ECHO: Efficient Covertly-Secure Three-party Computation with Applications to Private Machine Learning

Yufei Duan, Yun Li, Zhicong Huang, Cheng Hong, Tao Wei, and Chao Zhang

***Abstract*—Secure three-party computation with an honest majority is among the most efficient secure computation settings and is widely used in practice. However, achieving malicious security incurs significant overhead, often an order of magnitude higher than semi-honest protocols. Covert security provides a security–efficiency trade-off by detecting malicious behavior with a certain probability (e.g., 50%), deterring rational adversaries. Existing covert protocols mainly target two-party or dishonest-majority settings, with little work on efficient honest-majority three-party solutions.**

We present ECHO, a family of concretely efficient protocols for covertly secure honest-majority three-party computation. We explore the design space of cheating detection and identification, and develop optimized protocols for both arithmetic and Boolean circuits, targeting different performance goals such as low latency and reduced communication. For arithmetic circuits over rings, our asymmetric-MAC-based protocol achieves an online phase only $1.26\times$ slower than the semi-honest baseline and over $5.59\times$ faster than MAC-based maliciously secure protocol. For Boolean circuits, our method improves over the best malicious protocol by $5\times$. We also applied ECHO on practical PPML tasks. ECHO approaches semi-honest performance while providing up to $8\times$ speedup over malicious security.

***Index Terms*—Secure multi-party computation, three-party computation, covert security, privacy-preserving machine learning, honest majority.**

I. Introduction

Secure Multiparty Computation (MPC), introduced by Andrew C. Yao in 1982 [1], enables mutually distrustful parties to jointly compute a function on their private inputs while revealing only the output. In recent years, MPC has evolved from a purely theoretical construct to a practical technology, examples include private machine learning [2], [3], [4], [5], statistics [6], or voting [7].

The threat models of MPC can be classified into two types: **semi-honest security**, where adversaries follow the protocol but attempt to infer additional information, and **malicious security**, where adversaries may cheat and arbitrarily deviate from the protocol. Recent works [8], [9] have shown that malicious parties in MPC can compromise both data integrity and privacy, emphasizing the need for malicious security. However, current maliciously secure solutions impose significant performance penalties. For example, SOTA works [2], [10], [11], [12] utilize secret MACs (Message Authentication Codes), DZKPs (Distributed Zero-Knowledge Proofs), or Beaver triples [13] to guarantee malicious security, but they are $3 \sim 83\times$ slower compared to their semi-honest counterparts as we benchmarked.

In 2007, Aumann and Lindell [14] introduced **covert security** as an intermediate threat model to balance security and efficiency. Unlike malicious security's overwhelming anomaly detection guarantee, covert security provides probabilistic detection with deterrence factor ϵ thus reducing the verification overhead. For rational business parties, this security model is already powerful enough, because the possibility of public exposure and reputation loss will discourage malicious behaviors, especially since mounting a successful attack often necessitates multiple rounds of cheating without being caught [8].

However, there still exist significant gaps in designing concrete efficient covert protocols. Most prior works on covert security focus on two-party (2PC) [14], [15], [16], [17] or general n-party computation (n-PC) [18], [19] with a dishonest majority, while little attention has been paid to the three-party (3PC) honest-majority setting, which enables concretely efficient protocols and has been widely studied in the applied MPC literature [20], [21], [22], [23], [12], [5], [4]. The challenges of building concrete efficient covert 3PC protocol could be summarized as follows:

- Covert security is not merely malicious security with an adjusted soundness guarantee. Under the standard definition of Aumann and Lindell [14], covert security requires not only cheating detection but also cheater identification, a property that maliciously secure protocols are not required to provide.[1]
- Existing malicious MPC protocols with cheater identification or public verifiability can be instantiated with three parties [24], [25], [26], [27], but they retain the machinery required for dishonest-majority security, such as BMR/SPDZ, OT/HE, and broadcast or public bulletin boards. Thus, direct instantiation is secure but does not exploit 2-out-of-3 RSS or native $\mathbb{Z}_{2^k}/\mathbb{Z}_2$ computation, and does not yield the concrete efficiency targeted here.
- Unlike in 2PC [15], where any cheating detection directly reveals the cheating party, honest parties in 3PC face ambiguity when attempting to identify which of the remaining two parties is the cheater.
- While [19], [18] propose n-party covert protocols with cheater identification, they are restricted to input-independent preprocessing protocols or are costly to adapt

Y. Duan and C. Zhang are with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: dyf23@mails.tsinghua.edu.cn; chaoz@tsinghua.edu.cn).

Y. Li, Z. Huang, C. Hong, and T. Wei are with Ant Group, Hangzhou 310000, China (e-mail: liyun24@antgroup.com; zhicong.hzc@antgroup.com; vince.hc@antgroup.com; lenx.wei@antgroup.com).

Manuscript received xxx; revised xxx.

[1]Protocols like [5] ask all the parties to send inputs to the honest party to compute the function upon failure. While effective for guaranteed output delivery (G.O.D.), this approach still does not support cheater identification.

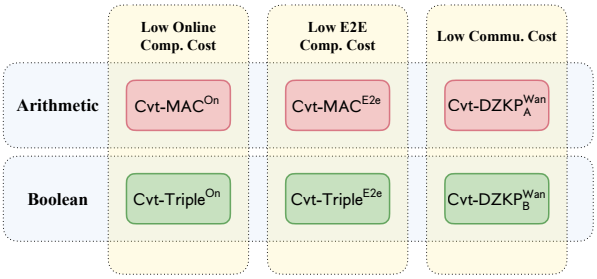


Fig. 1: Overview of our covertly secure 3PC protocols. "On": optimized for online computational efficiency, "E2E": optimized for end-to-end computational efficiency (online and offline), "Wan": optimized for communication efficiency, suitable for WAN settings.

to input-dependent protocols, and it's unexplored how to apply their protocols to practical MPC scenarios.

These observations leave open how to construct concretely efficient, covert protocols tailored to honest-majority 3PC, while supporting public cheater identification over both arithmetic and Boolean circuits.

A. Problem Definition and Threat Model

The data (e.g., model parameters and images) is secret shared among three parties. They want to jointly run a function (e.g., apply a neural network inference) on the data to obtain an output (e.g., inference result), without letting any party know anything beyond the output and public information (e.g., hyperparameters such as model structure and image size). The function is typically expressed as an arithmetic or a Boolean circuit. As rings like $\mathbb{Z}_{2^k}$ are more commonly adopted than fields (especially in PPML), in this work we mainly focus on arithmetic circuits over $\mathbb{Z}_{2^k}$ and Boolean circuits over $\mathbb{Z}_2$.

We assume only one party will be corrupted and try to run arbitrary malicious behavior to gain advantages. Our protocols aim to detect such malicious behavior with (at least) a given probability ϵ, and identify the malicious party after detecting such behavior; in addition, we allow the honest parties to generate a publicly verifiable certificate that can be validated by any third party to confirm the dishonesty of the cheater.

B. Technical Contributions

In this work, we present ECHO, a family of efficient covertly secure 3PC protocols with public verifiability for arithmetic circuits and Boolean circuits. We give an overview of our protocols in Figure 1, and compare our covertly secure protocols with existing SOTA 3PC protocols in the semi-honest and malicious model in communication cost in Table I.

At a high level, covert security can be achieved via: (1) detecting cheating behavior, then (2) identifying the cheater (with public verifiability). Below we first present our cheating detection approaches for $\mathbb{Z}_{2^k}$ and $\mathbb{Z}_2$ respectively, and then show how to identify the cheater with public verifiability.

1) Cheating Detection over $\mathbb{Z}_{2^k}$: We design three protocols to detect cheating behavior over $\mathbb{Z}_{2^k}$: the first two adapt existing MAC-based and DZKP-based cheating detection techniques using smaller ring gaps, and the third one is specially designed for low online cost.

TABLE I: Communication cost of different honest-majority 3PC protocols for arithmetic circuits over $\mathbb{Z}_{2^k}$ and k-bit parallel Boolean circuits over $\mathbb{Z}_2$. $\epsilon < 1$ is the deterrence factor. s and s^* denote the ring gap in malicious and covert security, respectively. Typically s is set to 46 and 64 in [28] and [11], and s^* is set to $3 \sim 12$ in ECHO. κ denote the computational security parameter. Communication costs are global. All costs are amortized over multiplication/AND gates.

Protocol	Threat Model	Offline Commu.	Online Commu.
[20]	Semi-honest	-	$3k$
[3]		k	$2k$
[28]	Malicious	-	$6(k+s)$
[11], [12]		-	$3k$
[21], [29]		$21k$	$6k$
[22]		$12k$	$9k$
[30], [29]‡		$9(k+s)$	$6k$
[31], [15]	Covert ($\epsilon = 50\%$)	-	$6\kappa k$
[16], [17]		-	$2\kappa k$
Cvt-MAC^E2e		-	$6(k+s^*)$
Cvt-MAC^On		$3(k+s^*)$	$2(k+s^*)$
Cvt-DZKP^Wan_A		-	$3k$
Cvt-DZKP^Wan_B		-	$3k$
Cvt-Triple^On		$6k$	$6k$
Cvt-Triple^E2e		-	$7.5k$

(‡): The authors of MP-SPDZ [29] adapt the techniques from [30] to the three-party honest-majority setting.

(1) Cvt-MAC$^{\text{E2e}}$, Cvt-DZKP$^{\text{Wan}}_{\text{A}}$: Adapting MACs and DZKPs. A key insight is that while the "ring gap" required for malicious security is moderate, it necessitates multi-precision arithmetic that incurs a drastic performance penalty. Specifically, for arithmetic circuits over the ring $\mathbb{Z}_{2^k}$, the two SOTA cheating detection techniques, MACs [28] and DZKPs [11], both perform the verification over a sufficiently large ring $\mathbb{Z}_{2^{k+s}}$, where, for example, $s = 46$ and 64 respectively for ensuring 2^{-40} soundness error. For $k = 64$, the ring will be lifted to $\mathbb{Z}_{2^{110}}$ and $\mathbb{Z}_{2^{128}}$, respectively. However, if we bound the soundness guarantee to 50%, the ring gap s will decrease to 3 and 12; for most applications, this allows us to embed the ring gap s into $\mathbb{Z}_{2^{64}}$, (i.e., set $k = 61$ and 52 respectively s.t. $k + s = 64$). Given the fact that multiplications over $\mathbb{Z}_{2^{64}}$ are $2 \sim 3\times$ faster than those over $\mathbb{Z}_{2^{110}}$ and $\mathbb{Z}_{2^{128}}$, this directly leads to a notable efficiency gain.

We denote the above two approaches by Cvt-MAC$^{\text{E2e}}$ and Cvt-DZKP$^{\text{Wan}}_{\text{A}}$, which inherently feature a high end-to-end computational efficiency and sublinear communication cost (that outperforms in WAN), respectively.

(2) Cvt-MAC$^{\text{On}}$: Achieving a Lightweight Online Phase. We notice that the verification overheads of the above MAC-based [28] and DZKP-based [11] approach are incurred online; however, one may expect a more efficient online phase if they can perform preprocessing. On the other hand, ASTRA [3] adopts an asymmetric online-efficient setting where P_0 dis-

tributes correlated randomness in the preprocessing phase, and P_1, P_2 evaluate the circuit in the online phase, but its triple-based cheating detection approach entails high offline communication costs.

