Dumbo-MPC: Efficient Fully Asynchronous MPC with Optimal Resilience

Yuan Su
Xi'an Jiaotong University
suyuan@stu.xjtu.edu.cn

Yuan Lu
Institute of Software CAS
luyuan@iscas.ac.cn

Jiliang Li
Xi'an Jiaotong University
jiliang.li@xjtu.edu.cn

Yuyi Wang
CRRC Zhuzhou Institute
yuyiwang920@gmail.com

Chengyi Dong
Xi'an Jiaotong University
2196113533@xjtu.stu.edu.cn

Qiang Tang
The University of Sydney
qiang.tang@sydney.edu.au

Abstract

Fully asynchronous multi-party computation (AMPC) has superior robustness in realizing privacy and guaranteed output delivery (G.O.D.) against asynchronous adversaries that can arbitrarily delay communications. However, none of these protocols are truly practical, as they either have sub-optimal resilience, incur cumbersome communication cost, or suffer from an online phase with extra cryptographic overhead. The only attempting implementation—HoneyBadgerMPC (hbMPC)—merely ensures G.O.D. in some implausible optimistic cases due to a non-robust offline pre-processing phase.

We propose Dumbo-MPC a concretely efficient AMPC-as-a-service design with all phases G.O.D. and optimal resilience against $t < n/3$ malicious parties (where n is the total number of parties). Same to hbMPC, Dumbo-MPC has a robust (almost) information-theoretic online phase that can efficiently perform online computations, given pre-processed multiplication triples. While for achieving all phases G.O.D., we design a novel dual-mode offline protocol that can robustly pre-process multiplication triples in asynchrony. The offline phase features $O(n)$ per-triple communication in the optimistic case, followed by a fully asynchronous fallback to a pessimistic path to securely restore G.O.D. in the bad case. To efficiently implement the pessimistic path, we devise a concretely efficient zk-proof for product relationship of secret shares over compact KZG polynomial commitments, which enables us to reduce the degree of two secret shares' product from $2t$ to t and could be of independent interest.

We also implement and extensively evaluate Dumbo-MPC (particularly its offline phase) in varying network settings with up to 31 AWS servers. To our knowledge, we provide the first implementation of AMPC with all-phase G.O.D. A recent asynchronous triple generation protocol from Groth and Shoup (GS23) is also implemented and experimentally compared. When $n = 31$, Dumbo-MPC generates 94 triples/sec (almost twice of GS23) in the pessimistic case and 349 triples/sec (6X of GS23) in the good case, such that 31 parties require only 2-8 min to prepare a private Vickrey auction of 100 bidders or 10-36 min for a mixing network of 2^{10} inputs.

1 Introduction

The paradigm of multi-party computation (MPC) as a service (MPCaaS) has recently gained significant interests as a promising approach to privacy-preserving distributed systems. Particularly, it is often seen as an enticing solution to overcome the challenge of privacy breaches in blockchains, thereby enabling private smart contracts [17, 91], anonymous broadcasts for transaction diffusion [5, 70], and numerous other tailored decentralized applications [73, 74].

Need for robust MPC in full asynchrony. In many aforementioned scenarios, ensuring G.O.D. is crucial for maintaining the availability of services, particularly in mission-critical applications where timely response is essential. Consider the case of MPCaaS-enabled private smart contracts [17, 91]: clients rely on MPCaaS to evaluate their transactions privately and deliver execution results based on contract clauses. If MPCaaS fails to ensure G.O.D., an adversary could forever block transaction execution, posing a severe denial-of-service threat that could completely censor the private smart contracts.

Despite the urgent demand of G.O.D., most practical MPC implementations [8, 38, 40, 67, 88] fail to provide such necessity [70]. For many additive secret sharing based MPC that tolerate $n-1$ malicious parties [40, 42, 88], it is inherently impossible to realize G.O.D. Even for many MPC protocols [8, 39, 41, 54, 56, 62] that are expected to be robust, their output delivery is conditioned on stringent network synchrony, and unsurprisingly, when they are deployed in an asynchronous network where messages can be arbitrarily delayed (e.g., MPCaaS servers experience unexpected communication interruption due to Internet glitches), their G.O.D. could be violated. Moreover, many MPC protocols [12, 41, 54, 56, 62] even suffer from privacy leakage in a fully asynchronous network, because they heavily rely on the strong assumption of network synchrony to "eject" suspiciously malicious nodes that are temporally unresponsive. As such, they might incorrectly eject honest parties in asynchrony, thereby eventually allowing malicious nodes learn all private inputs.

Given that, it becomes essential to consider more efficient

design of asynchronous MPC (AMPC), thus ensuring both privacy and G.O.D. in the unstable or even adversarial Internet environment to accommodate mission-critical applications.

1.1 Practical Obstacles of AMPC

Nevertheless, it is challenging to design efficient AMPC while preserving optimal $t < n/3$ resilience,[1] as one cannot distinguish a malicious party that sends nothing and an honest party whose messages are delayed due to network asynchrony. So the protocol has to proceed once receiving $n-t$ distinct parties' messages (which might only solicit $n-2t$ honest parties' messages and therefore omit the rest t honest parties).

Practicality issues of earlier theoretic studies. Despite that AMPC has been studied for more than 30 years, existing results are mostly theoretical, and essentially none of them was ever implemented for various efficiency issues. AMPC was initially studied in the unconditionally-secure setting [14, 15, 80, 83], but these early results have tremendous communication cost (at least n^4 overhead per gate). Some recent results with unconditional security reduce the asymptotic overhead to $O(n^2)$ or $O(n)$ per gate, but most of them [32, 34, 78] only tolerate $t < n/4$ malicious parties. The only unconditionally secure design that realizes linear per-gate communication with optimal $t < n/3$ resilience is a very recent study [55], but has a prohibitive $O(n^{14})$ circuit-independent overhead as well as inferior concrete efficiency (which requires thousands of secrets to be verifiably shared per triple, even if instantiating it from best computationally-secure components).[2]

In the cryptographic setting, the study of AMPC was initialized by Hirt et al. [63, 64], followed by a few theoretic improvements [33, 36] using somewhat/fully homomorphic encryption. Choudhury and Patra [33] realized linear per-gate communication overhead but have costly (n,t) threshold decryption of somewhat homomorphic encryption in its online phase (for multiplication gates). Coretti et al. [37] proposed an asynchronous version of 'BMR' [10] with constant rounds, but it has a costly process using another general-purpose AMPC to pre-compute distributed garbled circuits. Cohen [36] adopted threshold fully homomorphic encryption (tFHE) to get another constant-round AMPC, but it has undesired online computational cost due to expensive FHE evaluations.

hbMPC: trading robustness for efficiency. Until recently, an interesting attempt of HoneyBadgerMPC (hbMPC) [70] gives the first potentially practical (nearly-)asynchronous MPCaaS protocol. The work of hbMPC focuses on optimizing a robust online phase based on Shamir secret sharing [7, 9, 13] with $n/3$ resilience in the pre-processing model, with the price of an adapted *non-robust* offline phase [12, 41] to pre-process the needed multiplication triples. However, the overall robustness of hbMPC is impaired: when its non-robust offline phase is under network attacks, the pre-processing could be stalled; and subsequently, its online phase might also grind to a halt, as a result of forever waiting for the replenishment of multiplication triples. That said, the robustness of hbMPC still relies on a strong assumption that all n parties are honest to cooperatively make the offline phase progress. Unfortunately, such strong assumption could be elusive, especially in adversarial deployment environments like the open Internet.

GS23: a robust offline but with larger per-gate overhead. Very recently, Groth and Shoup (GS23) [57] make one step towards robust asynchronous triple generation by introducing an optimized asynchronous version of 'BGW' [13, 53] in the cryptographic setting. From a high level, it lets each party P_i invoke t-degree asynchronous complete (verifiable) secret sharing (ACSS) to share two secrets a_i and b_i, and adds shares from distinct parties to obtain two random shares $[\![a]\!]$ and $[\![b]\!]$. Then, every party computes $[\![ab]\!]_{2t} = [\![a]\!][\![b]\!]$ as $2t$-degree share of ab, and invokes t-degree ACSS again to re-share $[\![ab]\!]_{2t}$. When instantiating ACSS from (generalized) Pedersen's polynomial commitment, GS23 can let the final re-sharing of $[\![ab]\!]_{2t}$ also carry some zk-proof for the product relationship over Pedersen commitments, thus ensuring everyone to recover a correct t-degree share $[\![ab]\!]$ to complete triple generation. However, in GS23, each party verifiably shares 3 secrets per triple (two for a_i and b_i, and one for $[\![ab]\!]_{2t}$), which not only incurs concretely inferior performance, but also causes asymptotically large communication and computational cost, e.g., the per-triple communication is $O(n^2)$ in the good case and even $O(n^3)$ in the bad case.[3]

Can we harvest both robustness and efficiency in AMPC? On the one hand, the almost information-theoretic (I.T.) online phase of hbMPC is promisingly performant, capable of evaluating 8,000+ multiplication gates per second according to our evaluations in the pre-processing model (cf. Appendix I for test results), but it suffers from a non-robust offline phase, resulting in a major vulnerability in a fully asynchronous network. On the other hand, robust asynchronous offline protocols like GS23 offer superior robustness but are substantially less efficient, especially when hbMPC might optimistically execute in the good case, GS23 attains a throughput that is only 1/6 of hbMPC, as our evaluations reflect.

Facing this robustness-efficiency trade-off in state-of-the-art AMPC protocols, [4] we are asking the following question:

Can we push fully asynchronous MPC protocols with guaranteed output delivery closer to practice (i.e., as robust as GS23 while as efficient as hbMPC)?

[1] Note that the optimal resilience of asynchronous Byzantine agreement is $n/3$ [24], implying the same upper bound of resilience of robust AMPCs.

[2] Looking ahead, we only require n secrets to be shared per triple, which is concretely more efficient than [55] (and any its computationally-secure variant) for typical n, as they need to share thousands of secrets per triple.

[3] Though the state-of-the-art batched ACSS [84] attains $O(n)$ amortized communication cost per secret, its instantiation from Pedersen commitment like GS23 still incurs $O(n^2)$ amortized communication in the bad case.

[4] Besides closely related studies discussed in Introduction, we also review else relevant studies and thoroughly discuss their limitations in Appendix A.

Table 1: **Comparison of typical AMPC protocols**

Protocols	Thld. ($t <$)	I.T. online	asyn. G.O.D.	comm. / gate	
				Good	Bad
CHP13 [32]	$n/4$	✓	✓	$O(n)$	$O(n)$
CP17 [34]	$n/4$	✓	✓	$O(n)$	$O(n)$
DXKR23 [43]‡	$n/3$	✓	✗	$O(n^3)$	–
	$n/4$	✓	✓	$O(n^3)$	$O(n^3)$
hbMPC [70]	$n/3$	✓	✗	$O(n)$	–
HNP08 [64]	$n/3$	✗	✓	$O(n^2)$	$O(n^2)$
CP15 [33] †	$n/3$	✗	✓	$O(n)$	$O(n)$
CHL21 [30]	$n/3$	✗	✓	$O(n^2)$	$O(n^2)$
GS23 [57]	$n/3$	✓	✓	$O(n^2)$	$O(n^3)$
Dumbo-MPC	$n/3$	✓	✓	$O(n)$	$O(n^2)$

† CP15 performs threshold decryption of somewhat homomorphic encryption for each multiplication in online, which might cause serious efficiency issue.
‡ DXKR23 gives a random double-sharing protocol to generate t-degree and $2t$-degree shares of the same randomness. However, while using double shares for evaluating multiplication gates during the online phase in asynchrony with $n/3$ corruptions, the $2t$-degree secret shares might fail in reconstructing (as decoding might fail). Only if a lower non-optimal resilience (e.g. $n/4$) is allowed, the robustness of online evaluation can be restored.

1.2 Our Contribution

We answer the question affirmatively by designing a set of robust and concretely efficient AMPC protocols in the classic online-offline paradigm, called Dumbo-MPC, with amortized linear (resp. quadratic) per-gate communication in the good case (resp. the bad case). To achieve all-phase G.O.D. with optimal resilience, Dumbo-MPC introduces a throughput-optimized fully asynchronous offline protocol to robustly and efficiently pre-process multiplication triples in a batched manner. Dumbo-MPC also inherits hbMPC's almost I.T. online phase, which is robust and already promisingly performant in the pre-processing model, involving only Shamir secret sharing for crucial online efficiency (except for an asynchronous Byzantine agreement component that we also dedicatedly optimize). In sum, our contributions are:

- **New "hidden evaluation" interface of compact KZG polynomial commitment for conveniently proving product relation of committed secret shares.** To apply compact KZG polynomial commitment [66] for reducing the worst-case complexity of GS23, we first devise a new zero-knowledge proof scheme for proving the product relationship of secret shares committed to a triple of KZG commitments. For the purpose, we introduce a new "hidden evaluation" interface and properties to KZG scheme, allowing a party to convert the binding of secret shares from KZG polynomial commitment into Pedersen commitment. As such, it reduces the problem of proving product relationship over KZG commitments to the very standard problem of proving that over Pedersen commitments. When applying our technique to instantiate Dumbo-MPC using more compact KZG polynomial commitment, we reduce the worst-case communication complexity from GS23's $O(n^3)$ per-gate to $O(n^2)$. Moreover, our approach of proving product relation of Shamir secret shares over KZG commitments could be of independent interests, e.g., robust publicly auditable/accountable MPCs [65, 81].
- **Robust fully asynchronous offline protocol that can batchedly pre-process multiplication triples more efficiently.** In addition to our asymptotic improvement of the worst-case complexity of GS23, we further enhance concrete efficiency by reducing the number of secrets to be shared by an additional factor of 3, through a sub-protocol that extracts random shares using batching for efficiency. Our approach not only patches the concurrent composability of the state-of-the-art KZG-based batch ACSS [84], but also leverages KZG's homomorphism to redesign the GS23 protocol to extract $t+1$ random shares from $n-t$ shared secrets via hyper-invertible matrix. More importantly, this still maintains the crucial convenience of proving product relation of secret shares for valid triples, as we can compute the linear combinations of KZG polynomial commitments to bind all extracted random shares. Thereby, our triple generation only verifiably shares n secrets per triple, as opposed to that GS23 requires $3n$.[5]
- **Optimized offline fast path to harvest performance with secure fallback to always preserve G.O.D.** To harvest efficiency from optimistic conditions when no party misbehaves, we add a fast path optimized from hbMPC [12, 41, 70] to generate triples with amortized $O(n)$ communication overhead in the good case. To preserve G.O.D., we design a fallback mechanism enabling honest parties switch to our robust offline protocol when the fast path fails. The fallback is non-trivial, as a naive attempt (directly running the robust protocol after the fast path fails) might cause incorrect online multiplications due to network asynchrony (e.g., for a multiplication gate, someones use a triple from the fast path, but the others use another triple from the robust protocol). We efficiently resolve the threat by using a conceptually minimal asynchronous Byzantine agreement for a single bit. Moreover, our fast path saves 3 communication rounds compared to hbMPC, as a bonus of our fallback mechanism that eliminates the need of broadcasts to cross-check the consistency of generated triples.
- **The first implementation of AMPC with all-phase G.O.D. and extensive evaluations.** Combining with dedicated optimization of asynchronous consensus component for good-case latency, we then implemented a prototype of Dumbo-MPC. For fair comparison, we also

[5] Our triple generation not only surpasses GS23 in efficiency but is also concretely more efficient than many theoretical designs [35, 55] with similar or even better asymptotic complexities, cf. Appendix A for details.

implemented GS23 as a by-product. To our knowledge, we provide the *first* implementation of AMPC with all-phase G.O.D., which is also open-sourced.[6] Extensive evaluations were conducted to demonstrate the performance of Dumbo-MPC under varying network conditions, such as LAN and WAN settings, involving up to 31 AWS EC2 nodes.[7] As a highlight of our experimental results, Table 2 summarizes the pre-processing latency of GS23, hbMPC and Dumbo-MPC in a LAN setting for a task of private Vickrey auction with 100 bidders. Specifically, when $n = 31$, GS23 requires more than 13 minutes to pre-process all 44,571 multiplication triples needed by the Vickrey auction, and hbMPC might even have an infinite latency. In contrast, our pessimistic offline path takes less than 8 minutes, reducing the pre-processing latency by up to 41% compared to GS23.

Table 2: **Pre-processing latency (sec.) for private Vickery auction with 100 bidders (the bad case, in a LAN setting)**

Protocols	Scale $n =$		
	10	22	31
hbMPC [70]	∞	∞	∞
GS23 [57]	193	464	810
Dumbo-MPC	168 (↓ **13%**)	368 (↓ **21%**)	474 (↓ **41%**)

2 Problem Definition & Technique Overview

For the convenience of readers, we enumerate some notations widely used throughout the paper in Table 3.

Table 3: **Notations**

Notation	Description
n	the total number of parties
t	the maximum number of corrupted parties
N	the batch size of secret sharing
B	the batch size of triple generation
(pk_i, sk_i)	the public-secret key pair of P_i
$[\![r]\!]$, $[\![r]\!]_{2t}$	$(n, t+1)$ and $(n, 2t+1)$ Shamir's secret shares of r
$[\![r]\!]^i$	a Shamir secret share of r held by party P_i
$[n]$	short for $[1, \cdots, n]$
M_{ij}	the j-th element of i-th row of matrix M
κ	the bit length of security parameter
$\varepsilon(\cdot)$	negligible function

2.1 Problem: Asynchronous MPCaaS

Security model. We adopt the standard reliable asynchronous authenticated network with setup assumptions. We also consider the MPC-as-a-Service (MPCaaS) setting with malicious clients. Specifically, our model can be formalized as:

[6]The open-source code-base will soon be announced.

[7]To assess the feasibility of Dumbo-MPC, we primarily focus on evaluating its offline (which is same to most similar literature [40, 68, 69]) as the offline is the heaviest component bringing major performance bottleneck.

Public identities and trusted setup. There are n designated parties $\{P_1, \cdots, P_n\}$ (i.e., servers) that participate in the MPC protocol. There also exists a public key infrastructure such that each P_i gets and only gets its own secret key sk_i and additionally obtains the public keys $\{pk_i\}_{i\in[n]}$ of all parties. In addition, we assume that the common reference string $(g, g^{\alpha}, \cdots, g^{\alpha^t}, h, h^{\alpha}, \cdots, h^{\alpha^t}) \in \mathbb{G}^{2t+2}$ of KZG polynomial commitment [66] is honestly generated and published, which can be done through asynchronous distributed protocols [45].

Fully-connected asynchronous p2p network. We consider a standard asynchronous communication network that consists of secure point-to-point (p2p) channels between each pair of parties. Messages sent between honest parties can be arbitrarily delayed by the adversary, but they remain confidential and must eventually deliver without being tampered.

Adversary corrupting $t < n/3$ servers. We consider malicious, static, probabilistic polynomial-time (P.P.T.) bounded adversaries that can fully control up to $t < n/3$ corrupted parties. Here "static" means that the adversary chooses parties to corrupt before the protocol begins. Noticeably, $t < n/3$ is the optimal resilience of AMPC protocols with G.O.D. [24].

Probably malicious clients. We focus on the enticing MPCaaS model, where k clients submit their private inputs for secure computation. An inherent limitation of asynchrony is that up to c honest clients' inputs may be excluded from computation (where c is the number of malicious clients), because waiting for all k inputs might incur infinite time as corrupted clients might not provide any input. W.l.o.g., we assume $n = k$ and $t = c$ (i.e., each server plays another role of client) for presentation simplicity, unless otherwise specified.

Design goals. We aim to achieve concretely efficient AMPC, featuring robust offline and online phases (as illustrated in Figure 1), tailored for the MPCaaS setting with optimal resilience and all-phase G.O.D. More specifically,

Offline phase. Syntactically, n parties (servers) take system parameters as input, and each party P_i continuously outputs a linearized sequence of random shares $[\![r_1]\!]^i, [\![r_2]\!]^i \cdots$, and another sequence of multiplication triples $([\![a_1]\!]^i, [\![b_1]\!]^i, [\![a_1 b_1]\!]^i)$, $([\![a_2]\!]^i, [\![b_2]\!]^i, [\![a_2 b_2]\!]^i) \cdots$. The phase shall satisfy the following properties with overwhelming probability:

- *Validity*: for any j-th position in the output sequence of random shares, all honest parties must (eventually) hold consistent t-degree secret shares of some r_j; similarly, for any j-th position in the triples' sequence, all honest parties must (eventually) hold consistent t-degree secret shares of some a_j, b_j, $c_j = a_j b_j$.
- *Secrecy*: it is infeasible for any P.P.T. adversary to predict r_j (resp. a_j, b_j, $a_j b_j$) better than guessing, until the first honest node P_i spreads out its corresponding secret share.
- *Pre-processing liveness (Offline robustness)* : all honest parties' output sequences are ever-growing.

Online phase. Given the above robust offline phase that can continuously generate random shares and multiplication

triples, we then aim at realizing an online phase with MPCaaS interfaces to (probably malicious) clients. As such, for any agreed to evaluate function $f(\cdot)$, the phase can confidentially evaluate f and shall satisfy the next properties with overwhelming probability:

- *Privacy*. For any P.P.T. adversary, it learns nothing in addition to the evaluation result of f.
- *Guaranteed output delivery (online robustness)*. All honest parties (servers) eventually output the evaluation of $f(\cdot)$ on at least $n-t$ distinct clients' inputs.

Performance metrics. We are particularly interested in more efficient AMPC protocols. For the purpose, we primarily consider the key efficiency metric of *(amortized) communication complexity*, reflecting the expected number of bits sent by honest parties to generate each random share or multiplication triple. Moreover, we might estimate the concrete performance of AMPC protocols according to the number of secrets to be (verifiably) shared for each generated multiplication triple. Sometimes, the round of communication is also considered as another indicator to estimate protocols' performance.

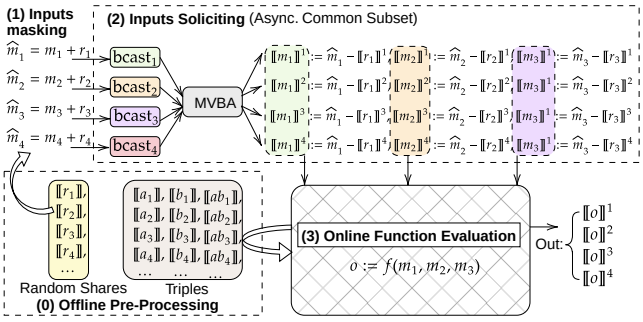


Figure 1: **Offline-online paradigm of AMPCaaS**

2.2 Challenges and Our Techniques

Challenge I: realize both robustness and efficiency while reducing the degree of shares' product. Asynchronous triple generation is at the heart of AMPC. One might immediately realize a "possible" design by adapting the seminal BGW protocol [13]: (i) All parties first collectively generate some t-degree shares $[\![a]\!]_t$ and $[\![b]\!]_t$ of unbiased randomness a and b, which can be done through a few existing asynchronous protocols [43, 44, 46]; (ii) Everyone locally computes $[\![ab]\!]_{2t} = [\![a]\!]_t[\![b]\!]_t$, which represents $2t$-degree secret share of ab; (iii) Then, each party invokes a "degree reduction" phase by re-sharing $[\![ab]\!]_{2t}$ through another t-degree verifiable secret sharing protocol, such that everyone can interpolate the received shares to obtain $[\![ab]\!]_t$, i.e., reducing the $2t$-degree $[\![ab]\!]_{2t}$ to t-degree $[\![ab]\!]_t$. The attempt appears to be enticing, as one seemingly can plug in the state-of-the-art asynchronous complete secret sharing (ACSS) [3, 84] to instantiate the idea. However, during degree reduction, t malicious parties can reshare arbitrary secret instead of the genuine product, making honest parties fail to decode the correct $[\![ab]\!]_t$ in asynchrony and thus causing the breach of robustness.

