

$$\begin{aligned} \overline{\mathbf{C}(\alpha_\ell)} &= \overline{\mathbf{A}(\alpha_\ell)} + \mathbf{a}_\ell * \overline{\mathbf{B}(\alpha_\ell)} + \mathbf{b}_\ell - \mathbf{a}_\ell * \overline{\mathbf{B}(\alpha_\ell)} + \mathbf{b}_\ell - \mathbf{b}_\ell * \overline{\mathbf{A}(\alpha_\ell)} + \mathbf{a}_\ell + \mathbf{c}_\ell + \mathbf{d}_\ell \\ &= \mathbf{A}(\alpha_\ell) * \mathbf{B}(\alpha_\ell) + (\Delta_1^{(\ell)} - \Delta_b^{(\ell)}) * \mathbf{B}(\alpha_\ell) + \mathbf{A}(\alpha_\ell) * (\Delta_2^{(\ell)} - \Delta_a^{(\ell)}) \\ &\quad + \Delta_a^{(\ell)} * (\overline{\mathbf{A}(\alpha_\ell)} + \mathbf{a}_\ell - \Delta_1^{(\ell)}) + \Delta_b^{(\ell)} * (\overline{\mathbf{B}(\alpha_\ell)} + \mathbf{b}_\ell - \Delta_2^{(\ell)}) \\ &\quad + \Delta_1^{(\ell)} * \Delta_2^{(\ell)} + \Delta_c^{(\ell)} + \mathbf{d}_\ell. \end{aligned}$$

If one of the requirements above is not satisfied for some $\ell \in [4m+3]$, then we have that $\mathbf{A}(\alpha_\ell) * \mathbf{B}(\alpha_\ell) \neq \overline{\mathbf{C}(\alpha_\ell)}$ except with negligible probability. This results in that $\overline{\mathbf{C}} - \mathbf{A} * \mathbf{B}$ is not a vector of zero polynomials and thus $\overline{\mathbf{C}}(r) \neq \mathbf{A}(r) * \mathbf{B}(r)$ except with negligible probability, by Schwartz-Zippel lemma and randomness of r . By verifiability of $\mathcal{F}_{\text{base-DSS}}$, to pass the verification, all parties must provide their shares of $[\mathbf{A}(r)]_{t'+k-1}$, $[\mathbf{B}(r)]_{t'+k-1}$, and $[\overline{\mathbf{C}}(r)]_{t'+k-1}$. Therefore, the triple verification cannot be passed. $\square$

Before giving the hybrid arguments, we first define the notation as follows. Recall that a virtual party is emulated by two corrupted parties is a corrupt virtual party, otherwise it is an honest virtual party.

- $\mathcal{E}_1$ denotes a subset of $[2m+2]$ that corresponds to triples distributed by corrupted virtual parties.
- $\mathcal{E}_2$ denotes a subset of $\{2m+3, \dots, 4m+3\}$ that corresponds to triples distributed by corrupted virtual parties.
- $\mathcal{D}_1$ denotes a subset of $[2m+2]$ that contains the first $|\mathcal{E}_2|$ identities that correspond to triples distributed by honest virtual parties.
- $\mathcal{D}$ is defined be $[2m+2] \setminus \mathcal{E}_1 \cup \mathcal{D}_1$.
- $\mathcal{D}_2$ is defined to be $\{2m+3, \dots, 4m+3\} \setminus \mathcal{E}_2$.

Hybrid arguments. Now we use hybrid arguments and make use of the lemma above to show the validity of [Sim_{auth-packed-triple}](#).

Hyb₀. This hybrid corresponds to real-world execution. In the end, the adversary outputs its outputs and honest parties output their outputs.

Hyb₁. In this hybrid, we simulate **packed triple distribution**.

Hyb_{1,1}. In this hybrid, in Step 1.1, instead of invoking [F_{rand-OLE}](#), Sim emulates by itself. The distributions in **Hyb_{1,1}** and **Hyb₀** are identical.

Hyb_{1,2}. In this hybrid, in Step 1.2, instead of invoking [F_{base-DSS}](#), Sim emulates itself. When an honest party needs to distribute a packed Shamir sharing, Sim first randomly samples shares of corrupted parties and then determines the whole sharing. For the ℓ -th packed triple, if $\ell \in \mathcal{D}_1$, Sim delays the sampling of honest parties' shares. The distributions in **Hyb_{1,2}** and **Hyb_{1,1}** are identical.

Hyb₂. In this hybrid, we simulate **packed triple extraction**.

Hyb_{2,1}. In this hybrid, in Step 2.2, we change the computation of honest parties' shares of $\{\llbracket \mathbf{A}(\alpha_\ell) + \mathbf{a}_\ell \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(\alpha_\ell) + \mathbf{b}_\ell \rrbracket_{t'+k-1}\}_{\ell=2m+3}^{4m+3}$. For each $\ell \in \{2m+3, \dots, 4m+3\}$, Sim first computes corrupted parties' shares of $\llbracket \mathbf{A}(\alpha_\ell) + \mathbf{a}_\ell \rrbracket_{t'+k-1}$ and $\llbracket \mathbf{B}(\alpha_\ell) + \mathbf{b}_\ell \rrbracket_{t'+k-1}$ and randomly samples the whole sharings.

For $\ell \in \mathcal{E}_2$, Sim computes $\llbracket \mathbf{A}(\alpha_\ell) \rrbracket_{t'+k-1} \leftarrow \llbracket \mathbf{A}(\alpha_\ell) + \mathbf{a}_\ell \rrbracket_{t'+k-1} - \llbracket \mathbf{a}_\ell \rrbracket_{t'+k-1}$ and $\llbracket \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1} \leftarrow \llbracket \mathbf{B}(\alpha_\ell) + \mathbf{b}_\ell \rrbracket_{t'+k-1} - \llbracket \mathbf{b}_\ell \rrbracket_{t'+k-1}$. Then Sim computes $\mathbf{A}, \mathbf{B}$ by interpolating evaluation points in $\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{D}$. For $\ell \in \mathcal{D}_1$, Sim computes $\llbracket \mathbf{a}_\ell \rrbracket_{t'+k-1} \leftarrow \llbracket \mathbf{A}(\alpha_\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{b}_\ell \rrbracket_{t'+k-1} \leftarrow \llbracket \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1}$, and computes $\llbracket \mathbf{c}_\ell \rrbracket_{t'+k-1}$ based on shares of corrupted parties and the secret $\mathbf{c}_\ell = \mathbf{a}_\ell * \mathbf{b}_\ell + \Delta_a^{(\ell)} * \mathbf{a}_\ell + \Delta_b^{(\ell)} * \mathbf{b}_\ell + \Delta_c^{(\ell)}$.

For $\ell \in \mathcal{D}_2$, Sim first computes $\llbracket \mathbf{A}(\alpha_\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1}$ and computes $\llbracket \mathbf{a}_\ell \rrbracket_{t'+k-1} \leftarrow \llbracket \mathbf{A}(\alpha_\ell) + \mathbf{a}_\ell \rrbracket_{t'+k-1} - \llbracket \mathbf{A}(\alpha_\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{b}_\ell \rrbracket_{t'+k-1} \leftarrow \llbracket \mathbf{B}(\alpha_\ell) + \mathbf{b}_\ell \rrbracket_{t'+k-1} - \llbracket \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1}$, and computes $\llbracket \mathbf{c}_\ell \rrbracket_{t'+k-1}$ based on shares of corrupted parties and the secret $\mathbf{c}_\ell = \mathbf{a}_\ell * \mathbf{b}_\ell + \Delta_a^{(\ell)} * \mathbf{a}_\ell + \Delta_b^{(\ell)} * \mathbf{b}_\ell + \Delta_c^{(\ell)}$.

By randomness of honest parties' shares of $\llbracket \mathbf{a}_\ell \rrbracket_{t'+k-1}, \llbracket \mathbf{b}_\ell \rrbracket_{t'+k-1}$ for $\ell \in \mathcal{D}_1 \cup \mathcal{D}_2$ and the one-to-one correspondence between the evaluation points in $\mathcal{D}_1$ and that in $\mathcal{E}_2$ given that in $\mathcal{E}_1 \cup \mathcal{D}$, the distributions in **Hyb_{2,1}** and **Hyb_{1,2}** are identical.

Hyb_{2,2}. In this hybrid, we simulate [AUTH-DEG-REDUCTION](#).

Hyb_{2,2,1}. In this hybrid, we simulate [RAND\(\$\Sigma_4\$ \)](#) in Step 1. When an honest party invokes [F_{base-DSS}](#) to distribute a random packed sharing, Sim first randomly samples shares of corrupted parties and then generate the whole sharing. We also simulate [RAND\(\$\Sigma_2\$ \)](#) in Step 1. When an honest party distributes a random sharing of the form $[\mathbf{o}]_{n'-1}$, Sim first randomly samples shares of corrupted parties and then samples the whole sharing subject to shares of corrupted parties and $\mathbf{o} = \mathbf{0}$. This makes no difference on the output distribution. The distributions in **Hyb_{2,2,1}** and **Hyb_{2,1}** are identical.

Hyb_{2,2,2}. In this hybrid, we change the computation of $[\mathbf{z}_\ell + \mathbf{u}_\ell]_{n'-1}$. Sim first randomly samples honest parties' shares of $[\mathbf{z}_\ell + \mathbf{u}_\ell]_{n'-1}$, computes $\mathbf{z}_\ell + \mathbf{u}_\ell$ further based on shares of corrupted parties. Sim then computes $\mathbf{u}_\ell \leftarrow (\mathbf{z}_\ell + \mathbf{u}_\ell) - \mathbf{z}_\ell$ and $\llbracket \mathbf{u}_\ell \rrbracket_{t'+k-1}$, and computes $[\mathbf{o}_\ell]_{n'-1} \leftarrow [\mathbf{z}_\ell + \mathbf{u}_\ell]_{n'-1} - [\mathbf{z}_\ell]_{t'+2k-2} - \llbracket \mathbf{u}_\ell \rrbracket_{t'+k-1}$. By randomness of honest parties' shares of $\llbracket \mathbf{u}_\ell \rrbracket_{t'+k-1}$ and $[\mathbf{o}_\ell]_{n'-1}$, the distributions in **Hyb_{2,2,2}** and **Hyb_{2,2,1}** are identical.

Hyb_{2,2,3}. In this hybrid, in Step 4, instead of invoking [F_{vrfy-linear-sharing}](#), Sim emulates by itself. The distributions in **Hyb_{2,2,3}** and **Hyb_{2,2,2}** are identical.

Hyb_{2,2,4}. In this hybrid, in Step 5, instead of invoking [F_{base-DSS}](#), Sim emulates by itself. The distributions in **Hyb_{2,2,4}** and **Hyb_{2,2,3}** are identical.

Hyb₃. In this hybrid, we simulate **triple verification**.

Hyb_{3,1}. In this hybrid, in Step 3.1, instead of invoking [F_{coin}](#), Sim emulates by sampling a random element r by itself. The distributions in **Hyb_{3,1}** and **Hyb_{2,2,4}** are identical.

Hyb_{3,2}. In this hybrid, in Step 2.3, instead of invoking [F_{base-DSS}](#), Sim emulates by itself and checks the correctness of active corrupted parties' shares. The distributions in **Hyb_{3,2}** and **Hyb_{3,1}** are identical.