We design a new online-efficient cheating detection approach by exploiting the asymmetric paradigm of ASTRA but using MACs. We observe that the correlated randomness generated by P_0 can be verified in the preprocessing phase, and if all the checks are passed, we can let P_0 sample the random MAC key (instead of the parties together generating a shared one as in the original MAC-based protocol [28]). With P_0 offering additional correlated randomness in the preprocessing phase, P_1, P_2 can *locally* authenticate each multiplication output in the online phase, and thus incur no extra (amortized) online communication cost over the semi-honest protocol of ASTRA (except that caused by the expansion of the ring). As a result, we can reduce the online communication cost from $6(k+s)$ bits per gate in [28] to $2(k+s)$ bits, with $3(k+s)$ offline communication bits.

We emphasize that this new approach may be of independent interest, e.g., it can be used for constructing maliciously secure protocols with all the asymptotic improvements maintained. In the covert model, we achieve ϵ deterrence factor by controlling the ring gap s as above. We denote the new online-efficient cheat detection approach by $\mathsf{Cvt\text{-}MAC}^{\mathsf{On}}$.

2) Cheating Detection over $\mathbb{Z}_2$: We also design three protocols for Boolean circuits: the first one adapts the DZKP-based approach [32], [23], and the latter two are based on triples with a specially designed cut-and-choose paradigm.

(1) $\mathsf{Cvt\text{-}DZKP}^{\mathsf{Wan}}_{\mathsf{B}}$: Adapting DZKPs. When verifying semi-honest behaviors over $\mathbb{Z}_2$, we need to lift the verification onto a large binary extension field to achieve the given soundness guarantee. Thus, in the covert model we can control the size of the extension field to achieve the specific deterrence factor.

(2) $\mathsf{Cvt\text{-}Triple}^{\mathsf{E2e}}$, $\mathsf{Cvt\text{-}Triple}^{\mathsf{On}}$: A New Cut-and-Choose Paradigm in the Covert Model. In SOTA triple-based cheating detection approaches [21], [22], [29], the parties typically generate random AND triples using the cut-and-choose paradigm in the preprocessing phase, and then verify semi-honest online AND triples using these random offline triples. Intuitively, in the covert model we can loosen the soundness guarantee by adapting the involved parameters, e.g., using a smaller bucket size B. However, we notice that for slightly large circuit sizes (e.g., 100) and moderate deterrence factors (e.g., 50% or 75%), even the smallest bucket size $B = 2$ will lead to exceeded soundness guarantees with unnecessary overheads. Therefore, we tailor a new cut-and-choose paradigm for the covert model, by allowing the parties to output some unchecked triples. Our new partial-bucketing construction instead checks only a random subset and outputs the unchecked triples alongside the checked ones. This enables us to significantly reduce the offline overheads by adjusting the number of initial triples and the proportion of the checked triples properly, compared to SOTA triple-based approaches [21], [22], [29] and a cut-and-choose-based approach [33].

We design two variants of the above approach; one features a higher online efficiency, and the other features a higher end-to-end computational efficiency. We denote them by $\mathsf{Cvt\text{-}Triple}^{\mathsf{On}}$ and $\mathsf{Cvt\text{-}Triple}^{\mathsf{E2e}}$, respectively.

3) Cheater Identification with Public Verifiability: We note that for the DZKP-based approaches ($\mathsf{Cvt\text{-}DZKP}^{\mathsf{Wan}}_{\mathsf{A}}$ and $\mathsf{Cvt\text{-}DZKP}^{\mathsf{Wan}}_{\mathsf{B}}$) and the one working in the asymmetric preprocessing setting ($\mathsf{Cvt\text{-}MAC}^{\mathsf{On}}$), cheater identification is almost free. For the others, we design a general cheater identification mechanism by using Commit-and-Prove Non-Interactive Zero-Knowledge (CP-NIZK) proofs.

(1) A Lightweight Cheater Identification Mechanism. We follow previous works [18], [19] using a signature and commitment scheme to make the parties accountable for their inputs and messages. We observe that in the three-party honest-majority setting, the key to identifying cheaters with almost free costs is to identify an honest party, who can act as a judge and identify the cheater by checking the other parties' views.

For the DZKP-based approaches, the core property to help identify an honest party is "re-computable verification" [23]. That is, an honest prover can always tell which verifier is cheating by re-computing the verifiers' messages. Therefore, once an abortion occurs, we can let the prover accuse one verifier, and the other verifier will be viewed as honest. For the asymmetric MAC-based approach, the re-computable verification property also holds for P_0. If the verification on P_0 is passed, then P_0 can be identified as an honest party; otherwise, we let P_0 accuse one verifier, and identify the left one party as honest. The above identification mechanisms introduce no heavy cryptographic primitives. and thus are efficient.

(2) A General Cheater Identification Mechanism. For the symmetric MAC-based and triple-based approaches, our idea is to enforce the parties to generate a zero-knowledge proof showing their honest behaviors after an abortion happens, i.e., proving that all its sent messages are faithfully computed according to the protocol specification w.r.t. its input, random tape and all the received messages; then the one who fails to provide a valid proof will be accused as the cheater.

(3) Achieving Public Verifiability. We observe that the three-party honest-majority setting is especially advantageous for achieving public verifiability, compared to the previous two-party or n-party dishonest-majority settings [18], [19], [17], [16]. In the three-party honest-majority setting, public verifiability follows naturally after identifying the cheater: the two honest parties can generate and sign a message that accuses the cheater, and then collect their signed message as a certificate. After that, any third party can confirm the dishonesty of the cheater by checking the two valid signed messages.

4) Support for PPML Building Blocks: We adapt standard PPML building blocks—including fixed-point arithmetic, truncation, comparison, and conversions between arithmetic and Boolean sharings—to our covertly secure protocols under the covert security model.

5) Implementation and Evaluations: We implement all protocols in MP-SPDZ [29]. Note that most previous covertly

secure MPC works are not open-sourced, except [17] for Boolean circuits. Take deterrence factor $\epsilon = 50\%$ for example, in the optimistic case, where no cheating is detected, [17] is $122\times$ slower than our work. In the pessimistic case, ECHO requires the parties to prove the correctness of their behavior using CP-NIZKs after an abort. When instantiated with VOLE-ZK, it achieves efficiency comparable to [17]. In arithmetic benchmarks, Cvt-$\mathsf{MAC}^{\mathsf{E2e}}$ achieves $2.38\times$ end-to-end speedup over the best maliciously secure MAC-based method [28]; Cvt-$\mathsf{DZKP}_{\mathsf{A}}^{\mathsf{Wan}}$ is $2.27\times$ faster than the best maliciously secure DZKP-based method [11]; more importantly, Cvt-$\mathsf{MAC}^{\mathsf{On}}$ achieves similar online costs as the semi-honest baselines, and is $3.56\times$ and $5.54\times$ faster than the SOTA triple-based maliciously secure method [30] in the online and offline phase, respectively.

In Boolean benchmarks, Cvt-$\mathsf{DZKP}_{\mathsf{B}}^{\mathsf{Wan}}$ is $1.73\times$ faster than the best maliciously secure DZKP-based method [12]; Cvt-$\mathsf{Triple}^{\mathsf{E2e}}$ delivers $5.36\times$ end-to-end improvement over the best maliciously secure triple-based method [22]; Cvt-$\mathsf{Triple}^{\mathsf{On}}$ matches the online cost of the best online-oriented maliciously secure method [21], [29] while reducing the offline cost by $2.89\times$.

II. Related Works

Below, we review SOTA maliciously and covertly secure protocols.

A. Maliciously Secure Protocols

Cheating Detection Techniques

(i) MAC-based Approach: At a high level, in MAC-based approaches [28], [34], the parties authenticate the inputs and all outputs of multiplication gates using a random secret MAC key. For finite fields, the randomness and secrecy of the MAC key guarantee that the probability of a malicious adversary cheating successfully is negligible provided that the field is sufficiently large. However, unlike fields, directly using MACs over $\mathbb{Z}_{2^k}$ would entail a large soundness error (i.e., $1/2$ in the worst case). Therefore, the previous work [28] proposed to perform the computation over a sufficiently larger ring $\mathbb{Z}_{2^{k+s}}$, which would reduce the soundness error to $2^{-s+\log(s+1)}$. However, as all the multiplication gates should be authenticated additionally, this approach incurs at least doubles communication cost of $2(k+s)$ bits per gate per party (compared to k bits of the semi-honest protocol). Our Cvt-$\mathsf{MAC}^{\mathsf{E2e}}$ retains the authentication mechanism of Abspoel et al. [28], but chooses a smaller ring gap s^* so that the soundness error matches the target deterrence factor rather than being negligible. In Cvt-$\mathsf{MAC}^{\mathsf{On}}$, P_0 supplies correlated MAC terms during preprocessing, allowing P_1, P_2 to construct the authenticated value locally; we further batch the cross-terms across gates, reducing the amortized cost to three elements offline and two online.

(ii) DZKP-based Approach: This approach is derived from distributed zero-knowledge proofs introduced by [32] and then developed in [23], [35], [11], [36], [37]. At a high level, it applies to secret-share-based semi-honest protocols where all involved data is distributed among the parties, and verifies the semi-honest executions of the parties in a distributed way. A key feature of this approach is that it only incurs a logarithmic communication cost. However, it requires a sufficiently large finite field to ensure a negligible soundness error. When working over the ring $\mathbb{Z}_{2^k}$, the SOTA work [11] proposes to combine the idea of the ring gap from [28] and the recursive techniques from [32], and achieves logarithmic communication cost. However, compared to the MAC-based approach, its computational cost is still relatively higher.

The DZKP-based approach also applies to Boolean circuits, but typically requires lifting the verification onto a sufficiently large binary extension field [32], [23] to guarantee negligible soundness errors. A recent work [12] proposed to alternatively lift the verification onto a prime field. By designing an efficient transformation method and applying a 61-bit Mersenne field, they can achieve higher concrete efficiency. Our Cvt-$\mathsf{DZKP}_{\mathsf{A}}^{\mathsf{Wan}}$ and Cvt-$\mathsf{DZKP}_{\mathsf{B}}^{\mathsf{Wan}}$ retain the existing DZKP constructions, but reduce the ring gap or extension-field size so that the soundness error matches $1-\epsilon$ rather than being negligible. The offline phase of our Cvt-$\mathsf{MAC}^{\mathsf{On}}$ uses DZKPs to verify the correctness of P_0's messages, and Cvt-$\mathsf{DZKP}_{\mathsf{A}}^{\mathsf{Wan}}$ and Cvt-$\mathsf{DZKP}_{\mathsf{B}}^{\mathsf{Wan}}$ follows [23] to identify an honest party.

(iii) Triple-based Approach: Triple is another classic technique used for achieving malicious security for Boolean circuits [38], [39], [40], [41], [21], [22], [3], based on the observation that any incorrect AND triple can be detected using a validated random AND triple with probability 100%. To efficiently generate validated AND triples, the previous work [21] adopted the cut-and-choose paradigm, where the parties generate and shuffle some initial triples, and check them via an open-then-bucket-check procedure (see details in Section V). For circuit size $N = 2^{20}$ and 40-bit soundness security, the protocol costs 7 bits to generate a validated triple, and 3 bits to compute and verify an online AND gate. The follow-up work [22] improves the approach of [21] by taking a smaller bucket size but adding an extra shuffle process on the online AND triples. This reduces the communication cost from 10 bits to 7 bits per gate per party. However, the online computational cost is slightly heavier. In MP-SPDZ framework [29], the authors optimized the online phase of [21] by letting the parties directly compute an AND gate using a validated AND triple. In this way, the online communication cost is reduced to 2 bits per gate. In our work, we observe that even the lowest security parameter used in prior schemes provides a soundness error of $1/N$, where N is the number of triples generated in each batch. Our construction instead checks only a random subset and deliberately retains both checked and unchecked triples. This requires a new soundness analysis and, for $\epsilon = 50\%$, reduces the global preprocessing cost from 21 to 6 bits per gate.