Therefore, to guarantee robustness during degree reduction, GS23 introduces batch ACSS (built from Pedersen's polynomial commitments) into the BGW framework [57] and uses zk-proof of product relation over Pedersen commitments to attest the re-sharing of correct $[\![ab]\!]_{2t} = [\![a]\!]_t[\![b]\!]_t$ as inspired by Gennaro et al. [53]. However, Pedersen polynomial commitment is large as broadcasting it already causes $O(n^2)$ bits, and GS23 optimizes this by letting ACSS dealer aggregate a batch of such polynomial commitments (after which, the k-th term in the aggregated commitment is a generalized Pedersen commitment that binds all polynomials' k-th coefficients). The approach has clear drawbacks: (i) it requires a concretely large number of $3n$ secrets to be (verifiably) shared per triple, as every party shares two secrets for generating two random sharings $[\![a]\!]$ and $[\![b]\!]$ and then re-shares $[\![a]\!][\![b]\!]$; (ii) the worst-case communication cost of the resulting batch ACSS is quadratic per secret, incurring cubic overhead per triple.

Our approach. We take advantage of succinct and homomorphic KZG polynomial commitment to realize an alternative batch triple generation protocol improving the enticing approach of GS23, both concretely and asymptotically. This nevertheless requires us to devise a new zk-proof scheme attesting the product relation of a triple of secret shares over KZG commitments. To this end, we introduce the new "hidden evaluation" interface to augment the legacy KZG scheme, enabling each party to compute a Pedersen commitment of a certain evaluation of the polynomial committed to the given KZG commitment. Our new interface can convert the proof-of-product over KZG commitments into the problem of proving that over Pedersen's, and hence, we can realize a more efficient triple generation protocol, as our augmented KZG scheme enables: (i) instantiate an efficient batch ACSS together with our careful patch for concurrent compatibility, thus asymptotically reducing the worst-case communication by another $O(n)$ factor; (ii) employ a more efficient randomness extraction technique via hyper-invertible matrix, reducing the number of shared secrets per triple from $3n$ to n.

Challenge II: fallback from fast path to pessimistic path might cause incorrect online computation. The non-robust offline protocol adapted from [8, 70] can optimistically generate triples with an amortized $O(n)$ communication overhead. However, when some malicious party starts to misbehave, this fast path protocol might fail to progress. In such bad cases, fallback is needed to restore robustness. But the tricky effect of network asynchrony is that some honest parties might already generate the r-th batch of triples from the fast path, while some other honest parties only obtain the $(r-1)$-th batch of triples from the fast path, which clearly results in a serious vulnerability of disagreed outputs after fallback.

Given this, if the honest nodes trivially quit from unresponsive fast path and then immediately start the pessimistic path, the online computation might no longer be correct, because

for a multiplication gate, some honest parties might use triple $([\![a]\!],[\![b]\!],[\![ab]\!])$ generated via fast path but some other parties might use another triple $([\![a']\!],[\![b']\!],[\![a'b']\!])$ generated by the pessimistic path. On the other side, completely withdrawing all fast path outcome after fallback might resolve the inconsistency issue but makes the fast path pointless.

Our approach. To resolve the above inconsistency issue caused by failed fast path, we take a more careful observation on the distribution of honest parties' fast path progress when they enter fallback. A key finding is that: for the honest parties, their progresses in the fast path are either the same or have a variance of at most one, when they detect the fast path failed or unresponsive. This is a simple fact because our fast path requires all parties to be responsive as it requires each party to wait for messages from all parties. If there is any honest party P_i starts fallback with only obtaining the r-th batch of fast-path triples, then no honest party can obtain the $(r+2)$-th batch of fast-path triples, because P_i would not collaborate in the $(r+2)$-th fast-path triple generation. This hints us at introducing a variant of asynchronous binary agreement for reaching consensus among two consecutive values (tcv-BA) [71]. Namely, every party takes its fast path progress r as input to tcv-ABA during fallback, then tcv-BA returns a common value R, using which, the honest parties can reach a unitary decision on preserving how many fast-path triples before entering the pessimistic path, thereby resolving any inconsistency that might cause incorrect online evaluation.

3 Preliminaries

Asynchronous multi-valued validated Byzantine agreement (MVBA) [2, 4, 27, 43, 48, 72, 89] is a variant of asynchronous Byzantine agreement with output satisfying a public boolean predicate. In particular, MVBA running among n parties is parameterized by a global predicate $Q : \mathbb{X} \times \mathbb{S} \to \{True, False\}$. Here $\mathbb{X}$ represents the domain of inputs and $\mathbb{S}$ denotes the domain of each party's internal states, and Q shall be (i) *monotonic*, i.e., it cannot switch from true to false as the honest party's state S_i evolves, and (ii) *eventually unanimous*, i.e., if $Q(x,S_i)$ is true due to some honest party P_i's state S_i, then eventually $Q(x,S_j)$ becomes true for every honest party P_j's internal state S_j. Such MVBA protocol satisfies the next properties with overwhelming probability: (i) *Termination*, if every honest node P_i starts the protocol with input x_i and internal states S_i s.t. $Q(x_i,S_i) = True$, all honest nodes would eventually output; (ii) *Agreement*, any two honest nodes P_i and P_j output y_i and y_j, respectively, then $y_i = y_j$; (iii) *External Validity*, every honest node P_i's output y_i can be (eventually) valid due to the predicate Q and P_i's internal states S_i.

Asynchronous two-consecutive-value Byzantine agreement (tcv-BA) [71] is an extended asynchronous binary Byzantine agreement where honest parties input two consecutive integers or the same integer. If all honest parties activate a tcv-BA protocol by inputting a value in $\{v, v+1\}$ where $v \in \mathbb{N}$, then the following properties would hold with overwhelming probability: (i) *Termination*, all honest parties would output some value; (ii) *Agreement*, any two honest parties' outputs are the same; (iii) *Validity*, if some honest parties output x, then at least one honest party inputs x.

4 Proof of Product Relation over KZG Polynomial Commitments

As aforementioned, we focus on improving the efficiency of the enticing GS23 protocol through succinct and homomorphic KZG polynomial commitment. This requires us to devise a zero-knowledge proof (zk-proof) scheme for the product relationship of certain polynomial evaluations over KZG commitments, ensuring that any malicious party either re-shares the correct product of $[\![ab]\!]^i_{2t} = [\![a]\!]^i \cdot [\![b]\!]^i$ or does nothing harmful. This section will elaborate on our approach to realizing such efficient zk-proof over KZG commitments.

KZG Polynomial Commitment w/ Hidden Evaluation

$\mathsf{Setup}(1^\kappa, t)$: generate bilinear pairing group $\mathcal{G} = (e, \mathbb{G}, \mathbb{G}_T)$ and randomly sample $\alpha, \tau \in \mathbb{Z}_q$. Let g be random generator of $\mathbb{G}$, set $h = g^\tau$ and return $SP = \{\mathcal{G}, \{g^{\alpha^i}, h^{\alpha^i}\}_{i=0}^t\}$.

$\mathsf{PolyCom}(SP, \phi(\cdot))$: sample random t-degree polynomial $\hat{\phi}(\cdot)$, and compute $C = g^{\phi(\alpha)} h^{\hat{\phi}(\alpha)}$.

$\mathsf{ProveEval}(SP, i, \phi(\cdot), \hat{\phi}(\cdot))$: compute $w_i = g^{\psi_i(\alpha)} h^{\hat{\psi}_i(\alpha)}$, where $\psi_i(x) = \frac{\phi(x)-\phi(i)}{x-i}$ and $\hat{\psi}_i(x) = \frac{\hat{\phi}(x)-\hat{\phi}(i)}{x-i}$, output $(i, \phi(i), \hat{\phi}(i), w_i)$.

$\mathsf{VerifyEval}(SP, C, i, \phi(i), \hat{\phi}(i), w_i)$: output 1 if $e(C/(g^{\phi(i)} h^{\hat{\phi}(i)}), g) = e(w_i, g^\alpha / g^i)$, output 0 otherwise.

Below are hidden evaluation interfaces

$\mathsf{HiddenEval}(SP, i, w_i, \phi(i), \hat{\phi}(i))$: compute $T_i = g^{\phi(i)} h^{\hat{\phi}(i)}$ and output (i, T_i, ω_i) where $\omega_i = w_i$.

$\mathsf{OpenHiddenEval}(SP, \phi(i), \hat{\phi}(i), T_i)$: if $T_i = g^{\phi(i)} h^{\hat{\phi}(i)}$, output 1, otherwise output 0.

$\mathsf{VerifyHiddenEval}(SP, C, i, T_i, \omega_i)$: if $e(C/T_i, g) = e(\omega_i, g^\alpha / g^i)$, output 1, otherwise output 0.

NOTE: we also use BatchVerifyEval and BatchVerifyHiddenEval to denote the batch versions of VerifyEval and VerifyHiddenEval, which simultaneously perform many verifications for efficiency.

Figure 2: **Our augmented KZG polynomial commitment with new hidden evaluation interfaces (KZG PCwHE).**

Adding hidden evaluation interfaces. To prove the product relation of secret shares committed to KZG polynomial commitments, we first introduce new algorithmic interfaces to the KZG scheme. Informally, these new interfaces enable the following: given a KZG polynomial commitment, a polynomial evaluation, and the valid evaluation proof, one can compute the Pedersen commitment of this evaluation and bind this Pedersen commitment to the given KZG commitment.

As illustrated in Figure 2, our construction of hidden evaluation stems from the observation that the VerifyEval function of the KZG scheme only requires a couple of Pedersen-style commitments, $g^{\phi(i)}h^{\hat{\phi}(i)}$ and $g^{\psi(\alpha)}h^{\hat{\psi}(\alpha)}$, to check the equality of two pairings, which hints at that they could be the expected hidden evaluation and corresponding proof. We then prove that our proposed augmented KZG scheme with hidden evaluation interfaces satisfies the properties of (i) *correctness*, (ii) *polynomial binding*, (iii) *evaluation binding*, (iv) *hiding* and (v) *hidden evaluation's binding*, assuming the hardness of discrete logarithm and t-strong Diffie-Hellman problems. We formalize these properties below.

Correctness: Given any $SP \leftarrow \mathsf{Setup}(1^\kappa, t)$ and any $\phi(\cdot) \in \mathbb{Z}_q[x]$, if $C \leftarrow \mathsf{PolyCom}(SP, \phi(\cdot))$, then:

- For any output of $\mathsf{ProveEval}(SP, i, \phi(\cdot), \hat{\phi}(\cdot))$, it can always pass the verification of VerifyEval.
- For any output of $\mathsf{HiddenEval}(SP, i, w_i, \phi(i), \hat{\phi}(i))$, it can always be accepted by VerifyHiddenEval.
- For any output of $\mathsf{HiddenEval}(SP, i, w_i, \phi(i), \hat{\phi}(i))$, there is $\mathsf{OpenHiddenEval}(SP, \phi(i), \hat{\phi}(i), T_i) = 1$.

Evaluation binding: For any P.P.T. adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} SP \leftarrow \mathsf{Setup}(1^\kappa, t); \\ (C, (i, \phi(i), \hat{\phi}(i), w_i), (i, \phi(i)', \hat{\phi}(i)', w_i')) \leftarrow \mathcal{A}(SP): \\ \mathsf{VerifyEval}(SP, C, i, \phi(i), \hat{\phi}(i), w_i) = 1 \wedge \\ \mathsf{VerifyEval}(SP, C, i, \phi(i)', \hat{\phi}(i)', w_i') = 1 \wedge \phi(i) \neq \phi(i)' \end{array}\right] \leq \varepsilon(\kappa)$$

Polynomial binding: For any P.P.T. adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} SP \leftarrow \mathsf{Setup}(1^\kappa, t); \\ (C, I_1, I_2, \{(\phi(i), \hat{\phi}(i), w_i)\}_{i \in I_1 \cup I_2}) \leftarrow \mathcal{A}(SP) \\ \text{where } I_1 \subset [n], I_2 \subset [n], \text{and } |I_1| = |I_2| = t+1: \\ \forall i \in I_1 \cup I_2, \mathsf{VerifyEval}(SP, C, i, \phi(i), \hat{\phi}(i), w_i) = 1 \wedge \\ (\phi_1(\cdot), \hat{\phi}_1(\cdot)) \leftarrow \mathit{Interpolate}(I_1, \{\phi(i), \hat{\phi}(i)\}_{i \in I_1}) \wedge \\ (\phi_2(\cdot), \hat{\phi}_2(\cdot)) \leftarrow \mathit{Interpolate}(I_2, \{\phi(i), \hat{\phi}(i)\}_{i \in I_2}) \wedge \\ \phi_1(\cdot) \neq \phi_2(\cdot) \end{array}\right] \leq \varepsilon(\kappa)$$

(Augmented) hiding: The polynomial commitment and all hidden evaluations should not reveal any additional information about the polynomial beyond t shares that are already known to the adversary. This is an enhanced hiding property of the original KZG scheme, which now also accounts for outputs from the hidden evaluation interface. Formally, for any P.P.T. adversary, a P.P.T. simulator $(\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$ exists, such that the next two distributions are identical:

Real world:

$$\left\{\begin{array}{l} SP \leftarrow \mathsf{Setup}(1^\kappa, t); (\phi(\cdot), I_1, I_2) \leftarrow \mathcal{A}(SP); \\ (C, \hat{\phi}(\cdot)) \leftarrow \mathsf{PolyCom}(SP, \phi(\cdot)); \\ \forall i \in I_1, (\phi(i), \hat{\phi}(i), w_i) \leftarrow \mathsf{ProveEval}(SP, i, \phi(\cdot), \hat{\phi}(\cdot)); \\ \forall i \in I_2, (T_i, \omega_i) \leftarrow \mathsf{HiddenEval}(SP, i, \phi(\cdot), \hat{\phi}(\cdot)) \\ : (C, \{i, \phi(i), \hat{\phi}(i), w_i\}_{i \in I_1}, \{i, T_i, \omega_i\}_{i \in I_2}) \text{ where } |I_1| \leq t \end{array}\right\}$$

Ideal world:

$$\left\{\begin{array}{l} (SP, st) \leftarrow \mathsf{Sim}_0(1^\kappa, t); (\phi(\cdot), I_1, I_2) \leftarrow \mathcal{A}(SP); \\ (C, c) \leftarrow \mathsf{Sim}_1(st); \\ (\{w_i\}_{i \in I_1}, \phi'(\cdot), \hat{\phi}'(\cdot)) \leftarrow \mathsf{Sim}_2(st, (C, c), \{i, \phi(i)\}_{i \in I_1}); \\ \{T_i, \omega_i\}_{i \in I_2} \leftarrow \mathsf{Sim}_3(st, (C, c), \phi'(\cdot), \hat{\phi}'(\cdot), I_2) \\ : (C, \{i, \phi(i), \hat{\phi}(i), w_i\}_{i \in I_1}, \{i, T_i, \omega_i\}_{i \in I_2}) \text{ where } |I_1| \leq t \end{array}\right\}$$

Hidden evaluation's binding: For all adversaries $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} SP \leftarrow \mathsf{Setup}(1^\kappa, t); \\ (C, i, (T_i, \omega_i, \phi'(i), \hat{\phi}'(i)), (\phi(i), \hat{\phi}(i), w_i)) \leftarrow \mathcal{A}(SP): \\ \mathsf{VerifyHiddenEval}(SP, C, i, T_i, \omega_i) = 1 \wedge \\ \mathsf{OpenHiddenEval}(SP, \phi'(i), \hat{\phi}'(i), T_i) = 1 \wedge \\ \mathsf{VerifyEval}(SP, C, i, \phi(i), \hat{\phi}(i), w_i) = 1 \wedge \hat{\phi}'(i) \neq \hat{\phi}(i) \end{array}\right] \leq \varepsilon(\kappa)$$

REMARKS. Clearly, our augmented KZG polynomial commitment with hidden evaluation (KZG PCwHE) satisfies all properties of the original KZG scheme [66]. Additionally, it provides a convenient interface to convert the binding of a secret share from a given KZG commitment to another Pedersen commitment that not only binds the secret share originally committed to KZG but also leaks nothing about the secret. For detailed proofs of KZG PCwHE, cf. Appendix B.

PoK of Product over KZG Commitments $\mathsf{PoK}_{\mathsf{Prod\_KZG}}$

The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ both know the public statement $(SP, C_a, C_b, C_c, i_a, i_b, i_c)$ where $\{C_x\}_{x \in \{a,b,c\}}$ are KZG polynomial commitments and $\{i_x\}_{x \in \{a,b,c\}}$ are points to evaluate polynomials. $\mathcal{P}$ has private witness $\{(\phi_x(i_x), \hat{\phi}_x(i_x), w_x)\}_{x \in \{a,b,c\}}$, s.t. $\forall x \in \{a,b,c\}$, $\mathsf{VerifyEval}(SP, C_x, i_x, \phi_x(i_x), \hat{\phi}_x(i_x), w_x) = 1$.

// Prover $\mathcal{P}$

- $\mathsf{PoK}_{\mathsf{Prod\_KZG}}.\mathcal{P}(\{(C_x, i_x, \phi_x(i_x), \hat{\phi}_x(i_x), w_x)\}_{x \in \{a,b,c\}})$:
  (a) compute $\mathsf{HiddenEval}(SP, i_x, w_x, \phi(i_x), \hat{\phi}(i_x))$ to obtain (T_x, ω_x) for each $x \in \{a,b,c\}$;
  (b) invoke the prover of the proof-of-product scheme for values committed to Pedersen commitments (cf. Figure 13) with taking $\{(\phi_x(i_x), \hat{\phi}_x(i_x))\}_{x \in \{a,b,c\}}$ as witness, and obtain the proof π regarding the product relation of (T_a, T_b, T_c);
  (c) output $\mathsf{proof} = (\{(T_x, \omega_x)\}_{x \in \{a,b,c\}}, \pi)$.

// Verifier $\mathcal{V}$

- $\mathsf{PoK}_{\mathsf{Prod\_KZG}}.\mathcal{V}(\{(C_x, i_x)\}_{x \in \{a,b,c\}}, \mathsf{proof})$:
  (a) parse proof as $(\{(T_x, \omega_x)\}_{x \in \{a,b,c\}}, \pi)$;
  (b) verify $\mathsf{VerifyHiddenEval}(SP, C_x, i_x, T_x, \omega_x)$=1 for $x \in \{a,b,c\}$;
  (c) verify the proof-of-product π regarding (T_a, T_b, T_c) (cf. Figure 13), return 1 if all checks pass and 0 otherwise.

Figure 3: **Zk-proof of product relation for a triple of evaluations of polynomials committed to KZG commitments.**

Proving product relationship over KZG. Given our augmented KZG scheme with hidden evaluation interfaces, we realize a convenient way to prove the product relationship of secret shares over a triple of (publicly known) KZG commitments, as illustrated in Figure 3. In particular, the proposed proof-of-knowledge zk-proof attests the following statement:

$$\mathbf{PoK}_{\mathsf{Prod\_KZG}}[\{(\phi_x(i_x), \hat{\phi}_x(i_x), w_x)\}_{x \in \{a,b,c\}}:$$
$$\phi_a(i_a) \cdot \phi_b(i_b) = \phi_c(i_c) \wedge$$
$$\mathsf{VerifyEval}(SP, C_x, i_x, \phi_x(i_x), \hat{\phi}_x(i_x), w_x) = 1, \forall x \in \{a,b,c\}]$$

which proves that the secret share $\phi_c(i_c)$ committed to KZG commitment C_c is indeed a correct product of two legitimate secrets' shares $\phi_a(i_a)$ and $\phi_b(i_b)$ that are respectively committed to two KZG commitments C_a and C_b. Intuitively, the

proof scheme first computes hidden evaluation as a Pedersen commitment for each secret share, and then proves the product relationship over these derived Pedersen commitments. In the next section when we instantiate our asynchronous triple generation protocol, this zk-proof plays a crucial role in preventing malicious parties from re-sharing arbitrary secrets instead of the genuine product of two expected shares.

Algorithm 1 Asynchronous random share generation ▷ Code of P_i

Input: N the batch size of BACSS, and other system parameters

Output: rnd_shares $= \{(C_k, [\![r_k]\!], [\![\hat{r}_k]\!], w_k)_{k=1}^{N(t+1)}\}$

// 1. Invoke BACSS to share randomness

$\mathcal{T}_i \leftarrow \emptyset$, shares$_i \leftarrow \emptyset$, rnd_shares $\leftarrow \emptyset$

activate BACSS_share$_j$ instance for every $j \in [n]$

uniformly sample N secrets $s_{i1}, \cdots, s_{iN}$ from $\mathbb{Z}_q$

invoke BACSS_share$_i$ as dealer to share $s_{i1}, \cdots, s_{iN}$

// Each P_i generate n private-public key pairs for IND-CCA PKE, so the j-th key pair is exclusively used in dealer P_j's BACSS.

// 2. Select $n-t$ completed BACSS instances

upon getting $(C_{jk}, [\![s_{jk}]\!]^i, [\![\hat{s}_{jk}]\!]^i, w^i_{jk})_{k\in[N]}$ from BACSS_share$_j$

  $\mathcal{T}_i \leftarrow \mathcal{T}_i \bigcup \{j\}$

  shares$_i \leftarrow$ shares$_i \bigcup \{(C_{jk}, [\![s_{jk}]\!]^i, [\![\hat{s}_{jk}]\!]^i, w^i_{jk})_{k\in[N]}\}$

  if $|\mathcal{T}_i| = 2t+1$

    call MVBA using a snapshot of $\mathcal{T}_i$ as input and $\mathcal{T}_i$ as state

    // MVBA's predicate waits for local $\mathcal{T}_i$ is a superset of input

wait for MVBA output $\mathcal{T}$, w.l.o.g., let $\mathcal{T} = \{P_1, \cdots, P_{2t+1}\}$

// 3. Extract random shares

for $k = 1$ to N

  $([\![r_{1k}]\!]^i, \cdots, [\![r_{(t+1)k}]\!]^i) \leftarrow M([\![s_{1k}]\!]^i, \cdots, [\![s_{(2t+1)k}]\!]^i)^T$

  $([\![\hat{r}_{1k}]\!]^i, \cdots, [\![\hat{r}_{(t+1)k}]\!]^i) \leftarrow M([\![\hat{s}_{1k}]\!]^i, \cdots, [\![\hat{s}_{(2t+1)k}]\!]^i)^T$

  for $l = 1$ to $t+1$

    $C_{lk} = \prod_{j=1}^{2t+1} (C_{jk})^{M_{lj}}, \quad w^i_{lk} = \prod_{j=1}^{2t+1} (w^i_{jk})^{M_{lj}}$

    rnd_shares $\leftarrow$ rnd_shares $\bigcup \{(C_{lk}, [\![r_{lk}]\!]^i, [\![\hat{r}_{lk}]\!]^i, w^i_{lk})\}$

return rnd_shares

5 Robust AMPC Offline Protocols

The core of Dumbo-MPC is a robust yet still efficient asynchronous offline phase. Building on our previously devised proof-of-product scheme over KZG commitments, this section will elaborate on our robust AMPC offline protocols.