Hyb_{3,3}. In this hybrid, in Step 3.2, if all requirements in Lemma 13 are satisfied, we change the computation of $\llbracket \mathbf{A}(r) \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(r) \rrbracket_{t'+k-1}$, and $\llbracket \mathbf{C}(r) \rrbracket_{t'+k-1}$. Denote the set $\{1, 2, \dots, m, r\}$ by $\mathcal{R}$. Concretely, Sim first computes corrupted parties' shares of $\llbracket \mathbf{A}(\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{C}(\ell) \rrbracket_{t'+k-1}$ for $\ell \in \mathcal{R}$. For each $\ell \in \mathcal{R}$, Sim then randomly samples $\mathbf{A}(\ell), \mathbf{B}(\ell)$, computes $\mathbf{C}(\ell) \leftarrow \mathbf{A}(\ell) * \mathbf{B}(\ell)$, and samples $\llbracket \mathbf{A}(\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{C}(\ell) \rrbracket_{t'+k-1}$ based on shares of corrupted parties and the secrets.

Sim computes $\llbracket \mathbf{A}(\alpha_\ell) \rrbracket_{t'+k-1}$ and $\llbracket \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1}$ for $\ell \in \mathcal{D}$ by interpolating the evaluation points in $\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{R}$ and computes $\llbracket \mathbf{C}(\alpha_\ell) \rrbracket_{t'+k-1}$ for $\ell \in \mathcal{D}$ by interpolating the evaluation points in $\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{R}$.

By randomness of $\{\llbracket \mathbf{A}(\alpha_\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{C}(\alpha_\ell) \rrbracket_{t'+k-1}\}_{\ell \in \mathcal{D}}$, the one-to-one correspondence between the evaluation points of $\mathbf{A}, \mathbf{B}$ in $\mathcal{D}$ and that in $\mathcal{R}$ given that in $\mathcal{E}_1 \cup \mathcal{E}_2$, and the one-to-one correspondence between the evaluation points of $\mathbf{C}$ in $\mathcal{R}$ and that in $\mathcal{D}$ given that in $\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{D}_1 \cup \mathcal{D}_2$, the distributions in $\mathbf{Hyb}_{3,3}$ and $\mathbf{Hyb}_{3,2}$ are identical.

Hyb_{3,4}. In this hybrid, in this hybrid, if all requirements in Lemma 13 are satisfied, we don't compute honest parties' shares of $\llbracket \mathbf{A}(\alpha_\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{B}(\alpha_\ell) \rrbracket_{t'+k-1}, \llbracket \mathbf{C}(\alpha_\ell) \rrbracket_{t'+k-1}$ for $\ell \in \mathcal{D} \cup \mathcal{D}_1 \cup \mathcal{D}_2$ and their shares of $\llbracket \mathbf{o}_\ell \rrbracket_{n'-1}, \llbracket \mathbf{u}_\ell \rrbracket_{t'+k-1}$ for $\ell \in [2m+1]$ used in **Auth-Deg-Reduction**. The distributions in $\mathbf{Hyb}_{3,4}$ and $\mathbf{Hyb}_{3,3}$ are identical.

Hyb_{3,5}. In this hybrid, if the requirements in Lemma 13 are not satisfied, however, the verification in Step 3 has been passed, Sim outputs $\perp$ and halts. By Lemma 13, the distribution in $\mathbf{Hyb}_{3,5}$ is statistically closed to that in $\mathbf{Hyb}_{3,4}$.

Hyb_{3,6}. In this hybrid, we simulate the interaction with the ideal functionality $\mathcal{F}_{\text{DSS}}$ as Step 7 of **Sim_{auth-packed-triple}**. Notice that if the requirements are not satisfied, since the protocol execution must be incorrect, a new dispute pair is guaranteed to be identified. Let honest parties output what they received from $\mathcal{F}_{\text{DSS}}$. The distributions in $\mathbf{Hyb}_{3,6}$ and $\mathbf{Hyb}_{3,5}$ are identical.

Noticing that $\mathbf{Hyb}_{3,5}$ already corresponds to the output distribution in the ideal world, we conclude that **Auth-Packed-Triples** securely computes Subfunctionality 1. of $\mathcal{F}_{\text{DSS}}$ in $\mathcal{F}_{\text{base-DSS}}, \mathcal{F}_{\text{coin}}, \mathcal{F}_{\text{vrfy-linear-sharing}}, \mathcal{F}_{\text{rand-OLE}}$ -hybrid model. $\square$

H.3 Proof for Lemma 3

Lemma 14 (Restatement of Lemma 3). ***Auth-Transpose** computes Subfunctionality 2. of $\mathcal{F}_{\text{DSS}}$, labeled with $(\text{Transpose}, m)$, with statistical security against any fully malicious adversary controlling up to t parties in the $\mathcal{F}_{\text{base-DSS}}, \mathcal{F}_{\text{coin}}$ -hybrid model.*

Proof (Proof of Lemma 3). We construct a simulator, **Sim_{auth-transpose}**, that interacts with $\mathcal{F}_{\text{DSS}}$, such that no environment $\mathcal{Z}$ can distinguish between an interaction with **Sim_{auth-transpose}** and an interaction with the adversary and **Auth-Transpose**. Let $\mathcal{H} \subseteq [n]$ denote the set of identities of honest parties.

Simulator 10: **Sim_{auth-transpose}** for **Auth-Transpose**

1. In Step 1.1, if P_i is corrupted, Sim emulates the interaction between $\mathcal{F}_{\text{base-DSS}}$ and the adversary and records the whole packed Shamir sharings; if P_i is honest, Sim randomly samples corrupted parties' shares and sends them to the adversary on behalf of $\mathcal{F}_{\text{base-DSS}}$.
2. Sim computes the difference $\{\Delta_{i,1}^{(\ell)}, \dots, \Delta_{i,k}^{(\ell)}\}_{\ell \in [N]}$ as follows. Suppose P_i distributes $[\mathbf{s}_{i,1*}^{(\ell)}]_{t'+k-1}, \dots, [\mathbf{s}_{i,k*}^{(\ell)}]_{t'+k-1}$ and $[\mathbf{s}_{i,*1}^{(\ell)}]_{t'+k-1}, \dots, [\mathbf{s}_{i,*k}^{(\ell)}]_{t'+k-1}$. Let $\mathbf{S}_i^{(\ell)} = (\mathbf{s}_{i,j*}^{(\ell)})_{j=1}^k$ and $\mathbf{s}_{i,*j}^{(\ell)}$ be the j -th column of $\mathbf{S}_i^{(\ell)}$. Sim then computes $\Delta_{i,j}^{(\ell)} \leftarrow \mathbf{s}_{i,*j}^{(\ell)} - \mathbf{s}_{i,j*}^{(\ell)}$ for $j \in [k]$.
3. In Step 2.1, Sim emulates $\mathcal{F}_{\text{coin}}$ by randomly sampling λ .
4. Sim checks whether the following requirements are satisfied.

– For each $i \in \mathcal{C} \setminus \text{Corr}$, $\ell \in [N]$, $\Delta_{i,1}^{(\ell)} = \dots = \Delta_{i,k}^{(\ell)} = \mathbf{0}$.

If the requirements are satisfied, Sim first computes corrupted parties' shares of $\{\llbracket \mathbf{r}_{j*} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{*j} \rrbracket_{t'+k-1}\}_{j=1}^k$, randomly samples $\mathbf{r}_{1*}, \dots, \mathbf{r}_{k*}$, and samples $\llbracket \mathbf{r}_{*1} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_{*k} \rrbracket_{t'+k-1}$ based on the secrets and shares of corrupted parties. In Step 2.2, Sim then broadcasts honest parties' shares on behalf of honest parties. In Step 2.3, Sim emulates $\mathcal{F}_{\text{base-DSS}}$ and checks whether the adversary provides correct shares.

Otherwise, Sim samples honest parties' shares of $\{[\mathbf{s}_{i,1*}^{(\ell)}]_{t'+k-1}, \dots, [\mathbf{s}_{i,k*}^{(\ell)}]_{t'+k-1}, [\mathbf{s}_{i,*1}^{(\ell)}]_{t'+k-1}, \dots, [\mathbf{s}_{i,*k}^{(\ell)}]_{t'+k-1}\}_{\ell \in [N]}$ and honestly follows the protocol in Step 2.2. In Step 2.3, Sim emulates $\mathcal{F}_{\text{base-DSS}}$. However, if the verifications in Steps 2.3 and 2.4 have been passed, Sim outputs $\perp$ and halts.

5. Sim interacts with $\mathcal{F}_{\text{DSS}}$.
 - If an active corrupted party P_i provides some incorrect shares in Step 2.2, Sim sends $(\text{reject}, \{i\})$ to $\mathcal{F}_{\text{DSS}}$.
 - If the adversary provides correct shares in Step 2.2,
 - if the requirements are satisfied, Sim computes active corrupted parties' shares of $(\llbracket \mathbf{r}_{j,1*}^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_{j,k*}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{j,*1}^{(\ell)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{r}_{j,*k}^{(\ell)} \rrbracket_{t'+k-1})$ for $\ell \in [N]$, $i \in [n-t]$ except for $(\ell, i) = (1, 1)$ and sends them together with **accept** to $\mathcal{F}_{\text{DSS}}$;
 - otherwise, Sim honestly follows the protocol in Step 2.5 to locate a dispute pair E and sends (reject, E) to $\mathcal{F}_{\text{DSS}}$.
6. Sim outputs what the adversary outputs and honest parties output what they receive from $\mathcal{F}_{\text{DSS}}$.

Hybrid arguments. Now we use hybrid arguments to show the validity of $\text{Sim}_{\text{auth-transpose}}$,

Hyb₀. This hybrid corresponds to real-world execution. In the end, the adversary outputs its outputs and honest parties output their outputs.

Hyb₁. In this hybrid, in Step 1.1, instead of invoking $\mathcal{F}_{\text{base-DSS}}$, Sim emulates itself. When an honest party needs to distribute a packed Shamir sharing, Sim first randomly samples shares of corrupted parties and then determines the whole sharing. The distributions in **Hyb₁** and **Hyb₀** are identical.

Hyb₂. In this hybrid, in Step 2.1, instead of invoking $\mathcal{F}_{\text{coin}}$, Sim emulates by randomly sampling λ . The distributions in **Hyb₂** and **Hyb₁** are identical.

Hyb₃. In this hybrid, in Step 2.3, instead of invoking $\mathcal{F}_{\text{base-DSS}}$, Sim emulates and checks the correctness of active corrupted parties' shares. The distributions in **Hyb₃** and **Hyb₂** are identical.

Hyb₄. In this hybrid, in Step 2.2, if the requirements in Step 4 of $\text{Sim}_{\text{auth-transpose}}$ are satisfied, we change the computation of honest parties' shares.

For $\ell \in [N]$, $i \in [n-t]$, Sim computes corrupted parties' shares of $\{\llbracket \mathbf{r}_{i,j*}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{i,*j}^{(\ell)} \rrbracket_{t'+k-1}\}_{j \in [k]}$, and randomly samples $\mathbf{r}_{i,1*}^{(\ell)}, \dots, \mathbf{r}_{i,k*}^{(\ell)}$. Let $\mathbf{R}_i^{(\ell)} = (\mathbf{r}_{i,j*}^{(\ell)})_{j=1}^k$ and $\mathbf{r}_{i,*j}^{(\ell)}$ be the j -th column of $\mathbf{R}_i^{(\ell)}$. Sim randomly samples honest parties' shares of $\{\llbracket \mathbf{r}_{i,j*}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{i,*j}^{(\ell)} \rrbracket_{t'+k-1}\}_{j \in [k]}$ based on the secrets and shares of corrupted parties.