Cheater Identification Techniques

A common pattern in designing efficient MPC over arithmetic circuits with cheater identification is to bind the secret sharing with authentication. Concretely, prior constructions employ linearly homomorphic information-theoretic signatures

on individual shares [24], homomorphic commitments together with transcript-based public auditing [25], openable pairwise MACs and committed correlation inputs supporting vindicating release [26], or publicly verifiable PRF-based MACs [27]. If any inconsistency is detected, the protocol releases or replays the prescribed evidence, such as verification keys and tags so that the deviation can be attributed. Particularly, [25] uses an honest-majority assumption. It uses Shamir secret sharing and NIZK to identify the cheater. Although these protocols can be instantiated for 3PC simply by setting $n = 3$, many of their costly components (like HE, OT, Shamir secret sharing) are designed for more general settings and stronger security guarantees. In contrast, ECHO exploits the 3PC structure to achieve efficient cheater identification while avoiding many costly cryptographic primitives.

B. Covertly Secure Protocols

Cheating Detection Techniques

In the literature [14], [15], [16], [17], [42], [19], [18], covert security is typically achieved using the cut-and-choose paradigm [43] by running t instances of the semi-honest protocol in parallel and randomly opening $t-1$ of them for verification. This provides a deterrence factor of $1-\frac{1}{t}$, however, incurring a factor-t overhead of the semi-honest protocol. Another work by Chen [33] proposes a cut-and-choose-based approach to achieve both covert and malicious security for 3PC Boolean circuits. The parties execute s instances in parallel while jointly sampling a random, secret-shared selection vector $[\vec{c}]$ of length s. After committing to all execution transcripts, they open $\vec{c}$, reveal and verify all transcripts corresponding to $c_i = 1$, and use MPC to check that the outputs of all instances with $c_i = 0$ are consistent. However, when s is small, $\vec{c}$ may be the all-one vector with non-negligible probability 2^{-s}, in which case every instance is selected for verification and no instance remains to produce an output. This failure probability substantially limits the practicality of their construction when instantiated as a covert protocol with a small number of instances.

Cheater Identification with Public Verifiability

In recent works [18], [19], public verifiability is further achieved using time-lock puzzles [44]. However, the paradigm only applies to input-independent preprocessing protocols due to privacy requirements, while the most efficient RSS-based 3PC protocol [20] does not fall into this category. To compile the 3PC protocol [20] into a covertly secure one, other techniques such as the player virtualization paradigm [45], [46] are required, which would incur an explosion in communication and computational overhead. Asharov et al. [15] use signed OT and digital signatures, which provide the honest party with publicly verifiable evidence of cheating. Attema et al. [47] adapt the same cut-and-choose compiler to the honest-majority setting and replace time-lock puzzles with publicly verifiable secret sharing (PVSS), allowing the honest parties to reconstruct committed random tapes even if a detected cheater aborts. This avoids the trusted setup and actively secure puzzle-generation protocol of prior compilers, but still targets input-independent protocols and incurs the factor-t cost of parallel executions.

Our work considers the honest-majority 3PC setting and exploits several efficiency-enhancing properties specific to this setting. For lightweight identification, we exploit honest-majority 3PC in our DZKP-based and asymmetric MAC-based protocols to identify an honest internal judge. For the other protocols, our general mechanism adds commitments and message signatures in the optimistic execution and invokes CP-NIZK only after an abortion, thereby avoiding disclosure of private inputs and random tapes. Once a cheater is detected, the honest-majority assumption guarantees that both parties other than the accused party are honest. An external verifier therefore needs only to verify two accusation signatures.

III. Preliminaries

A. Notation

Let $\mathsf{P} = \{P_0, P_1, P_2\}$ be a set of three parties. We use P_j where $j \in \{0,1,2\}$ to denote a certain party in this set, and $P_{j\pm1}$ to denote the next $(+)$ or previous $(-)$ party. We use $\mathbb{Z}_2$ for the binary ring of modulus 2, and $\mathbb{Z}_{2^k}$ for a ring of modulus 2^k. We may explicitly use $=_k$ for congruence modulo 2^k, and use $:=$ for assignment. For simplicity, we use $[n]$ to denote the set $\{1,\cdots,n\}$, and $[n_1,n_2]$ for $\{n_1,\cdots,n_2\}$. We use $\vec{v}$ for a vector.

Let κ be the computational security parameter and σ the statistical security parameter. We use $\mathsf{negl}(\cdot)$ to denote a negligible function. We use H for a pre-image and collision-resistant hash function and PRG for a pseudo-random generator with a key $K \in \{0,1\}^\kappa$ (the target domain will be clear from context). We will use some shared keys to generate shared randomness, including a global key K known by all the parties, and another two keys K_t for $t \in \{1,2\}$ known by P_0 and P_t, respectively.

B. Secure Multiparty Computation

We follow the standard ideal/real simulation paradigm, and use the covert notions from previous works [14], [15], [19], [18].

Covert Security with Public Verifiability. We consider the strongest security notion in the covert model, i.e., *strong explicit cheat formulation* (SECF). The ideal adversary can send a special message (cheat, i) to the ideal functionality, and $\mathcal{F}_{\text{SECF}}$ allows the cheating happens with probability $1-\epsilon$; in this case, the ideal adversary $\mathcal{S}$ can learn the honest parties' inputs, and decide the output of the functionality. Covert security with public verifiability property further enhances a covertly secure protocol with two algorithms Blame and Judge. The algorithm Blame is used for an honest party to distill a certificate cert_j from its view after it outputs a cheater P_i, denoted by $\mathsf{cert}_j = \mathsf{Blame}(\mathsf{view}_j)$. The algorithm Judge can be executed by any third party, which takes as input an honest party's certificate cert_j and outputs the identity of a cheating party, i.e., $\mathsf{id}_i = \mathsf{Judge}(\mathsf{cert}_j)$. We give a detailed description of the definition in Appendix A.

Security up to Additive Attacks. In this work we follow the literature to consider a multiplication functionality that is secure up to additive attacks [48], [49], [34], [28], [11], [12], i.e., the only way an adversary can cheat when computing a multiplication gate, is to add a non-zero error to the result. It captures the property of many existing semi-honest multiplication protocols [20], [3], [50], including the ones we use in this work. Starting from these semi-honest multiplication protocols, the goal of our protocols is to detect such additive attacks given a deterrence factor.

C. Sharing Semantics

Additive Secret Sharing (ASS): A value x is said to be additively secret-shared between parties P_1, P_2, if each party P_j for $j \in \{1,2\}$ hold a random additive share x_j s.t. $x = x_1 + x_2$. We denote it by $[x] = (x_1, x_2)$.

Replicated Secret Sharing (RSS) [51]: A value x is said to be replicated-secret-shared among parties P_0, P_1, P_2, if each party P_j for $j \in \{0,1,2\}$ holds two random additive shares x_j, x_{j-1} s.t. $x = x_0 + x_1 + x_2$. We denote it by $[\![x]\!] = (x_0, x_1, x_2)$.

We may use superscripts to specify the underlying domain of this share, i.e., $[\![\cdot]\!]^{\mathsf{A}}, [\![\cdot]\!]^{\mathsf{B}}$ denote RSS over the arithmetic ring $\mathbb{Z}_{2^k}$ and the binary ring $\mathbb{Z}_2$ respectively.

Pairwise Consistency. We say $[\![x]\!]$ is pairwise consistent if the additive share that is commonly held by the two honest parties is consistent. We adopt a procedure "cmpview" from [21] for the parties to check their common shares. Specifically, for each pair of two parties P_j, P_{j+1} holding a common additive share x_j, we let P_j send a hash digest $\mathsf{H}(x_j)$ to P_{j+1}, who then checks if the digest is consistent with the share x_j held by itself. If any inconsistency is detected, the parties will abort.

Reconstruction. To reconstruct a secret x to a party P_j, we let P_{j+1} send x_{j+1} to P_j, who then sums the three shares up to recover x. To open x to all the parties, we simply run the above procedure for each $j \in \{0,1,2\}$. However, a malicious party may send an incorrect share during reconstruction. We use cmpview to detect such behaviors: since P_{j+1} and P_{j-1} both hold x_{j+1}, we additionally let P_{j-1} send a hash digest $\mathsf{H}(x_{j+1})$ to P_j, who then will check consistency of the received x_{j+1} and $\mathsf{H}(x_{j+1})$. We denote the above procedure by Rec.

Linearity. RSS is linearly homomorphic, and thus addition and multiplication-by-constant can be locally computed by the parties. Specifically, given two sharings $[\![x]\!], [\![y]\!]$, and any public values a, b, they can locally compute $[\![z]\!] := [\![ax+by]\!]$ by each party P_j setting $(z_{j-1}, z_j) := (ax_{j-1}+by_{j-1}, ax_j+by_j)$. For any public constant c, obtaining $[\![x+c]\!]$ only requires any two parties adding c to their common additive share of x.

Multiplication. To compute $[\![z]\!]$ where $z = x \cdot y$ given $[\![x]\!], [\![y]\!]$ (which is denoted by $[\![x]\!] \cdot [\![y]\!]$ in this work), first note that each P_j can locally compute an additive share of z, i.e., $z_j := x_j y_j + x_j y_{j-1} + x_{j-1} y_j$ s.t. $z = z_0 + z_1 + z_2$. To convert it into a replicated secret sharing, each P_j needs to pass around its share z_j to the next party, and to guarantee privacy, z_j should be masked with an additive share of 0, which can be locally generated using PRG with shared keys [51].

Protocol 3.1: Π_{Mul} **(Multiplication in ASTRA Sharing)**
Input: $[\![x]\!] = (r_{x,1}, \delta_x, r_{x,2})$ and $[\![y]\!] = (r_{y,1}, \delta_y, r_{y,2})$.
Preprocessing:

1) P_0 computes $r_{xy} := r_x \cdot r_y$ where $r_x = r_{x,1} + r_{x,2}$, $r_y = r_{y,1} + r_{y,2}$, and then shares r_{xy} to P_1, P_2.
2) P_0 and P_t for $t \in \{1,2\}$ run $r_{z,t} := \mathsf{PRG}_{\mathsf{K}_t}(\cdot)$. P_0 sets its share of z as $(r_{z,1}, r_{z,2})$.

Online:

1) P_1 computes $\delta_{z,1} := \delta_x\delta_y + \delta_x r_{y,1} + \delta_y r_{x,1} + r_{xy,1} - r_{z,1}$. P_2 computes $\delta_{z,2} := \delta_x r_{y,2} + \delta_y r_{x,2} + r_{xy,2} - r_{z,2}$. P_1, P_2 exchange $\delta_{z,1}, \delta_{z,2}$ with each other.
2) P_1, P_2 locally compute $\delta_z := \delta_{z,1} + \delta_{z,2}$, and set their share as $(\delta_z, r_{z,1})$ and $(\delta_z, r_{z,2})$ respectively.

ASTRA Secret Sharing: We introduce a variant of the RSS scheme from ASTRA [3]. In this scheme, a secret x is masked by a randomness r_x s.t. $x = \delta_x + r_x$, and

- δ_x is held by P_1, P_2,
- r_x is held by P_0, and is additively shared between P_1, P_2.

Concretely, P_0 holds randomness $(r_{x,1}, r_{x,2})$ s.t. $r_x = r_{x,1} + r_{x,2}$, and P_1, P_2 hold $(\delta_x, r_{x,1})$ and $(\delta_x, r_{x,2})$, respectively. It's clearly the form of RSS, and we can denote it by $[\![x]\!] = (r_{x,1}, \delta_x, r_{x,2})$.