5.1 Asynchronous Random Share Generation

We start by presenting an asynchronous random share generation (AsyRanShGen) protocol, optimized through batching for enhanced efficiency, capable of generating t-degree random shares with an amortized communication cost of $O(\kappa n)$. We employ a randomness extraction method utilizing hyper-invertible matrices (where every square sub-matrix is invertible) [12], which was recently adapted by Das et al. for the asynchronous setting, though without batching [43].

Patch the concurrent composability for batch ACSS. One might suggest directly integrating the state-of-the-art batch asynchronous complete secret sharing (BACSS) hbACSS [84] into Das et al.'s design to implement a batched AsyRanShGen protocol.[8] Here BACSS allows a dealer to share a batch of secrets $(s_1, \cdots, s_N)$ across the whole network, such that (i) each honest party P_i would output a batch of N secret shares (where the k-th output $[\![s_k]\!]^i$ corresponds to a share of the k-th input secret s_k), or (ii) probably all honest parties eventually output nothing, if the dealer is malicious. However, we identify a flaw of the above seemingly viable idea due to lacking concurrent composability in the original hbACSS, which might even cause realistic secrecy attacks allowing a malicious dealer copy scripts from another honest dealer and thus learn the secrets (cf. Appendix C.3 for detailed attacks).

We therefore make necessary yet still minimal adaptions of hbACSS to patch its concurrent composability: (i) use IND-CCA secure public key encryption (PKE) instead of IND-CPA PKE in the original hbACSS design; (ii) specify that each party wouldn't re-use PKE keys in different dealers' hbACSS instances. Such enhancement enables us to prove the crucial secrecy of concurrent hbACSS instances. Syntactically, we also explicitly require that the honest parties in BACSS additionally output N KZG polynomial commitments besides N secret shares, such that each output KZG commitment fixes a polynomial whose evaluations correspond to the k-th output shares of all parties. The communication complexity of this adapted hbACSS still is $O(\kappa nN + \kappa n^2 \log n)$. When letting the batch size sufficiently large such as $N \geq \Omega(n \log n)$, the amortized communication cost is $O(\kappa n)$ per shared secret. For space limit, we refrain from representing the whole protocol of the adapted hbACSS, and defer its details to Appendix C.2.

AsyRanShGen protocol. Algorithm 1 describes the protocol for batchedly generating random shares atop concurrently composable BACSS protocols, which executes as:

1. *Share randomness*. Each P_i samples a batch of random secrets, and subsequently starts as a dealer of one BACSS protocol to verifiably share them in a batched manner.
2. *Solicit sufficient (probably biased) randomness shares*. In the asynchronous setting, one cannot wait for the terminations of all parties' BACSS protocols, so once a party P_i outputs in $n-t$ distinct dealers' BACSS protocols, it activates an MVBA by taking the indices of these finished BACSS as input. MVBA ensures that all honest parties can select a common set of $n-t$ BACSS protocols with distinct dealers, and all honest parties can eventually obtain shares from these $n-t$ solicited BACSS protocols.
3. *Extract uniform randomness via hyper-invertible matrix*. Among the $n-t$ selected BACSS protocols, at least $t+1$ of them shall have honest dealers that indeed share uniform randomness sampled from $\mathbb{Z}_q$, and at most t of them could be chosen by the adversary.

[8] Note that hbACSS [84] is more preferred than another state-of-the-art BACSS protocol Bingo [3] due to significantly better concrete efficiency, as the computation and communication of Bingo are about 3 times of hbACSS.

Thus, to extract as many as possible random shares from these $n-t$ BACSS protocols, we leverage the idea of randomness extraction via Vandermonde matrix $M \in \mathbb{Z}_q^{(t+1)\times(2t+1)}$, where the j-th row is $M_j = (\eta_j^0, \eta_j^1, \cdots, \eta_j^{2t})$ and $\eta_j \in \mathbb{Z}_q$ is distinct for each j. So given $2t+1$ BACSS protocols (each of which batchedly shares N secrets), the honest parties can extract a number of $(t+1)*N$ random shares.

The polynomial commitments and evaluation proofs related to the extracted random shares can also be locally computed by each honest party (without interacting), due to the additive homomorphism of KZG scheme. Soon in the next subsection, we will leverage these resulting polynomial commitments for robust triple generation.

Security of AsyRanShGen. The properties of AsyRanShGen can be summarized as the following security theorem.

Theorem 1 (Termination, Consistency, Secrecy and Randomness of AsyRanShGen)**.** *If all honest parties activate the AsyRanShGen protocol (running among n parties against up to t malicious corruptions), then the following properties hold with all but negligible probability:*

- *Termination. All honest parties would eventually output a set* $\mathsf{rnd\_shares} = \{(C_k, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i)\}_{k=1}^{N(t+1)}$.
- *Consistency. The output* rnd_shares *of all honest parties are "consistent", namely: (i) they carry the same* $\{C_k\}_{k=1}^{N(t+1)}$*; (ii) for each* $(C_k, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i) \in$ rnd_shares, $\mathsf{VerifyEval}(SP, C_k, i, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i)$=1.
- *Secrecy and randomness. The adversary cannot predicate each* r_k *better than guessing over* $\mathbb{Z}_q$.

We defer the full proof of the above security theorem to Appendix G.1, as most analysis is rather standard and mirrors the literature [18, 43, 58]. Particularly considering that we already patch the concurrent composability of BACSS, it enables us construct a simulator that always simulates decryption queried by adversary in face of malicious BACSS dealers, and such simulated decryption wouldn't leak secrets of other concurrent BACSS instances.

Complexities of AsyRanShGen. In the AsyRanShGen protocol, $\Theta(n)$ instances of BACSS are involved. If each BACSS has a sufficient batch size $N = \Omega(n \log n)$, the communication cost of n BACSS protocols becomes $O(\kappa n^3 \log n)$ in total. Additionally, one MVBA protocol with n-bit input is invoked, and its state-of-the-art instantiation costs $O(\kappa n^3)$ bits [4, 47]. In sum, AsyRanShGen can batchedly generate $N(t+1)$ random shares with a total communication cost of $O(\kappa n^3 \log n)$ bits, which corresponds to an amortized communication complexity of $O(\kappa n)$ per generated random share.

5.2 Asynchronous Random Triple Generation

Here we are ready to present the protocol of asynchronous random multiplication triple generation (AsyRanTriGen), which can be thought of an asynchronous, optimal-resilient, and batched version of the seminal BGW paradigm and is also concretely and asymptotically more efficient than GS23.

Algorithm 2 Asynchronous mul. triples generation ▷ Code of P_i

Input: B batch size of triple generation and other system parameters
Output: $\mathsf{Beaver\_triples} = \{([\![a_k]\!]^i, [\![b_k]\!]^i, [\![c_k]\!]^i)\}_{k=1}^{B}$

$\mathcal{T}_i \leftarrow \emptyset$, $\mathsf{prod\_shares}_i \leftarrow \emptyset$, $\mathsf{Beaver\_triples} \leftarrow \emptyset$
// 1. Generate random shares
activate AsyRanShGen with taking as input $N = (2B/(t+1))$ randomness, and wait for AsyRanShGen returns $\mathsf{rnd\_shares} = \{(C_k, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i)\}_{k\in[2B]}$
let first half of rnd_shares be $\{(C_{ak}, [\![a_k]\!]^i, [\![\hat{a}_k]\!]^i, w_{ak}^i)\}_{k\in[B]}$ and last half of rnd_shares be $\{(C_{bk}, [\![b_k]\!]^i, [\![\hat{b}_k]\!]^i, w_{bk}^i)\}_{k\in[B]}$
// 2. Re-share $2t$-degree product
for $k = 1$ to B
  $[\![c_k]\!]_{2t}^i \leftarrow [\![a_k]\!]^i \cdot [\![b_k]\!]^i$
  $(C_{ck}^i, \hat{\phi}_{ck}^i(\cdot)) \leftarrow \mathsf{PolyCom}(SP, \phi_{ck}^i(\cdot))$, s.t. $\phi_{ck}^i(0) = [\![c_k]\!]_{2t}^i$
  $w_{ck,0}^i \leftarrow \mathsf{ProveEval}(SP, 0, \phi_{ck}^i(\cdot), \hat{\phi}_{ck}^i(\cdot))$
  $\mathsf{proof}_k^i \leftarrow \mathsf{PoK}_{\mathsf{Prod\_KZG}}.\mathcal{P}\left(\left\{\begin{array}{c}(C_{ak}, i, [\![a_k]\!]^i, [\![\hat{a}_k]\!]^i, w_{ak}^i),\\ (C_{bk}, i, [\![b_k]\!]^i, [\![\hat{b}_k]\!]^i, w_{bk}^i),\\ (C_{ck}^i, 0, \phi_{ck}(0), \hat{\phi}_{ck}(0), w_{ck,0}^i)\end{array}\right\}\right)$
  cf. Fig 3 for details of the above proof of product relationship
activate $\mathsf{BACSS\_share}_j'$ instance for every $j \in [n]$
invoke $\mathsf{BACSS\_share}_i'$ as dealer to re-share $\{[\![c_k]\!]_{2t}^i\}_{k\in[B]}$ using KZG commitments $\{C_{ck}^i\}_{k\in[B]}$ with broadcasting $\{\mathsf{proof}_k^i\}_{k\in[B]}$
// 3. Wait for $n-t$ completed re-shares of product
upon receiving output from $\mathsf{BACSS\_share}_j'$ instance:
  if $\mathsf{PoK}_{\mathsf{Prod\_KZG}}.\mathcal{V}\left(\left\{\begin{array}{c}(C_{ak}, j),\\ (C_{bk}, j),\\ (C_{ck,0}^j, 0)\end{array}\right\}, \mathsf{proof}_k^j\right)$=1 for $\forall\, k \in [B]$:
    $\mathsf{prod\_shares}_i \leftarrow \mathsf{prod\_shares}_i \bigcup \{([\![[\![c_k]\!]_{2t}^j]\!]^i)_{k\in[B]}\}$
    $\mathcal{T}_i \leftarrow \mathcal{T}_i \bigcup \{j\}$
    if $|\mathcal{T}_i| = 2t+1$
      call MVBA′ using a snapshot of $\mathcal{T}_i$ as input and $\mathcal{T}_i$ as state
// 4. Locally interpolate t-degree share of product
wait for MVBA output $\mathcal{T}$, w.l.o.g., let $\mathcal{T} = \{P_1, \cdots, P_{2t+1}\}$
for $k = 1$ to B
  $[\![c_k]\!]^i \leftarrow \sum_{j=1}^{2t+1} \lambda_j [\![[\![c_k]\!]_{2t}^j]\!]^i$, where λ_j represents the evaluation of the j-th Lagrange interpolating polynomial at zero point
  $\mathsf{Beaver\_triples} \leftarrow \mathsf{Beaver\_triples} \bigcup \{([\![a_k]\!]^i, [\![b_k]\!]^i, [\![c_k]\!]^i)\}$
return Beaver_triples

AsyRanShGen protocol. As illustrated in Algorithm 2, the protocol proceeds through running the following four steps:

1. *Batchedly generate random shares*. All parties execute the AsyRanShGen protocol presented in the earlier subsection, where each party P_i randomly chooses $N = \frac{2B}{t+1}$ secrets to share via its BACSS instance and would obtain $\{(C_k, [\![s_k]\!]^i, [\![\hat{s}_k]\!]^i, w_k^i)_{k\in[2B]}$, i.e., $2B$ random shares along with all related KZG commitments and evaluations proofs. Presentation-wise, we partition the output set into two halves, denoted as $\{(C_{ak}, [\![a_k]\!]^i, [\![\hat{a}_k]\!]^i, w_{ak}^i)\}_{k\in[B]}$ and $\{(C_{bk}, [\![b_k]\!]^i, [\![\hat{b}_k]\!]^i, w_{bk}^i)\}_{k\in[B]}$, respectively.

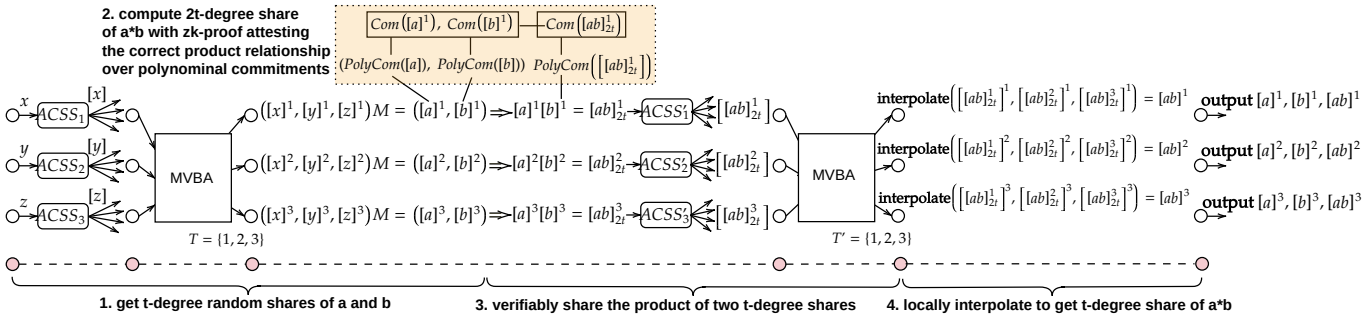


Figure 4: **The overview of asynchronous multiplication triple generation protocol (exemplified without batching).**

2. *Locally compute $2t$-degree shares of products*. Given the output obtained from the AsyRanShGen protocol, every party P_i computes $[\![c_k]\!]^i_{2t} = [\![a_k]\!]^i \cdot [\![b_k]\!]^i$ for each $k \in [B]$.
3. *Re-share the correct $2t$-degree product*. Then P_i invokes another BACSS protocol as dealer to re-share $[\![c_k]\!]^i_{2t}$ using KZG polynomial commitment C^i_{ck}.
   We also require P_i to compute and broadcast some additional $\{\mathsf{proof}^i_k\}$ attesting the correctness of the re-shared $2t$-degree product $[\![c_k]\!]^i_{2t}$. Particularly, each $\mathsf{proof}^i_k = \{T^i_{ak}, \omega^i_{ak}, T^i_{bk}, \omega^i_{bk}, T^i_{ck,0}, \omega^i_{ck,0}, \pi^i_k\}$ is a zero-knowledge proof of the product relationship over KZG commitments C_{ak}, C_{bk} and C^i_{ck} (as earlier described in Figure 3). Here $(T^i_{ak}, \omega^i_{ak})$ are a hidden evaluation (i.e. a Pedersen commitment) and a hidden evaluation proof regarding KZG commitment C_{ak}, attesting that T^i_{ak} commits the i-th evaluation of the polynomial fixed by C_{ak}; Similarly, ω^i_{bk} attests T^i_{bk} commits an evaluation of the polynomial fixed by C_{bk} at point i; While $\omega^i_{ck,0}$ proves that $T^i_{ck,0}$ commits the 0-point secret of the polynomial committed to C^i_{ck}.
4. *Reconstruct t-degree shares of products to obtain triples*. After everyone re-shares its local $2t$-degree shares, each P_i activates another MVBA protocol to decide a common set of $2t+1$ completed re-sharings. Finally, each P_i locally interpolates the t-degree share $[\![c_k]\!]^i$ of the product. As P_i also has shares $[\![a_k]\!]^i$ and $[\![b_k]\!]^i$, so it can return a random triple $([\![a_k]\!]^i, [\![b_k]\!]^i, [\![c_k]\!]^i)$ for each $k \in [B]$.

Security of AsyRanTriGen. The security of AsyRanTriGen can be summarized by the following theorem.

Theorem 2 (Termination, Validity, and Secrecy of AsyRanTriGen)**.** *In an AsyRanTriGen protocol running among n parties with up to $t < n/3$ corruptions, each party P_i outputs a batch of triples $\{([\![a_k]\!]^i, [\![b_k]\!]^i, [\![c_k]\!]^i)\}^B_{k=1}$, and the following properties shall hold with all but negligible probability:*

- *Termination. If all honest parties activate the protocol, all of them would output $\{([\![a_k]\!]^i, [\![b_k]\!]^i, [\![c_k]\!]^i)\}^B_{k=1}$.*
- *Validity. For each $k \in B$ and $x \in \{a,b,c\}$, honest parties can interpolate their output shares $\{[\![x_k]\!]^i\}$ to obtain a unique t-degree polynomial whose zero-point is x_k, and the interpolated $\{x_k\}_{x\in\{a,b,c\}}$ satisfying $a_k \cdot b_k = c_k$.*
- *Secrecy. For any output triple $([\![a_k]\!], [\![b_k]\!], [\![c_k]\!])$, adversary learns nothing about a_k, b_k and c_k except $a_k \cdot b_k = c_k$.*

We provide the intuitions of proofs hereunder, and defer details of security analysis to Appendix G.2 for space limit:

- For termination, this is trivial, as its violation is reducible to breach either of (i) the correctness of BACSS; (ii) the completeness of BACSS; (iii) the termination of MVBA.
- For validity, the arguments are: (i) the random shares $[\![a]\!]$ and $[\![b]\!]$ obtained by honest parties must be consistent to fixed t-degree polynomials, because they are evaluations bounded to the same polynomial commitments; (ii) the honest parties' secret shares $[\![c]\!]$ must correspond to the same t-degree polynomial with $c = a \cdot b$, because otherwise, either the hidden evaluation's binding of our augmented KZG scheme is violated or the knowledge soundness of the proof of product relationship is breached.
- For secrecy, we construct a P.P.T. simulator using the setup trapdoor of KZG commitment and black-box simulators of product proof and IND-CCA encryption, so we can get into a hybrid world where the adversary interacts with the above simulator, and the adversary's view in this hybrid world is computationally indistinguishable from its view in the real-world execution. Observing that the adversary in the final hybrid world can only information-theoretically get t shares of $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$, secrecy is therefore proven in the real-world execution.

Complexities of AsyRanTriGen. An AsyRanTriGen protocol that generates B triples involves an AsyRanShGen instance generating $2B$ random shares, n BACSS protocol instances (each of which shares B secrets), and another MVBA instance. For sufficiently large batch B, the communication of n BACSS instances with batch size B becomes the dominating factor, resulting in amortized $O(\kappa n^2)$ bits per triple.

6 Adding Fast Path for Triple Generation

Now we elaborate on how to further harvest efficiency from optimistic conditions using a concretely and asymptotically more efficient non-robust fast path, with preserving G.O.D.

for all possible bad cases, through a simplistic yet still fully asynchronous fallback mechanism.

Fast path protocol. Algorithm 3 describes a non-robust *optimistic* random triple generation protocol (OptRanTriGen) adapted from [12, 41, 70]. For completeness, we repeat the protocol here with our tailored optimization (which removes $O(n)$ broadcasts and thus saves 3 additional rounds), while deferring its security analysis and the used algorithms of plain Shamir's secret sharing to Appendices G.3 and F, respectively.

In OptRanTriGen, each party shares the same random secret to all parties twice, using t-degree and $2t$-degree plain Shamir secret sharing without verifiability. Then, everyone waits for these shares from all parties in order to extract $t+1$ random double sharings $(\llbracket r_1\rrbracket, \llbracket r_1\rrbracket_{2t}), \cdots, (\llbracket r_{t+1}\rrbracket, \llbracket r_{t+1}\rrbracket_{2t})$ optimistically. Finally, all parties try to reduce the degree of $\llbracket ab\rrbracket_{2t}$ to get $\llbracket ab\rrbracket$, by (i) optimistically reconstructing $ab-r$ by interpolating the $2t$-degree shares $\llbracket ab-r\rrbracket_{2t}$ and (ii) adding $\llbracket r\rrbracket$ and $ab-r$ to get $\llbracket ab\rrbracket$. As such, the fast path can simply execute a sequence of such OptRanTriGen protocols, as long as the latest OptRanTriGen instance can output triples in time.

Algorithm 3 OptRanTriGen adapted from [12, 41, 70] ▷ Code of P_i

Input: System parameters
Output: Beaver_triples $= \{(\llbracket a_k\rrbracket, \llbracket b_k\rrbracket, \llbracket c_k\rrbracket)_{k=1}^{t+1}\}$
// 1. Generate random double sharings
1: uniformly sample a randomness s_i
2: call Share(s_i, t) and Share$(s_i, 2t)$, i.e., use t-degree and $2t$-degree plain Shamir secret sharing to share the same s_i
3: **wait** for all shares $(\llbracket s_j\rrbracket^i, \llbracket s_j\rrbracket^i_{2t})$ sent from all n parties $\{P_j\}$:
4: $(\llbracket r_1\rrbracket^i, \cdots, \llbracket r_n\rrbracket^i) \leftarrow M(\llbracket s_1\rrbracket^i, \cdots, \llbracket s_n\rrbracket^i)^T$
5: $(\llbracket r_1\rrbracket^i_{2t}, \cdots, \llbracket r_n\rrbracket^i_{2t}) \leftarrow M(\llbracket s_1\rrbracket^i_{2t}, \cdots, \llbracket s_n\rrbracket^i_{2t})^T$
6: **for** each $P_j \in \{P_{t+2}, \cdots, P_n\}$:
7: P_i exclusively sends $(\llbracket r_j\rrbracket^i, \llbracket r_j\rrbracket^i_{2t})$ to P_j
8: **if** $P_i \in \{P_{t+2}, \cdots, P_n\}$:
9: **wait** for $(\llbracket r_i\rrbracket^j, \llbracket r_i\rrbracket^j_{2t})$ sent from all n parties $\{P_j\}_{j\in[n]}$:
10: **verify** that both secret shares have the correct degree
11: **verify** that they can be decoded to get the same r_j
12: **abort** if any verification fails, **continue** otherwise
//The counterpart of line 12 in [12, 41, 70] reliably broadcasts a bit ("OK"/"ABORT") to cross-check the consistency of double sharing. Our fallback (Alg. 4) allows to remove the broadcasts.
13: rnd_dou_sha $\leftarrow \{(\llbracket r_1\rrbracket^i, \llbracket r_1\rrbracket^i_{2t}), \cdots, (\llbracket r_{t+1}\rrbracket^i, \llbracket r_{t+1}\rrbracket^i_{2t})\}$
// 2. Generate random sharings
14: repeat lines 1-12 twice without doing $2t$-degree sharing
15: if P_i does not abort in line 14, it obtains $2(t+1)$ random shares rnd_sha $\leftarrow \{(\llbracket a_1\rrbracket^i, \llbracket b_1\rrbracket^i), \cdots, (\llbracket a_{t+1}\rrbracket^i, \llbracket b_{t+1}\rrbracket^i)\}$
// 3. Perform degree reduction
16: **for** each $1 \le k \le t+1$: compute $\llbracket c_k\rrbracket^i_{2t} = \llbracket a_k\rrbracket^i \cdot \llbracket b_k\rrbracket^i$
17: invoke BatchRec$(\llbracket c_1\rrbracket^i_{2t} - \llbracket r_1\rrbracket^i_{2t}, \cdots, \llbracket c_{t+1}\rrbracket^i_{2t} - \llbracket r_{t+1}\rrbracket^i_{2t})$ and wait for it returns $(c_1 - r_1, \cdots, c_{t+1} - r_{t+1})$
18: **for** each $1 \le k \le t+1$: compute $\llbracket c_k\rrbracket^i = c_k - r_k + \llbracket r_k\rrbracket^i$
19: **return** Beaver_triples $= \{(\llbracket a_k\rrbracket^i, \llbracket b_k\rrbracket^i, \llbracket c_k\rrbracket^i)_{k=1}^{t+1}\}$

Final protocol of dual-mode triple generation. Clearly, the fast path built from OptRanTriGen cannot guarantee G.O.D. Considering the reconstruction of $2t$-degree shares in line 15 of Algorithm 3, it has to wait for all parties' shares to verify they are consistent w.r.t. a unique $2t$-degree polynomial. That means, if any party crashes, OptRanTriGen would stuck forever. Therefore, we introduce a fallback mechanism as described in Algorithm 4, which can securely restore robustness in the bad case that the fast path fails to progress in time or encounters inconsistent verification.