Sim computes honest parties' shares of $\{\llbracket \mathbf{s}_{i,j*}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{s}_{i,*j}^{(\ell)} \rrbracket_{t'+k-1}\}_{\ell \in [N], i \in \mathcal{H}, j \in [k]}$ using the one-to-one correspondence defined by $\mathbf{M}_{\mathcal{H}}$, the $(n-t) \times (n-t)$ submatrix of the Vandermonde matrix $\mathbf{M}$ consisting of the columns corresponding to identities of honest parties. Due to the randomness of $\{\llbracket \mathbf{s}_{i,j*}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{s}_{i,*j}^{(\ell)} \rrbracket_{t'+k-1}\}_{\ell \in [N], i \in \mathcal{H}, j \in [k]}$ and the one-to-one correspondence, the distributions in **Hyb₄** and **Hyb₃** are identical.

Hyb₅. In this hybrid, in Step 2.2, if the requirements in Step 4 of $\text{Sim}_{\text{auth-transpose}}$ are satisfied, we change the computation of honest parties' shares. Sim first computes corrupted parties' shares of $\{\llbracket \mathbf{r}_{j*} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{*j} \rrbracket_{t'+k-1}\}_{j \in [k]}$ and randomly samples $\mathbf{r}_{1*}, \dots, \mathbf{r}_{k*}$. Let $\mathbf{R} = (\mathbf{r}_{j*})_{j=1}^k$ and $\mathbf{r}_{*j}$ be the j -th column of $\mathbf{R}$. Sim then randomly samples honest parties' shares of $\{\llbracket \mathbf{r}_{j*} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{*j} \rrbracket_{t'+k-1}\}_{j \in [k]}$ based on the secrets and shares of corrupted parties. For $j \in [k]$, Sim then computes

$$\begin{aligned}
 & (\llbracket \mathbf{r}_{1,j*}^{(1)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{1,*j}^{(1)} \rrbracket_{t'+k-1}) \\
 &= (\llbracket \mathbf{r}_{j*} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{*j} \rrbracket_{t'+k-1}) \\
 & - \sum_{1 \leq \ell \leq N, 1 \leq i \leq n-t, (\ell, i) \neq (1, 1)} \lambda^{(\ell-1) \cdot (n-t) + i - 1} \cdot (\llbracket \mathbf{r}_{i,j*}^{(\ell)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{i,*j}^{(\ell)} \rrbracket_{t'+k-1}).
 \end{aligned}$$

By randomness of $\{\llbracket \mathbf{r}_{1,j*}^{(1)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}_{1,*j}^{(1)} \rrbracket_{t'+k-1}\}_{j=1}^k$, the distributions in **Hyb₅** and **Hyb₄** are identical.

Hyb₆. In this hybrid, if the requirements are not satisfied, however, the verifications in Steps 2.3 and 2.4 have been passed, Sim outputs $\perp$ and halts. By Schwartz-Zippel Lemma, if the requirements are not satisfied and the adversary provides correct shares in Step 2.2, the verification in Step 2.4 will fail except with negligible probability. The distribution in **Hyb₆** is statistically closed to that in **Hyb₅**.

Hyb₇. In this hybrid, we simulate the interaction with the ideal functionality $\mathcal{F}_{\text{DSS}}$ as Step 5 of $\text{Sim}_{\text{auth-transpose}}$. Notice that if requirements are not satisfied, since the protocol execution must be incorrect, a new dispute pair is guaranteed to be identified. Let honest parties output what they received from $\mathcal{F}_{\text{DSS}}$. The distributions in **Hyb₇** and **Hyb₆** are identical.

Noticing that **Hyb₇** already corresponds to the output distribution in the ideal world, we conclude that **AUTH-TRANPOSE** securely computes Subfunctionality 2. of $\mathcal{F}_{\text{DSS}}$ in $\mathcal{F}_{\text{base-DSS}}, \mathcal{F}_{\text{coin}}$ -hybrid model. $\square$

H.4 Proof for Lemma 4

Lemma 15 (Restatement of Lemma 4). ***DSS-RECON** computes Subfunctionality 3. of $\mathcal{F}_{\text{DSS}}$, labeled with **Recon**, with statistical security against any fully malicious adversary controlling up to t parties in the $\mathcal{F}_{\text{base-DSS}}, \mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{coin}}$ -hybrid model.*

Proof (Proof of Lemma 4). We construct a simulator, $\text{Sim}_{\text{DSS-recon}}$, that interacts with $\mathcal{F}_{\text{DSS}}$, such that no environment $\mathcal{Z}$ can distinguish between an interaction with $\text{Sim}_{\text{DSS-recon}}$ and an interactions with the adversary and **DSS-RECON**. Let $\mathcal{H} \subseteq [n]$ denote the set of identities of honest parties.

Simulator 11: $\text{Sim}_{\text{DSS-recon}}$ for **DSS-RECON**

1. **Sim** invokes the ideal functionality $\mathcal{F}_{\text{DSS}}$ and receives corrupted parties' shares of $[\![\mathbf{x}_1]\!]_{t'+k-1}, \dots, [\![\mathbf{x}_M]\!]_{t'+k-1}$. **Sim** follows the protocol to uniformly divide the computation into n^2 segments and initializes $i \leftarrow 1$.
2. In **setup** of **DSS**, **Sim** invokes $\text{Sim}_{\text{rand-depack}}$. In case of success, **Sim** learns corrupted parties' shares of $\{[\![\mathbf{r}_\ell]\!]_{t'+k-1}, \langle r_{\ell,1} \rangle_t, \dots, \langle r_{\ell,k} \rangle_t\}_{\ell \in [M]}$.
3. In Step 2.1, **Sim** computes corrupted parties' shares of $[\![\mathbf{u}_\ell]\!]_{t'+k-1}$, randomly samples $[\![\mathbf{u}_\ell]\!]_{t'+k-1}$ based on shares of corrupted parties, and sends shares of honest parties to P_{king} .
4. In Step 2.2, if P_{king} is honest, **Sim** honestly follows the protocol. **Sim** receives $\mathbf{u}_\ell^{(i)}$ from P_{king} on behalf of honest P_i .
5. In Step 3.1, **Sim** emulates $\mathcal{F}_{\text{coin}}$ by randomly sampling λ .
6. In Step 3.2, **Sim** honestly broadcasts shares of honest parties and emulates $\mathcal{F}_{\text{base-DSS}}$ to check the correctness of active corrupted parties' shares. If an active corrupted party P_i provides some incorrect shares in Step 3.2, **Sim** invokes **DISPUTE-CONTROL** to update $\mathcal{P}_{\text{active}}, \text{Corr}$, and Disp .
 - If $|\mathcal{P}_{\text{active}}| \geq t' + 2k - 1$, if $i < n^2$, set $i \leftarrow i + 1$, assign relay parties, and go to Step 3.
 - Otherwise, **Sim** sends **reject** and $\mathcal{C}' \leftarrow \text{Corr}$ to $\mathcal{F}_{\text{DSS}}$.
7. In Steps 3.3 and 3.4, **Sim** honestly follows the protocol. If the verification fails, **Sim** honestly follows the protocol to locate a dispute pair E and invokes **DISPUTE-CONTROL** to update $\mathcal{P}_{\text{active}}, \text{Corr}$, and Disp .
 - If $|\mathcal{P}_{\text{active}}| \geq t' + 2k - 1$, if $i < n^2$, set $i \leftarrow i + 1$, assign relay parties, and go to Step 3.
 - Otherwise, **Sim** sends **reject** and $\mathcal{C}' \leftarrow \text{Corr}$ to $\mathcal{F}_{\text{DSS}}$.
8. If all detectable secret sharings have been depacked, **Sim** sends **confirm** to $\mathcal{F}_{\text{DSS}}$ and continues the simulation below.
9. In Step 4, for each corrupted receiver P_o , for each output $x_{\ell,j}$ belonging to P_o , **Sim** receives the secret $x_{\ell,j}$ from $\mathcal{F}_{\text{DSS}}$, and passes it to the adversary to emulate $\mathcal{F}_{\text{VSS}}$.

Simulator 12: $\text{Sim}_{\text{rand-depack}}$ for **RAND-DEPACK**

1. In Step 1.1, if P_i is corrupted, **Sim** emulates the interaction between $\mathcal{F}_{\text{base-DSS}}, \mathcal{F}_{\text{VSS}}$ and the adversary and records the whole Shamir sharings; if P_i is honest, **Sim** randomly samples corrupted parties' shares and sends shares of active corrupted parties to the adversary on behalf of $\mathcal{F}_{\text{base-DSS}}$ and $\mathcal{F}_{\text{VSS}}$.

2. For each active corrupted party P_i and $\ell \in [N]$, suppose P_i distributes $\llbracket \mathbf{s}_i^{(\ell)} \rrbracket_{t'+k-1}$ and $\langle \overline{s_{i,1}^{(\ell)}} \rangle_t, \dots, \langle \overline{s_{i,k}^{(\ell)}} \rangle_t$. Sim computes $\mathbf{s}_i^{(\ell)}, s_{i,1}^{(\ell)}, \dots, s_{i,k}^{(\ell)}$, and $\Delta_{i,j}^{(\ell)} \leftarrow \overline{s_{i,j}^{(\ell)}} - s_{i,j}^{(\ell)}$ for $j \in [k]$.
3. In Step 2.1, Sim emulates $\mathcal{F}_{\text{coin}}$ by randomly sampling λ .
4. Sim checks whether the following requirements are satisfied.
 - For each $i \in \mathcal{C} \setminus \text{Corr}$, $\ell \in [N]$, $\Delta_{i,1}^{(\ell)} = \dots = \Delta_{i,k}^{(\ell)} = 0$.
 If the requirements are satisfied,
 - (a) Sim first computes corrupted parties' shares of $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$, randomly samples $\mathbf{r}$, and computes $\langle r_1 \rangle_t, \dots, \langle r_k \rangle_t$ based on the secrets and shares of corrupted parties.
 - (b) In Step 2.2, Sim broadcasts shares of honest parties.
 - (c) In Step 2.3, Sim emulates $\mathcal{F}_{\text{base-DSS}}$ to check shares of active corrupted parties.
 - (d) In Step 2.4, Sim emulates $\mathcal{F}_{\text{VSS}}$ to output $r_1, \dots, r_k$.
 - (e) Sim computes corrupted parties' shares of $\llbracket \mathbf{r}_i^{(\ell)} \rrbracket_{t'+k-1}, \langle r_{i,1}^{(\ell)} \rangle_t, \dots, \langle r_{i,k}^{(\ell)} \rangle_t$ for $(\ell, i) \in [N] \times [n-t]$.
 Otherwise,
 - (a) Sim samples honest parties' shares of $\{\llbracket \mathbf{s}_i^{(\ell)} \rrbracket_{t'+k-1}, \langle s_{i,1}^{(\ell)} \rangle_t, \dots, \langle s_{i,k}^{(\ell)} \rangle_t\}_{\ell \in [N]}$ for $i \in \mathcal{H}$ and honestly follows the protocol in Step 2.2.
 - (b) In Steps 2.3 and 2.4, Sim emulates $\mathcal{F}_{\text{base-DSS}}$ and $\mathcal{F}_{\text{VSS}}$, respectively.
 - (c) In Step 2.5, Sim honestly follows the protocol. If the verification has been passed, Sim outputs $\perp$ and halts. Otherwise, in Step 2.6, Sim honestly follows the protocol to locate a dispute pair E .
5. Sim interacts with $\mathcal{F}_{\text{DSS}}$.
 - If $|\mathcal{P}_{\text{active}}| < t' + 2k - 1$, Sim sends **reject** and $\mathcal{C}' \leftarrow \text{Corr}$ to $\mathcal{F}_{\text{DSS}}$.