ASTRA sharing inherits the linearity property from the RSS scheme. Pairwise consistency and the reconstruction procedure also follow naturally.

Multiplication. The multiplication protocol of ASTRA [3] slightly differs from that of the RSS scheme. For each multiplication gate computing $[\![z]\!] := [\![x]\!] \cdot [\![y]\!]$, we have $z = (\delta_x + r_x)(\delta_y + r_y) = \delta_x\delta_y + \delta_x r_y + \delta_y r_x + r_x r_y$. As P_1, P_2 already hold δ_x, δ_y and $[r_x], [r_y]$, we let P_0 first provide an additive sharing of $r_{xy} := r_x r_y$ to P_1, P_2 (which can be done in the preprocessing phase); then P_1, P_2 can compute an additive sharing $[z] := \delta_x\delta_y + \delta_x[r_y] + \delta_y[r_x] + [r_{xy}]$ in the online phase. By letting P_0 generate a shared random mask $[r_z] = (r_{z,1}, r_{z,2})$ with P_1, P_2 (in the preprocessing phase), P_1, P_2 can mask $[z]$ by computing $[\delta_z] := [z] - [r_z]$. Then P_1, P_2 recover δ_z, and the parties can obtain a replicated secret sharing of z by setting $[\![z]\!] := (r_{z,1}, \delta_z, r_{z,2})$. We present the protocol in Protocol 3.1.

D. Ideal Functionalities

We use some basic functionalities as our building blocks.

- $\mathcal{F}_{\text{rand}}$: This functionality samples a random r, and distributes a sharing $[\![r]\!]$ to the parties. It can be instantiated non-interactively using PRG and shared keys.
- $\mathcal{F}_{\text{coin}}$: This functionality samples a random r, and sends the public r to the parties. It can be instantiated by first calling $\mathcal{F}_{\text{rand}}$ to obtain $[\![r]\!]$ then opening it.

- $\mathcal{F}_{\text{checkzero}}$: This functionality checks if an additively shared value equals 0 over a ring $\mathbb{Z}_{2^k}$. It receives two shares x_1, x_2 from two parties P_1, P_2, and checks if $x_1 + x_2 =_k 0$. We can use the instantiation from [28].
- $\mathcal{F}_{\text{input}}$: This functionality allows a party to share its input x to the other parties. We can follow previous works [20], [3] to instantiate it. In Appendix D we give the instantiation of $\mathcal{F}_{\text{input}}$ with the ASTRA sharing scheme.
- $\mathcal{F}_{\text{VrfyIP}}$: This functionality is formally introduced by [23], [35], [11], which checks inner-product relations that are secret-shared among the parties and are known by a single party. The single party knowing the secrets, say P_j, acts as a prover, while the other two parties P_{j-1}, P_{j+1} holding the shares act as verifiers. We review the formal description and instantiation of the functionality from [11] in Appendix D. In this work, we will use the adapted functionality $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$ in the covert model, which detects incorrect distributed relations with probability ϵ.

E. Signature Schemes

We use a signature scheme that is existentially unforgeable under chosen-message attacks. We give a formal description of the signature scheme in Appendix B due to page limit.

F. CP-NIZK Proof Schemes

A commitment scheme [52] is a tuple of three algorithms $\mathsf{COM} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open})$ that allows a party to first commit a secret message then opening it (with some auxiliary information). It should be *binding* and *hiding*. The binding property guarantees that it's hard for an adversary to find two different messages with the same commitment, and the hiding property ensures that the commitment doesn't reveal any information about the message. We give a formal definition of commitments in Appendix B-B.

Let R be a polynomial-time decidable relation on the pair (s, w), where s is an NP statement, and w is the witness. We write $\mathsf{R}(s, w) = 1$ if relation R holds on (s, w), and 0 otherwise. A Non-Interactive Zero-Knowledge (NIZK) proof of knowledge is a tuple of algorithms $\mathsf{ZK} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Vrfy})$ that allows a party to prove the satisfaction of a relation R on a public statement s and secret witness w, without revealing any information about w. It should have the properties of *completeness*, *knowledge soundness*, and *zero-knowledge*. Completeness guarantees that an honest verifier will always accept a proof generated by an honest prover on a satisfied relation; knowledge soundness guarantees that a malicious prover cannot fake a proof to convince the verifier if the relation is not satisfied with w; zero-knowledge ensures that no extra information is revealed other than the truth of the statement. Formal definitions can be found in Appendix B-B.

We adopt the notion of *Commit-and-Prove* NIZK (CP-NIZK) proofs of knowledge from LegoSNARK [53], [54]. Given a commitment scheme COM and a relation R, a CP-NIZK proof shows the knowledge of a secret witness w for a public statement s s.t. $\mathsf{R}(s, w) = 1$, and additionally, $w = (u, \mathsf{aux})$ can be split into a message part u and an auxiliary part aux, and the secret message is in accord with a specified commitment. $\mathsf{Com}_u = \mathsf{COM.Commit}(\mathsf{pp}; u)$. More specifically, it demonstrates the following conjunctive relations: $\mathsf{COM.Open}(\mathsf{pp}; \mathsf{Com}_u, u, r) = 1 \wedge \mathsf{R}(s, (u, \mathsf{aux})) = 1$. Existing works [55], [56], [57], [58], [59], [60] have well explored this type of proof system, and can be adopted in this work.

IV. Cheating Detection on Arithmetic Circuits

In this section, we present three cheating detection approaches for arithmetic circuits over $\mathbb{Z}_{2^k}$: the first two are based on smaller ring gaps with high end-to-end efficiency or low communication cost, and the last one focuses on online efficiency.

A. Approaches Based on Smaller Ring Gaps

For SOTA maliciously secure approaches based on MACs [28] and DZKPs [11], they need to increase the underlying ring by a "gap" of at least 40 bits, to ensure 2^{-40} soundness error. Intuitively, we can construct covertly secure protocols by using a smaller ring gap s s.t. the soundness error is bounded by $1-\epsilon$. We use $\epsilon = 50\%$ for a concrete example. Given an arithmetic circuit over $\mathbb{Z}_{2^{64}}$ of size $m = 2^{20}$, we can set $s = 3$ by using MACs [28, Lemma 2]. Compared to the malicious setting where 40-bit statistical security is required (where $s = 46$), the communication cost is reduced from 220 bits to 134 bits per gate. If we use DZKPs, we can set $s = 12$ [11, Lemma G.3] (with the recursion parameter $q = 16$). The amortized communication cost is only 64 bits. We denote the above two approaches by $\mathsf{Cvt\text{-}MAC}^{\mathsf{E2e}}$ and $\mathsf{Cvt\text{-}DZKP}^{\mathsf{Wan}}_{\mathsf{A}}$, respectively. $\mathsf{Cvt\text{-}MAC}^{\mathsf{E2e}}$ has a better end-to-end performance in LAN due to its relatively low computational cost, while $\mathsf{Cvt\text{-}DZKP}^{\mathsf{Wan}}_{\mathsf{A}}$ can outperform $\mathsf{Cvt\text{-}MAC}^{\mathsf{E2e}}$ in WAN due to its low communication burden[2].

Interestingly, even a small relaxed soundness guarantee may significantly lower the demand on the size of ring gap s: even if we set ϵ to 95% in the above example, it suffices to set $s = 8$ by using MACs and $s = 12$ (with a larger recursion factor $q = 32$) by using DZKPs. Meanwhile, we notice another key fact: multiplications over 2^{64+s} (where $1 \le s \le 64$) is approximately $2 \sim 3\times$ slower than that over 2^{64} for modern CPUs, and a small or moderate ring size is sufficient for many real-world applications. For example, previous works [61], [62], [63] have adopted a ring size of approximately $32 \sim 41$ bits. In such cases, to achieve malicious security, the ring has to be expanded to 80 bits using MACs, and approximately 100 bits using DZKPs. However, if we turn to the covert model, we can safely use 8 or 12 extra bits even with $\epsilon = 95\%$, and the final expanded ring is still bounded in 64 bits. This directly leads to a $2 \sim 3\times$ reduction in computational burden.

[2]The soundness analyses and security proofs of $\mathsf{Cvt\text{-}MAC}^{\mathsf{E2e}}$ and $\mathsf{Cvt\text{-}DZKP}^{\mathsf{Wan}}_{\mathsf{A}}$ naturally follow [28], [11], and thus we omit the details here.

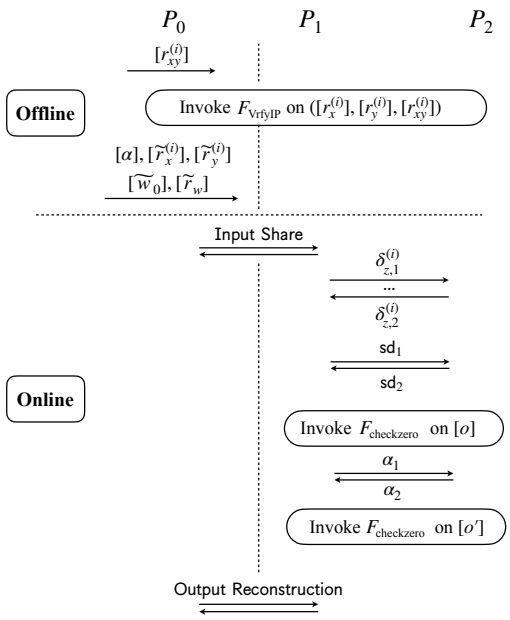


Fig. 2: Workflow of Cvt-MAC$^{\mathsf{On}}$.

B. A New Online-Efficient Approach

We notice that both the MAC-based and DZKP-based approaches are input-dependent protocols, i.e., the verification is conducted entirely after the inputs are known, and thus all the additional overheads are incurred online. This may be suboptimal for settings where parties can preprocess and demand maximal online efficiency. Another previous work ASTRA [3] adopts the relatively online-efficient triple-based approach to achieve malicious security; however, the online cost is still two times that of the semi-honest protocol, and the offline communication burden is heavy. This also holds for another SOTA triple-based work [30]. We thus aim to design an optimized protocol with a lightweight online cost and a moderate offline burden.

We start from the preprocessing model of ASTRA [3]. Our key observation is that the semi-honest protocol of ASTRA [3] is asymmetric: unlike the RSS-based protocol [20] where the parties act all the same, in ASTRA P_0 distributes offline randomness for P_1, P_2 to consume online. Motivated by this, we design different verification mechanisms for P_0 and P_1, P_2 with an asymmetric use of MACs, and achieve a lightweight online cost, with even lower total communication burden than the normal MAC-based protocol [28]. We denote this approach by Cvt-MAC$^{\mathsf{On}}$, and elaborate it below.

Verification on P_0. We now take a deep look at P_0's behavior in the semi-honest protocol of ASTRA. For each multiplication gate computing $[\![z]\!] = [\![x]\!] \cdot [\![y]\!]$, P_0 additively shares $r_{xy} = r_x \cdot r_y$ to P_1, P_2. As P_1, P_2 hold $[r_x], [r_y]$, we can view $([r_x], [r_y], [r_{xy}])$ as an additively shared multiplication triple. Since P_0 is silent during the online circuit evaluation phase, and the output reconstruction procedure is robust, the only way it can corrupt the final outputs is to offer incorrect multiplication triples. Thus, it suffices for P_1, P_2 to check the correctness of these triples to check P_0's behavior. This can be handled by the $\mathcal{F}_{\text{VrfyIP}}^{\epsilon}$ functionality (see Section III-D). Specifically, as P_1, P_2 hold additive shares of these triples which are known by P_0, they can locally convert them into replicated secret shared shares (by locally setting an extra additive share to zero), and then invoke $\mathcal{F}_{\text{VrfyIP}}^{\epsilon}$ to check these triples.