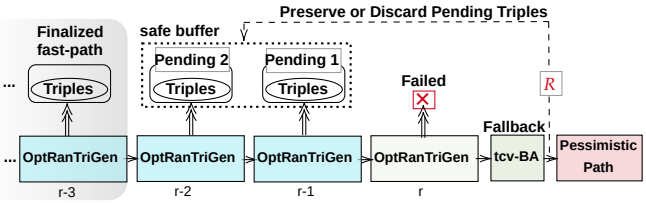


Figure 5: **Dual-mode triple generation with fallback**

The subtlety of fallback in an asynchronous network is that different honest parties might quit the fast path with different progresses. Figure 5 illustrates how our fallback mechanism resolves this threat. We first let each party maintain a safe buffer to temporarily withhold the outputs of the latest two OptRanTriGen from immediate output (as their immediate output might cause disagreement if unexpectedly encountering fallback). Given the safe buffer, once a party P_i aborts in the r-th OptRanTriGen if detecting misbehavior or timeout, it can just invoke a two-consecutive-value Byzantine agreement (tcv-BA) with input $r-1$ (i.e., the index of the latest completed OptRanTriGen). Then, tcv-BA outputs R, and P_i reacts accordingly, to discard or preserve triples in its safe buffer. Finally, the honest parties would have totally consistent fast-path output after tcv-BA completes, and they can activate the pessimistic path by running AsyRanTriGen to recover G.O.D.

Security analysis of dual-mode triple generation. The security of Algorithm 4 can be summarized as follows.

Theorem 3 (Validity, Secrecy, and Liveness of Dual-mode Triple Generation)**.** *Algorithm 4 securely realizes the desired properties of AMPC's offline phase of multiplication triple.*

Here we brief the intuitions of proving the above security theorem and defer details to Appendix G.3 for space limit:

- For validity, we first prove a lemma: if any honest party outputs in the r-th OptRanTriGen of fast path, then all honest parties already output triples up to the (r-1)-th OptRanTriGen. Then, for the validity and agreement of tcv-BA, all honest parties must obtain a common R during fallback, and R represents an OptRanTriGen instance where at least an honest party has outputted. Therefore, all honest parties can finalize consistent fast-path triples with the same progress according to R-1.
- For secrecy, it is straightforward as both fast and pessimistic paths have provable secrecy, and our sequential composition of them would inherit their secrecy.

- Liveness is immediate, as (i) everyone can quit from a failed fast path after timer expires, (ii) the termination of tcv-BA and AsyRanTriGen then ensure the pessimistic path to always progress even if in the worst case.

Algorithm 4 Dual-mode offline triple generation ▷ Code for P_i

Input: System parameters
Output: A sequence of multiplication triple shares

```
// Fast path
initialize a timer Δ that expires after τ
initialize safe buffer Pending_1 = ∅ and Pending_2 = ∅
for r ∈ {1,2,3,···}:
    reset timer Δ and activate an OptRanTriGen_r instance for r
    if OptRanTriGen_r returns Beaver_triples_r before Δ expires:
        if Pending_2 not empty: output Pending_2
        Pending_2 = Pending_1, Pending_1 = Beaver_triples_r
    else //If OptRanTriGen_r either aborts or doesn't output in time
        activate tcv-BA with input r − 1 and wait for it returns R
        //Preserve fast-path triples until the (R-1)-th OptRanTriGen
        if R = r − 2: discard Pending_2 and Pending_1
        if R = r − 1: output Pending_2 and discard Pending_1
        if R = r: output Pending_2 and Pending_1
        break
// Pessimistic path
while true: run an AsyRanTriGen instance and output its result
```

Complexities. Our fast path prepares triples with amortized $O(n)$ communication, in the good case of a synchronous network without actual corruptions. In the worst case, our pessimistic path attains an amortized $O(n^2)$ per-triple overhead. Both paths achieve concrete improvements as well: (i) the fast path reduces the number of communication rounds by 3 compared to hbMPC; (ii) the pessimistic path reduces the number of shared secrets by a factor of 3 compared to GS23.

7 Towards Efficient AMPC-as-a-Service

Realizing Dumbo-MPC **as robust AMPCaaS.** Given our robust offline protocols, we can directly instantiate the robust offline phase of Dumbo-MPC using them. While for the online phase, it can mostly inherit the already performant and robust design from hbMPC, which consists of three main sub-phases: (i) function ordering, (ii) input solicitation and (iii) function evaluation. Here "function ordering" enables AMPC servers reach an agreement on the function to evaluate, "input solicitation" lets AMPC servers gather the private inputs from a sufficient number of clients (which usually is realized by a special variant of asynchronous Byzantine agreement known as asynchronous common subset, and can also be dedicatedly optimized as discussed in Appendix E), and finally, "function evaluation" lets AMPC servers privately evaluate the function on the solicited private inputs.

Notably, each above online sub-phase can be realized via black-box invocation of standard techniques, and they are neither the bottleneck of efficiency nor the obstacles to robustness, so our primary focus is the design and implementation of a more efficient robust offline phase throughout the paper. For those reasons, we wouldn't repeat the online phase design here, and refer readers interested in its details to Appendix E.

Optimizing consensus component. Dumbo-MPC heavily relies on MVBA—a specific asynchronous Byzantine agreement protocol, to overcome network asynchrony by explicitly reaching consensus in both the online and offline phases. For efficiency of this critical component, we put forth the notion of optimistically terminable asynchronous MVBA (otMVBA) and give its generic construction. Our otMVBA construction is an MVBA that can deterministically and responsively terminate in 5 asynchronous rounds, under optimistic conditions like the network is synchronous and a certain party is honest.

As opposed, earlier MVBAs [4, 47, 59, 72] terminate in about 7 rounds after at least one invocation of randomized coin flipping, even if in the best conditions.[9] Our core idea to realize optimistic termination is forbidding the arbitrary choice of output during the pessimistic case, by enforcing a non-leader party to either (i) obtain a threshold signature attesting that no honest party would ever output in the good case, or (ii) wait for another mutually exclusive threshold signature fixing the good-case output. Readers interested in otMVBA can find sufficient details in Appendix H.

8 Implementation and Evaluations

We implemented Dumbo-MPC, especially its offline phase.[10] Triple generation protocol of GS23 [57] is also implemented for fair comparison. We finally evaluate and experimentally compare Dumbo-MPC, hbMPC [70] and GS23, in different network settings with varying system scales and different fault numbers. Note that we do not evaluate the incomparable protocols like DXKR23 [43] for generating double random sharing, as their pre-processed $2t$-degree secret shares could fail in reconstruction during the online phase, compromising either online robustness or optimal $n/3$ resilience.

8.1 Implementation and Experiment Setup

Implementation. All evaluated protocols are written in the same language of Python 3 as forks of the open-source implementations of hbACSS [84] and hbMPC [70]. The p2p channels utilize unauthenticated TCP sockets, and concurrent tasks are managed by Python's asyncio library. We use pairing friendly elliptic curve BLS12-381 for Dumbo-MPC, atop which Boldyreva's BLS threshold signature [19] and KZG polynomial commitment [66] are implemented. For GS23, we use secp256k1 curve to implement Pedersen commitments. All elliptic curve implementations are from gnark-crypto [23]. We choose a 256-bit prime field for Shamir secret sharing.

[9] Ditto [52] once presented 2-chain VABA and claimed it as an MVBA with 5-round optimistic latency, but its recentest version withdraws the result.

[10] Note that the online phase is not efficiency bottleneck, and it can be mostly forked from hbMPC [70] with adaptions to its consensus components.

Experiment setup. We evaluate Dumbo-MPC and GS23 [57] using AWS EC2 c6a.8xlarge instances installing Ubuntu 20.04 LTS and equipped with 32 vCPUs and 64 GB memory.[11] We run tests with varying scales for n = 4, 10, 22, and 31 parties in the same AWS region at Virginia, which reflects a LAN deployment setting. As an affordable and reproducible WAN benchmarking approach, we utilize Linux TC tool to restrict the upload bandwidth of each instance to 500 Mbps and set the network's round-trip time (RTT) to 150 ms, followed by re-running all tests in this simulated WAN setting.

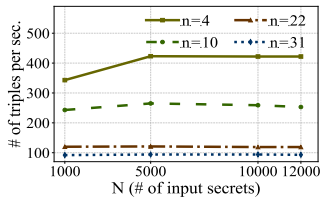


Figure 6: **Throughput v.s. batch in AsyRanTriGen.**

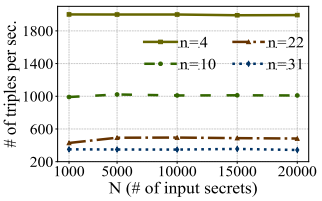


Figure 7: **Throughput v.s. batch in OptRanTriGen.**

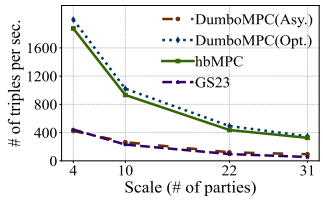


Figure 8: **Throughput v.s. scale in the LAN setting.**

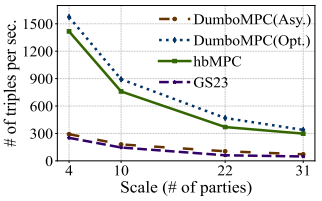


Figure 9: **Throughput v.s. scale in the WAN setting.**

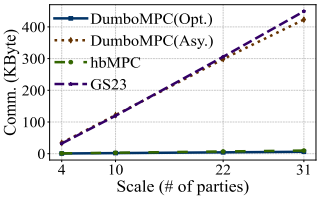


Figure 10: **Communication cost per node per triple.**

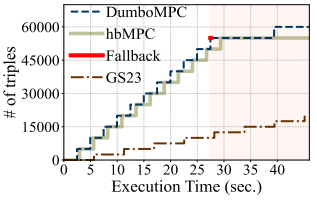


Figure 11: **Execution with fallback for $n = 4$ (in LAN).**

8.2 Evaluation results

Choices of batch size. Dumbo-MPC has a critical parameter of batch size, which specifies the number of random secrets taken as input in each triple generation protocol. Clearly, the throughput of triples is closely related to the choice of batch size: A larger batch size might render a higher throughput, as a result of amortizing the fixed overheads, but an unnecessarily large batch might cause dramatic increment of latency. We evaluate Dumbo-MPC under varying batch sizes (1000, 5000, 10000, 15000 and 20000) in the LAN setting at different scales of n= 4, 10, 22 and 31. Figures 6 and 7 plot the trade-off between throughput and batch size in AsyRanTriGen and OptRanTriGen, respectively. Clearly, while the batch size increases, throughput starts to grow rapidly but soon becomes stable. In almost all scales, the throughput reaches a plateau after the batch size ≥ 5000, thus leading us to choose a fixed batch size of 5000 throughout all later experiments.

Triple throughput. Figures 8 and 9 plot the throughput performance of Dumbo-MPC compared with GS23 and hbMPC, in the LAN and WAN settings, respectively. Though throughputs of all evaluated protocols decrease while the system scale is larger, AsyRanTriGen of Dumbo-MPC outperforms GS23 if $n \geq 10$ in LAN, and is always superior to GS23 in WAN. In particular, when $n = 31$, AsyRanTriGen realizes a throughput of 94 triples/sec in LAN, 2X that of GS23. For our OptRanTriGen protocol, it can optimistically generate 349 triples/sec when $n = 31$ in the LAN setting, resulting in a throughput 6X of GS23 and about 10% larger than hbMPC. Moreover, restricting bandwidth to 500 Mbps and RTT to 150 ms only causes marginal decrement in triple throughput. When $n = 31$ in WAN, OptRanTriGen (resp. AsyRanTriGen) generates 339 (resp. 72) triples/sec, 7X (resp. 2X) that of GS23.

Communication cost. Figure 10 reports the amortized per-node communication for each triple. As expected, OptRanTriGen and hbMPC have the least communication (i.e., constant per-node overhead about 10 KB for each triple). GS23 exhibits the worst asymptotic behavior and is concretely worse than AsyRanTriGen, since GS23 needs to share $3n$ secret per triple but AsyRanTriGen only shares n secrets per triple.

Performance with fallback. To understand the performance of the evaluated protocols in the bad case, we examine an execution that begins with optimistic conditions and later enters a bad case due to a malicious party. As Figure 11 plots, although both Dumbo-MPC and hbMPC can outperform GS23 during the good case, hbMPC grinds to a halt when a party misbehaves (as shown by the red region in Figure 11). Clearly, Dumbo-MPC achieves the best of both AsyRanTriGen and OptRanTriGen, harvesting efficiency in good case and simultaneously preserving robustness in all situations.

Additional tests. For space limit, we defer more evaluation results like the impact of crash nodes and the pre-processing latency of specific tasks (auction and shuffling) to Appendix I.

9 Conclusion

We design Dumbo-MPC—a set of concretely efficient AMPC protocols in the classic offline-online paradigm, to circumvent the severe robustness-efficiency trade-off of existing AMPC protocols like GS23 and hbMPC. Dumbo-MPC is also the first implemented AMPC with all-phase G.O.D., featuring a novel dual-mode robust offline phase dedicated optimizations for each crucial sub-protocols. Extensive experiments showcase its promising performance in various applications.

[11] Note that we choose the high-profile EC2 instance to accommodate large memory requirement instead of achieving parallelization speed-up, cf. our experiments conducted on different EC2 instances (Appendix I.2), which reveals there is no performance gain through parallelization over extra CPUs.

Acknowledgment

We sincerely thank Hanwen Feng for many fruitful discussions about the concurrent (un)composability of some state-of-the-art batch asynchronous complete secret sharing protocol.

References

[1] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Perfect asynchronous mpc with linear communication overhead. In *Proc. EUROCRYPT 2024*, pages 280–309, 2024.

[2] Ittai Abraham, Gilad Asharov, Arpita Patra, and Gilad Stern. Perfectly secure asynchronous agreement on a core set in constant expected time. *Cryptology ePrint Archive*, 2023.

[3] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation. pages 39–70, 2023.

[4] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proc. PODC 2019*, pages 337–346, 2019.

[5] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder–scalable, robust anonymous committed broadcast. In *Proc. CCS 2020*, pages 1233–1252, 2020.

[6] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Asynchronous verifiable information dispersal with near-optimal communication. *Cryptology ePrint Archive*, 2022.

[7] Gilad Asharov and Yehuda Lindell. A full proof of the bgw protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, 2017.

[8] Assi Barak, Martin Hirt, Lior Koskas, and Yehuda Lindell. An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants. In *Proc. CCS 2018*, pages 695–712, 2018.

[9] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Proc. CRYPTO 1991*, pages 420–432, 1992.

[10] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513, 1990.

[11] Zuzana Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In *Proc. TCC 2006*, pages 305–328, 2006.

[12] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In *Proc. TCC 2008*, pages 213–230, 2008.

[13] M Ben-Or, Avi Wigderson, and S Goldwasser. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proc. STOC 1988*, pages 1–10, 1988.

[14] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proc. STOC 1993*, pages 52–61, 1993.

[15] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience. In *Proc. PODC 1994*, pages 183–192, 1994.

[16] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proc. PODC 1994*, pages 183–192, 1994.

[17] Fabrice Benhamouda, Shai Halevi, and Tzipora Halevi. Supporting private data on hyperledger fabric with secure multiparty computation. *IBM Journal of Research and Development*, 63(2/3):3–1, 2019.

[18] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. SPRINT: High-throughput robust distributed schnorr signatures. In *Advances in Cryptology – EUROCRYPT 2024*, pages 62–91, 2024.

[19] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Proc. PKC 2003*, pages 31–46, 2002.

[20] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *Proc. CRYPTO 2018*, pages 565–596, 2018.

[21] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *Proc. CRYPTO 2018*, pages 565–596, 2018.

[22] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proc. TCC 2005*, pages 325–341, 2005.

[23] Gautam Botrel, Thomas Piellard, Youssef El Housni, Arya Tabaie, Gus Gutoski, and Ivo Kubjas. Consensys/gnark-crypto: v0.11.2, 2023.

[24] Gabriel Bracha and Sam Toueg. Resilient consensus protocols. In *Proc. PODC 1983*, pages 12–26, 1983.

[25] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual cryptology conference*, pages 868–886, 2012.

[26] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325, 2012.

[27] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Proc. CRYPTO 2001*, pages 524–541, 2001.

[28] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 2005.

[29] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *Proc. SRDS 2005)*, pages 191–201, 2005.

[30] Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang. On communication-efficient asynchronous mpc with adaptive security. In *Proc. TCC 2021*, pages 35–65, 2021.

[31] Ashish Choudhury. Optimally-resilient unconditionally-secure asynchronous multi-party computation revisited. Cryptology ePrint Archive, Paper 2020/906, 2020.

[32] Ashish Choudhury, Martin Hirt, and Arpita Patra. Asynchronous multiparty computation with linear communication complexity. In Yehuda Afek, editor, *Proc. DISC 2013*, pages 388–402, 2013.

[33] Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous mpc with linear communication complexity. In *Proc. ICDCN 2015*, 2015.

[34] Ashish Choudhury and Arpita Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Transactions on Information Theory*, 63(1):428–468, 2017.

[35] Ashish Choudhury and Arpita Patra. On the communication efficiency of statistically secure asynchronous mpc with optimal resilience. *Journal of Cryptology*, 36(2):13, 2023.

[36] Ran Cohen. Asynchronous secure multiparty computation in constant time. In *Proc. PKC 2016*, pages 183–207. 2016.

[37] Sandro Coretti, Juan Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In *Proc. ASIACRYPT 2016*, pages 998–1021, 2016.

[38] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Proc. PKC 2009*, pages 160–179, 2009.

[39] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In *Proc. CRYPTO 2008*, pages 241–261, 2008.

[40] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure mpc for dishonest majority – or: Breaking the spdz limits. In *Proc. ESORICS 2013*, pages 1–18, 2013.

[41] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Proc. CRYPTO 2007*, pages 572–590, 2007.

[42] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Proc. CRYPTO 2012*, pages 643–662, 2012.

[43] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *Proc. USENIX Security 23*, pages 5359–5376, 2023.

[44] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *Proc. CCS 2021*, pages 2705–2721, 2021.

[45] Sourav Das, Zhuolun Xiang, and Ling Ren. Powers of tau in asynchrony, 2024.

[46] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *Proc. SP*, pages 2518–2534, 2022.

[47] Sisi Duan, Xin Wang, and Haibin Zhang. Fin: Practical signature-free asynchronous common subset in constant time. In *Proc. CCS 2023*, pages 815–829, 2023.

[48] Sisi Duan, Xin Wang, and Haibin Zhang. Practical signature-free asynchronous common subset in constant time. In *Proc. CCS 2023*, 2023.

[49] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[50] Adi Fiat, Amosand Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proc. CRYPTO 1987*, pages 186–194, 1987.

[51] Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 135–155, 1987.

[52] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *Proc. FC 2022*, pages 296–315. Springer, 2022.

[53] Rosario Gennaro, Michael O Rabin, and Tal Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proc. PODC 1998*, pages 101–111, 1998.

[54] Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional mpc with guaranteed output delivery. In *Proc. CRYPTO 2019*, pages 85–114, 2019.

[55] Vipul Goyal, Chen-Da Liu-Zhang, and Yifan Song. Towards achieving asynchronous mpc with linear communication and optimal resilience. In *Proc. CRYPTO 2024*, pages 170–206, 2024.

[56] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority mpc. In *Proc. CRYPTO 2020*, pages 618–646, 2020.

[57] Jens Groth and Victor Shoup. Design and analysis of a distributed ecdsa signing service. Cryptology ePrint Archive, Paper 2022/506, 2022.

[58] Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. In *Proc. EUROCRYPT 2024*, pages 370–400, 2024.

[59] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Speeding dumbo: Pushing asynchronous bft closer to practice. 2022.

[60] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous bft protocols. In *Proc. CCS 2020*, pages 803–818, 2020.

[61] James Hendricks, Gregory R Ganger, and Michael K Reiter. Verifying distributed erasure-coded data. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 139–146, 2007.

[62] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In *Proc. CRYPTO 2006*, pages 463–482, 2006.

[63] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience. In *Proc. EUROCRYPT 2005*, pages 322–340, 2005.

[64] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Asynchronous multi-party computation with quadratic communication. In *Automata, Languages and Programming*, pages 473–485, 2008.

[65] Sanket Kanjalkar, Ye Zhang, Shreyas Gandlur, and Andrew Miller. Publicly auditable mpc-as-a-service with succinct verification and universal setup. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 386–411, 2021.

[66] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Proc. ASIACRYPT 2010*, pages 177–194, 2010.

[67] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proc. CCS 2020*, pages 1575–1590, 2020.

[68] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proc. CCS 2016*, pages 830–842, 2016.

[69] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: making spdz great again. In *Proc. EUROCRYPT 2018*, pages 158–189, 2018.

[70] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *Proc. CCS 2019*, pages 887–903, 2019.

[71] Yuan Lu, Zhenliang Lu, and Qiang Tang. Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined bft. In *Proc. CCS 2022*, pages 2159–2173, 2022.