Hybrid arguments. Now we use hybrid arguments to show the validity of $\text{Sim}_{\text{DSS-recon}}$.

Hyb₀. This hybrid corresponds to real-world execution. In the end, the adversary outputs its outputs and honest parties output their outputs.

Hyb₁. In this hybrid, we simulate Rand-Depack .

Hyb_{1,1}. In this hybrid, in Step 1.1, instead of invoking $\mathcal{F}_{\text{base-DSS}}$ and $\mathcal{F}_{\text{VSS}}$, Sim emulates by itself. For each honest party P_i , Sim first randomly samples corrupted parties' shares and then generates the whole sharings. The distributions in **Hyb_{1,1}** and **Hyb₀** are identical.

Hyb_{1,2}. In this hybrid, in Step 2.1, instead of invoking $\mathcal{F}_{\text{coin}}$, Sim emulates by randomly sampling λ . The distributions in **Hyb_{1,2}** and **Hyb_{1,1}** are identical.

Hyb_{1,3}. In this hybrid, in Step 2.3 and 2.4, instead of invoking $\mathcal{F}_{\text{base-DSS}}$ and $\mathcal{F}_{\text{VSS}}$, Sim emulates $\mathcal{F}_{\text{base-DSS}}$ and $\mathcal{F}_{\text{VSS}}$ by itself. The distributions in **Hyb_{1,3}** and **Hyb_{1,2}** are identical.

Hyb_{1,4}. In this hybrid, if the requirements in Step 4 of $\text{Sim}_{\text{rand-depack}}$ are satisfied, in Step 2.2, Sim changes the computation of honest parties' shares.

For $\ell \in [N], i \in [n-t]$, Sim first computes corrupted parties' shares of $(\llbracket \mathbf{r}_i^{(\ell)} \rrbracket_{t'+k-1}, \langle r_{i,1}^{(\ell)} \rangle_t, \dots, \langle r_{i,k}^{(\ell)} \rangle_t)$. Sim randomly samples $\mathbf{r}_i^{(\ell)}$, and randomly samples honest parties' shares of $(\llbracket \mathbf{r}_i^{(\ell)} \rrbracket_{t'+k-1}, \langle r_{i,1}^{(\ell)} \rangle_t, \dots, \langle r_{i,k}^{(\ell)} \rangle_t)$ based on the secrets and shares of corrupted parties. After that, Sim computes $(\llbracket \mathbf{s}_i^{(\ell)} \rrbracket_{t'+k-1}, \langle s_{i,1}^{(\ell)} \rangle_t, \dots, \langle s_{i,k}^{(\ell)} \rangle_t)$ for $\ell \in [N], i \in \mathcal{H}$ using the one-to-one correspondence defined by the matrix $\mathbf{M}_{\mathcal{H}}$.

Due to the randomness of $\{\llbracket \mathbf{s}_i^{(\ell)} \rrbracket_{t'+k-1}, \langle s_{i,1}^{(\ell)} \rangle_t, \dots, \langle s_{i,k}^{(\ell)} \rangle_t\}_{\ell \in [N], i \in \mathcal{P} \setminus \mathcal{C}}$ and the one-to-one correspondence, the distributions in **Hyb_{1,4}** and **Hyb_{1,3}** are identical.

Hyb_{1,5}. In this hybrid, if the requirements in Step 4 of $\text{Sim}_{\text{rand-depack}}$ are satisfied, in Step 2.2, Sim changes the computation of honest parties' shares.

Sim first computes corrupted parties' shares of $\llbracket \mathbf{r} \rrbracket_{t'+k-1}, \langle r_1 \rangle_t, \dots, \langle r_k \rangle_t$, randomly samples $\mathbf{r}$, and randomly samples honest parties' shares of $\llbracket \mathbf{r} \rrbracket_{t'+k-1}, \langle r_1 \rangle_t, \dots, \langle r_k \rangle_t$. After that, Sim computes

$$\begin{aligned} & (\llbracket \mathbf{r}_1^{(1)} \rrbracket_{t'+k-1}, \langle r_{1,1}^{(1)} \rangle_t, \dots, \langle r_{1,k}^{(1)} \rangle_t) \\ &= (\llbracket \mathbf{r} \rrbracket_{t'+k-1}, \langle r_1 \rangle_t, \dots, \langle r_k \rangle_t) \\ & \quad - \sum_{\ell \in [N], i \in [n-t], (\ell, i) \neq (1, 1)} \lambda^{(\ell-1) \cdot (n-t) + i - 1} \cdot (\llbracket \mathbf{r}_i^{(\ell)} \rrbracket_{t'+k-1}, \langle r_{i,1}^{(\ell)} \rangle_t, \dots, \langle r_{i,k}^{(\ell)} \rangle_t). \end{aligned}$$

By randomness of $(\llbracket \mathbf{r}_1^{(1)} \rrbracket_{t'+k-1}, \langle r_{1,1}^{(1)} \rangle_t, \dots, \langle r_{1,k}^{(1)} \rangle_t)$, the distributions in **Hyb**_{1,5} and **Hyb**_{1,4} are identical.

Hyb_{1,6}. In this hybrid, if the requirements in Step 4 of **Sim**_{rand-depack} are satisfied, in Step 2.2, Sim changes the computation of honest parties' shares. We don't compute honest parties' shares of $\llbracket \mathbf{r}_i^{(\ell)} \rrbracket, \langle r_{i,1}^{(\ell)} \rangle_t, \dots, \langle r_{i,k}^{(\ell)} \rangle_t$ for $\ell \in [N], i \in [n-t]$. The distributions in **Hyb**_{1,6} and **Hyb**_{1,5} are identical.

Hyb_{1,7}. In this hybrid, if the requirements in Step 4 of **Sim**_{rand-depack} are not satisfied, if the verification in Step 2.5 has been passed, Sim outputs $\perp$ and halts. By Schwartz-Zippel Lemma, if the requirements are not satisfied and the adversary provides correct shares in Step 2.2, the verification in Step 2.5 will fail except with negligible probability. The distribution in **Hyb**_{1,7} is statistically closed to that in **Hyb**_{1,6}.

Hyb_{1,8}. In this hybrid, we simulate the interaction with $\mathcal{F}_{\text{DSS}}$. If $|\mathcal{P}_{\text{active}}| < t' + 2k - 1$, Sim sends **reject** and $\mathcal{C}' \leftarrow \text{Corr}$ to $\mathcal{F}_{\text{DSS}}$. The distributions in **Hyb**_{1,8} and **Hyb**_{1,7} are identical.

Hyb₂. In this hybrid, in Step 2.1 of **DSS-RECON**, we change the computation of honest parties' shares of $\llbracket \mathbf{u}_\ell \rrbracket_{t'+k-1}$. Sim first computes corrupted parties' shares of $\llbracket \mathbf{u}_\ell \rrbracket_{t'+k-1}$ and randomly samples shares of honest parties based on shares of corrupted parties. Sim then computes $\llbracket \mathbf{r}_\ell \rrbracket_{t'+k-1} \leftarrow \llbracket \mathbf{u}_\ell \rrbracket_{t'+k-1} - \llbracket \mathbf{x}_\ell \rrbracket_{t'+k-1}$. Due to the randomness of honest parties' shares of $\llbracket \mathbf{r}_\ell \rrbracket_{t'+k-1}$, the distributions in **Hyb**₂ and **Hyb**_{1,8} are identical.

Hyb₃. In this hybrid, in Step 3.1, instead of invoking $\mathcal{F}_{\text{coin}}$, Sim emulates it by randomly sampling λ . The distributions in **Hyb**₃ and **Hyb**₂ are identical.

Hyb₄. In this hybrid, in Step 3.2, instead of invoking $\mathcal{F}_{\text{base-DSS}}$, Sim emulates by itself to check the correctness of active corrupted parties' shares. The distributions in **Hyb**₄ and **Hyb**₃ are identical.

Hyb₅. In this hybrid, we simulate the interaction with $\mathcal{F}_{\text{DSS}}$.

- If $|\mathcal{P}_{\text{active}}| < t' + 2k - 1$, Sim sends **reject** and $\mathcal{C}' \leftarrow \text{Corr}$ to $\mathcal{F}_{\text{DSS}}$.
- If all detectable secret sharings have been successfully depacked, Sim sends **confirm** to $\mathcal{F}_{\text{DSS}}$, and receives corrupted parties' outputs from $\mathcal{F}_{\text{DSS}}$. Sim uses corrupted parties' shares received from $\mathcal{F}_{\text{DSS}}$ for simulation in Step 2.1 and doesn't compute honest parties' shares of $\llbracket \mathbf{x}_\ell \rrbracket_{t'+k-1}$. In Step 4, Sim uses the corrupted parties' outputs received from $\mathcal{F}_{\text{DSS}}$ to emulate $\mathcal{F}_{\text{VSS}}$.

If the verification in Step 3.3 has been passed, by Schwartz-Zippel Lemma, we have that $\mathbf{u}_\ell = \overline{\mathbf{u}_\ell^{(i)}}$ for $\ell \in [M]$, $i \in \mathcal{H}$, which means all parties obtain correct shares of $\langle x_{\ell,j} \rangle_t$. The distribution in **Hyb**₅ is statistically closed to that in **Hyb**₄.

Noticing that **Hyb**₅ already corresponds to the output distribution in the ideal world, we conclude that **DSS-RECON** securely computes Subfunctionality 3. of $\mathcal{F}_{\text{DSS}}$ in $\mathcal{F}_{\text{base-DSS}}, \mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{coin}}$ -hybrid model. $\square$

H.5 Proof for Lemma 6

Lemma 16 (Restatement of Lemma 6). *Let $k = (n + 5)/12$ and $t = (n - 1)/2$. **MAIN** computes $\mathcal{F}_{\text{main}}$ in $\mathcal{F}_{\text{DSS}}, \mathcal{F}_{\text{verfy-linear-sharing}}, \mathcal{F}_{\text{coin}}, \mathcal{F}_{\text{main-client-server}}^*$ -hybrid model with statistical security against a fully malicious adversary which corrupts at most t parties.*

Proof (Proof of Lemma 6). We construct a simulator, **Sim**_{main}, that interacts with $\mathcal{F}_{\text{main}}$, such that no environment $\mathcal{Z}$ can distinguish between an interaction with **Sim**_{main} and an interaction with the adversary and **MAIN**. Let $\mathcal{H} \subseteq [n]$ denote the set of identities of honest parties.

Simulator 13: Sim_{main} for MAIN

Sim invokes **DISPUTE-CONTROL** to initialize $\mathcal{P}_{\text{active}}, \text{Corr}, \text{Disp}$. In order to emulate $\mathcal{F}_{\text{DSS}}$, Sim initializes a database to record authenticated packed Shamir sharings. Sim invokes the adversary and the ideal functionality $\mathcal{F}_{\text{main}}$ and samples the internal randomness of the adversary.