We instantiate $\mathcal{F}_{\text{VrfyIP}}^{\epsilon}$ using DZKPs by adapting the involved parameters as in Section IV-A (a formal description is given in Appendix D). Due to the sublinear communication cost inherited from DZKPs, the verification procedure incurs no (amortized) communication burden.

Verification on P_1, P_2. Recall that to authenticate a computation over $\mathbb{Z}_{2^k}$ using MACs, the ring has to be lifted to $\mathbb{Z}_{2^{k+s}}$ (but only the least significant k bits of the outputs are effective). In the normal usage of MACs, the parties together sample a random shared MAC key $[\![\alpha]\!]$ where $\alpha \in \mathbb{Z}_{2^{k+s}}$; then each input v is shared over $\mathbb{Z}_{2^{k+s}}$ and authenticated by computing $[\![\widetilde{v}]\!] :=_{k+s} [\![\alpha]\!] \cdot [\![v]\!]$. For each multiplication gate computing $[\![z]\!] :=_{k+s} [\![x]\!] \cdot [\![y]\!]$, the parties authenticate z by computing $[\![\widetilde{z}]\!] :=_{k+s} [\![\widetilde{x}]\!] \cdot [\![y]\!]$ (or $[\![\widetilde{z}]\!] :=_{k+s} [\![x]\!] \cdot [\![\widetilde{y}]\!]$, equivalently). After evaluating all multiplication gates, they sample random secret coefficients and perform a random linear combination on all $\{[\![v]\!], [\![z]\!]\}$ and also the authenticated $\{[\![\widetilde{v}]\!], [\![\widetilde{z}]\!]\}$ (by securely computing two inner products), then opening the MAC key α to check the consistency of the two resulting shares.

Our key insight here is that, as we now only concern the correct execution of P_1, P_2, we can view P_0 as a trusted party and let it sample the MAC key α and additively share necessary correlated randomness to P_1, P_2, who then are able to compute an authenticated $\widetilde{z}$ in the ASS form rather than the RSS form. This allows P_1, P_2 to obtain $[\widetilde{z}]$ in a non-interactive way, and thus incurs no extra (amortized) online communication cost over the semi-honest protocol. We detail it below.

(i) Asymmetric Usage of MACs: Observe that for each multiplication gate computing $[\![z]\!] := [\![x]\!] \cdot [\![y]\!]$, we have $\alpha z = \alpha(\delta_x + r_x)(\delta_y + r_y) = \alpha\delta_x\delta_y + \alpha\delta_x r_y + \alpha\delta_y r_x + \alpha r_{xy}$. Let $\widetilde{z} := \alpha z$ be the authenticated value of z. By defining $\widetilde{r}_x := \alpha r_x$, $\widetilde{r}_y := \alpha r_y$, and $\widetilde{r}_{xy} := \alpha r_{xy}$, we have

$$[\widetilde{z}] = \delta_x\delta_y[\alpha] + \delta_x[\widetilde{r}_y] + \delta_y[\widetilde{r}_x] + [\widetilde{r}_{xy}]. \tag{1}$$

Therefore, for P_1, P_2 to obtain $[\widetilde{z}]$, it suffices for P_0 to distribute the authenticated randomness $[\widetilde{r}_x], [\widetilde{r}_y], [\widetilde{r}_{xy}]$ to P_1, P_2, which can be done in the preprocessing phase.

However, the above process still requires an extra communication of $3m$ (where m is the circuit size) ring elements over the semi-honest protocol, the same as the normal usage of MACs. Below we show how to further reduce it to $2m+1$.

(ii) Compressing Communication: Let $\{z^{(i)}\}_{i=1}^m$ be the outputs of m multiplication gates produced in the online phase. Suppose there are m random public coefficients $\{\eta^{(i)}\}_{i=1}^m$ over $\mathbb{Z}_{2^{k+s}}$. We let the parties apply a random linear combination to $\{[\![z^{(i)}]\!]\}_{i=1}^m$ and obtain a combined result $[\![w]\!] := \sum_{i=1}^m \eta^{(i)} \cdot [\![z^{(i)}]\!]$. Now we define $\widetilde{w} := \sum_{i=1}^m \eta^{(i)} \cdot \widetilde{z}^{(i)}$, which is an authenticated value of w. Naively, P_1, P_2 are able to locally obtain $[\widetilde{w}]$ given the authenticated randomness

$[\widetilde{r}_x^{(i)}], [\widetilde{r}_y^{(i)}], [\widetilde{r}_{xy}^{(i)}]$ and the public coefficients. However, we notice that the communication of all $[\widetilde{r}_{xy}^{(i)}]$ can be saved by sharing only one combined value. Specifically, by Equation (1), we have

$$[\widetilde{w}] = \sum_{i=1}^{m} \eta^{(i)}(\delta_x^{(i)}\delta_y^{(i)}[\alpha] + \delta_x^{(i)}[\widetilde{r}_y^{(i)}] + \delta_y^{(i)}[\widetilde{r}_x^{(i)}]) + \sum_{i=1}^{m} \eta^{(i)} \cdot [\widetilde{r}_{xy}^{(i)}].$$

Let $\widetilde{w}_0 := \sum_{i=1}^{m} \eta^{(i)} \widetilde{r}_{xy}^{(i)}$. Clearly P_0 can compute and distribute $\widetilde{w}_0$ in the preprocessing phase. Then P_1, P_2 can locally derive $[\widetilde{w}]$ given $\{[\widetilde{r}_x^{(i)}]\}_{i=1}^m, \{[\widetilde{r}_y^{(i)}]\}_{i=1}^m$ and $[\widetilde{w}_0]$, which indicates P_0 only needs to share $2m+1$ elements.

The public random coefficients should be known by P_1, P_2 after all multiplication gates are computed. The parties can call $\mathcal{F}_{\text{coin}}$ to generate them; however, we notice that it can be adapted to the preprocessing model by P_0 first sharing a random seed sd to P_1, P_2 together with a commitment of sd, and derive all $\{\eta^{(i)}\}_{i=1}^m$ using PRG with sd in the preprocessing phase, and then P_1, P_2 can recover sd to derive $\{\eta^{(i)}\}_{i=1}^m$ after computing all multiplication gates in the online phase. In this way, P_0 can still keep silent during the online phase until the reconstruction procedure.

(iii) Consistency Check: The remaining task is to check consistency between $[\![w]\!]$ and $[\widetilde{w}]$. To this end, we let P_1, P_2 derive another additive sharing of $\widetilde{w}' := \alpha \cdot w$ from $[\alpha]$ and $[\![w]\!]$ with the help of P_0. Given that $[\widetilde{w}'] = [\alpha](\delta_w + [r_w]) = \delta_w[\alpha] + [\alpha \cdot r_w]$ and $r_w = \sum_{i=1}^{m} \eta^{(i)} r_z^{(i)}$, we ask P_0 to compute r_w and then share $[\widetilde{r}_w] := [\alpha \cdot r_w]$ in the preprocessing phase; then P_1, P_2 are able to locally compute $[\widetilde{w}']$ in the online phase. After this, the final consistency check can be performed by calling $\mathcal{F}_{\text{checkzero}}$ on $[o] := [\widetilde{w}] - [\widetilde{w}']$.

(iv) Additional Check on P_0***:*** Note that a corrupted P_0 may share incorrect $\{\widetilde{r}_x^{(i)}\}_{i=1}^m, \{\widetilde{r}_y^{(i)}\}_{i=1}^m, \widetilde{w}_0$ or $\widetilde{r}_w$ to P_1, P_2, and try to extract extra information according to the behaviors of P_1, P_2 (i.e., whether they abort or not). Thus, it's necessary for P_1, P_2 to additionally check the authenticated randomness. Similarly, we can invoke $\mathcal{F}_{\text{VrfyIP}}^{\epsilon}$ on these triples. However, to avoid heavy computational burdens in the preprocessing phase, we opt for checking these values online in a more lightweight way: after all the multiplication gates are checked, we let P_1, P_2 directly open α to check the consistency of these values. The communication cost can be saved by applying a random linear combination and then invoking $\mathcal{F}_{\text{checkzero}}$ on the combined result. We claim that opening α doesn't violate privacy even if P_1 or P_2 is corrupted, as the procedure is performed after authenticating P_1 and P_2's computations. If a corrupted P_1 or P_2 can successfully cheat in the previous steps (by introducing some additive error), it must have already guessed necessary information about α. Thus the revealing of α doesn't give it any extra information.

We present the workflow of our protocol in Figure 2, and give a formal description as well as a security proof in Appendix E. In summary, our protocol only requires communicating about $3m$ in the offline phase and $2m$ ring elements in the online phase using PRG (as a contrast, the original MAC-based protocol [28] incurs a total communication of $6m$ ring elements in the online phase).

V. Cheating Detection on Binary Circuits

Unlike arithmetic circuits, using MACs over a larger ring to authenticate AND gate computation would significantly degrade the online efficiency (as ring operations are more expensive than Boolean ones). Therefore, we opt for the SOTA triple-based approach [13], and design a new cut-and-choose paradigm tailored for the covert model with better online/end-to-end efficiency. We also adapt the DZKP-based approach [32], [23] to the covert model, considering its optimal sublinear communication cost.

A. Adapting DZKPs with Smaller Field Size

As DZKPs cannot directly apply to $\mathbb{Z}_2$ due to their large soundness error, previous maliciously secure protocols [32], [23] proposed to lift the verification onto a sufficiently large binary extension field $GF(2^n)$ to guarantee negligible soundness error. By using the recursive techniques, the soundness error is bounded by $O(\frac{\log m}{|\mathbb{F}|})$ [23, Theorem 7.1]. In our case, we only need the extension field $GF(2^n)$ to be moderately large such that the soundness error is bounded by $1-\epsilon$. Concretely, setting $n = 4, 8$ or 16 is sufficient for most choices of ϵ. We denote the adapted protocol by $\mathsf{Cvt\text{-}DZKP}_{\mathsf{B}}^{\mathsf{Wan}}$.

Note that the recent alternative DZKP protocol [12] over Mersenne prime fields (recall Section II-A) cannot apply, as it has a restriction on circuit size to avoid overflow, and a small field size here would significantly limit practical applications.

B. Covertly Secure Cut-and-Choose Method

We now introduce our triple-based approaches with a new cut-and-choose paradigm tailored for the covert model.

Achieving Efficient Online with Lower Offline Costs. We first review how to compute an AND gate using a random triple. Assume the parties are given a random AND triple $([\![a]\!], [\![b]\!], [\![c]\!])$ where $c = a \cdot b$; to compute an AND gate $[\![z]\!] := [\![x]\!] \cdot [\![y]\!]$, the parties locally compute $[\![\rho]\!] := [\![x]\!] \oplus [\![a]\!]$ and $[\![\sigma]\!] := [\![y]\!] \oplus [\![b]\!]$, then opening ρ, σ and computing $[\![z]\!]$ by

$$[\![z]\!] = \rho \cdot \sigma \oplus \rho \cdot [\![b]\!] \oplus \sigma \cdot [\![a]\!] \oplus [\![c]\!]. \tag{2}$$

Note that the online computation is (relatively) lightweight given prepared random triples, and thus we use this approach to construct our online-efficient protocol $\mathsf{Cvt\text{-}Triple}^{\mathsf{On}}$. Since the offline cost is heavy, we aim to optimize the offline procedure with lower costs.