[72] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proc. PODC 2020*, pages 129–138, 2020.

[73] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *Proc. SP 2021*, pages 1348–1366, 2021.

[74] Fabio Massacci, Chan Nam Ngo, Jing Nie, Daniele Venturi, and Julian Williams. Futuresmex: secure, distributed futures market exchange. In *Proc. SP 18*, pages 335–353, 2018.

[75] Ueli Maurer. Unifying zero-knowledge proofs of knowledge. In *International Conference on Cryptology in Africa*, pages 272–286. Springer, 2009.

[76] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proc. CCS 2016*, pages 31–42, 2016.

[77] Christian Mouchet, Bertrand Elliott, and Hubaux Jean Pierre. An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption. *Journal of Cryptology*, 36(10):1–20, 2023.

[78] Arpita Patra, Ashish Choudhury, and C Pandu Rangan. Efficient asynchronous verifiable secret sharing and multiparty computation. *Journal of Cryptology*, 28:49–109, 2015.

[79] Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous multiparty computation with optimal resilience. Cryptology ePrint Archive, Paper 2008/425, 2008.

[80] B Prabhu, K Srinathan, and C Pandu Rangan. Asynchronous unconditionally secure computation: An efficiency improvement. In *Proc. INDOCRYPT 2002*, pages 93–107, 2002.

[81] Marc Rivinius, Pascal Reisert, Daniel Rausch, and Ralf Küsters. Publicly accountable robust multi-party computation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2430–2449, 2022.

[82] Victor Shoup and Nigel P Smart. Lightweight asynchronous verifiable secret sharing with optimal resilience. *Journal of Cryptology*, 37(3):27, 2024.

[83] K Srinathan and C Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In *Proc. INDOCRYPT 2000*, pages 117–129, 2000.

[84] Yurek Thomas, Luo Licheng, Fairoze Jaiden, Kate Aniket, and Miller Andrew. hbacss: How to robustly share many secrets. In *Proc. NDSS 2022*, pages 1187–1201, 2022.

[85] Antoine Urban and Matthieu Rambaud. Robust multiparty computation from threshold encryption based on rlwe. *Cryptology ePrint Archive*, 2024.

[86] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, 1961.

[87] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *Proc. SP 18*, pages 926–943, 2018.

[88] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *Proc. CCS 2017*, pages 39–56, 2017.

[89] Thomas Yurek, Zhuolun Xiang, Yu Xia, and Andrew Miller. Long live the honey badger: Robust asynchronous {DPSS} and its applications. In *Proc. USENIX Security 23*, pages 5413–5430, 2023.

[90] Haibin Zhang and Sisi Duan. Pace: Fully parallelizable bft from reproposable byzantine agreement. In *Proc. CCS 2022*, pages 3151–3164, 2022.

[91] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.

A More Discussions on Related Studies and Alternative Design Choices

A.1 Related studies

MPC has attracted a great amount of attentions since its formulation in 1980's, and it has been a subject involving a huge line of studies. Here we go over a few relevant results and discuss their limitations in the context of realizing more efficient fully asynchronous MPC.

MPC protocols based on additive secret sharing, such as SPDZ [40] and EMP [88] and its descendants [67], can tolerate dishonest majority with resilience $t < n$. However, they would be aborted if there is any corrupted party, and thus might fail to realize guaranteed output delivery. The lack of G.O.D. (or more general, the loss of fairness) is inherent for any dishonest majority MPC protocols without extra setups.

Robust MPC in the synchronous setting with honest majority has been extensively studied [12, 51, 54, 56, 62], and $O(n)$ per-gate communication complexity has been achieved in the setting, but their robustness and even privacy might be violated in an asynchronous network. In particular, dispute control [11], that is restarting the computation after detecting and eliminating faulty parties, is used to handle faulty parties

for $t < n/3$ [12, 41] and $t < n/2$ [41, 56, 62] to construct secure MPC protocols in synchronous network. However, such protocols depending on the ability to time out nodes are vulnerable to unresponsive attack [70]. As such, if t honest nodes are temporarily isolated from the network due to asynchrony, the honest nodes would be removed from the system, and the remaining malicious parties may jeopardize the privacy of (clients') private inputs. HyperMPC [8] is designed to adapt to low bandwidth MPC based on [12] in the $n/3$ setting, and fails to guarantee output as it works with $2t$-shares both in the offline and online phase. In the offline phase, random double sharing of security with abort is needed to generate double shares (e.g., degree-t shares and a degree-$2t$ shares of some random secrets). During online phase, parties have to reconstruct $2t$-shares, which is infeasible to provide guaranteed output delivery in $n/3$ asynchronous setting.

In the asynchronous setting, MPC protocols usually employ asynchronous agreement protocols (e.g., asynchronous common subset [16, 47, 60, 76, 90]) to evaluate the function with only the $n-t$ fastest parties' inputs. The topic was initially studied in the setting of computationally-unbounded adversaries [14, 15, 32, 34, 78, 80, 83]. Perfectly secure asynchronous MPC without any error probability recently was also designed for $(t < n/4)$ with linear per-gate communication overhead [1]. Unconditionally secure asynchronous MPC protocols can also be designed to realize optimal $t < n/3$ tolerance (in the presence of negligible small error probability). In this setting, the first design [16] incurs $O(n^{11})$ per-gate communication complexity, and later was improved to $O(n^5)$ [79] and $O(n^4)$ [31, 35]. Very recently, Goyal et al. [55] proposed the first unconditionally secure AMPC with linear per-gate communication and optimal $t < n/3$ resilience, but unfortunately, it has a prohibitive $O(n^{14})$ circuit-independent communication overhead (which might be not surprisingly as the stringent setting of unconditional information-theory setting brings extra design overhead).

For computationally secure AMPC using cryptography, several protocols are proposed based on additive/somewhat threshold homomorphic encryption [30, 33, 64], where arithmetic operations are performed on ciphertexts in the online phase. In [33], threshold somewhat homomorphic encryption (e.g., BGN cryptosystem [22]) is used as a primitive to construct encrypted MPC protocol, which might take expected $\tilde{O}(2^{\sqrt{w}})$ time to decrypt ciphertext using Pollard's lambda method (where w is the bit-length of secret share field), likely causing its implementation only suitable for Boolean circuits (or arithmetic circuits defined over very small fields). hbMPC [70] does not require any homomorphic encryption, and it is proposed to guarantee output in the online phase but only provides security with abort in the offline phase (since its offline requires to reconstruct $2t$-shares).

A recent work of Das et al. [43] presents an efficient asynchronous random double sharing (ARDS) protocol, but the result alone fails to robustly reconstruct $2t$-shares while evaluating a multiplication gate in the online phase for $t < n/3$. Very recently, a couple of studies [57, 82] achieve guaranteed output delivery in asynchrony, but with cubic per-gate communication overhead in the worst case.

Other theoretic asynchronous MPC protocols feature a constant-round online phase independent of the circuit depth, making use of different underlying cryptographic primitives, such as FHE [36] and pseudo-random generator [37]. But they either exhibit a large communication overhead that is quadratic in n and linear in circuit size (to privately compute a distributed version of garbled circuit through another AMPC protocol like ours) [37], or/and suffer from large online computational cost like expensive FHE self-ootstrappings that are linear to the circuit's multiplicative depth [36].

A.2 Efficiency issues of alternative designs

Noticeably, there indeed exist a few alternative design choices of asynchronous triple generation with asymptotic complexities that are similar to or even better than ours, but their concrete performance could be problematic. Here we briefly explain the reasons of their inferior performance.

Possible instantiations of Goyal et al. [55]. The AMPC protocol of Goyal et al. [55] achieves a communication cost of $O(|C|n + Dn^2 + n^6\kappa + n^7)$ plus $O(n^2)$ invocations of ACSS to share $O(|C|)$ degree-t Shamir sharings, where $|C|$ is circuit size and D is circuit depth. Though Goyal et al. [55] originally focus on the information theoretic setting, we nevertheless can instantiate the protocol by more efficient computationally-secure components, using the best-possible (computationally-secure) ACSS [3, 84] and coin flipping [28]. However, for concrete efficiency, the protocol of Goyal et al. [55] needs to verifiably share hundreds or thousands of secrets for each triple, if using their suggested security parameters. Though we require n secrets to be shared for each triple (which is asymptotically worse), our approach is still concretely more efficient for typical system scales like several dozens of parties.

Possible instantiations of Choudhury et al. [35]. Noticeably, if we employ the state-of-the-art computationally-secure ACSS, such as hbACSS and Bingo, to instantiate the recent (statistically-secure) AMPC framework from Choudhury et al. [35], an AMPC with quadratic per-gate communication overhead can also be realized, but its concrete efficiency is significantly worse than our design, as it requires $6n$ secrets to be shared for generating each triple, which is six times larger than ours and two times larger than GS23. Shoup and Smart [82] also observe that a happy path can be embedded to the AMPC framework of Choudhury et al. [35] by (optimistically) waiting for messages sent from more than $n-f$ parties, but this happy path has an execution flow same to the original fully asynchronous protocol, which involves too many redundant communication rounds to circumvent asynchrony, making it concretely less efficient than our dedicated fast path particularly tailored for the good case.

AMPC from tSHE/tFHE. In the Shamir-based tFHE/tSHE scheme, smudging noise could be amplified by Lagrange coefficients during reconstruction, resulting in a modulus increase on the order of $O(n!^3)$ [20]. To mitigate this modulus growth, binary coefficients have been utilized in constructing tFHE/tSHE schemes, but such approach incurs significant space overhead, as the size of each secret key share grows to at least $O(n^{4.2})$. Some recent works [77, 81] employ a thresholdizer that uses additively shared secret keys to construct tFHE/tSHE with further using Shamir secret sharing to reshare each additive key share, but this thresholdizer essentially requires all nodes to remain online for perform decryption, otherwise when some party is not responsive due to trigger some timeout, it would invoke the reconstruction of Shamir secret sharing to publicly open this lost additive share, making it clearly unsuitable for asynchronous settings (because an asynchronous adversary can force to recover all honest parties' additive key shares).

B PCwHE: Definition and Security Proofs

B.1 Definition of PCwHE

For presentation simplicity, Section 4 refrains from giving a formal definition of polynomial commitment with hidden evaluation (PCwHE), and only describes a concrete construction of such augmented polynomial commitment atop the KZG scheme. Here for rigorousness, we formally define the notion of polynomial commitment with "hidden evaluation" below.

Definition 1. *A polynomial commitment with hidden evaluation is a tuple consisting of the following algorithms:*

- $\mathsf{Setup}(1^\kappa, t) \to SP$: given a security parameter κ and an upper bound t restricting the degree of any polynomial to be committed, it generates system parameter SP.
- $\mathsf{PolyCom}(SP, \phi(\cdot), \hat{\phi}(\cdot)) \to C$: it computes a commitment C for a given polynomial $\phi(\cdot)$ with a random polynomial $\hat{\phi}(\cdot)$ using system parameter SP. $\mathsf{PolyCom}(SP, \phi(\cdot)) \to (C, \hat{\phi}(\cdot))$ also denotes it with a random $\hat{\phi}(\cdot)$ as auxiliary output.
- $\mathsf{ProveEval}(SP, i, \phi(\cdot), \hat{\phi}(\cdot)) \to (i, \phi(i), \hat{\phi}(i), w_i)$: given a point i and polynomials $\phi(\cdot)$ and $\hat{\phi}(\cdot)$, it returns polynomial evaluations $\phi(i)$ and $\hat{\phi}(i)$ as well as an evaluation proof w_i.
- $\mathsf{VerifyEval}(SP, C, i, \phi(i), \hat{\phi}(i), w_i) \to \{0,1\}$: it checks whether $\phi(i)$ and $\hat{\phi}(i)$ are correct evaluations of the polynomials committed to C at the point i. If the verification succeeds, it outputs 1, otherwise it returns 0.
- $\mathsf{HiddenEval}(SP, i, \phi(\cdot), \hat{\phi}(\cdot)) \to (i, T_i, \omega_i)$: given a point i and polynomials $\phi(\cdot)$ and $\hat{\phi}(\cdot)$, it outputs a hidden evaluation T_i (which can be analog to a commitment of $\phi(i)$) and a hidden evaluation's proof ω_i. Sometimes, we require another form of $\mathsf{HiddenEval}'(SP, i, w_i, \phi(i), \hat{\phi}(i)) \to (i, T_i, \omega_i)$, which takes only evaluations and corresponding proof (instead of the whole polynomials) as input but can compute the same output.
- $\mathsf{OpenHiddenEval}(SP, \phi(i), \hat{\phi}(i), T_i) \to \{0,1\}$: this is analog to the opening of commitment, which verifies whether $\phi(i)$ and $\hat{\phi}(i)$ are committed behind T_i, and returns 0 (reject) or 1 (accept).
- $\mathsf{VerifyHiddenEval}(SP, C, i, T_i, \omega_i) \to \{0,1\}$: given a polynomial commitment C, a hidden evaluation T_i, and a hidden evaluation proof ω_i, it checks whether T_i commits the evaluation of the polynomial committed to C at the point i, and returns a bit of verification result (0 as rejected and 1 as accepted).

Adapting from [66], we require that a polynomial commitment scheme (with hidden evaluation interfaces) shall satisfy the properties of *correctness*, *polynomial binding*, *evaluation binding*, *hiding* and *hidden evaluation's binding*, with overwhelming probability. We wound not repeat them here as they have been already formalized in Section 4.

B.2 Security proofs of the PCwHE construction from augmented KZG

Theorem 4. *Assuming the hardness of the discrete logarithm problem and t-strong Diffie-Hellman (t-SDH) problem over the given bilinear group, the augmented KZG scheme presented in Figure 2 securely realizes a polynomial commitment scheme with hidden evaluation, i.e., ensures correctness, polynomial binding, evaluation binding, hiding and hidden evaluation's binding with overwhelming probability.*

Proof. Here we prove the properties of our augmented KZG polynomial commitment with hidden evaluation below.

Correctness. This is trivial, as evaluations and hidden evaluations can always pass verification if they and their associating proofs are correctly computed.

Evaluation binding. Suppose that there exists an adversary $\mathcal{A}$ that breaks the evaluation binding property of commitment C, and computes two tuples $(i, \phi(i), \hat{\phi}(i), w_i)$ and $(i, \phi(i)', \hat{\phi}(i)', w_i')$ for index i that are both verified by $\mathsf{VerifyEval}$ with non-negligible probability. Then, an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ as a sub-routine can be constructed to break the t-SDH assumption with non-negligible probability. The detailed proof follows that of Theorem 3.3 in the original KZG paper [66], and we refrain from repeating it here.

Polynomial binding. Suppose that for the honestly generated system parameter SP, the adversary $\mathcal{A}$ outputs the tuple $(C, I_1, I_2, \{(\phi(i), \hat{\phi}(i), w_i)\}_{i \in I_1 \cup I_2})$, where $|I_1| = |I_2| = t+1$ and for $\forall i \in I_1 \cup I_2$, $(i, \phi(i), \hat{\phi}(i), w_i)$ is accepted by $\mathsf{VerifyEval}$ regarding the commitment C with non-negligible probability. Then consider the following two cases:

- $I_1 \cap I_2 \neq \emptyset$: This indicates that there exists at least one index $i \in I_1$ and $j \in I_2$ where $i = j$ such that the two tuples

$(i,\phi(i),\hat{\phi}(i),w_i)$ and $(j,\phi(j),\hat{\phi}(j),w_j)$ are successfully verified by VerifyEval. Following from the proof of evaluation binding, we can construct an algorithm $\mathcal{B}$ to break the t-SDH assumption with non-negligible probability.

- $I_1 \cap I_2 = \emptyset$: The adversary $\mathcal{A}$ outputs two set of shares $(I_1, \{\phi(i),\hat{\phi}(i)\}_{i\in I_1})$ and $(I_2, \{\phi(i),\hat{\phi}(i)\}_{i\in I_2})$ that are verified by VerifyEval for the same commitment C where the index set $I_1 \cap I_2 = \emptyset$ and $|I_1| = |I_2| = t+1$. Then, an algorithm $\mathcal{B}$ can interpolate two polynomials $\phi_1(\cdot)$ and $\phi_2(\cdot)$ where $\phi_1(\cdot) \neq \phi_2(\cdot)$. There are two polynomials $\phi_1(\cdot) \neq \phi_2(\cdot)$ that are committed to same commitment C. By the proof of Theorem 3.3 in [66], we can construct an algorithm $\mathcal{B}$ to break the DL assumption with non-negligible probability.

Therefore, the probability to break the polynomial binding property is equal to break the t-SDH assumption and DL assumption, which are negligible.

Hidden evaluation binding. Suppose that for honestly generated system parameter SP, the adversary $\mathcal{A}$ outputs the tuple $(C,(i,\phi(i),\hat{\phi}(i),T_i,w_i),(i,\phi(i)',\hat{\phi}(i)',T_i',w_i'))$ where $\phi(i) \neq \phi(i)'$ and $\hat{\phi}(i) \neq \hat{\phi}(i)'$, which can be both verified by VerifyHiddenEval and OpenHiddenEval. Consider the following two cases:

- $T_i \neq T_i'$: this implies that two accepted tuples $(i,\phi(i),\hat{\phi}(i),w_i)$ and $(i,\phi(i)',\hat{\phi}(i)',w_i')$ are produced for same index i. However, this violates the evaluation binding property.
- $T_i = T_i'$: this indicates that $T_i = g^{\phi(i)}h^{\hat{\phi}(i)} = T_i' = g^{\phi(i)'}h^{\hat{\phi}(i)'}$, which breaks the binding property of Pedersen commitment - this holds as long as computing discrete logarithms is intractable in $\mathbb{G}$.

Therefore, the probability to break the hidden evaluation binding property is equal to break the evaluation binding of KZG commitment and the binding property of Pedersen commitment, which are negligible.

Hiding. We will prove that $\mathcal{A}$'s view is identically distributed in the real protocol execution and the ideal world through a sequence of games down below.

- **Game 0**: This corresponds to the real-world execution.
- **Game 1**: Same as Game 0 except that the secret key α and τ is uniformly sampled from known $\mathbb{Z}_q$. Game 1 is indistinguishable from Game 0 as the distribution of SP is identical in both games.
- **Game 2**: This corresponds to the simulated world. The difference between Game 1 and Game 2 is that the commitment, witness, and hidden evaluation are generated by Sim_1, Sim_2 and Sim_3 of Figure 12. It is obvious that all witnesses and hidden evaluations can pass the verification given the commitment. Consider the worst case that $|I_1| = t$ and $|I_2| = n$, the outputs of the simulator are $(C, \{i,\phi'(i),\hat{\phi}'(i),w_i\}_{i\in I_1}, \{i,T_i,\omega_i\}_{i\in I_2})$ where $C = g^c$ is uniformly distributed in $\mathbb{G}$, $T_i = g^{\phi'(i)}h^{\hat{\phi}'(i)}$ and $\omega_i = w_i = g^{\frac{\phi'(\alpha)-\phi'(i)}{\alpha-i}}h^{\frac{\hat{\phi}'(\alpha)-\hat{\phi}'(i)}{\alpha-i}}$. In the real world, there must exist a hiding polynomial $\hat{\phi}(\cdot) = (\phi'(\cdot)-\phi(\cdot))\cdot\tau^{-1}+\hat{\phi}'(\cdot)$ such that the output is completely identical to that in Game 2. Thus, Game 2 is identically distributed as Game 1.

Hiding Simulator for KZG with Hidden Evaluation

1. $\mathsf{Sim}_0(1^\kappa,t)$: choose random trapdoors (α,τ) and run $\mathsf{KZG.Setup}(1^\kappa,t)$ using the trapdoors, then output (SP,st) where $st = (\alpha,\tau)$.
2. $\mathsf{Sim}_1(st)$: return $C \leftarrow g^c$ where c is uniformly sampled from $\mathbb{Z}_q$.
3. $\mathsf{Sim}_2(SP,(C,c),\{i,\phi(i)\}_{i\in I_1})$:
   (1) set $Q \leftarrow I_1$ and $Q' \leftarrow \emptyset$. Choose $i \in [n]/I_1, \phi(i) \leftarrow \mathbb{Z}_q$ and set $Q \leftarrow Q \cup \{i\}$ until $|Q| > t$.
   (2) interpolate $\phi'(\cdot)$ from $\{(i,\phi(i))_{i\in Q}\}$.
   (3) compute $\hat{\phi}'(\alpha) \leftarrow \frac{c-\phi'(\alpha)}{\tau}$.
   (4) set $Q' \leftarrow \{\alpha\}$. Choose $k \in [n], \hat{\phi}'(k) \leftarrow \mathbb{Z}_q$ and set $Q' \leftarrow Q' \cup \{k\}$ until $|Q'| = t+1$.
   (5) interpolate $\hat{\phi}'(\cdot)$ from $\{(k,\hat{\phi}'(k))_{k\in Q'}\}$.
   (6) for $\forall i \in I_1$, compute $w_i \leftarrow g^{\frac{\phi'(\alpha)-\phi'(i)}{\alpha-i}}h^{\frac{\hat{\phi}'(\alpha)-\hat{\phi}'(i)}{\alpha-i}}$.
   (7) return $(\{w_i\}_{i\in I_1}, \phi'(\cdot), \hat{\phi}'(\cdot))$.
4. $\mathsf{Sim}_3(SP,\phi'(\cdot),\hat{\phi}'(\cdot),I_2)$:
   (1) for $\forall i \in I_2$, compute $T_i \leftarrow g^{\phi'(i)}h^{\hat{\phi}'(i)}$, and $\omega_i \leftarrow g^{\frac{\phi'(\alpha)-\phi'(i)}{\alpha-i}}h^{\frac{\hat{\phi}'(\alpha)-\hat{\phi}'(i)}{\alpha-i}}$.
   (2) return $(\{T_i,\omega_i\}_{i\in I_2})$.

Figure 12: **Hiding simulator** $(\mathsf{Sim}_0,\mathsf{Sim}_1,\mathsf{Sim}_2,\mathsf{Sim}_3)$ **for KZG polynomial commitment w/ hidden evaluation**.

C Batch ACSS

C.1 Definition

For completeness, we formally define batch ACSS with additively homomorphic commitment (BACSS) down below.

Definition 2 (BACSS)**.** *Syntactically, a (t,n) BACSS protocol sharing B secrets consists of the next two sub-protocols:*

- Share. The dealer inputs a batch of secrets $(s_1,\cdots,s_N)$ and samples N t-degree polynomials to share these secrets to all parties. For every P_i, it can obtain $\{(C_k,[\![s_k]\!]^i,[\![\hat{s}_k]\!]^i,w_k^i)_{k\in[N]}\}$, namely, the i-th shares of all secrets, and all corresponding polynomial commitments and evaluation proofs. Here shares $([\![s_k]\!]^i,[\![\hat{s}_k]\!]^i)$ can be verified due to polynomial commitment C_k and evaluation proof w_k^i, such that $[\![s_k]\!]^i$ is indeed the evaluation of polynomial committed to C_k at the point $x = i$.
- Rec. Each party P_i takes $\{(C_k,[\![s_k]\!]^i,[\![\hat{s}_k]\!]^i,w_k^i)_{k\in[N]}\}$ the output of Share phase as input, such that the honest parties can selectively recover any k-th secret s_k.