1. In the preprocessing phase, uniformly divide the preprocessing phase into n^2 segments and initialize $i \leftarrow 1$.
 - (a) In Step 1.1, Sim emulates the interaction between $\mathcal{F}_{\text{DSS}}$ and the adversary.
 - (b) Upon receiving **accept** from the adversary, Sim records shares of corrupted parties. If $i < n^2$, set $i \leftarrow i + 1$ and go to Step 1.a; otherwise, go to Step 2.
 - (c) Upon receiving (**reject**, E) from the adversary, Sim invokes **DISPUTE-CONTROL** to update $\mathcal{P}_{\text{active}}, \text{Corr}, \text{Disp}$.
 - If $|\mathcal{P}_{\text{active}}| \geq t' + 2k - 1$, go to Step 1.a.
 - Otherwise, Sim outputs **abort** and $\mathcal{C}'$ on behalf of $\mathcal{F}_{\text{DSS}}$ and goes to Step 4.
2. In the online phase, uniformly divide the circuit into n^2 segments and initialize $i \leftarrow 1$.
 - (a) In Step 2.1, if it is an input segment, Sim invokes **Sim_{input}**; if it is a computation segment, Sim invokes **Sim_{comp-seg}**.
 - (b) In case of success, if $i < n^2$, set $i \leftarrow i + 1$ and go to Step 2.a; otherwise, go to Step 3.
 - (c) Otherwise, all parties have identified a new dispute pair E . Sim invokes **DISPUTE-CONTROL** to update $\mathcal{P}_{\text{active}}, \text{Corr}, \text{Disp}$.
 - If $|\mathcal{P}_{\text{active}}| \geq t' + 2k - 1$, assign relay parties. For each honest relay party P_r who passes messages from an honest party to a corrupted party, since Sim can simulate the message, Sim uses it to simulate P_r . Go to Step 2.a.
 - Otherwise, go to Step 4.
3. In Step 3, Sim emulates the interaction between $\mathcal{F}_{\text{DSS}}$ and the adversary.
 - Upon receiving **confirm**, Sim sends circuit inputs of corrupted parties to $\mathcal{F}_{\text{main}}$ and honest parties provide their inputs. Sim then receives circuit outputs from $\mathcal{F}_{\text{main}}$.
 - Upon receiving **reject** and $\mathcal{C}'$, Sim outputs **abort** and $\mathcal{C}'$ on behalf of $\mathcal{F}_{\text{DSS}}$ and goes to Step 4.
4. **Restart.** Sim emulates $\mathcal{F}_{\text{main-client-server}}^*$ by passing the circuit inputs received from the adversary to $\mathcal{F}_{\text{main}}$ and sending the circuit outputs received from $\mathcal{F}_{\text{main}}$ to the adversary.
5. Sim outputs what the adversary outputs and honest parties output what they received from $\mathcal{F}_{\text{main}}$.

Simulator 14: Sim_{comp-seg} for COMP-SEG

In the beginning, Sim has learnt corrupted parties' shares of the input sharings of this computation segment and the preprocessing data used in this segment.

1. In Step 2.1, Sim invokes **Sim_{rand}**(Σ_2).
2. In Step 2.2, Sim invokes **Sim_{rand-linear-mask}**.
3. In Step 3.1, Sim invokes **Sim_{select}** and **Sim_{sharing-trans}**.
4. In Step 3.2, for addition gates, Sim computes corrupted parties' shares as $[z]_{t'+k-1} \leftarrow [x]_{t'+k-1} + [y]_{t'+k-1}$; for multiplication gates, Sim invokes **Sim_{mult}**.
5. In Step 3.3, Sim invokes **Sim_{sharing-trans}**.
6. In Step 4.1, Sim emulates $\mathcal{F}_{\text{vrfy-linear-sharing}}$ and checks whether P_{king} distributes valid degree- $(k-1)$ packed Shamir sharings.
7. In Step 4.2, Sim invokes **Sim_{vrfy-sharing-trans}**.
8. In Step 4.3, Sim invokes **Sim_{vrfy-transcripts}**.
9. In Step 5, Sim emulates the interaction between $\mathcal{F}_{\text{DSS}}$ and the adversary.

In case of success, **Sim** learns corrupted parties' shares of each output sharing of this computation segment.

Simulator 15: Sim_{mult} for **MULT**

In the beginning, **Sim** has computed corrupted parties' shares of the input sharings $[\mathbf{x}]_{t'+k-1}$, $[\mathbf{y}]_{t'+k-1}$, and has learnt corrupted parties' shares of the packed triple $([\mathbf{a}]_{t'+k-1}, [\mathbf{b}]_{t'+k-1}, [\mathbf{c}]_{t'+k-1})$ and the random mask $[\mathbf{r}]_{t'+k-1}$.

1. In Step 1, **Sim** first computes corrupted parties' shares of $[\mathbf{v}_1]_{t'+k-1}$ and $[\mathbf{v}_2]_{t'+k-1}$, randomly samples shares of honest parties based on shares of corrupted parties, and sends them to P_{king} .
2. In Step 2, if P_{king} is honest, **Sim** honestly follows the protocol. **Sim** receives from P_{king} honest parties' shares of $[\mathbf{v}_1]_{k-1}$ and $[\mathbf{v}_2]_{k-1}$.
3. In Step 3, **Sim** randomly samples honest parties' shares of $[\mathbf{e}]_{n'-1}$ and sends them to P_{king} .
4. In Step 4, if P_{king} is honest, **Sim** honestly follows the protocol. **Sim** receives from P_{king} honest parties' shares of $[\mathbf{e}]_{k-1}$.

In the end, **Sim** learns corrupted parties' shares of the output sharing $[\mathbf{z}]_{t'+k-1}$.

Simulator 16: $\text{Sim}_{\text{rand-linear-mask}}$ for **RAND-LINEAR-MASK**

In the beginning, **Sim** has learnt corrupted parties' shares of each authenticated random linear mask for transpose.

1. In Step 1.a, for $\ell \in [k]$, **Sim** samples random elements as honest parties' shares of $[\mathbf{v}_{*\ell} + \mathbf{u}_{*\ell}]_{n'-1}$ and sends them to P_{king} on behalf of honest parties.
2. In Step 1.b, if P_{king} is honest, **Sim** honestly follows the protocol. For $\ell \in [k]$, **Sim** receives from P_{king} honest parties' shares of $[\mathbf{v}_{*\ell} + \mathbf{u}_{*\ell}]_{k-1}$ and $[\mathbf{v}_{\ell*} + \mathbf{u}_{\ell*}]_{k-1}$.
3. In Step 2, **Sim** emulates $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_3)$ to check whether the sharings received from P_{king} in Step 1.b are valid degree- $(k-1)$ packed Shamir sharings and their secrets satisfy the transpose relation.

In case of success, **Sim** learns corrupted parties' shares of $([\mathbf{r}]_{t'+k-1}, [\mathbf{L}(\mathbf{r})]_{t'+k-1})$ for each linear transformation $\mathbf{L}$.

Simulator 17: $\text{Sim}_{\text{sharing-trans}}$ for **SHARING-TRANS**

In the beginning, **Sim** has computed corrupted parties' shares of $[\mathbf{x}]_{t'+k-1}$ and the random linear mask $([\mathbf{r}]_{t'+k-1}, [\mathbf{L}(\mathbf{r})]_{t'+k-1})$.

1. In Step 1, **Sim** computes corrupted parties' shares of $[\mathbf{x} + \mathbf{r}]_{t'+k-1}$, randomly samples honest parties' shares based on shares of corrupted parties, and sends them to P_{king} .
2. In Step 2, if P_{king} is honest, **Sim** honestly follows the protocol. **Sim** receives from P_{king} honest parties' shares of $[\mathbf{x} + \mathbf{r}]_{k-1}$ and $[\mathbf{L}(\mathbf{x} + \mathbf{r})]_{k-1}$, from P_{king} .

In the end, **Sim** learns corrupted parties' shares of $[\mathbf{L}(\mathbf{x})]_{t'+k-1}$.

Simulator 18: $\text{Sim}_{\text{select}}$ for **SELECT**

In the beginning, **Sim** has computed corrupted parties' shares of $[\mathbf{x}_1]_{t'+k-1}, \dots, [\mathbf{x}_k]_{t'+k-1}$ and $[\mathbf{r}]_{t'+k-1}$.

1. In Step 1, Sim samples random elements as honest parties' shares of $[e]_{n'-1}$ and sends them to P_{king} .
2. In Step 2, if P_{king} is honest, Sim honestly follows the protocol. Sim receives from P_{king} honest parties' shares of $[e]_{k-1}$.

In the end, Sim learns corrupted parties' shares of $[x]_{t'+k-1}$.

Simulator 19: $\text{Sim}_{\text{vrfy-sharing-trans}}$ for **VRIFY-SHARING-TRANS**

In the beginning, Sim has learnt honest parties' shares of $\{[x_\ell + r_\ell]_{k-1}, [L(x_\ell + r_\ell)]_{k-1}\}_{\ell \in [m]}$.

1. In Step 1, for $\ell \in [m]$ and $i \in [k]$, Sim computes honest parties' shares of $[o_{\ell,i}]_{2k-2}$.
2. In Step 2, Sim emulates $\mathcal{F}_{\text{vrfy-linear-sharing}}$ to check whether the sharings computed above are valid degree- $(2k-2)$ packed Shamir sharings with each secret having its components summing to 0.

Simulator 20: $\text{Sim}_{\text{vrfy-transcripts}}$ for **VRIFY-TRANSCRIPTS**

1. In Step 1, Sim invokes $\text{Sim}_{\text{de-linearization}}$.
2. In Step 2, Sim invokes $\text{Sim}_{\text{dimension-reduction}}$.
3. In Step 3, Sim invokes $\text{Sim}_{\text{randomization}}$.
4. In Step 4, Sim invokes $\text{Sim}_{\text{vrfy-single-transcript}}$.

Simulator 21: $\text{Sim}_{\text{extend-deg-reduction}}$ for **EXTEND-DEG-REDUCTION**

In the beginning, Sim has learnt honest parties' shares of $[\vec{v}_1]_{k-1}$, $[\vec{v}_2]_{k-1}$, $[\vec{d}]_{k-1}$ and corrupted parties' shares of $[\vec{u}]_{t'+k-1}$, $[\vec{r}]_{t'+k-1}$.

1. In Step 1, Sim randomly samples honest parties' shares of $[e]_{n'-1}$ and sends them to P_{king} on behalf of honest parties.
2. In Step 2, if P_{king} is honest, Sim honestly follows the protocol. Sim receives honest parties' shares of $[e]_{k-1}$ from P_{king} .

In the end, Sim has learnt honest parties' shares of $[e]_{n'-1}$ and $[e]_{k-1}$.

Simulator 22: $\text{Sim}_{\text{vrfy-single-transcript}}$ for **VRIFY-SINGLE-TRANSCRIPT**

1. Recall Sim has learnt honest parties' shares of $[v_1]_{k-1}$, $[v_2]_{k-1}$, $[d]_{k-1}$, $[e]_{n'-1}$, $[e]_{k-1}$ and corrupted parties' shares of $[v_1]_{k-1}$, $[v_2]_{k-1}$, $[d]_{k-1}$, $[\vec{u}]_{t'+k-1}$, $[\vec{r}]_{t'+k-1}$. Sim first computes v_1 , v_2 , d , and e from $[v_1]_{k-1}$, $[v_2]_{k-1}$, $[d]_{k-1}$, and $[e]_{n'-1}$, respectively. Sim then randomly samples u and computes $r \leftarrow e - v_1 * v_2 - d * u$. Sim samples honest parties' shares of $[u]_{t'+k-1}$, $[r]_{t'+k-1}$ based on the secrets and shares of corrupted parties, and computes $[o]_{n'-1} \leftarrow [e]_{n'-1} - [v_1]_{k-1} \cdot [v_2]_{k-1} - [d]_{k-1} \cdot [u]_{t'+k-1} - [r]_{t'+k-1}$.
2. In Steps 1 and 2, Sim honestly follows the protocol.
3. In Step 3.a, Sim emulates $\mathcal{F}_{\text{DSS}}$ to check the correctness of active corrupted parties' shares.
4. In Step 3.b, Sim honestly follows the protocol to check the correctness of $[o]_{n'-1}$. In case of failure, Sim invokes $\text{Sim}_{\text{FL-ZeroSharing}}$.
5. In Step 3.c, Sim honestly follows the protocol.