First recall that the previous work [21] proposed to use the classical cut-and-choose paradigm to generate offline random triples. Assume there are N AND gates in the Boolean circuit. The parties first run the RSS-based semi-honest AND protocol to generate $NB + C$ initial random triples where B, C are parameters. Then they randomly shuffle these triples, open and check the first C of them, and then divide the unopened triples into N buckets, each of size B. For each bucket, the parties perform the bucket-check procedure, where the first triple is checked using the remaining $B-1$ triples respectively.[3] If all

[3]To check the first triple (say $([\![x]\!], [\![y]\!], [\![z]\!])$) with another one (say, $([\![a]\!], [\![b]\!], [\![c]\!])$), the parties compute the RHS of Equation (2), and check if the difference of the result and $[\![z]\!]$ is a sharing of zero.

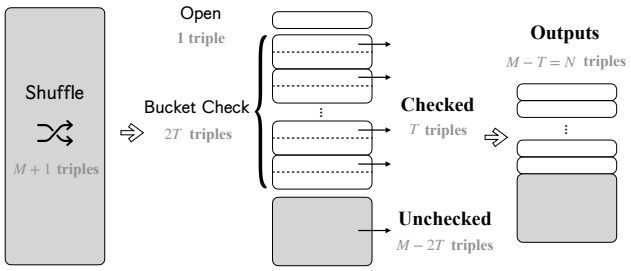


Fig. 3: Triple generation procedure of $\mathsf{Cvt\text{-}Triple}^{\mathsf{On}}$. ($M, T$ are parameters s.t. $M - T = N$.)

the checks are passed, they output all the first triples in these buckets, yielding N validated triples in total. As analyzed in [21], the soundness error is bounded by $2^{-\sigma}$ provided that $\sigma \leq \frac{(BN+B)!}{N \cdot B! \cdot (BN)!}$ and $C = B$. For example, for $N = 2^{20}$, we can set $B = C = 3$ to achieve $\sigma = 40$ bit security, and this requires a (global) communication of 21 bits per gate.

In [21], every retained triple belongs to a bucket and is checked against $B-1$ other triples. In the covert model, merely choosing the smallest bucket size $B = 2$ still checks every retained triple. It reduces offline communication to 12 bits per gate but leaves soundness error roughly $1/N$, which is unnecessarily small for moderate N and a target deterrence such as 50%. Our change is instead to bucket-check only a random subset and deliberately retain unchecked triples. Choose M, T with $M - T = N$ and $0 < T \leq M/2$, generate and shuffle $M+1$ semi-honest triples, and open the first triple. Of the remaining M triples, place only $2T$ into T buckets of size 2 and apply the unchanged check from [21]. If all checks pass, retain one triple per bucket together with the remaining $M-2T$ unchecked triples, yielding $M-T = N$ output triples.

For intuition, ignoring rounding, the conservative choice $M = (1+\epsilon)N$ and $T = \epsilon N$ uses ϵN triples to check a uniformly selected ϵN-subset of N output candidates. Each bad candidate is therefore selected with probability ϵ. The tighter parameter choices below follow from the exact analysis rather than this single-bad-triple intuition.

We give an illustration of the above procedure in Figure 3. Unlike the analysis of [21], our analysis must account for unchecked bad triples and pairs of bad triples that pass a bucket check. In the ideal-shuffle and ideal-consistency hybrid, Appendix F proves the exact worst-case algebraic soundness error

$$\epsilon_{\mathsf{snd}}(M, T) = \frac{\max\{M - 2T, 1\}}{M + 1}$$

over any number of bad initial triples. We choose M, T so that this error is at most $1 - \epsilon$. Concretely, for $\epsilon = 50\%$, we can set $M = \frac{4(N+1)}{3}$ and $T = \frac{M}{4} + 1$. Compared to the malicious setting, the offline communication cost is reduced from 21 bits to 6 bits. For $\epsilon = 75\%$ and $\epsilon = 87.5\%$, we can set $M = \frac{8N}{5}$ and $\frac{16N}{9}$, with $T = \frac{3M}{8}$ and $\frac{7M}{16}$, and the offline communication cost is roughly 8 and 10 bits, respectively.

Achieving Higher End-to-End Efficiency. The previous work [22] has proposed a variant of the cut-and-choose approach that has better end-to-end efficiency, by using a smaller bucket size in the preprocessing phase at the cost of a slightly higher online cost. Specifically, the parties only generate $N(B-1)+C$ random triples, and perform the bucket-check procedure as in [21]. Unlike [21], now the output triples are not guaranteed to be validated (the soundness error is not negligible, as B is now reduced by 1). Then in the online phase, the parties first compute N AND gates using the semi-honest multiplication protocol, then apply a random shuffle to these online triples, which are then checked with those N random offline triples. Intuitively, the random shuffle applied to the online triples guarantees that the overall soundness guarantee is nearly the same as [21].

Following [22], we design a variant of the cut-and-choose protocol in the covert model with higher end-to-end efficiency, denoted by $\mathsf{Cvt\text{-}Triple}^{\mathsf{E2e}}$. Similarly, we reduce the number of the initial triples by N: instead of $M+1$, we now let the parties generate $M+1-N = T+1$ (as $M-T = N$) initial triples and randomly shuffle them. Then the parties open and check the first triple as above; the remaining T triples will be used for later online verification. After computing N online triples, the parties apply a random shuffle to them, and divide these triples into two parts: the first T online triples will be paired and checked with those T random offline triples, and the remaining $N - T = M - 2T$ online triples will not be checked. In this way, the overall soundness guarantee is still guaranteed to be at least ϵ with proper parameters, and the number of initial triples to be generated is directly reduced by N, leading to higher end-to-end efficiency. The soundness analysis of $\mathsf{Cvt\text{-}Triple}^{\mathsf{E2e}}$ naturally follows that of $\mathsf{Cvt\text{-}Triple}^{\mathsf{On}}$.

Remark 5.1: Our new cut-and-choose paradigm tailored for the covert model can be extended to more types of correlated randomness beyond AND triples, such as (e)daBits [64], truncation pairs [2], dot-product/matrix triples, and so on. This enables us to construct concretely efficient covert protocols for mixed circuits.

VI. Cheater Identification

Previous sections establish how to detect a cheating behavior, but a failed check may not reveal which party caused it. In this section, we show how to identify cheaters with public verifiability. Unlike previous covertly secure works [19], [18] and maliciously secure works with cheater identification [24], [25], [26], [27], we observe that in the three-party honest-majority setting, we can identify cheaters without introducing any other cryptographic primitives (other than signatures and commitments) for some protocols with specific properties. For the other protocols, we can identify cheaters by using CP-NIZK proofs.

A. Lightweight Cheater Identification

Our key observation is that in the three-party honest-majority setting, once an honest party is identified after an abortion, it can act as a judge and identify the cheater by checking the other parties' views. The two parties reveal their views to the judge (privacy is preserved since the judge is honest). To prevent forged views, each party commits to its input and random tape and signs every message; binding

commitments and signatures make inconsistencies detectable. We show how to identify an honest party for specific protocols.

To identify the cheater, we use the technique of [23]: each party authenticates the final outputs under its own secret MAC key, and the protocol computes both the outputs and their MACs. If P_i sends an incorrect share to P_j, it must forge a valid MAC under P_j's key to avoid detection; otherwise P_i is identified. Since authentication is only applied to outputs, the overhead is negligible.

Identifying an Honest Party for DZKP-based Approach. We follow [23] to identify an honest party for DZKP-based approaches. If abortion happens when verifying P_j as the prover, then either P_j is malicious or a verifier cheats. We form a conflict pair by letting P_j accuse one verifier P_t for $t \in \{j-1, j+1\}$, so (P_j, P_t) contains a cheater. In the latter case, we use "re-computable verification" [23]: an honest prover can re-compute both verifiers' messages and pinpoint the cheating verifier. With an honest majority, the remaining verifier (not accused) is honest.

Identifying an Honest Party for Asymmetric MAC-based Approach. For the asymmetric MAC-based approach, we make use of the asymmetric preprocessing property to identify an honest party: if P_0's behavior has been successfully verified, then P_0 can be identified as an honest party, otherwise we let P_0 accuse another party, and the left one party will be viewed as honest. Note that if a malicious P_1 or P_2 tries to cheat during the verification of P_0, it can only manipulate two procedures: (1) the offline DZKP verification on P_0's randomness $\{([r_x^{(i)}], [r_y^{(i)}], [r_{xy}^{(i)}])\}_{i=1}^m$, or (2) the online consistency check of the authenticated randomness $\{[\widetilde{r_x^{(i)}}]\}_{i=1}^m, \{[\widetilde{r_y^{(i)}}]\}_{i=1}^m, [\widetilde{w}_0]$ and $[\widetilde{r}_w]$. Under any circumstance, an honest P_0 can always tell the cheater since it knows all the involved randomness.

B. General Cheater Identification

The above cheater identification approach cannot apply to symmetric MAC-based and triple-based approaches, as identifying the honest party is challenging without additional primitives. For the symmetric MAC-based and triple-based approaches, our idea is to enforce the parties to generate a zero-knowledge proof showing their honest behaviors after an abortion happens, i.e., proving that all its sent messages are faithfully computed according to the protocol specification w.r.t. its input, random tape and all the received messages; then the one who fails to provide a valid proof will be accused as the cheater. We design a general cheater identification mechanism using CP-NIZK proofs: upon abortion, each party generates a zero-knowledge proof demonstrating faithful execution with respect to commitments (to its input and random tape) and signed messages. Due to page limits, we defer the detailed protocol to [65, Appendix G].

C. Achieving Public Verifiability

We observe that in the three-party honest-majority setting, public verifiability follows naturally after identifying the cheater. Suppose P_j is identified to be malicious. As the other two parties must be honest, we can let each honest party sign an accusation message $(\mathsf{corrupted}, i)$ and exchange it with each other, and the Blame algorithm just outputs the two signed accusation messages as a certificate. After this, any third party running the Judge algorithm just validates the two signatures w.r.t. the accusation message; as the signatures are generated by the honest parties, the check must pass, and the Judge algorithm will output the identity of the cheating party correctly. In [65, Appendix G] we give formal descriptions and security proofs of the two algorithms.

VII. ECHO: Covertly Secure PPML Framework

In this section, we present a covertly secure PPML framework with public verifiability. To this end, we adapt existing protocols for PPML primitives to covert security.

A. Fixed-Point Representation

We use fixed-point arithmetic, where a k-bit signed fixed-point $\widetilde{x}$ is represented with two parts: an e-bit integer part and a d-bit decimal part s.t. $k = e + d$. We map the fixed-point number $\widetilde{x}$ into an element x in $\mathbb{Z}_{2^k}$ by setting $x := 2^d \cdot \widetilde{x}$, and addition and multiplication operations are mapped accordingly.

B. Share Conversion

In the PPML framework, we sometimes need to transform the shares back and forth over $\mathbb{Z}_2$ and $\mathbb{Z}_{2^k}$, referred to as A2B and B2A, respectively. We consider a typical case of B2A, where a shared bit is converted into the arithmetic domain, commonly referred to as Bit2A.

We follow the previous work [2] to transform arithmetic shares into binary shares (A2B). Given an arithmetic sharing $[\![x]\!]^{\mathsf{A}} = (x_0, x_1, x_2)$, to compute a binary secret sharing of x, we let each party P_j decompose x_j into bits $x_j^{(0)}, \cdots, x_j^{(k-1)}$, and then the parties locally generate the binary sharing of these bits, denoted by $[\![\vec{x}_j]\!]^{\mathsf{B}} := ([\![x_j^{(0)}]\!]^{\mathsf{B}}, \cdots, [\![x_j^{(k-1)}]\!]^{\mathsf{B}})$. Then they run binary adders to compute $x_0 + x_1 + x_2$, taking input as binary sharings $[\![\vec{x}_0]\!]^{\mathsf{B}}, [\![\vec{x}_1]\!]^{\mathsf{B}}$ and $[\![\vec{x}_2]\!]^{\mathsf{B}}$. By carefully designing the binary adders (i.e., using fuller adders and parallel prefix adders), the previous work [2] can reduce three addition operands into two in one round, and obtain the final binary sharing in $O(\log k)$ rounds, with $O(k)$ communication cost. In the covert model, we can apply our covertly secure protocol for binary circuits to obtain a covertly secure A2B protocol.