BACSS satisfies the basic properties of ACSS: correctness, termination and secrecy, which are defined as follows.

- **Correctness**. If the dealer is honest, then for every $k \in [N]$, each honest party eventually outputs a share $\phi_k(i)$ where $\phi_k(\cdot)$ is a random polynomial with $\phi_k(0) = s_k$.
- **Completeness**. If any honest party receives output, then for every $k \in [N]$, there exists a unique degree-t polynomial $\tilde{\phi}_k(\cdot)$ such that each honest party P_i eventually outputs $\tilde{\phi}_k(i)$. Moreover, if the dealer is honest, $\tilde{s}_k = s_k$.
- **Secrecy**. If the dealer is honest, then a computationally bounded adversary learns no information about $\phi_k(\cdot)$ except for the shares of corrupted parties for every $k \in [N]$, except with negligible probability.

Additionally, we require BACSS to satisfy the next property of additive homomorphism (over multiple sessions):

- **Additive Homomorphism**. For each honest party P_i, if it obtains $\{(C_k, [\![s_k]\!]^i, [\![\hat{s}_k]\!]^i, w_k^i)_{k=1}^{N'}\}$ from a few BACSS protocols (i.e., multiple sessions), then for any s that is a linear combination of $(s_1, \cdots, s_{N'})$, P_i can locally and deterministically compute its exclusive share $[\![s]\!]^i$ of s through $\{[\![s_k]\!]^i\}_{k=1}^{N'}$, a polynomial commitment C binding some polynomial $\phi(\cdot)$ with $\phi(0) = s$ through $\{C_k\}_{k=1}^{N'}$, and also an evaluation proof w_i attesting that $[\![s]\!]^i$ is an evaluation of $\phi(\cdot)$ at the point $x = i$.

C.2 Instantiation with patch for concurrent compatibility

We instantiate a concurrently composable BACSS on top of hbACSS [84] using KZG polynomial commitment. In the instantiation, we patch the original hbACSS in the following way: (i) a public key encryption with IND-CCA security (instead of IND-CPA) is used for encrypting secret shares; (ii) Each party P_i selects n public keys $\{pk_{i1}, \cdots, pk_{in}\}$, where pk_{ij} is exclusively used in a BACSS protocol if it has a designated dealer P_j. The detailed protocol flow mostly inherits [84] and is presented in Algorithm 5.

Note that in the above description, (Disperse, Retrieve) represent the tuple of sub-protocols consisting of (batch) asynchronous verifiable information dispersal (AVID) [6, 29, 61]. Disperse enables a dealer verifiably disperse a batch of messages (e.g. n messages) to the whole network, and Retrieve enables all parties in the network to help a certain party P_j to recover some specific dispersed messages (e.g. the j-th dispersed message). Since the construction of such batch AVID is rather standard [84], we refrain from repeating them.

Algorithm 5 BACSS_Share$_i(N)$ (from [84] with our patch for concurrent composability) ▷ Code for P_i

Setups (that are already established before the protocol executes):

each P_i samples n private keys $\{sk_{ij}\}_{j\in[n]}$ for IND-CCA secure PKE and computes the corresponding public keys $pk_{ij} = g^{sk_{ij}}$;
publish the set of public keys $\{pk_{i1}, \cdots, pk_{in}\}_{i\in[n]}$ for each P_i;
publish the common reference string SP of KZG commitment.

Dealer P_l:

if $i = l$, dealer P_l waits for input secrets $(s_1, \cdots, s_N)$ and runs:

sample N random degree-t polynomial $\phi_1(\cdot), \cdots, \phi_N(\cdot)$ such that for $k \in [N]$, each $\phi_k(0) = s_k$, where s_k is randomly sampled from $\mathbb{Z}_q$
$\mathbf{C} = \{C_1, \cdots, C_N\} \leftarrow \{\mathsf{PolyCom}(SP, \phi_k(\cdot))\}_{k\in[N]}$ ▷ compute a batch of polynomial commitments
ReliableBroadcast($\mathbf{C}$)
for $j = 1$ to n ▷ Disperse the shares and witnesses
  for $k = 1$ to N
    $\{\phi_k(j), \hat{\phi}_k(j), w_k^j\} \leftarrow \mathsf{ProveEval}(SP, j, \phi_k(\cdot), \hat{\phi}_k(\cdot))$
  $z_j \leftarrow \mathsf{Enc}_{pk_{jl}}(\{\phi_k(j), \hat{\phi}_k(j), w_k^j\}_{k\in[N]})$
Disperse($\{z_j\}_{j\in[n]}$)

Every party:

// Wait for broadcasts
wait to receive $\mathbf{C} = \{C_1, \cdots, C_N\} \leftarrow$ ReliableBroadcast
wait for Disperse to complete
// Decrypt and validate
$z_i \leftarrow \mathsf{Retrieve}(i)$
$\{\phi_k(i), \hat{\phi}_k(i), w_k^i\}_{k\in[N]} \leftarrow Dec_{sk_{il}}(z_i)$
if $\mathsf{BatchVerifyEval}(SP, \mathbf{C}, \{\phi_k(i), \hat{\phi}_k(i), w_k^i\}_{k\in[N]}) \neq 1$ or decryption fails
  sendall $(Implicate, sk_{il})$
otherwise, valid shares $\{(C_k, \phi_k(i), \hat{\phi}_k(i), w_k^i)\}_{k\in[N]}$ are owned, so sendall OK
// Bracha-style agreement
on receiving OK from $2t+1$ parties,
  sendall READY
on receiving READY from $t+1$ parties and do not send READY,
  sendall READY
wait to receive READY from $2t+1$ parties,
  if all owner shares are valid **then**
    output shares $\{(C_k, \phi_k(i), \hat{\phi}_k(i), w_k^i)\}_{k\in[N]}$
// Handling Implication
on receiving $(Implicate, sk_{jl})$ from some P_j,
  ignore if already in Share Recovery or $pk_{jl} \neq g^{sk_{jl}}$
  $z_j \leftarrow \mathsf{Retrieve}(j)$
  if $\mathsf{BatchVerifyEval}(SP, \mathbf{C}, j, \{\phi_k(j), \hat{\phi}_k(j), w_k^j\}_{k\in[N]}) \neq 1$ or decryption fails
    proceed to Share Recovery below

// Share Recovery
P_i has already received commitment $\{C_k\}_{k=1}^N$
for each set of $t+1$ secrets in N
  if $i \leq t+1$: $C_i' \leftarrow C_i$
  else
    $C_i' \leftarrow \prod_{k=1}^{t+1} C_k^{\lambda_k}$, where λ_k is the value of k-th Lagrange interpolating polynomial at zero-point
  if previously receiving valid shares
    interpolate $\psi_i(\cdot)$ from $\{(k, \phi_k(i))\}_{k\in[t+1]}$

interpolate $\hat{\psi}_i(\cdot)$ from $\{(k, \hat{\phi}_k(i))\}_{k\in[t+1]}$
for $j = 1$ to n
if $j \le t+1$: $w_i^j \leftarrow w_j^i$
else $w_i^j \leftarrow \prod_{k=1}^{t+1} (w_k^i)^{\lambda_k}$, where λ_k is the value of k-th Lagrange interpolating polynomial at zero point
send $(R_1, \psi_i(j), \hat{\psi}_i(j), w_i^j)$ to P_j
// R_1 phase
set $\mathcal{R}_{1i} \leftarrow \emptyset$
upon receiving $(R_1, \psi_j(i), \hat{\psi}_j(i), w_j^i)$ from P_j
if VerifyEval$(C_i', j, \psi_j(i), \hat{\psi}_j(i), w_j^i) = 1$
$\mathcal{R}_{1i} \leftarrow \mathcal{R}_{1i} \bigcup \{j\}$
if $|\mathcal{R}_{1i}| \ge t+1$
interpolate $\varphi_i(\cdot)$ from $\{(k, \psi_k(i))\}_{k\in\mathcal{R}_{1i}}$
interpolate $\hat{\varphi}_i(\cdot)$ from $\{(k, \hat{\psi}_k(i))\}_{k\in\mathcal{R}_{1i}}$
for each P_j
send $(R_2, \varphi_i(j), \hat{\varphi}_i(j))$ to P_j
// R_2 phase
set $\mathcal{R}_{2i} \leftarrow \emptyset$
upon receiving $(R_2, \varphi_j(i), \hat{\varphi}_j(i))$ from P_j
$\mathcal{R}_{2i} \leftarrow \mathcal{R}_{2i} \bigcup \{j\}$
if $|\mathcal{R}_{2i}| \ge 2t+1$
robustly interpolate $\psi_i(\cdot)$ from $\{(k, \varphi_k(i))\}_{k\in\mathcal{R}_{2i}}$
robustly interpolate $\hat{\psi}_i(\cdot)$ from $\{(k, \hat{\varphi}_k(i))\}_{k\in\mathcal{R}_{2i}}$
for $k = 1$ to $t+1$
$(k, \phi_k(i), \hat{\phi}_k(i), w_k^i) \leftarrow$ ProveEval$(SP, i, \psi_i(\cdot), \hat{\psi}_i(\cdot))$ ▷ $\phi_k(i) = \psi_i(k)$ and $\hat{\phi}_k(i) = \hat{\psi}_i(k)$
output shares $\{C_k, \phi_i(k), \hat{\phi}_i(k), w_k^i\}_{k\in[t+1]}$

C.3 Secrecy vulnerability without our patch of concurrent compatibility

If one carelessly instantiates BACSS directly using the original hbACSS [84] without our patch of concurrent compatibility, there could be a serious threat of privacy when simultaneously executing multiple such BACSS protocols. The core idea of this attack is twofold: (i) a malicious party can obtain all messages from an honest dealer with non-negligible probability, and (ii) the malicious party disperses these messages within its own BACSS, raising a valid implication to recover the secrets of the honest dealer. Here we describe the concrete attack of secrecy, for two concurrent BACSS protocols (BACSS_h and BACSS_m), where BACSS_h and BACSS_m are executed by $n = 7$ parties with $t = 2$ malicious participants (P_m, P_k), BACSS_h has an honest dealer, and BACSS_m has a malicious dealer. The attack is performed as follows:

1. *Waiting for up to t messages from the honest dealer.* In BACSS_h, the honest dealer P_h computes secret shares for all parties and encrypts shares using the public key of party P_j, resulting in ciphertexts z_j of size $2\lambda + l$ bits (assuming the ciphertext space is λ bits). Through specific error-correcting coding (e.g., polynomial interpolation), z_j is encoded as $(z_{j_0}, z_{j_1}, \cdots, z_{j_6})$, where z_{j_m} and z_{j_k} represent the first 2λ bits of the ciphertext z_j. The dealer then sends z_{j_i} to party P_i for each $i \in [6]$.

   Noticeably, the adversary can at most receive t messages from the honest dealer P_h. W.o.l.g., say the adversary learns z_{j_0} and z_{j_1} for each $j \in [6]$.
2. *Guess the ciphertext of shares from honest dealer with non-negligible probability.* Since the adversary learns z_{j_0} and z_{j_1} for each $j \in [6]$, it can attempt to guess the remaining l bits of the ciphertext with a probability of $\frac{1}{2^l}$. Here l can be relatively small depending on the chunking parameter, and w.l.o.g., let us say l is a single bit. Thus, the adversary can recover all ciphertext fragments $(z_{j_0}, z_{j_1}, \cdots, z_{j_6})$ for each $j \in [6]$ with high probability. E.g., when $l = 1$, the probability of correctly obtaining all 7 shares' ciphertext is $(1/2)^5$, which is clearly non-negligible. [12]

   Note that the fundamental reason why the adversary can guess the ciphertext of secret shares is straightforward, because erasure coding itself would ensure any confidentiality, and the adversary can likely recover the original message by exploiting its own t encode fragments.
3. *Paste messages from honest dealer into the malicious BACSS instance.* In BACSS_m, the adversary instructs the malicious dealer P_m to do:
   (a) Copy all polynomial commitments and evaluation proofs from BACSS_h, and paste them into the corresponding messages of BACSS_m;
   (b) Encode the guessed ciphertext $(z_0, z_1, \cdots, z_6)$ from the honest dealer, and paste mostly of the correct encoding fragments into the corresponding messages of BACSS_m, except for that z_k (i.e. the ciphertext that is supposed to be encrypted by another malicious party's public key) is replaced by a random string of bits.
4. *Force the secrets to be opened by "implication" in the malicious BACSS instance*. In BACSS_m, the malicious party P_k can send an implication message to all honest parties. The honest parties can validate the implication message (as it indeed receives a random string instead of the genuine ciphertext, thus the decrypted shares are invalid), and thus will engage in the Share recovery procedure to recover all the secrets of the honest dealer P_h.

This completes the attack, and with non-negligible probability, the adversary steals the honest dealer's secrets.

D Zk-Proof of Product Relationship over a Triple of Pedersen Commitments

Recall that we reduce the key problem of proving product relation over KZG polynomial commitments to the problem of proving product relationship over Pedersen commitments, which is folklore and concretely very efficient [75, 87].

[12] If the protocol is set up with an authentication channel but not a private channel, malicious parties can intercept all messages from the honest dealer.

For completeness, we repeat this standard proof-of-product-relationship scheme over a triple of Pedersen commitments in Figure 13. Note that we describe its non-interactive variant due to Fiat-Shamir heuristic [50], in the random oracle model of cryptographic hash function $\mathcal{H}$.

Theorem 5. *The proof of product scheme over Pedersen commitments (described in Figure 13) satisfies the following properties in the random oracle model:*

- *Completeness: if $\mathcal{P}$ is honest, then $\mathcal{V}$ will accept its proof with probability 1.*
- *Proof-of-knowledge: for any PPT adversary $\mathcal{P}^*$, there exists a PPT extractor* Ext *interacting with $\mathcal{P}^*$, such that if $\mathcal{V}$ accepts the proof returned by $\mathcal{P}^*$ with non-negligible probability, then* Ext *can extract witness satisfying statement with non-negligible probability.*
- *Zero-knowledge: for any PPT adversary $\mathcal{V}^*$, there exists a simulator $\mathcal{S}$, such that on input any valid statement (g,h,T_a,T_b,T_c) with satisfying witness, the distribution of $\mathcal{S}$'s output is computationally indistinguishable from that of $\mathcal{P}$'s output.*

Proof of Product over Pedersen Commitments

Both the prover $\mathcal{P}$ and verifier $\mathcal{V}$ know the public statement (g,h,T_a,T_b,T_c). $\mathcal{P}$ also knows private witness $a,\hat{a},b,\hat{b},c,\hat{c}$ such that $T_a = g^a h^{\hat{a}}$, $T_b = g^b h^{\hat{b}}$ and $T_c = g^c h^{\hat{c}}$.

// Prover

- $\mathbf{PoK}_{\mathbf{Prod\_Ped}}.\mathcal{P}((g,h,T_a,T_b,T_c),(a,\hat{a},b,\hat{b},c,\hat{c}))$:

(a) randomly sample $e_1,\cdots,e_5 \in \mathbb{Z}_p$;
(b) compute $\beta = g^{e_1}h^{e_2}, \gamma = g^{e_3}h^{e_4}, \delta = T_a{}^{e_3}h^{e_5}$;
(c) compute $x = \mathcal{H}(g||h||T_a||T_b||T_c||\beta||\gamma||\delta)$;
(d) compute that

$$z_1 = e_1 + xa,\ z_2 = e_2 + x\hat{a},$$
$$z_3 = e_3 + xb,\ z_4 = e_4 + x\hat{b},$$
$$z_5 = e_5 + x(\hat{c} - \hat{a}b);$$

(e) let $\pi_1 = (\beta,\gamma,\delta)$ and $\pi_2 = (z_1,\cdots,z_5)$;
(f) output $\pi = (\pi_1,\pi_2)$.

// Verifier

- $\mathbf{PoK}_{\mathbf{Prod\_Ped}}.\mathcal{V}((g,h,T_a,T_b,T_c),\pi)$:

(a) parse π as $((\beta,\gamma,\delta),(z_1,\cdots,z_5))$;
(b) compute $x = \mathcal{H}(g||h||T_a||T_b||T_c||\beta||\gamma||\delta)$;
(c) check that

$$\beta\cdot(T_a)^x = g^{z_1}h^{z_2},\ \gamma\cdot(T_b)^x = g^{z_3}h^{z_4},\ \delta\cdot(T_c)^x = (T_a)^{z_3}h^{z_5}$$

(d) if all verifications pass, return 1, otherwise return 0.

Figure 13: **Non-interactive variant of the folklore proof of product relation over Pedersen commitments [75, 87].**

E Online Phase Protocols

This appendix presents the deferred description of online phase, which mostly inherits the already-performant online phase of hbMPC. While executing the online phase, the MPCaaS servers first reach a consensus on the description of functions to evaluate. Then, an input soliciting process is initiated to determine a consensus on a subset of client inputs, utilizing asynchronous consensus primitives. Finally, all servers evaluate the function using random shares and multiplication triples that are pre-prepared in the offline phase. In greater detail, these steps can be instantiated as follows.

Step 1. Order the functions to evaluate. All clients first submit the functions to be computed to a "function" pool (which can be analog to the transaction pool or mempool in BFT consensus). The MPCaaS servers can execute some asynchronous atomic broadcast (i.e., BFT consensus) to agree on the sequence of functions to evaluate, as shown in Figure 14. W.l.o.g., the agreed sequence functions to be evaluated can be $\{f_2, f_1, \cdots, f_k\}$.

Step 2. Soliciting (masked) inputs from clients. Each designated client firstly queries the shares of a certain randomness that is already pre-shared among the MPCaaS servers, and thus recovers a randomness mask r that can be used to hide its actual input m. The mask input $\hat{m}$ would be $r+m$.

Then, an input soliciting process executes to let MPCaaS servers reach a consensus on selecting a subset of $k-c$ different clients' masked inputs, utilizing an asynchronous consensus common subset (ACS) protocol. Namely, each client would provide its masked input to ACS, and the output of ACS would return a set of at least $k-c$ available (masked) inputs to MPCaaS servers for their online function evaluation.

Once ACS is completed, each MPCaaS server can compute the secret share $[\![m]\!]$ of each client's input by computing $\hat{m} - [\![r]\!]$, where $[\![r]\!]$ is the share of the pre-processing masking randomness held by the MPCaaS server.

Step 3. Online evaluation. All servers engage in a standard robust online phase of secure MPC to evaluate the function, utilizing random shares and Beaver triples to ensure confidentiality and integrity.

- Addition gate. The server can apply the linear function on the inputs of the gate to obtain the output.
- Multiplication gate. The server can process $t+1$ multiplication gates simultaneously and execute the BatchRec algorithm [70] to robustly evaluate the $t+1$ multiplication gates. Concretely, let $([\![x_1]\!],[\![y_1]\!]),\cdots,([\![x_{t+1}]\!],[\![y_{t+1}]\!])$ be the inputs to these $t+1$ multiplication gates and $([\![a_1]\!],[\![b_1]\!],[\![c_1]\!]),\cdots,([\![a_{t+1}]\!],[\![b_{t+1}]\!],[\![c_{t+1}]\!])$ be the associated Beaver triples. Inputting $([\![x_1-a_1]\!],\cdots,[\![x_{t+1}-a_{t+1}]\!])$ to BatchRec, servers can efficiently and robustly open $(x_1-a_1,\cdots,x_{t+1}-a_{t+1})$ and similarly obtain $(b_1-y_1,\cdots,b_{t+1}-y_{t+1})$[13]. Then each server can locally computes $[\![x_iy_i]\!] = (x_i-a_i)[\![y_i]\!] + (y_i-b_i)[\![x_i]\!] - (x_i-a_i)(y_i-b_i) + [\![c_i]\!]$ for each $i \in [t+1]$.

[13] To improve efficiency of batch reconstruct, each server can maintain a local blocklist to identify dishonest servers sending incorrect shares, thereby disregarding shares from servers on the blocklist and minimizing tentative interpolation. The efficiency of adding blocklist is shown in Appendix I.1.

After finishing the circuit evaluation, servers can obtain the secret shared outputs, and reconstruct the final output by collecting enough secret shared outputs.

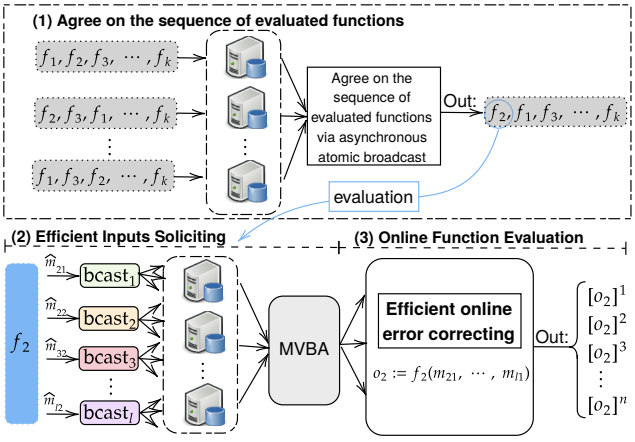


Figure 14: **Illustration of AMPC's online phase.**

Optimal asynchronous common subset (ACS) for soliciting clients' inputs. Although the input solicitation step of the online phase can be built from any black-box ACS to achieve consensus, the ACS instantiation in hbMPC is sub-optimal as it incurs expected $O(\log n)$ rounds and $O(\kappa n^3)$ communication cost. We instead adapt the state-of-the-art ACS [27, 72] for input solicitation, making the circuit-independent part of online cost to be expected $O(1)$ rounds and $O(\kappa n^2)$ bits. [14]

Particularly, we adopt an asymptotically optimal instantiation of CKPS01 ACS [27] using the communication-efficient MVBA extension protocol [72], which proceeds as: (i) Each client initially multicasts an "encrypted" input (that is the input masked by a randomness shared among AMPC servers) and a digital signature to all servers; (ii) Subsequently, each AMPC server waits for a sufficient number of "encrypted" inputs signed by distinct clients; (iii) Then, each AMPC server uses the vector of input-signature pairs from the clients to invoke a communication-efficient MVBA protocol, and the external validity of MVBA is set to check if there is a sufficient number of input ciphertexts signed by distinct clients; (iv) Finally, MVBA returns a common vector of input-signature pairs to every AMPC server, and the private inputs carried by this vector would be used by all servers as inputs for function evaluations.

F Plain Shamir Secret Sharing

Though we mainly adopt asynchronous *verifiable* secret sharing in our pessimistic offline path for realizing robust and efficient triple generation, we still widely use the plain Shamir secret sharing (adapted for asynchrony) in the optimistic offline path and the online phase (to avoid extra cryptographic overheads). For reference of the readers, this appendix presents these standard pertinent algorithms below.