Simulator 23: Sim_{de-linearization} for DE-LINEARIZATION

In the beginning, Sim has learnt honest parties' shares of $\{\vec{v}_1^{(i)}\}_{k-1}, [\vec{v}_2^{(i)}]_{k-1}, [\vec{d}^{(i)}]_{k-1}, [e^{(i)}]_{n'-1}, [e^{(i)}]_{k-1}\}_{\ell=1}^M$ and corrupted parties' shares of $\{\llbracket \vec{u}^{(i)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}\}_{\ell=1}^M$.

1. In Step 1, Sim emulates $\mathcal{F}_{\text{coin}}$ by randomly sampling λ .
2. In Step 2, Sim computes corrupted parties' shares of $[\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}, [\vec{d}]_{k-1}, \llbracket \vec{u} \rrbracket_{k-1}, \llbracket \mathbf{r} \rrbracket_{t'+k-1}, [\mathbf{o}]_{n'-1}, [e]_{n'-1}, [e]_{k-1}$.

In the end, Sim learns honest parties' shares of $[\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}, [\vec{d}]_{k-1}, [e]_{n'-1}, [e]_{k-1}$ and corrupted parties' shares of $\llbracket \vec{u} \rrbracket_{t'+k-1}, \llbracket \mathbf{r} \rrbracket_{t'+k-1}$.

Simulator 24: Sim_{compress} for COMPRESS

In the beginning, Sim has learnt honest parties' shares of $\{\vec{v}_1^{(i)}\}_{k-1}, [\vec{v}_2^{(i)}]_{k-1}, [\vec{d}^{(i)}]_{k-1}, [e^{(i)}]_{n'-1}, [e^{(i)}]_{k-1}\}_{\ell=1}^M$ and corrupted parties' shares of $\{\llbracket \vec{u}^{(i)} \rrbracket_{t'+k-1}, \llbracket \mathbf{r}^{(i)} \rrbracket_{t'+k-1}\}_{\ell=1}^M$.

1. In Step 3, Sim invokes $\text{Sim}_{\text{extend-deg-reduction}}$.
2. In Step 5, Sim emulates $\mathcal{F}_{\text{coin}}$ by randomly sampling λ .

In the end, Sim has learnt honest parties' shares of $[\vec{V}_1(\lambda)]_{k-1}, [\vec{V}_2(\lambda)]_{k-1}, [\vec{D}(\lambda)]_{k-1}, [E(\lambda)]_{n'-1}, [E(\lambda)]_{k-1}$ and corrupted parties' shares of $\llbracket \vec{U}(\lambda) \rrbracket_{t'+k-1}, \llbracket R(\lambda) \rrbracket_{t'+k-1}$.

Simulator 25: Sim_{dimension-reduction} for DIMENSION-REDUCTION

In the beginning, Sim has learnt honest parties' shares of $[\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}, [\vec{d}]_{k-1}, [e]_{n'-1}, [e]_{k-1}$ and corrupted parties' shares of $\llbracket \vec{u} \rrbracket_{t'+k-1}, \llbracket \mathbf{r} \rrbracket_{t'+k-1}$.

1. In Step 2, Sim invokes $\text{Sim}_{\text{extend-deg-reduction}}$.
2. In Step 4, Sim invokes $\text{Sim}_{\text{compress}}$.

Simulator 26: Sim_{randomization} for RANDOMIZATION

In the beginning, Sim has learnt honest parties' shares of $[\vec{v}_1]_{k-1}, [\vec{v}_2]_{k-1}, [\vec{d}]_{k-1}, [e]_{n'-1}, [e]_{k-1}$ and corrupted parties' shares of $\llbracket \vec{u} \rrbracket_{t'+k-1}, \llbracket \mathbf{r} \rrbracket_{t'+k-1}$.

1. In Step 1, Sim invokes $\text{Sim}_{\text{extend-deg-reduction}}$.
2. In Step 3, Sim invokes $\text{Sim}_{\text{extend-deg-reduction}}$.
3. In Step 5, Sim invokes $\text{Sim}_{\text{compress}}$.

Simulator 27: Sim_{FL-ZeroSharing} for FL-ZEROSHARING

1. If P_v is honest, Sim honestly follows the protocol to simulate P_v .
2. In Step 2, for each honest P_i , Sim samples random elements as corrupted parties' shares of $[\mathbf{o}'_i]_{n'-1}$ and sends them to the adversary.
3. In Step 3, Sim randomly samples honest parties' shares of $\sum_{i \in \mathcal{H}} [\mathbf{o}'_i]_{n'-1}$ based on shares of corrupted parties, computes honest parties' shares of $[\mathbf{o}']_{n'-1}$, and broadcasts them.
4. In Step 4, Sim honestly follows the protocol.

5. In Step 5, for each active corrupted P_i , Sim has learnt honest parties' shares of $[\tilde{o}_i]_{n'-1}$; for each honest P_i , Sim has learnt corrupted parties' shares of $[\tilde{o}_i]_{n'-1}$. Sim samples honest parties' shares of $\{[\tilde{o}_i]_{n'-1}\}_{i \in \mathcal{H}}$ based on shares of corrupted parties and such that $\sum_{i \in [n]} [\tilde{o}_i]_{n'-1} = [\tilde{o}]_{n'-1}$. Sim sends honest parties' shares of $\{[\tilde{o}_i]_{n'-1}\}_{i \in [n]}$ to P_v .
6. In Step 6 and 7, Sim honestly follows the protocol.

Simulator 28: $\text{Sim}_{\text{input}}$ for **INPUT**

1. In Step 1, Sim invokes $\text{Sim}_{\text{rand}}(\Sigma_2)$.
2. In Step 2, Sim invokes $\text{Sim}_{\text{rand-linear-mask}}$.
3. In Step 3.b, if P_i is honest, Sim samples random elements as $\mathbf{x}^{(\ell)} + \mathbf{r}^{(\ell)}$ and sends active corrupted parties' shares of $[\mathbf{x}^{(\ell)} + \mathbf{r}^{(\ell)}]_{k-1}$ to the adversary; if P_i is corrupted, Sim receives honest parties' shares of $[\mathbf{x}^{(\ell)} + \mathbf{r}^{(\ell)}]_{k-1}$ from the adversary.
4. In Step 3.d, Sim emulates $\mathcal{F}_{\text{vrfy-linear-sharing}}$ and checks whether P_i distributed valid degree- $(k-1)$ packed Shamir sharings.
5. Sim simulates Steps 4, 5 and 6 of **INPUT** as Steps 5-9 $\text{Sim}_{\text{comp-seg}}$.

In the end, Sim learns corrupted parties' shares of the output sharings $\llbracket \mathbf{z}^{(1)} \rrbracket_{t'+k-1}, \dots, \llbracket \mathbf{z}^{(m')} \rrbracket_{t'+k-1}$.

Lemma 17. *When invoking **COMP-SEG** to compute a segment, starting with correctly authenticated input packed Shamir sharings, if the **segment verification** has been passed, except with negligible probability, then all parties have correctly authenticated their shares of the output packed Shamir sharings of the segment.*

Proof. The **segment verification** contains three parts: 1) all parties invoke $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_1)$ to check whether P_{king} distributes valid degree- $(k-1)$ packed Shamir sharings, 2) invoke **VRIFY-SHARING-TRANS** to verify whether P_{king} correctly perform sharing transformation, and 3) invoke **VRIFY-TRANSCRIPTS** to verify the correctness of the transcripts for degree reduction. The effectiveness of the first two verifications is clear. We then focus on the effectiveness of **VRIFY-TRANSCRIPTS**. Considering the generation of the final virtual transcript, we have that

- except with probability $M/|\mathbb{F}|$, if one of M input transcripts is incorrect, the resulting transcript of **DE-LINEARIZATION** is incorrect;
- except with probability $(2M-1)/|\mathbb{F}|$, if one of M input transcripts is incorrect, the resulting transcript of **COMPRESS**(M) is incorrect;
- except with probability $(2\kappa-1)/|\mathbb{F}|$, if the input transcript is incorrect, the resulting transcript of **DIMENSION-REDUCTION**(κ) is incorrect;
- except with probability $(2s+1)/|\mathbb{F}|$, if the input transcripts is incorrect, the resulting transcript of **RANDOMIZATION**(s) is incorrect.

Hence, by union bound, except with negligible probability, if one of the input transcripts is incorrect, the resulting transcript in Step 3 of **VRIFY-TRANSCRIPTS** is incorrect. Therefore, invoking **VRIFY-SINGLE-TRANSCRIPT** will detect the error and identify a dispute pair. This completes the effectiveness of **segment verification**. $\square$

Similarly, when invoking **INPUT** to handle an input segment, in case of success, except with negligible probability, all parties have correctly authenticated the output sharings of the segment.

Hybrid arguments. Now we use hybrid arguments and make use of the lemma above to show the validity of Sim_{main} .

Hyb₀. This hybrid corresponds to real-world execution. In the end, the adversary outputs its outputs and honest parties output their outputs.

Hyb₁. In this hybrid, we simulate the **preprocessing phase**. Instead of invoking $\mathcal{F}_{\text{DSS}}$ to prepare the preprocessing data, Sim emulates the interaction between the adversary and $\mathcal{F}_{\text{DSS}}$. Sim records shares of

corrupted parties and delays the sampling of shares of honest parties. The distributions in **Hyb**₁ and **Hyb**₀ are identical.

Hyb₂. In this hybrid, we simulate **INPUT**(P_j).

Hyb_{2,1}. In this hybrid, in Step 1, we simulate **RAND**(Σ_2).

Hyb_{2,1,1}. In this hybrid, in Step 1, for each honest P_i , Sim first randomly samples shares of corrupted parties and generates the whole sharing $[\mathbf{o}^{(i)}]_{n'-1}$ based on shares of corrupted parties. The distributions in **Hyb**_{2,1,1} and **Hyb**₁ are identical.

Hyb_{2,1,2}. In this hybrid, we change the computation of $\{[\mathbf{o}_{(i)}]_{n'-1}\}_{i \in \mathcal{H}}$. For each $i \in [n-t]$, Sim first computes corrupted parties' shares of $[\mathbf{o}_{(i)}]_{n'-1}$ and samples shares of honest parties based on shares of corrupted parties. Sim then computes $([\mathbf{o}^{(i)}]_{n'-1})_{i \in \mathcal{H}}$ as $\mathbf{M}_{\mathcal{H}}^{-1} \cdot (([\mathbf{o}_{(i)}]_{n'-1})_{i \in [n-t]}^\top - \mathbf{M}_{\mathcal{C}} \cdot ([\mathbf{o}^{(i)}]_{n'-1})_{i \in \mathcal{C}})$. Due to the randomness of $\{[\mathbf{o}_{(i)}]_{n'-1}\}_{i \in \mathcal{H}}$, and the one-to-one correspondence between $([\mathbf{o}^{(i)}]_{n'-1})_{i \in \mathcal{H}}$ and $([\mathbf{o}_{(i)}]_{n'-1})_{i \in [n-t]}^\top$ defined by $\mathbf{M}_{\mathcal{H}}$, the distributions in **Hyb**_{2,1,2} and **Hyb**_{2,1,1} are identical.