For Bit2A, we utilize the fact that for any $x, y \in \mathbb{Z}_2$, we have $x \oplus y = x + y - 2xy$. Given a shared bit $[\![b]\!]^{\mathsf{B}} = (b_0, b_1, b_2)$, we have $b = b_0 \oplus b_1 \oplus b_2$. Thus, to obtain an arithmetic sharing of b, we let the parties first locally generate replicated secret sharings $[\![b_0]\!]^{\mathsf{A}}, [\![b_1]\!]^{\mathsf{A}}, [\![b_2]\!]^{\mathsf{A}}$ over $\mathbb{Z}_{2^k}$, and then invoke the secure arithmetic multiplication protocol to compute each XOR operation. This requires two secure multiplications, and we can directly apply our covertly secure protocol here.

C. Truncation

Truncation is typically invoked after arithmetic multiplications to adjust the decimal part of the results. It takes as input a sharing $[\![x]\!]^{\mathsf{A}}$ of integer $x \in \mathbb{Z}_{2^k}$, and outputs a

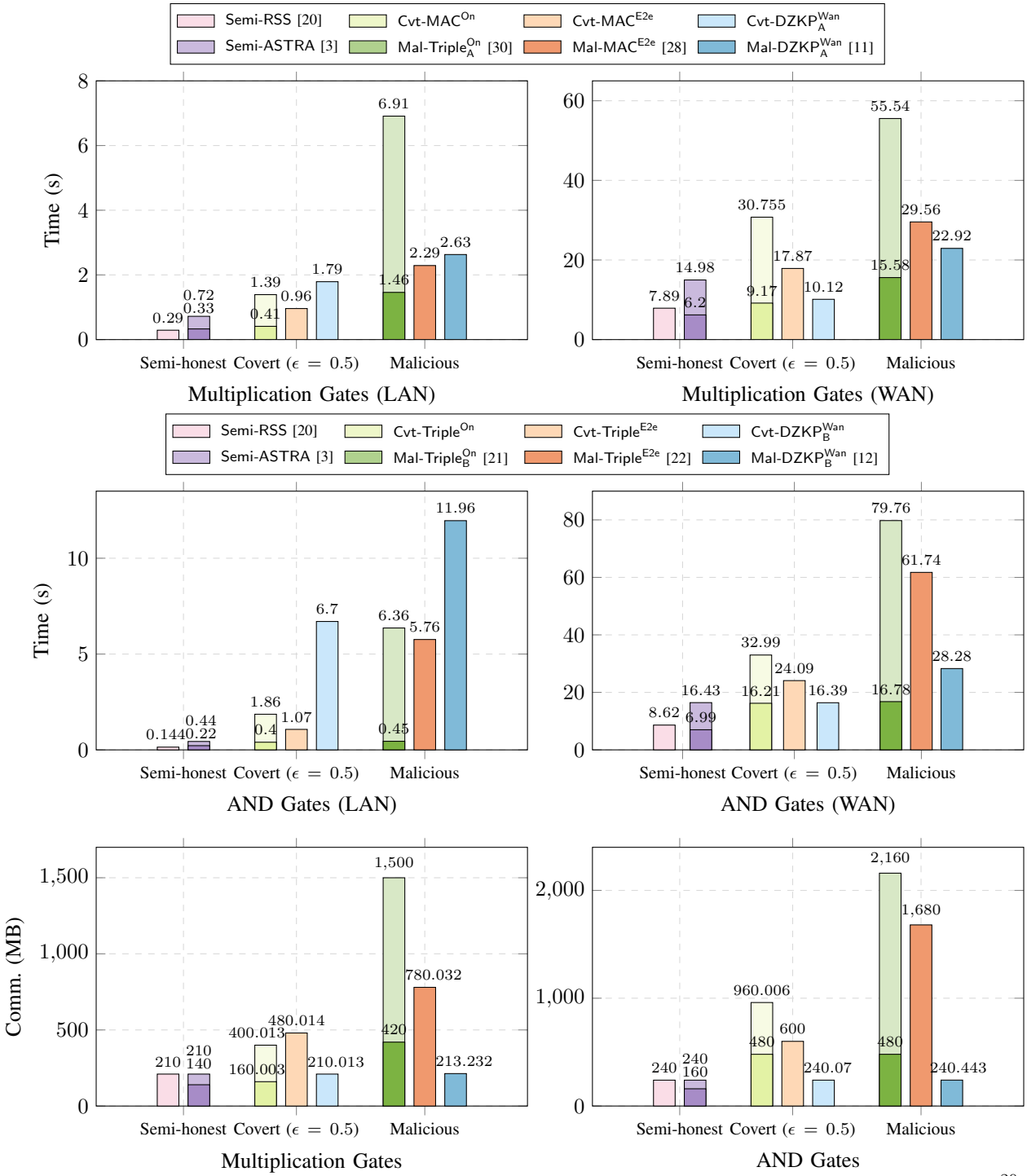


Fig. 4: Time (s) and communication cost (MB) for computing depth-10 circuits where each layer consists of 2^{20} Multiplication Gates over $\mathbb{Z}_{2^{52}}$ or 2^{20} 64-bit parallel AND gates. For protocols in the preprocessing model, the darker (bottom) portion of each bar represents the online cost, while the total bar represents the end-to-end cost. The legends for the communication cost plots are reused from the corresponding time cost plots.

sharing of x' where $x' = \lfloor x/2^d \rfloor$, and d is the bit length of the decimal part in the fixed-point representation. In this framework, we adopt probabilistic truncation, where it rounds $x/2^d$ to $x' = \lfloor x/2^d \rfloor + w$ where $w \in \{0, 1\}$ is biased towards the nearest integer to $x/2^d$.

In this work, we opt for probabilistic truncation based on the online-efficient approach from prior works [66], [29]. We assume a "small-gap" on the truncated value x to guarantee the correctness of the truncation results [66], [29], [10], [5], i.e., by requiring that the most significant bit of x is zero. (this can be done by adding x with 2^{k-1} before truncation then subtracting 2^{k-d-1} from the truncated value afterward [66], [29]). This approach involves the parties preparing random shared bits $[\![r^{(i)}]\!]$ and $r = \sum_{i=1}^{k} [\![r^{(i)}]\!]$ over $\mathbb{Z}_{2^k}$ in the preprocessing phase. Following [66], we let two parties propose random bits and XOR them using our covert security protocol, and additionally check each secret is from $\mathbb{Z}_2$ using the verification techniques from [10] (more details refer to [66], [10]). To address the security concerns raised by Li et al. [12], we adopt the approach from [67], with a slight modification to the functionality to ensure security.

D. Comparison

The comparison operation is commonly used in activation functions (like ReLU) and Max Pooling layers in neural networks. A secure comparison protocol over ring $\mathbb{Z}_{2^k}$ takes as input two sharings $[\![a]\!], [\![b]\!]$, and outputs the sharing of z where $z = 0$ denotes $a \geq b$ and $z = 1$ otherwise. We follow previous works [2] to decompose secure comparison into the following steps: each party locally computes $[\![z]\!]^{\mathsf{A}} = [\![a]\!]^{\mathsf{A}} - [\![b]\!]^{\mathsf{A}}$; then the value $[\![z]\!]^{\mathsf{A}}$ is converted to a binary share $[\![z]\!]^{\mathsf{B}}$ via the A2B operation, and the most significant bit $[\![z_0]\!]^{\mathsf{B}}$ is extracted and converted back to arithmetic sharing using Bit2A, yielding $[\![z_0]\!]^{\mathsf{A}}$ as the output of the comparison.

E. (E)daBits

A daBit (doubly-authenticated Bit) [68] is a random bit shared over both the arithmetic and the binary domain, i.e., $([\![r]\!]^{\mathsf{A}}, [\![r]\!]^{\mathsf{B}})$ where $r \in \{0, 1\}$. An EdaBit (Extended daBit) [64] is a generalization of daBit, consisting of n random bits shared over the binary domain $[\![r^{(0)}]\!]^{\mathsf{B}}, \cdots, [\![r^{(n-1)}]\!]^{\mathsf{B}}$ and an integer r shared over the arithmetic domain s.t. $r = \sum_{i \in [n]} r^{(i)} \cdot 2^i$. (E)dabits are commonly used in PPML to help evaluate non-linear functions, such as truncation, comparison, share conversion, etc. The generation of (E)dabits typically involves the cut-and-choose paradigm, and in the covert model we can accordingly apply our cut-and-choose paradigm (described in Section V-B) to the generation procedures.

VIII. Evaluation

A. Experimental Setup

We implemented our covertly secure protocols based on the MP-SPDZ framework [29]. [4]. Our experiments were run on three different cloud instances under a LAN, each equipped with a 32-core Intel(R) Xeon(R) Platinum 6982P-C 3GHz CPU and 256GB memory. We report results in the LAN and an emulated WAN with 100Mbps bandwidth and 40ms one-way latency using the Linux command *tc*. All the microbenchmarks were run in a single thread, and all PPML tasks were run in 16 threads.

[4]Our code is in https://anonymous.4open.science/r/Covert-3PC-5E45.

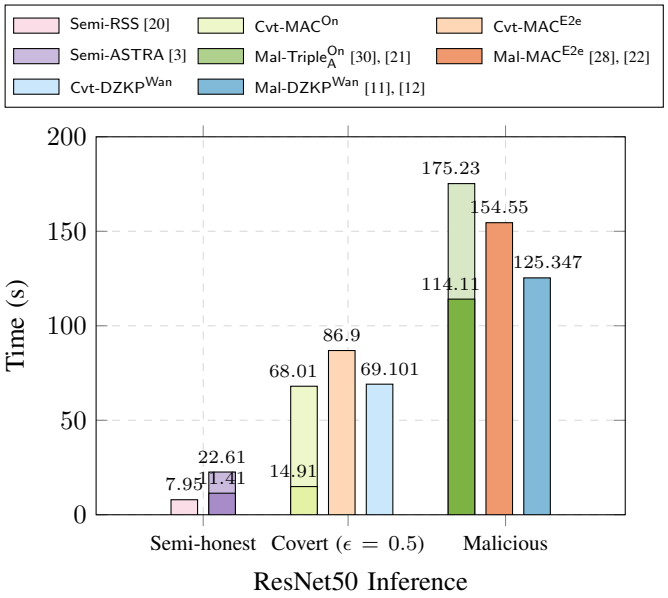


Fig. 5: LAN PPML (ResNet50) runtime. For protocols with offline/online phases (ASTRA, Cvt$^{\mathsf{ON}}$, Mal-MAC$^{\mathsf{E2e}}$+Mal-Triple$^{\mathsf{On}}_{\mathsf{B}}$), the darker portion denotes online cost; the full bar indicates end-to-end cost.

B. Microbenchmark Results

We compare with maliciously secure MAC-based [28], DZKP-based [11] and triple-based [30] protocols for arithmetic circuits (denoted by Mal-MAC$^{\mathsf{E2e}}$, Mal-DZKP$^{\mathsf{Wan}}_{\mathsf{A}}$, Mal-Triple$^{\mathsf{On}}_{\mathsf{A}}$ respectively). For Boolean circuits, we compare with maliciously secure triple-based [21], [22] and DZKP-based protocols [12] (denoted by Mal-Triple$^{\mathsf{On}}_{\mathsf{B}}$, Mal-Triple$^{\mathsf{E2e}}$, Mal-DZKP$^{\mathsf{Wan}}_{\mathsf{B}}$ respectively). We show the time cost and communication cost of the tested protocols in Figure 4, and give detailed analyses below.