Share. This simply shares a secret s with a reconstruction threshold $t+1$ (i.e., t-degree sharing). Specifically, a degree-t polynomial $\phi(\cdot)$ is sampled such that $\phi(0)=s$, and each share $[\![s]\!]_t^j = \phi(j)$ is the evaluation of the polynomial $\phi(\cdot)$ at point i. The following Algorithm 6 presents $\mathsf{Share}(s,t)$, which allows an honest dealer to correctly share a secret s using degree-t polynomial across a network consisting of n parties.

Algorithm 6 $\mathsf{Share}(s,t)$ ▷ Code for P_i

1: P_i samples a random degree-t polynomial $\phi(\cdot)$ with $\phi(0)=s$
2: **for** $j=1$ to n
3: send $[\![s]\!]^j = \phi(j)$ to party P_j

Interpolation in asynchrony. Algorithm 7 outlines the standard approach for robustly decoding t-degree sharing in an asynchronous setting $(t<n/3)$ [12, 70]. Initially, a party attempts to interpolate a degree-t polynomial $\phi(\cdot)$ upon receiving any $t+1$ shares. If the resulting $\phi(\cdot)$ matches the first $2t+1$ received shares, it is deemed correct. In case of failure in the optimistic scenario, a party must await additional shares to attempt error correction. In the worst case, with t incorrect shares received, $3t+1$ total shares are required for correcting t errors and determining a degree-t polynomial coinciding with all $2t+1$ honest shares.

Algorithm 7 $\mathsf{Robust\text{-}Interpolate}(l,[\![s]\!])$ [12, 70] ▷ Code for P_l

1: send share $[\![s]\!]^i$ to P_l
2: **if** $i=l$
3: **wait** for receiving sufficient shares of s: $\{[\![s]\!]^j\}_{j\in[n]}$ up to t erasures ($[\![s]\!]^j \in \mathbb{Z}_q \bigcup \{\perp\}$)
4: case of t erasures:
5: interpolate a polynomial ϕ from any $t+1$ points
6: **if** ϕ coincides with all $2t+1$ point
7: **return** ϕ
8: **else**
9: **return** $\perp$
10: case of $t-e$ erasures:
11: run RSDecode decoding to correct up to e errors and **return** ϕ

Batch reconstruction. Algorithm 8 describes the amortized batch public reconstruction (BatchRec) of t-sharings in the $(t<n/3)$ setting [12, 41, 70]. The approach involves using a Vandermonde matrix M to expand the shared secrets $[\![s_1]\!],\cdots,[\![s_{t+1}]\!]$ into a set of shares $[\![y_1]\!],\cdots,[\![y_n]\!]$. In the first round, each party P_i invokes $\mathsf{Robust\text{-}Interpolate}(i,[\![y_i]\!])$ to reconstruct a share y_i. In the second round, each party P_i utilizes $\mathsf{Robust\text{-}Interpolate}(i,y_i)$ to reconstruct a polynomial encoding $s_1,\cdots,s_{t+1}$.

Algorithm 8 $\mathsf{BatchRec}([\![s_1]\!],\cdots,[\![s_{t+1}]\!])$ [70]

[14] Note that we assume the number of clients is n and their input is κ-bit.

1: let M be the $(n, t+1)$ Vandermonde matrix $M_{ij} = \eta_i^j$ evaluating a degree-t polynomial at $(\eta_1, \cdots, \eta_n)$
2: compute $([\![y_1]\!], \cdots, [\![y_n]\!])^T := M([\![s_1]\!], \cdots, [\![s_{t+1}]\!])$
3: invoke Robust-Interpolate$(i, [\![y_i]\!])$ to decode y_i
4: invoke Robust-Interpolate(i, y_i) to decode $s_1, \cdots, s_{t+1}$
5: **return** $s_1, \cdots, s_{t+1}$

G Security Proofs of Offline Protocols

G.1 Proof of AsyRanShGen

Below is the deferred security proof of AsyRanShGen. We will prove its properties of termination, consistency, secrecy and randomness, one by one, as follows.

Lemma 1 (Termination)**.** *All honest parties would eventually output a set rnd_shares*$=\{(C_k, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i)\}_{k=1}^{N(t+1)}$.

Proof. In the AsyRanShGen protocol, the underlying primitives are BACSS protocol and MVBA protocol. According to termination property of BACSS protocol, the BACSS protocol will eventually terminate and output shares (along with corresponding commitments and evaluation proofs) $\{(C_{jk}, [\![s_{jk}]\!]^i, [\![\hat{s}_{jk}]\!]^i, w_{jk}^i)_{k=1}^N\}$ for each honest party. For the MVBA protocol, the termination property ensures that every honest party will terminate with set $\mathcal{T}$. By the deterministic random extraction algorithm, each honest party can locally compute the set rnd_shares $= \{(C_k, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i)\}_{k=1}^{N(t+1)}$. Therefore, the AsyRanShGen protocol will terminate. □

Lemma 2 (Consistency)**.** *For all honest parties, their output rnd_shares sets are "consistent". Here "consistent" means: (i) their* $\{C_k\}_{k=1}^{N(t+1)}$ *are same; (ii) for each* $(C_k, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i) \in$ *rnd_shares*, VerifyEval$(SP, C_k, i, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i) = 1$.

Proof. The termination property of MVBA protocol enables all honest parties to agree on the output $\mathcal{T}$ ($\mathcal{T} \geq n-t$), in which all BACSS instance is terminated at all honest parties. That is, after completing MVBA protocol every honest party P_i has obtain shares $\{(C_{jk}, [\![s_{jk}]\!]^i, [\![\hat{s}_{jk}]\!]^i, w_{jk}^i)_{k=1}^N\}$ from party P_j ($j \in \mathcal{T}$). According to the correctness property of BACSS protocol and binding property of KZG commitment scheme, any subset of $t+1$ shares $[\![s_{jk}]\!]$ lies on the polynomial $\phi_{jk}(\cdot)$ committed to C_{jk} and unique identity $\phi_{jk}(0) = s_{jk}$. Then, every honest party P_i can compute random shares $\{(C_k, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i)_{k=1}^{N(t+1)}\}$ by Vandermonde matrix determined by MVBA's output $\mathcal{T}$. The linear combination in random extraction phase and the additive homomorphic property of BACSS protocol ensure that every honest party P_i outputs the correct shares $\{(C_k, [\![r_k]\!]^i, [\![\hat{r}_k]\!]^i, w_k^i)_{k=1}^{N(t+1)}\}$, where any subset of $t+1$ shares $[\![r_k]\!]$ lie on the polynomial $\phi_k(\cdot)$ that committed to C_k and has unique identity $\phi_k(0) = r_k$. □

Lemma 3 (Randomness and Secrecy)**.** *For each output* $(C_k, [\![r_k]\!], [\![\hat{r}_k]\!], w_k) \in$ *rnd_shares, the adversary cannot predicate* r_k *better than guessing over* $\mathbb{Z}_q$.

Proof. In the following, we will prove AsyRanShGen protocol satisfies randomness and secrecy.

Randomness. In the AsyRanShGen protocol, each party obtains the shares $\{[\![s_j]\!]^i\}_{j\in\mathcal{T}}$ from party P_j with secret s_j. At least $t+1$ secrets are uniformly sampled from $\mathbb{Z}_q$ (w.l.o.g., $\{s_1, \cdots, s_{t+1}\}$), and at most t secrets are sampled from an arbitrary distribution independent of $\mathbb{Z}_q$ (w.l.o.g., $\{s_{t+2}, \cdots, s_{2t+1}\}$). Then, $(r_1, \cdots, r_{t+1}) = M(s_1, \cdots, s_{2t+1}) = M_{[t+1]}(s_1, \cdots, s_{t+1}) + M_{[t+1,2t+1]}(s_{t+2}, \cdots, s_{2t+1})$ where $M_{[t+1]} = (M_{ij})_{i\in[t+1]}^{j\in[t+1]}$ is invertible by the definition of hyper-matrix, and $M_{[t+1,2t+1]} = (M_{ij})_{i\in[t+1]}^{j\in[t+2,2t+1]}$. The vector $(s_1, \cdots, s_{t+1})$ is uniformly random in $\mathbb{Z}_q^{t+1}$, thus $M_{[t+1]}(s_1, \cdots, s_{t+1}) + M_{[t+1,2t+1]}(s_{t+2}, s_{2t+1})$ is uniformly random in $\mathbb{Z}_q^{t+1}$. Since $\{s_{t+2}, \cdots, s_{2t+1}\}$ are sampled from an arbitrary distribution independent of $\mathbb{Z}_q$, it follows that $M_{[t+1]}(s_1, \cdots, s_{t+1}) + M_{[t+1,2t+1]}(s_{t+2}, \cdots, s_{2t+1})$ are uniformly random in $\mathbb{Z}_q^{t+1}$, i.e., $r_1, \cdots, r_{t+1}$ are uniformly random in $\mathbb{Z}_q^{t+1}$.

Secrecy. The following games are designed to prove the secrecy of AsyRanShGen protocol. Let $\mathcal{H}$ denote the set of honest parties, and $\mathcal{C}$ denote the set of corrupted parties.

Game 0. The adversary $\mathcal{A}$ that has corrupted t parties interacts with the remaining honest nodes to run the AsyRanShGen protocol, mirroring the real-world execution of the AsyRanShGen protocol.

Game 1. The simulator randomly samples α, τ from $\mathbb{Z}_q$, and publishes $\{g, h = g^\tau, (g^\alpha, \cdots, g^{\alpha^t}), (h^\alpha, \cdots, h^{\alpha^t})\}$ as common reference string. Then, the simulator participates in the BACSS_share protocol and the MVBA protocol on behalf of all honest parties belonging to $\mathcal{H}$. Note that for those BACSS_share$_j$ $\forall j \in \mathcal{H}$, the simulator would pick random polynomials $\phi_j(\cdot), \hat{\phi}_j(\cdot)$ as input to BACSS_share$_j$. Also note that each corrupted party P_i ($i \in \mathcal{C}$) would receive $\{(C_j, [\![s_j]\!]^i, [\![\hat{s}_j]\!]^i, w_j^i)\}_{j\in\mathcal{T}}$, where $[\![s_j]\!]^i = \phi_j(i), [\![\hat{s}_j]\!]^i = \hat{\phi}_j(i)$.

The distributions of **Game 1** and **Game 0** are trivially identical, as the simulator just generates common reference strings and runs other sub-protocols on behalf of honest parties in the exactly same way without any additional change.

Game 2. Instead of running BACSS_share$_j$, the simulator simulates BACSS_share$_j$ scripts sent to the corrupted parties with only t, n, α, τ and shares $\{[\![s_j]\!]^i\}_{j\in\mathcal{H}, i\in\mathcal{C}}$. Such simulation can be done by the following steps:

- The simulator randomly picks $\phi'_j(0) \in \mathbb{Z}_q$ for $\forall j \in \mathcal{H}$, and uses $\phi'_j(0)$ and shares $\{[\![s_j]\!]^i\}_{i\in\mathcal{C}}$ to interpolate a polynomial $\phi'_j(\cdot)$. Then, the simulator uses the trapdoor τ to compute $\hat{\phi}'_j(i)) = (\phi_j(i) - \phi'_j(i))\tau^{-1} + \hat{\phi}_j(i))$, and sends $(C_j, \phi'_j(i), \hat{\phi}'_j(i), w_j^i)$ to corrupted party P_i ($i \in \mathcal{C}$).
- The simulator replaces all encrypted shares sent to the honest party by the ciphertext of encrypting $\{0^*\}$ string with equal length of plain-text. In the different sessions,

distinct encryption keys of intended receivers are used to encrypt plaintext. Thus, the adversary learns nothing about the ciphertexts sent to the honest parties.

- In the Share Recovery phase, the adversary can initiate the implication of a specific party and require honest parties to publish the received message of the implicated party (i.e., access the decryption oracle of the implicated party). This falls into the following two cases:

  1) if the implicated party is an honest party, the simulator can call the IND-CCA simulator to respond with $\perp$ indicating decryption failure.

  2) if the implicated party is a corrupted party, the simulator can use the IND-CCA simulator to construct valid decryption of ciphertext that received from the implicated party on behalf of honest parties.

The difference between **Game 2** and **Game 1** arises from the manner in which shares are generated, and the distribution of shares received from the implicated party. The distribution of commitment, hide randomness, and witness is identical to the distribution in **Game 1**. The unconditional hiding property of the KZG commitment scheme reveals no additional information about the honest shares. Different encryption keys are used in each dealing session to encrypt messages, preventing the adversary from learning the shares of honest parties in the implication. The IND-CCA security of the encryption scheme guarantees that the distribution of shares of the implicated party is identical to the distribution in **Game 1**. Thus, the distribution of **Game 2** is the same as **Game 1**.

Then, the adversary $\mathcal{A}$ can obtain shares $\{[\![s_j]\!]^i\}_{j\in\mathcal{H},i\in\mathcal{C}}$, $\{s_i\}_{i\in\mathcal{C}}$ and random shares $\{[\![r_k]\!]^i\}_{k\in[t+1]}$, and construct the following equation system (w.l.o.g., assume $\mathcal{H} = \{P_1,\cdots,P_{t+1}\}$ and $\mathcal{C} = \{P_{t+2},\cdots,P_{2t+1}\}$):

$$\begin{cases} s_1+\eta_1 s_2+\cdots+\eta_1^t s_{t+1}-r_1 = -(\eta_1^{t+1}s_{t+2}+\cdots+\eta_1^{2t}s_{2t+2}) \\ \vdots \\ s_1+\eta_{t+1} s_2+\cdots+\eta_{t+1}^t s_{t+1}-r_{t+1} = -(\eta_{t+1}^{t+1}s_{t+2}+\cdots+\eta_{t+1}^{2t}s_{2t+2}) \end{cases}$$

The above equation system has $t+1$ equations with $2t+2$ unknown values. The probability to obtain $\{s_i,r_i\}_{k\in t+1}$ is negligible. Again, even t shares $\{[\![s_j]\!]^i\}_{j\in\mathcal{H},i\in\mathcal{C}}$ that are fully determined by the adversary where $|\mathcal{C}| = t$, the adversary still learns no additional information of honest parties' secrets $\{s_j\}_{j\in\mathcal{H}}$ and secrets $\{r_k\}_{k\in[t+1]}$. Therefore, the shares $\{[\![r_k]\!]\}_{k\in[t+1]}$ are uniform and random.

Moreover, even if random shares $\{[\![r_1]\!],\cdots,[\![r_t]\!]\}$ have all been opened and t secrets $\{r_1,\cdots,r_t\}$ are public, the above equation system still has $t+2$ unknowns but only $t+1$ equations. This guarantees that the last remaining random sharing $[\![r_{t+1}]\!]$ would remain secrecy even if all other secrets $\{[\![r_1]\!],\cdots,[\![r_t]\!]\}$ have been revealed. As the distribution of **Game 2** is identical to the real world execution, thus the secrecy in the real world (**Game 0**) is proven. □

G.2 Proof of AsyRanTriGen

Hereunder is the deferred full security proof of AsyRanShGen. We will prove its properties of termination, validity, and secrecy, one by one, as follows.

Lemma 4 (Termination)**.** *If all honest parties activate the AsyRanShGen protocol, then any honest party would output* $\{([\![a_k]\!],[\![b_k]\!],[\![c_k]\!])\}_{k=1}^{B}$.

Proof. In the AsyRanTriGen protocol, the underlying primitives are the AsyRanShGen protocol and MVBA protocol. According to termination property of AsyRanShGen protocol, the AsyRanShGen protocol will eventually terminate and output a set random shares (along with corresponding commitments and evaluation proofs) $\{(C_{ak},[\![a_k]\!]^i,[\![\hat{a}_k]\!]^i,w_{ak}^i)\}_{k\in[2B]}$ for each honest party P_i. For the MVBA protocol, the termination property ensures that every honest party will terminate with the same set $\mathcal{T}$. After MVBA completes, each honest party locally interpolates t-degree share of product with shares of parties in $\mathcal{T}$, and outputs triples $\{([\![a_k]\!],[\![b_k]\!],[\![c_k]\!])\}_{k=1}^{B}$. □

Lemma 5 (Validity)**.** *(i) For each $k\in B$ and $x\in\{a,b,c\}$, the honest parties can interpolate their output shares $[\![x_k]\!]$ to obtain a unique t-degree polynomial whose zero-point is x_k, despite the adversary; (ii) for every $k\in B$, the interpolated $\{x_k\}_{x\in\{a,b,c\}}$ satisfying $a_k\cdot b_k = c_k$.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that can break the validity property. Then we can use $\mathcal{A}$ as a sub-routine to obtain the scripts when executing the AsyRanTriGen protocol. That is $(\{[\![a_k]\!],[\![b_k]\!],[\![c_k]\!],T_{ak},T_{bk},T_{ck},\pi_k\}_{k\in[N]}) \leftarrow \mathcal{A}(\{[\![a_k]\!],[\![b_k]\!]\}_{k\in[N]})$ where $c_k \neq a_k\cdot b_k$. The above scripts fall into the following cases:

1. The hidden evaluations (T_{ak},T_{bk},T_{ck}) fail to satisfy the product relation, which breaks the binding property of Pedersen commitment scheme.
2. The product proof π_k fails to prove that (T_{ak},T_{bk},T_{ck}) satisfies the product relation, which breaks the soundness property of proof of product protocol.
3. All random share $[\![a_k]\!]$ or $[\![b_k]\!]$ do not lie in a unique t-degree polynomial, which breaks the consistency property of AsynShTriGen protocol.
4. All random share $[\![c_k]\!]$ do not interpolate a unique t-degree polynomial, which violates the completeness and additive homomorphism of BACSS.

Therefore, the AsyRanShGen protocol satisfies validity. □

Lemma 6 (Secrecy)**.** *For any generated multiplication triple $([\![a_k]\!],[\![b_k]\!],[\![c_k]\!])$, the adversary learns nothing about a_k, b_k and c_k except that c_k is the product of a_k and b_k.*

Proof. In the following games, we will prove that the adversary learns nothing from asynchronous triples generation except c_k is the product of a_k and b_k.

Game 0. The adversary $\mathcal{A}$ runs the AsyRanTriGen protocol with t corruptions, which corresponds to real world execution.

Game 1. The simulator randomly samples α, τ from $\mathbb{Z}_q$, and publishes $\{g, h = g^{\tau}, (g^{\alpha}, \cdots, g^{\alpha^t}), (h^{\alpha}, \cdots, h^{\alpha^t})\}$ as common reference string. Then, the simulator participates in the BACSS_share protocol and the MVBA protocol on behalf of all honest parties belonging to $\mathcal{H}$. Note that for those BACSS_share$_j$ $\forall j \in \mathcal{H}$, the simulator would pick random polynomials $\phi_j(\cdot), \hat{\phi}_j(\cdot)$ as input to BACSS_share$_j$ on behalf of the dealer. Also note that each corrupted party P_i $(i \in \mathcal{C})$ would receive $\{(C_j, [\![s_j]\!]^i, [\![\hat{s}_j]\!]^i, w_j^i)\}_{j \in \mathcal{T}}$, where $[\![s_j]\!]^i = \phi_j(i), [\![\hat{s}_j]\!]^i = \hat{\phi}_j(i)$.

The distributions of **Game 1** and **Game 0** are trivially identical, as the simulator just generates common reference strings and runs other sub-protocols on behalf of honest parties in the exactly same way without any additional change.

Game 2. Instead of running BACSS_share$_j$ in the AsyRanShGen, the simulator simulates BACSS_share$_j$ scripts sent to corrupted party with only t, n, α, τ and shares $\{[\![s_j]\!]^i\}_{j \in \mathcal{H}, i \in \mathcal{C}}$. Concretely, the simulator randomly picks $\phi'_j(0) \in \mathbb{Z}_q$ for $\forall j \in \mathcal{H}$, and uses $\phi'_j(0)$ and shares $\{[\![s_j]\!]^i\}_{i \in \mathcal{C}}$ to interpolate a polynomial $\phi'_j(\cdot)$. The simulator uses the trapdoor τ to compute $\hat{\phi}'_j(i) = (\phi_j(i) - \phi'_j(i))\tau^{-1} + \hat{\phi}_j(i)$, and sends $(C_j, \phi'_j(i), \hat{\phi}'_j(i), w_j^i)$ to party P_i $(i \in \mathcal{C})$. Then, each party runs MVBA and random extraction to produce random shares. After completing AsyRanShGen protocol, each party P_i $(i \in [n])$ possesses two shares $\{(C_a, [\![a]\!]^i, [\![\hat{a}]\!]^i, w_a^i)\}$ and $\{(C_b, [\![b]\!]^i, [\![\hat{b}]\!]^i, w_b^i)\}$. Then, each party P_i follows the AsyRanTriGen protocol to run BACSS_share$_i$ and MVBA, and outputs triples $([\![a]\!], [\![b]\!], [\![c]\!])$. Each corrupted party P_i $(i \in \mathcal{C})$ possesses $\{(C_{cj}, \phi_{cj}(i), \hat{\phi}_{cj}(i), w_{cj}^i)\}_{j \in \mathcal{H}}$ and $\{\mathsf{proofs}_j\}_{j \in \mathcal{H}}$ where $\mathsf{proofs}_j = (w_a^j, w_b^j, w_{c,0}^j, T_a^j, T_b^j, T_c^j, \pi^j)$. Note that, in the Share Recovery phase of BACSS, the simulator can always simulate valid messages of honest parties received from the implicated party since the underlying public key encryption scheme has IND-CCA security. Specifically, the adversary can initiate implication of a specific party and require honest parties to publish the received message of the implicated party (i.e., access the decryption oracle of the implicated party). This falls into the following two cases:

1) if the implicated party is an honest party, the simulator can call the IND-CCA simulator to respond with $\perp$ indicating decryption failure.

2) if the implicated party is a corrupted party, the simulator can use the IND-CCA simulator to construct valid decryption of ciphertext that received from the implicated party on behalf of honest parties.

The difference between **Game 2** and **Game 1** arises from the manner in which shares are generated, and the distribution of shares received from the implicated party. The distribution of commitment, hide randomness, and witness is identical to the distribution in **Game 1**. The unconditional hiding property of the KZG commitment scheme reveals no additional information about the honest shares. Different encryption keys are used in each dealing session to encrypt messages, preventing the adversary from learning the shares of honest parties. The IND-CCA security of encryption scheme guarantees that the distribution of shares of the implicated party is identical to the distribution in **Game 1**. Thus, the distribution of **Game 2** is the same as **Game 1**.