Hyb_{2,1,3}. In this hybrid, we don't compute shares of $\{[\mathbf{o}^{(i)}]_{n'-1}\}_{i \in \mathcal{H}}$ of honest parties. The distributions in **Hyb**_{2,1,3} and **Hyb**_{2,1,2} are identical.

Hyb_{2,2}. In this hybrid, in Step 2 of **INPUT**, we simulate **RAND-LINEAR-MASK**(N).

Hyb_{2,2,1}. In this hybrid, in Step 1.a, we change the computation of honest parties' shares of $[\mathbf{v}_{*\ell} + \mathbf{u}_{*\ell}]_{n'-1}$ for $\ell \in [k]$. Sim samples random elements as shares of honest parties, and computes the secrets $\mathbf{v}_{*\ell} + \mathbf{u}_{*\ell}$, and computes $\mathbf{u}_{*\ell} \leftarrow (\mathbf{v}_{*\ell} + \mathbf{u}_{*\ell}) - \mathbf{v}_{*\ell}$. Sim samples honest parties' shares of $[\mathbf{u}_{*\ell}]_{t'+k-1}$ based on the secrets and shares of corrupted parties, and computes $[\mathbf{o}_\ell]_{n'-1} \leftarrow [\mathbf{u}_{*\ell} + \mathbf{v}_{*\ell}]_{n'-1} - \sum_{j=1}^k [\mathbf{L}_{*,\ell j}]_{k-1} \cdot [\mathbf{r}_{*j}]_{t'+k-1} - [\mathbf{u}_{*\ell}]_{t'+k-1}$. Due to the randomness of $\{[\mathbf{u}_{*\ell}]_{t'+k-1}, [\mathbf{o}_\ell]_{n'-1}\}_{\ell=1}^k$, the distributions in **Hyb**_{2,2,1} and **Hyb**_{2,1,3} are identical.

Hyb_{2,2,2}. In this hybrid, in Step 2, instead of invoking $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_3)$, Sim emulates $\mathcal{F}_{\text{vrfy-linear-sharing}}$ and checks the consistency of the sharings received from P_{king} in Step 1.b. The distributions in **Hyb**_{2,2,2} and **Hyb**_{2,2,1} are identical.

Hyb_{2,2,3}. In this hybrid, Sim delays the sampling of shares of honest parties of $([\mathbf{r}]_{t'+k-1}, [\overline{\mathbf{L}(\mathbf{r})}]_{t'+k-1})$ for each linear transformation $\mathbf{L}$ until the end of **RAND-LINEAR-MASK**. The distributions in **Hyb**_{2,2,3} and **Hyb**_{2,2,2} are identical.

Hyb_{2,3}. In this hybrid, in Step 3 of **INPUT**, we simulate the step of **sharing input**.

Hyb_{2,3,1}. In this hybrid, in Step 3.b, if P_j is honest, we change the computation of $[\mathbf{x}^{(\ell)} + \mathbf{r}^{(\ell)}]_{k-1}$. Sim first randomly samples $\mathbf{x}^{(\ell)} + \mathbf{r}^{(\ell)}$, computes $\mathbf{r}^{(\ell)} \leftarrow (\mathbf{x}^{(\ell)} + \mathbf{r}^{(\ell)}) - \mathbf{x}^{(\ell)}$, and samples honest parties' shares of $[\mathbf{r}^{(\ell)}]_{t'+k-1}$ based on the secrets and shares of corrupted parties. Due to the randomness of $[\mathbf{r}^{(\ell)}]_{t'+k-1}$, the distributions in **Hyb**_{2,3,1} and **Hyb**_{2,2,3} are identical.

Hyb_{2,3,2}. In this hybrid, in Step 3.d, instead of invoking $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_1)$, Sim emulates $\mathcal{F}_{\text{vrfy-linear-sharing}}$ and checks the consistency of the sharings received from P_{king} in Step 3.b. The distributions in **Hyb**_{2,3,2} and **Hyb**_{2,3,1} are identical.

Hyb_{2,4}. In this hybrid, in Step 4 of **INPUT**, we simulate **SHARING-TRANS**. In Step 1, we change the computation of $[\mathbf{x} + \mathbf{r}]_{t'+k-1}$. Sim first computes corrupted parties' shares of $[\mathbf{x} + \mathbf{r}]_{t'+k-1}$, samples honest parties' shares based on shares of corrupted parties, and computes $[\mathbf{r}]_{t'+k-1} \leftarrow [\mathbf{x} + \mathbf{r}]_{t'+k-1} - [\mathbf{x}]_{t'+k-1}$. Due to the randomness of $[\mathbf{r}]_{t'+k-1}$, the distributions in **Hyb**_{2,4} and **Hyb**_{2,3,2} are identical.

Hyb_{2,5}. In this hybrid, in Step 5 of **INPUT**, we simulate the **segment verification**.

Hyb_{2,5,1}. In this hybrid, in Step 5.1, instead of invoking $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_1)$, Sim emulates $\mathcal{F}_{\text{vrfy-linear-sharing}}$ and checks the consistency of the sharings received from P_{king} in Step 4. The distributions in **Hyb**_{2,5,1} and **Hyb**_{2,4} are identical.

Hyb_{2,5,2}. In this hybrid, in Step 5.2, we simulate **VRFY-SHARING-TRANS**. In Step 2, instead of invoking $\mathcal{F}_{\text{vrfy-linear-sharing}}(\Sigma_2)$, Sim emulates $\mathcal{F}_{\text{vrfy-linear-sharing}}$ and checks the consistency of the sharings computed in Step 1. The distributions in **Hyb**_{2,5,2} and **Hyb**_{2,5,1} are identical.

Hyb_{2,5,3}. In this hybrid, in Step 5.3, we simulate **VRFY-TRANSCRIPTS**.

Hyb_{2,5,3,1}. In this hybrid, in Step 1, we simulate **DE-LINEARIZATION**. In Step 1, instead of invoking $\mathcal{F}_{\text{coin}}$, Sim emulates it by randomly sampling λ . The distributions in **Hyb**_{2,5,3,1} and **Hyb**_{2,5,2} are identical.

Hyb_{2,5,3,2}. In this hybrid, in Step 2 of **VRFY-TRANSCRIPTS**, we simulate **DIMENSION-REDUCTION**(κ).

Hyb_{2,5,3,2,1}. In this hybrid, in Step 2, we simulate **EXTEND-DEG-REDUCTION**. In Step 1, we change the computation of honest parties' shares of $[e]_{n'-1}$. Sim first samples random elements as honest parties' shares of $[e]_{n'-1}$, reconstructs the secrets $\mathbf{e}$, and computes $\mathbf{r} \leftarrow \mathbf{e} - \vec{v}_1 \odot \vec{v}_2 - \vec{d} \odot \vec{u}$. Sim then randomly samples $[\mathbf{r}]_{t'+k-1}$ based on the secrets and shares of corrupted parties and computes $[\mathbf{o}]_{n'-1} \leftarrow [e]_{n'-1} - [\vec{v}_1]_{k-1} \odot [\vec{v}_2]_{k-1} - [\vec{d}]_{k-1} \odot [\vec{u}]_{t'+k-1} - [\mathbf{r}]_{t'+k-1}$. Due to the randomness of $[\mathbf{r}]_{t'+k-1}$ and $[\mathbf{o}]_{n'-1}$, the distributions in **Hyb_{2,5,3,2,1}** and **Hyb_{2,5,3,1}** are identical.

Hyb_{2,5,3,2,2}. In this hybrid, in Step 4 of **DIMENSION-REDUCTION**, we simulate **COMPRESS**(κ).

Hyb_{2,5,3,2,2,1}. In this hybrid, in Step 3, we simulate **EXTEND-DEG-REDUCTION** as above. The distributions in **Hyb_{2,5,3,2,2,1}** and **Hyb_{2,5,3,2,1}** are identical.

Hyb_{2,5,3,2,2,2}. In this hybrid, in Step 5 of **COMPRESS**, instead of invoking $\mathcal{F}_{\text{coin}}$, Sim emulates it by randomly sampling λ . The distributions in **Hyb_{2,5,3,2,2,2}** and **Hyb_{2,5,3,2,2,1}** are identical.

Hyb_{2,5,3,3}. In this hybrid, in Step 3 of **VRFY-TRANSCRIPTS**, we simulate **RANDOMIZATION**(κ). In Step 1 and 3, we simulate **EXTEND-DEG-REDUCTION** as above. In Step 5, we simulate **COMPRESS**($s+1$) as above. The distributions in **Hyb_{2,5,3,3}** and **Hyb_{2,5,3,2,2,2}** are identical.

Hyb_{2,5,3,4}. In this hybrid, in Step 1, we change the computation of the resulting transcript of **RANDOMIZATION**, which is denoted by $([v_1]_{k-1}, [v_2]_{k-1}, [d]_{k-1}, [u]_{t'+k-1}, [r]_{t'+k-1}, [o]_{n'-1}, [e]_{n'-1}, [e]_{k-1})$. Sim first computes corrupted parties' shares of $[u]_{t'+k-1}$ and randomly samples the whole sharing based on shares of corrupted parties. Sim then computes v_1, v_2, d , and e from $[v_1]_{k-1}, [v_2]_{k-1}, [d]_{k-1}$, and $[e]_{n'-1}$. Then Sim computes $\mathbf{r} \leftarrow \mathbf{e} - v_1 * v_2 - d * u$, and randomly samples $[\mathbf{r}]_{t'+k-1}$ from the secrets and shares of corrupted parties. Sim computes $[o]_{n'-1} \leftarrow [e]_{n'-1} - [v_1]_{k-1} \cdot [v_2]_{k-1} - [d]_{k-1} \cdot [u]_{t'+k-1} - [\mathbf{r}]_{t'+k-1}$. Then Sim computes $[u^{(0)}]_{t'+k-1}$ by interpolating $\{[u^{(i)}]_{t'+k-1}\}_{i=1}^s$ and $[u]_{t'+k-1}$, and computes $[\mathbf{r}^{(0)}]_{t'+k-1}$ by interpolating $\{[\mathbf{r}^{(i)}]_{t'+k-1}\}_{i=1}^{2s}$ and $[\mathbf{r}]_{t'+k-1}$. Due to the randomness of $[u^{(0)}]_{t'+k-1}$, the distributions in **Hyb_{2,5,3,4}** and **Hyb_{2,5,3,3}** are identical.

Hyb_{2,5,3,5}. In this hybrid, in Step 4 of **VRFY-TRANSCRIPTS**, we simulate **VRFY-SINGLE-TRANSCRIPT**.

Hyb_{2,5,3,5,1}. In this hybrid, in Step 3.1, instead of invoking $\mathcal{F}_{\text{DSS}}$ with **Check**, since Sim has learnt shares of corrupted parties, Sim emulates $\mathcal{F}_{\text{DSS}}$ to check the correctness of active corrupted parties' shares. The distributions in **Hyb_{2,5,3,5,1}** and **Hyb_{2,5,3,4}** are identical.

Hyb_{2,5,3,5,2}. In this hybrid, in Step 3.2 of **VRFY-SINGLE-TRANSCRIPT**, we simulate **FL-ZERO SHARING**.

Hyb_{2,5,3,5,2,1}. In this hybrid, in Step 2, for each honest P_i , Sim first samples random elements as corrupted parties' shares of $[o'_i]_{t'+k-1}$ and then generates the whole sharing. The distributions in **Hyb_{2,5,3,5,2,1}** and **Hyb_{2,5,3,5,1}** are identical.