Game 3. Instead of running BACSS_share$_j$ in the AsyRanTriGen, the simulator simulates the transcripts sent to corrupted parties with only t, n, α, τ and $\{[\![[\![c]\!]^j_{2t}]\!]^i = \phi_{cj}(i), [\![\hat{c}]\!]^i = \hat{\phi}_{cj}(i)\}_{i \in \mathcal{C}, j \in \mathcal{H}}$. For each $j \in \mathcal{H}$, the simulator random samples $\phi'_{cj}(0)$ from $\mathbb{Z}_q$, and uses $\phi'_{cj}(0)$ and $\{[\![[\![c]\!]^j_{2t}]\!]^i\}_{i \in \mathcal{C}}$ to interpolate a polynomial $\phi'_{cj}(\cdot)$. Then, the simulator uses the trapdoor τ to compute $\hat{\phi}'_{cj}(i) = (\phi_{cj}(i) - \phi'_{cj}(i))\tau^{-1} + \hat{\phi}_{cj}(i)$ and compute $\hat{c}'_j = ([\![c]\!]^j_{2t} - \phi'_{cj}(0))\tau^{-1} + \hat{\phi}_{cj}(0)$. This ensures that the underlying commitment $T_c^j = g^{\phi'_{cj}(0)} h^{\hat{c}'_j}$ remains unchanged. Finally, the simulator sends $(C_{cj}, \phi'_{cj}(i), \hat{\phi}'_{cj}(i), w_{cj}^i)$ and $\mathsf{proofs}_j = (w_a^j, w_b^j, w_{c,0}^j, T_a^j, T_b^j, T_c^j, \pi^j)$ to party P_i $(i \in \mathcal{C})$ on behalf of honest party P_j.

The difference between **Game 3** and **Game 2** is that we simulate the transcripts sent to the corrupted parties in the BACSS protocol. The distribution of hide randomness is identical to the distribution in **Game 2**. Thus, **Game 3** maintains the same distribution as **Game 2**.

Game 4. $\mathcal{A}$ runs the protocol as **Game 3** except that we simulate the transcripts of proof of product protocol. By Theorem 5, there exists a simulator that can generate the transcripts π'^j on behalf of honest party P_j $(j \in \mathcal{H})$, which has the identical distribution of transcripts generated by the proof of product protocol. Thus, the distribution of **Game 4** is the same as **Game 3**.

Lastly, consider the view of $\mathcal{A}$ in **Game 4**. The perfect zero knowledge property reveals nothing about the underlying witness $([\![a]\!], [\![b]\!], [\![c]\!])$. $\mathcal{A}$ learns at most t shares of a, b and c, which reveals no information about the secrets a, b and c. The soundness and perfect zero knowledge ensures that $\mathcal{A}$ learns nothing except the fact $c = a \cdot b$.

Since **Game 0** has the same distribution as **Game 4**, the AsyRanTriGen protocol satisfies the secrecy property. □

G.3 Proof of dual-mode triple generation

Lemma 7 (Mostly Consistency of Fast Path)**.** *If some honest parties have outputted in the k-th OptRanTriGen, then all honest parties have already outputted consistent triple shares in the* $(k-1)$*-th OptRanTriGen.*

Proof. When some honest parties have successfully outputted triple shares in the k-th shot, it implies that they must have received all shares of all parties in the instance OptRanTriGen$_{k-1}$. This indicates that neither party has experienced a timeout or detected inconsistent shares. Thus, all honest party can generate consistent triple shares in the $(k-1)$-th shot of OptRanTriGen instance. □

Lemma 8 (Fast Path Secrecy)**.** *For any beaver triples shares* $([\![a_k]\!],[\![b_k]\!],[\![c_k]\!])$ *outputted in the fast path, the adversary learns nothing except* $c_k = a_k \cdot b_k$.

Proof. The security of the $(n, t+1)$ Shamir secret sharing scheme ensures that less than or equal to t shares reveal no information about the remaining share. Thus, the random shares $[\![a_k]\!],[\![b_k]\!]$ keep secrecy. With random shares $[\![a]\!]$ and $[\![b]\!]$, all parties can locally compute $[\![ab]\!]_{2t} = [\![a]\!] \cdot [\![b]\!]$, and they can attempt to perform degree reduction to get $[\![ab]\!]$ by (i) opening $2t$-degree share $[\![ab]\!]_{2t} - [\![r]\!]_{2t}$ to get $ab-r$ and (ii) then adding $[\![r]\!]$ and $ab-r$ to get $[\![ab]\!]$. As random double shares mask triple shares during the degree reduction, the secrecy of triple shares relies on that of random double shares. We claim that the random double shares remain secrecy below. The adversary knows (at most) t of the input shares $\{[\![s_1]\!],\cdots,[\![s_n]\!]\}$ (those provided by corrupted players), and t of the output shares $\{[\![r_1]\!],\cdots,[\![r_n]\!]\}$ (with $k > t$, those reconstructed towards corrupted players). When fixing these $2t$ shares, there are $4t+2$ unknowns for total $n = 3t+1$ equations. The adversary cannot learn $[\![r_1]\!],\cdots,[\![r_{t+1}]\!]$ and $[\![r_1]\!]_{2t},\cdots,[\![r_{t+1}]\!]_{2t}$. Hence, the outputted random double shares are uniformly random, unknown to the adversary. □

Lemma 9 (Unitary Fallback)**.** *Conditioned on that all honest parties initially entered the fast path, if any honest party exits the fast path, then all honest parties will leave the phase and invoke tcv-BA.*

Proof. An honest node leaves a fast path, due to either of the following two reasons: (1) the fast path cannot make progress in time (i.e. fails to output triples before timeout), or (2) it detects inconsistent triple shares. For both situations, once any honest node exits, all honest parties will at least realize the timeout and then enter the tcv-BA phase. □

Lemma 10 (Fallback Termination and Consistency)**.** *If all honest parties enter tcv-BA, then all of them would eventually enter the pessimistic path with consistent secret shares of fast-path triples.*

Proof. We first prove that tcv-BA must terminate. By Lemma 7, all honest will input the shot number of the latest successful OptRanTriGen instance to the tcv-BA, where the input of honest parties to tcv-BA is either of two consecutive shot numbers. Thus, due to the termination of tcv-BA, the primitive must terminate.

Second, all honest parties must obtain a common R from tcv-BA by the validity and agreement of tcv-BA, and R represents a fast path shot where at least one honest party has outputted. All honest parties with R can decide to output or discard the triples in the two safe buffers Pending_1 and Pending_2. Thus, all honest parties would preserve the consistency of triple after fast path (as an implication of Lemma 7). □

Theorem 3 (Validity, Secrecy, and Liveness of Dual-mode Triple Generation)**.** *Algorithm 4 securely realizes the desired properties of AMPC's offline phase of multiplication triple.*

Proof. We prove the dual-mode triple generation satisfies the following properties: validity, secrecy, and liveness.

Validity. The validity of fast path follows from Lemma 10, that is all honest parties would output consistent triples in the same shot of fast path after the tcv-BA terminates. The validity of AsyRanTriGen (c.f. Lemma 5) further ensures that the honest parties can output consistent triples in the pessimistic path.

Secrecy. The secrecy stems from the fact that both fast and pessimistic paths have secrecy, c.f., Lemma 8 and Theorem 6.

Liveness. The dual-mode triple generation consists of three phases: fast path, tcv-BA phase and pessimistic path. Lemma 9 guarantees that all honest parties can leave a failed fast path without "getting stuck" and enter the tcv-BA phase; then Lemma 10 ensures the honest parties eventually enter the pessimistic path; finally, the termination of AsyRanTriGen ensures the pessimistic path can output an ever-growing sequence of triples' shares in the worst case (c.f. Lemma 4). □

H MVBA with Optimistic Termination

We give a generic construction of optimistically terminable asynchronous multi-value validated Byzantine agreement (ot-MVBA) from any MVBA (as Algorithm 9 describes), which has two main phases and an accompanying daemon:

- **Optimistic path**: This fast path has a simple 5-round execution flow with a linear number of messages to exchange via a designated leader, and it executes as: leader $\xrightarrow{\text{VALUE}}$parties$\xrightarrow{\text{PREVOTE}}$leader$\xrightarrow{\text{PREVOTEQC}}$parties$\xrightarrow{\text{VOTE}}$leader $\xrightarrow{\text{VOTEQC}}$parties. If an honest party can complete these steps before a timer expires, then it makes an output, performs another multicast of FINISH message carrying the output and the 2nd "vote" quorum certificate (QC), and finally terminates;
- **Fallback path**: If a party has not yet received a valid "vote" QC from leader upon the timer expires, it enters the fallback path and immediately multicasts FALLBACK message, which carries a "pre-vote" QC together with leader's proposal (if receiving them) or just contains a signature on "no QC received". Then, it waits for $n-f$

FALLBACK messages from distinct parties, if there is at least one FALLBACK message carrying valid "pre-vote" QC, then invokes an underlying MVBA protocol with this FALLBACK message, otherwise, all $n-f$ FALLBACK messages carry signatures on "no QC received", then invokes the underlying MVBA protocol with the party's own input (together with the received $n-f$ signatures on "no QC received"). Finally, the party waits for MVBA returns, and outputs what MVBA outputs.

- **Termination Daemon**: Noticeably, it is possible that some honest party might have outputted and terminated before timeout, so the fallback path's MVBA might lack honest parties to participate and cannot return anything. Nevertheless, we let honest parties multicast a FALLBACK message before terminating, and then let each party wait for a valid FALLBACK message. Such that, a party can output the value carried by FALLBACK message, then multicasts the same message and terminates.

Algorithm 9 Optimistically Terminable MVBA (otMVBA) for P_i with session identifier (SID) built from an underlying MVBA

Input: x_i satisfying $Q(x_i) = True$
Output: some value satisfying the predicate Q

1: start a timer that expires after Δ
2: $P_\ell \leftarrow \mathsf{Permute}(\{P_i\}_{i\in[n]})$ using SID as seed and select the 1st party as leader

// Optimistic path for every node
3: **upon** receiving VALUE(x_ℓ) from P_ℓ:
4: **wait for** x_ℓ satisfying $Q(x_\ell) = True$
5: $\sigma_i^1 \leftarrow Sign_i(\mathsf{SID}||\text{"prevote"}||\mathsf{H}(x_\ell))$
6: **send** PREVOTE(σ_i^1) to P_ℓ
7: **wait for** PREVOTEQC(h, Σ^1) from P_ℓ:
8: **verify** $h = \mathsf{H}(x_\ell)$ and Σ^1 contains $n-f$ valid "prevote" signatures signed by different nodes
9: $\sigma_i^2 \leftarrow Sign_i(\mathsf{SID}||\text{"vote"}||\mathsf{H}(x_\ell))$
10: **send** VOTE(σ_i^2) to P_ℓ
11: **wait for** VOTEQC(h, Σ^2) from P_ℓ:
12: **verify** $h = \mathsf{H}(x_\ell)$ and Σ^2 contains $n-f$ valid "vote" signatures signed by different nodes
13: **multicast** FINISH("terminate", x_ℓ, Σ^2) message to all nodes
14: **output** x_ℓ and **terminate**

// Optimistic path for leader only
15: **if** $P_\ell = P_i$:
16: **multicast** VALUE(x_ℓ) message to all parties
17: **wait for** $n-f$ PREVOTE(σ_j^1) messages carrying valid signatures $\{\sigma_j^1\}_{|n-f|}$ on $SID||$"prevote"$||\mathsf{H}(x_\ell)$ from distinct P_j
18: **multicast** PREVOTEQC($\mathsf{H}(x_\ell)$, $\Sigma^1 = \{(j,\sigma_j^1)\}_{|n-f|}$) to all
19: **wait for** $n-f$ VOTE(σ_j^2) messages carrying valid signatures $\{\sigma_j^2\}_{|n-f|}$ on SID$||$"vote"$||\mathsf{H}(x_\ell)$ from distinct P_j
20: **multicast** VOTEQC($\mathsf{H}(x_\ell)$, $\Sigma^2 = \{(j,\sigma_j^2)\}_{|n-f|}$) to all

// Fallback path for every party (which would decide to still use the leader's input or some other party's valid input)
21: **upon** timer expires (*before receiving valid* Σ^2 *and terminating*):
22: **if** receive valid Σ^1: **multicast** FALLBACK("leader", x_ℓ, Σ^1)
23: **else**: ▷ receive neither VOTEQC Σ^2 nor PREVOTEQC Σ^1
24: $\sigma_i^0 \leftarrow Sign_i(\mathsf{SID}||\text{"no QC received"})$
25: **multicast** FALLBACK("non-leader", $\perp$, σ_i^0) to all
26: **wait for** valid FALLBACK($*$, $*$, $*$) from $n-f$ parties
27: **if** $n-f$ received FALLBACK messages contain at least
28: one valid FALLBACK("leader", x_ℓ, Σ^1):
29: **invoke** MVBA with input ("leader", x_ℓ, Σ^1)
30: **else**: ▷ receive $n-f$ FALLBACK("non-leader", $\perp$, σ_j^0)
31: **invoke** MVBA with ("non-leader", x_i, $\{(j,\sigma_j^0)\}_{n-f}$)
32: **wait for** MVBA returns ($*$, x, $*$)
33: **output** x and wait for MVBA terminates to **terminate**

// Predicate $Q'(x)$ of the underlying MVBA in fallback path:
34: **parse** x as ("leader", x_ℓ, Σ^1) or ("non-leader", x_j, Σ^0)
35: **if** former case: **check** Σ^1 contains $n-f$ valid signatures on x_ℓ
36: **else**: **check** Σ^0 has $n-f$ valid signatures on SID$||$"non-leader" and $Q(x_j) = True$

// Termination daemon for every node
37: **upon** receive FINISH("terminate", x_ℓ, Σ^2) from any party:
38: **verify** Σ^2 has $n-f$ valid signatures on SID$||$"vote"$||\mathsf{H}(x_\ell)$ from distinct parties
39: **multicast** FINISH("terminate", x_ℓ, Σ^2) message to all parties
40: **output** x_ℓ and **terminate**

The above otMVBA construction satisfies all desired properties of MVBA defined in Section 3, and additionally, realizes the next " optimistic deterministic termination" property.

Definition 3 (Optimistic deterministic termination). *An asynchronous (multi-valued) validated byzantine agreement protocol is said optimistically deterministically terminable, if it satisfies the next property:*

- *While network is synchronous, there exists a non-empty set of optimistic conditions, such that all honest nodes can output and terminate, deterministically and responsively.*

Informally, when the selected leader is honest and the network stays synchronous, all honest parties can output and terminate from otMVBA without invoking any randomized subroutines in 5 asynchronous rounds; and even if in the worst case, all honest parties can still decide a unitary output satisfying the external validity condition, in expected constant rounds. The security properties of our otMVBA construction can be formalized as the next theorem.

Theorem 6. *The otMVBA protocol described in Algorithm 9 satisfies agreement, termination, external validity and optimistic deterministic termination, conditioned on that the underlying invoked MVBA protocol satisfies agreement, termination, and external validity.*

Proof. Here we prove agreement, termination, external validity, and optimistic deterministic termination one by one.

Termination. To prove the property, we argue that all honest parties would output and terminate, in either of lines 14, 33, or 40 of Algorithm 9. We consider the next two cases:

Case 1): Some honest party outputs in line 14/40. If any honest party outputs from the fast path in line 14/40, then at least this honest party would multicast a valid FINISH message. As such, all honest parties must eventually output and terminate, because they at least would receive this valid FINISH message, and then output and terminate in line 40.

Case 2): None of honest parties output in line 14/40. If none honest party outputs from the fast path in line 14/40, then the condition in line 26 must be triggered for every honest party, because at least the $n-f$ honest parties would multicast FALLBACK messages. So all honest parties would enter the underlying MVBA protocol with valid proposal. Thus, all honest parties would output and terminate in line 33.

In both cases, all honest parties would eventually output and terminate.

Agreement. We prove that any two honest parties P_i and P_j would have the same output, by considering the next cases:

Case 1): Both P_i and P_j output in line 33. In this case, the outputs of P_i and P_j trivially are equal, as they are both from the underlying MVBA, which satisfies agreement.

Case 2): Both P_i and P_j output in line 14 or 40. In this case, P_i and P_j also have the same output, because there is only a unique value having a valid quorum certificate Σ^2.

Case 3): P_i outputs in line 14/40 and P_j in line 33. This case is more involved, and we can consider the following fact: If an honest party P_i outputs in line 14/40, then the condition of line 30 wouldn't be triggered for any honest party P_j, with all but negligible probability. This fact is true, because if P_i outputs in line 14/40, then there is no sufficient honest nodes sending FALLBACK("non-leader", $\perp$, σ_j^0) messages. Therefore, the output of MVBA must be validated by a certificate Σ^1, which is also same to the value validated by certificate Σ^2, indicating that P_i and P_j must output the same value (as their outputs are validated by Σ^1 and Σ^2, respectively).

In all above cases, the honest parties have the same output.

External Validity. This is trivial to see, as all parties check the external validity condition before signing, indicating any output is verified by at least $f+1$ honest parties.

Optimistic Deterministic Termination. To prove the desired property, considering an honest dealer P_ℓ and synchronous network, it is clear to see that the honest leader P_ℓ can always generate a valid quorum certificate Σ^2 regarding its input x_ℓ, after four rounds of communication. As such, after one more round of communication, all parties can receive the valid FINISH("terminate", x_ℓ, Σ^2) message, and then output and terminate according to line 40. In such good case, no randomized subroutine is ever invoked. □

I Additional Experiments

I.1 Benchmarking online phase

To evaluate the efficiency of the (almost) I.T. online phase mostly inherited from hbMPC, we execute a typical mixing-net task among 4 AWS EC2 instances (c6a.8x charge), to shuffle a varying number of distinct clients' inputs. Table 4 reports the test results. We also evaluate an optimization to fasten online error correction in the malicious setting, via a local blocklist (banning senders of incorrect t-degree shares), which was earlier implemented in hbMPC [70] as well. In sum, the online phase can efficiently evaluate the circuit of mixing-net, evaluating the function with 4096 inputs in 133 seconds with up to 4 EC2 instances. This can be translated into a rate of consuming about 9000 triples per second.

Clearly, the already performant and robust online phase is neither the bottleneck of efficiency nor the weak point of robustness. This fact validates our key design choice of inheriting the almost I.T. online phase and focusing on improving the efficiency of robust offline protocols, as the offline protocols are the primary obstacles to both robustness and efficiency.

Table 4: **Online performance of evaluating a switching network for shuffling 4096 messages [70] ($n=4$).**

# of inputs	# of triples	Execution time		Consumption rate (per/sec.)	
		blocklist	blocklist (w.o)	blocklist	blocklist (w.o)
64	4608	0.61	0.77	7589	6023
128	13444	1.65	1.69	8128	7967
256	32768	3.88	4.23	8449	7739
512	82944	10.27	10.92	8079	7597
1024	204800	23.38	24.11	8759	8495
2048	495616	55.91	56.72	8866	8739
4096	1179648	133.81	141.03	8816	8365

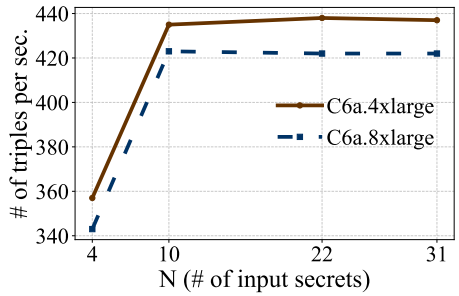


Figure 15: **Throughput of AsyRanTriGen ($n=4$) when testing with c6a.8xlarge and c6a.4xlarge instances.**

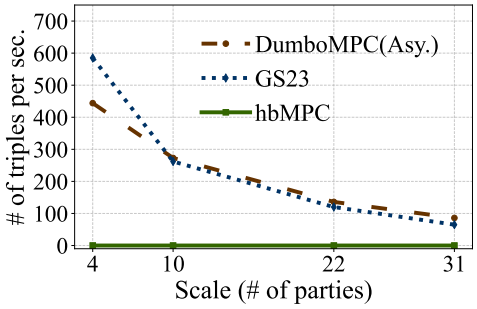
I.2 Types of AWS instances

Figure 15 visualizes the throughput of our asynchronous triple generation protocol when executed on different types of AWS EC2 instances, including high-profile c6a.8xlarge instances (with 48 vCPUs) and low-profile c6a.4xlarge instances (with

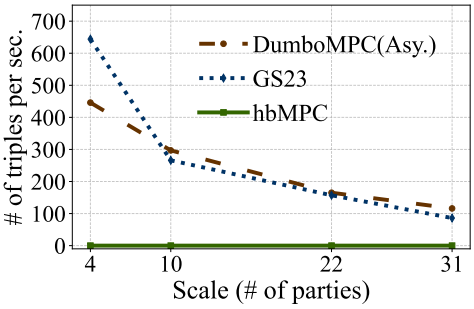
16 vCPUs). It is evident that the potential for parallelization using high-profile instances does not improve throughput. However, we still stick with the high-profile EC2 instances during benchmarking, due to the large memory requirements, in particular to accommodate the memory required by large system scales.

I.3 Throughout with different crash faults

To evaluate the performance of our offline protocols with varying number of faulty parties, we compare Dumbo-MPC with GS23 and hbMPC, in scenarios involving (i) a single crash node and (ii) t crash nodes for different scales with n =4, 10, 22 and 31 parties. All tests are executed with the same batch size of 5000 in the LAN setting. As illustrated in Figure 16, hbMPC remains zero throughputs since it must always wait for messages from all nodes, lacking fault tolerance. The throughputs of Dumbo-MPC and GS23 decrease as the system scale increases, and Dumbo-MPC starts to outperform GS23 when $n \geq 10$. When $n = 31$ with $t = 10$ faults, Dumbo-MPC realizes a throughput of 116 triples/sec, which can generate triples 40% faster than GS23 under the exactly same experiment setting.



(a) Throughput under 1 crash node

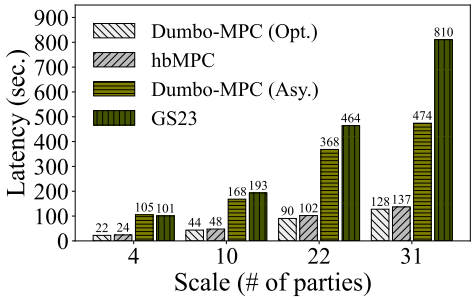


(b) Throughput under t crash nodes

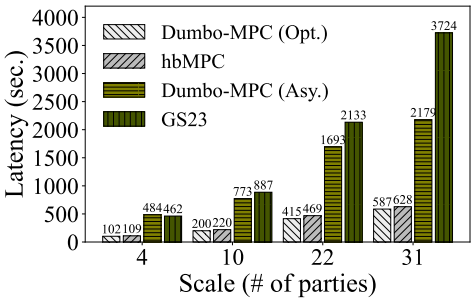
Figure 16: **Triple throughput under different number of crash nodes (batch size is 5000).**

I.4 Pre-processing latency for typical tasks

To demonstrate the feasibility of Dumbo-MPC for realistic applications, we further evaluate the pre-processing latency of



(a) Vickery auction



(b) Switching network

Figure 17: **Latency of pre-processing sufficient triples for Vickery auction and Switching network.**

Dumbo-MPC regarding two typical tasks—Vickrey auction [86] and switching network [70], as illustrated in Figure 17. The former task requires 44,571 triples for 100 inputs, and the latter one needs 204,800 triples for 2^{10} inputs. The pessimistic path of Dumbo-MPC (AsyRanTriGen) takes about 8 minutes and 36 minutes to complete the pre-processing of these two tasks, respectively. In contrast, GS23 needs twice time for the same pre-processing tasks.