Hyb_{2,5,3,5,2,2}. In this hybrid, in Step 3, we change the computation of $[o']_{n'-1}$. Sim first computes corrupted parties' shares of $\sum_{i \in \mathcal{H}} [o'_i]_{n'-1}$, randomly samples shares of honest parties based on shares of corrupted parties, and then computes $[o']_{n'-1} \leftarrow \sum_{i \in \mathcal{H}} [o'_i]_{n'-1} + \sum_{i \in \mathcal{C}} [o'_i]_{n'-1}$. Sim randomly samples $\{[o'_i]_{n'-1}\}_{i \in \mathcal{H}, i \neq i'}$ and then computes $[o'_{i'}]_{n'-1} \leftarrow \sum_{i \in \mathcal{H}} [o'_i]_{n'-1} - \sum_{i \in \mathcal{H}, i \neq i'} [o'_i]_{n'-1}$, where i' denotes the smallest identity of an honest party. Due to the randomness of $[o'_{i'}]_{n'-1}$, the distributions in **Hyb_{2,5,3,5,2,2}** and **Hyb_{2,5,3,5,2,1}** are identical.

Hyb_{2,5,3,5,2,3}. In this hybrid, in Step 5, for each honest P_i , we change the computation of $[\tilde{o}_i]_{n'-1}$. Suppose we decompose $[o]_{n'-1}$ as $[o]_{n'-1} = \sum_{i \in [n]} [o_i]_{n'-1}$, where $[o_i]_{n'-1}$ is distributed by P_i . For $i \in \mathcal{H} \setminus \{i'\}$, Sim first computes corrupted parties' shares of $[\tilde{o}_i]_{n'-1}$, randomly samples shares of honest parties based on shares of corrupted parties, and computes $[o'_i]_{n'-1} \leftarrow [\tilde{o}_i]_{n'-1} - [o_i]_{n'-1}$ if $[\tilde{o}]_{n'-1} = [o]_{n'-1} + [o']_{n'-1}$. As for $i = i'$, Sim computes $[\tilde{o}_{i'}]_{n'-1} \leftarrow [\tilde{o}]_{n'-1} - \sum_{i \in [n]} [\tilde{o}_i]_{n'-1}$. Due to the randomness of $\{[o'_i]_{n'-1}\}_{i \in \mathcal{H}}$, the distributions in **Hyb_{2,5,3,5,2,3}** and **Hyb_{2,5,3,5,2,2}** are identical.

Hyb_{2,5,3,5,2,4}. In this hybrid, if $[\tilde{o}]_{n'-1} = [o]_{n'-1} + [o']_{n'-1}$, we don't compute honest parties' shares of $\{[o'_i]_{n'-1}\}_{i \in \mathcal{H}}$. The distributions in **Hyb_{2,5,3,5,2,4}** and **Hyb_{2,5,3,5,2,3}** are identical.

Hyb_{2,6}. In this hybrid, in Step 6 of **INPUT**, instead of invoking $\mathcal{F}_{\text{DSS}}$ with **Auth**, Sim emulates $\mathcal{F}_{\text{DSS}}$ and records shares of corrupted parties in case of success. The distributions in **Hyb_{2,6}** and **Hyb_{2,5,3,5,2,4}** are identical.

Hyb₃. In this hybrid, in Step 2.1 of **MAIN**, we simulate **COMP-SEG**.

Hyb_{3,1}. In this hybrid, in Step 2.1, we simulate **RAND**(Σ_2) as in **Hyb_{2,1,3}**. Sim learns corrupted parties' shares of $\{[o_{(i)}]_{n'-1}\}_{i \in [n-t]}$. In Step 2.2, we simulate **RAND-LINEAR-MASK** as in **Hyb_{2,3,2}**. Sim learns

corrupted parties' shares of $(\llbracket \mathbf{r} \rrbracket_{t'+k-1}, \llbracket \mathbf{L}(\mathbf{r}) \rrbracket_{t'+k-1})$ for each linear transformation $\mathbf{L}$. The distributions in $\mathbf{Hyb}_{3,1}$ and $\mathbf{Hyb}_{2,6}$ are identical.

Hyb_{3,2}. In this hybrid, in Step 3, we simulate **segment evaluation**.

Hyb_{3,2,1}. In this hybrid, in Step 3.1, we simulate **SELECT**. In Step 1, we change the computation of $[e]_{n'-1}$. Sim first samples random elements as honest parties' shares of $[e]_{n'-1}$, reconstructs the secrets $\mathbf{e}$, and computes $\mathbf{r} \leftarrow \mathbf{e} - \sum_{\ell=1}^k \mathbf{e}_\ell * \mathbf{x}_\ell$. Sim then samples $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$ based on shares of corrupted parties and the secrets, and computes $[\mathbf{o}]_{n'-1} \leftarrow [\mathbf{e}]_{n'-1} - \sum_{\ell=1}^k [\mathbf{e}_\ell]_{k-1} \cdot [\mathbf{x}_\ell]_{t'+k-1} - \llbracket \mathbf{r} \rrbracket_{t'+k-1}$. Due to the randomness of $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$ and $[\mathbf{o}]_{n'-1}$, the distributions in $\mathbf{Hyb}_{3,2,1}$ and $\mathbf{Hyb}_{3,1}$ are identical.

Hyb_{3,2,2}. In this hybrid, in Step 3.1, we simulate **SHARING-TRANS** as in $\mathbf{Hyb}_{2,4}$. The distributions in $\mathbf{Hyb}_{3,2,2}$ and $\mathbf{Hyb}_{3,2,1}$ are identical.

Hyb_{3,2,3}. In this hybrid, in Step 3.2, we simulate **MULT**.

Hyb_{3,2,3,1}. In this hybrid, in Step 1, we change the computation of $[\mathbf{v}_1]_{t'+k-1}$ and $[\mathbf{v}_2]_{t'+k-1}$. Sim first computes corrupted parties' shares of $[\mathbf{v}_1]_{t'+k-1}$ and $[\mathbf{v}_2]_{t'+k-1}$, and randomly samples shares of honest parties based on shares of corrupted parties. Sim then computes $\llbracket \mathbf{a} \rrbracket_{t'+k-1} \leftarrow [\mathbf{v}_1]_{t'+k-1} - [\mathbf{x}]_{t'+k-1}$ and $\llbracket \mathbf{b} \rrbracket_{t'+k-1} \leftarrow [\mathbf{v}_2]_{t'+k-1} - [\mathbf{y}]_{t'+k-1}$, reconstructs the secrets $\mathbf{a}$ and $\mathbf{b}$, computes $\mathbf{c} \leftarrow \mathbf{a} * \mathbf{b}$, and randomly samples $\llbracket \mathbf{c} \rrbracket_{t'+k-1}$ based on shares of corrupted parties and the secrets. Due to the randomness of $\llbracket \mathbf{a} \rrbracket_{t'+k-1}$ and $\llbracket \mathbf{b} \rrbracket_{t'+k-1}$, the distributions in $\mathbf{Hyb}_{3,2,3,1}$ and $\mathbf{Hyb}_{3,2,2}$ are identical.

Hyb_{3,2,3,2}. In this hybrid, in Step 3, we change the computation of $[e]_{n'-1}$. Sim first samples random elements as honest parties' shares of $[e]_{n'-1}$, reconstructs the secrets $\mathbf{e}$, and computes $\mathbf{r} \leftarrow \mathbf{e} - \mathbf{v}_1 * \mathbf{v}_2 + \mathbf{a} * \mathbf{v}_2 + \mathbf{b} * \mathbf{v}_1 - \mathbf{c}$. Sim then randomly samples $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$ based on shares of corrupted parties and the secrets, and computes $[\mathbf{o}]_{n'-1} \leftarrow [\mathbf{e}]_{n'-1} - [\mathbf{v}_1]_{k-1} \cdot [\mathbf{v}_2]_{k-1} + \llbracket \mathbf{a} \rrbracket_{t'+k-1} \cdot [\mathbf{v}_2]_{k-1} + \llbracket \mathbf{b} \rrbracket_{t'+k-1} \cdot [\mathbf{v}_1]_{k-1} - \llbracket \mathbf{r} \rrbracket_{t'+k-1}$. Due to the randomness of $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$ and $[\mathbf{o}]_{n'-1}$, the distributions in $\mathbf{Hyb}_{3,2,3,2}$ and $\mathbf{Hyb}_{3,2,3,1}$ are identical.

Hyb_{3,2,4}. In this hybrid, in Step 3.3 of **COMP-SEG**, we simulate **SHARING-TRANS** as in $\mathbf{Hyb}_{2,4}$. The distributions in $\mathbf{Hyb}_{3,2,4}$ and $\mathbf{Hyb}_{3,2,3,2}$ are identical.

Hyb_{3,3}. In this hybrid, in Step 4, we simulate **segment verification** as in $\mathbf{Hyb}_{2,5,3,4,2,4}$. The distributions in $\mathbf{Hyb}_{3,3}$ and $\mathbf{Hyb}_{3,2,4}$ are identical.

Hyb_{3,4}. In this hybrid, in Step 5, instead of invoking $\mathcal{F}_{\text{DSS}}$ with **Auth**, Sim simulates $\mathcal{F}_{\text{DSS}}$ and records shares of corrupted parties in case of success. The distributions in $\mathbf{Hyb}_{3,4}$ and $\mathbf{Hyb}_{3,3}$ are identical.

Hyb₄. In this hybrid, in Step 3 of **MAIN**, instead of invoking $\mathcal{F}_{\text{DSS}}$ with **Recon**, Sim emulates $\mathcal{F}_{\text{DSS}}$ by itself. In case of success, since Sim has learnt circuit inputs of corrupted parties, Sim provides them to the ideal functionality $\mathcal{F}_{\text{main}}$. At the meantime, honest parties provide their circuit inputs. Sim then receives corrupted parties' circuit outputs and uses them to emulate $\mathcal{F}_{\text{DSS}}$. By Lemma 17, the distribution in $\mathbf{Hyb}_4$ is statistically closed to that in $\mathbf{Hyb}_{3,4}$.

Hyb₅. In this hybrid, if parties restart the computation, instead of invoking $\mathcal{F}_{\text{main-client-server}}^*$, Sim emulates it as follows. Sim first passes the circuit inputs of corrupted parties received from the adversary to $\mathcal{F}_{\text{main}}$. At the meanwhile, honest parties also provide their circuit inputs to $\mathcal{F}_{\text{main}}$. Sim then receives corrupted parties' circuits outputs and passes them to the adversary. The distributions in $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ are identical.

Hyb₆. In this hybrid, we don't use inputs of honest parties. During the simulation, except the virtual transcript generated by **RANDOMIZATION** in **VRIFY-TRANSCRIPTS**, we don't compute honest parties' shares of $\llbracket \mathbf{u} \rrbracket_{t'+k-1}$, $\llbracket \mathbf{r} \rrbracket_{t'+k-1}$, and $[\mathbf{o}]_{n'-1}$. The distributions in $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ are identical.

Noticing that $\mathbf{Hyb}_6$ already corresponds to the output distribution in the ideal world, we conclude that **MAIN** securely computes $\mathcal{F}_{\text{main}}$ in $\mathcal{F}_{\text{DSS}}, \mathcal{F}_{\text{coin}}, \mathcal{F}_{\text{vrfy-linear-sharing}}, \mathcal{F}_{\text{main-client-server}}^*$ -hybrid model. $\square$