Multiplication Gates.

In the LAN setting, the online time of Cvt-MAC$^{\mathsf{On}}$ is only $1.42\times$ and $1.24\times$ of the semi-honest protocols Semi-RSS and Semi-ASTRA, respectively; compared to the SOTA malicious protocol Mal-MAC$^{\mathsf{E2e}}$, Cvt-MAC$^{\mathsf{On}}$ achieves a $5.59\times$ speedup in the online phase. This gain comes from two parts. Specifically, by reducing the ring gap from $\mathbb{Z}_{2^{128}}$ to $\mathbb{Z}_{2^{64}}$, Cvt-MAC$^{\mathsf{E2e}}$ achieves a $2.38\times$ speedup compared to Mal-MAC$^{\mathsf{E2e}}$. And by moving computation from online to pre-

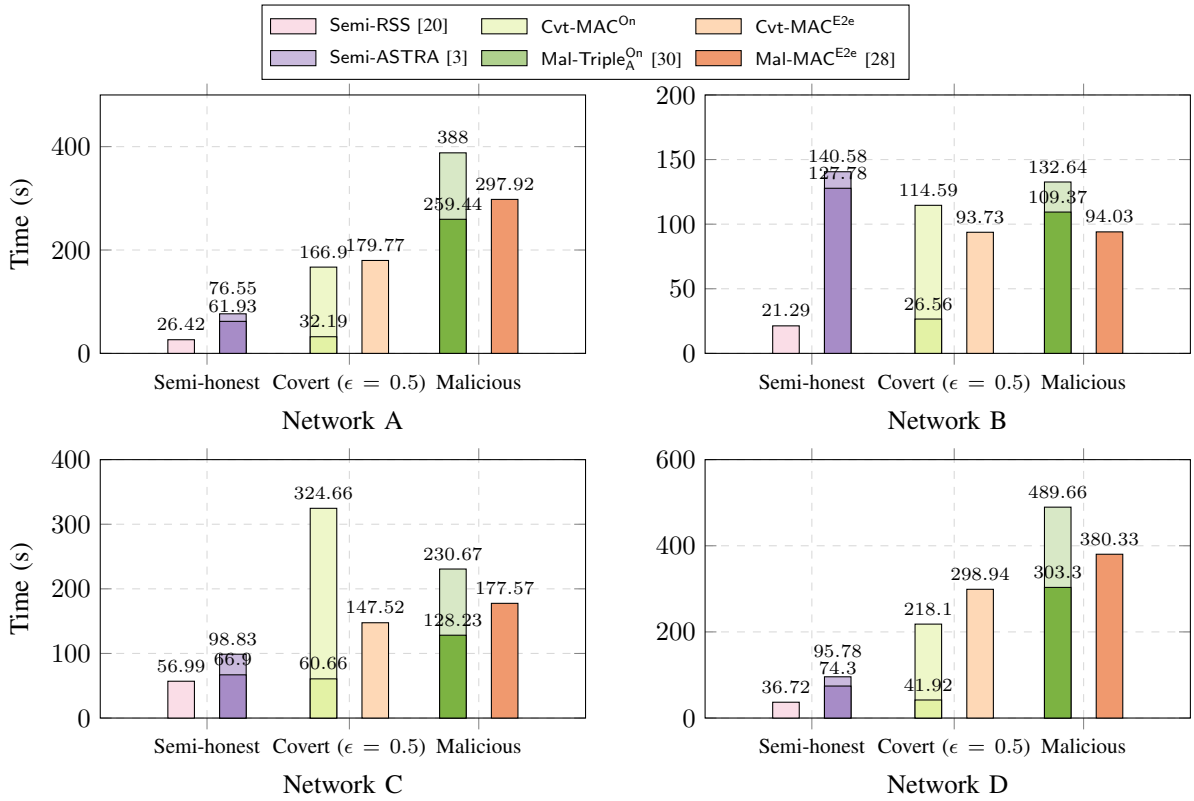


Fig. 6: LAN PPML runtime for training on different networks. For protocols with offline/online phases (ASTRA, Cvt-MAC$^{\text{On}}$, Mal-Triple$^{\text{On}}_{\text{A}}$), the darker portion denotes online cost; the full bar indicates end-to-end cost.

processing, Cvt-MAC$^{\text{On}}$ achieves a $2.34\times$ speedup compared to Cvt-MAC$^{\text{E2e}}$.

Under the WAN setting, as Cvt-MAC$^{\text{On}}$'s online communication remains lightweight, it incurs only $1.16\times$ online overhead over Semi-RSS and $1.04\times$ over Semi-ASTRA. For end-to-end performance, DZKP-based protocols outperform the others due to their low communication burden, and our covertly secure protocol Cvt-DZKP$^{\text{Wan}}_{\text{A}}$ is $2.27\times$ faster than the maliciously secure counterpart Mal-DZKP$^{\text{Wan}}_{\text{A}}$ due to the reduction of the size of the computation ring.

The communication cost of Cvt-MAC$^{\text{E2e}}$ is only $0.62\times$ of Mal-MAC$^{\text{E2e}}$ by using a smaller ring size. Moreover, the online communication cost of Cvt-MAC$^{\text{On}}$ is just $0.76\times$ and $1.14\times$ that of Semi-RSS and Semi-ASTRA, where the overhead compared to Semi-ASTRA is introduced by the ring gap; compared to the maliciously secure protocol Mal-Triple$^{\text{On}}_{\text{A}}$, the online and offline communication cost of Cvt-MAC$^{\text{On}}$ is only $0.38\times$ and $0.22\times$, respectively. To be more specific, it reduces $1.63\times$ commmunication by reducing the ring gap and $3\times$ communication by our efficient online design.

AND Gates. In the LAN setting, Cvt-Triple$^{\text{On}}$ and the SOTA online-efficient malicious protocol Mal-Triple$^{\text{On}}_{\text{B}}$ have similar online efficiencies (as their online procedures are the same); nevertheless, Cvt-Triple$^{\text{On}}$ reduces the offline time by $2.89\times$ due to our tailored cut-and-choose paradigm that checks fewer triples. Compared with the covertly secure protocol [17], which runs 130.9s with 30.07GB communication cost, our protocol is $122\times$ faster, and the communication cost is $51.32\times$ lower. Note, however, that [17] incurs substantial overhead for its blame and judge procedures regardless of whether cheating is detected, whereas our costs for blame and judge procedures are incurred only after cheating is detected (we show these costs in Section VIII-D).

In the WAN setting, the DZKP-based protocols show advantages: the overall running time of our covertly secure protocol Cvt-DZKP$^{\text{Wan}}_{\text{B}}$ is only $1.9\times$ of Semi-RSS, and is even slightly faster than Semi-ASTRA; compared with the maliciously secure counterpart Mal-DZKP$^{\text{Wan}}_{\text{B}}$, Cvt-DZKP$^{\text{Wan}}_{\text{B}}$ is $1.73\times$ faster due to the lower computation cost.

On communication, both Cvt-Triple$^{\text{On}}$ and Cvt-Triple$^{\text{E2e}}$ benefit from our new cut-and-choose paradigm, Cvt-Triple$^{\text{On}}$ reduces the communication cost of Mal-Triple$^{\text{On}}_{\text{B}}$ by $4.2\times$ in the offline phase, and Cvt-Triple$^{\text{E2e}}$ reduces the overall communication cost of Mal-Triple$^{\text{E2e}}$ by $2.8\times$.

C. PPML Task Results

We compare with semi-honest protocols ([20], [3]) and malicious protocols ([28] for arithmetic and [21], [22] for Binary). We evaluate ResNet50 inference in Figure 5 and training on Network A, B, C, and D in Figure 6.

ResNet50 Inference.

As shown in Figure 5, we evaluate ResNet50 inference on the ImageNet dataset under the LAN setting. Since the MAC-based protocol doesn't support probabilistic truncation and share splitting in MP-SPDZ, we ran the default compile setting for Cvt-MAC$^{\text{E2e}}$ and Mal-MAC$^{\text{E2e}}$ [28], [22], which are slower than circuit with other settings. Our online-efficient protocol

Cvt-MAC$^{\mathsf{On}}$ achieves less than $1\times$ overhead compared to the semi-honest baselines Compared to the maliciously secure protocol Mal-Triple$^{\mathsf{On}}$ [30], [21], our Cvt-MAC$^{\mathsf{On}}$ achieves a substantial $7.65\times$ speedup. For end-to-end efficiency, our protocol Cvt-MAC$^{\mathsf{E2e}}$ provides a $1.77\times$ speedup compared to the maliciously secure counterpart Mal-MAC$^{\mathsf{E2e}}$. For DZKP-based protocols Cvt-DZKP$^{\mathsf{Wan}}$ and Mal-DZKP$^{\mathsf{Wan}}$ [12], [11], our protocols achieve a $1.81\times$ speedup.

NN Training.

As shown in Figure 6, we evaluate our protocols on four neural network training tasks (Networks A, B, C, and D) on MNIST dataset. We trained one epoch with batch size 128. Specifically, for Network B and C, we used only 5000 training samples to reduce the training time.

Our online-efficient protocol Cvt-MAC$^{\mathsf{On}}$ achieves overhead ranging from 6% to 25% compared to the semi-honest baseline Semi-RSS, demonstrating that covert security adds minimal cost over semi-honest protocols in practical PPML workloads. Compared to the maliciously secure protocol Mal-Triple$^{\mathsf{On}}$, our Cvt-MAC$^{\mathsf{On}}$ is up to $8.06\times$ faster, showing significant performance advantages. For end-to-end efficiency, our Cvt-MAC$^{\mathsf{E2e}}$ achieves speedups of at most $1.66\times$ compared to the maliciously secure counterpart Mal-MAC$^{\mathsf{E2e}}$.

We note that in some experiments, Semi-ASTRA exhibits slower performance than certain malicious baselines, which might be attributed to implementation-specific issues in MP-SPDZ rather than fundamental protocol limitations.

D. Cheater Identification

We evaluate the cost of our cheater identification mechanism using ZKPs based on the VOLE (vector oblivious linear evaluation)-in-the-Head technique [69], [70]. As we evaluated, proving the execution of a Boolean circuit consisting of 2^{20} 64-bit parallel AND gates takes 43.42s, and requires approximately 5 bits of communication per AND gate, yielding a proof size of 1.25GB per party. Recall that [17] takes 130.9s for such a circuit and communicates 30.07GB. We thus conclude that even in the pessimistic case, our protocol still outperforms [17] in both running time and communication. If we switch the ZKP backend to a succinct CP-NIZK scheme such as LegoSNARK [53], the communication cost will be further significantly reduced. For example, for an arithmetic circuit of 2^{20} multiplication gates, our experimental results show it only takes less than one millisecond to verify a proof using LegoSNARK [53], and the proof size is just 34 bytes. The prover time is relatively high, about tens of seconds. However, we emphasize that this cheater identification procedure is only conducted when using the symmetric MAC-based and triple-based approaches after a cheating attempt is detected. In other words, our general cheater identification protocol incurs zero overhead in the optimistic case.

IX. Conclusion

In this work, we thoroughly investigate the methods to achieve covert security for secure 3PC, and introduce concretely efficient covertly secure 3PC protocol family ECHO. ECHO includes six protocols for both arithmetic and Boolean circuits, tailored for end-to-end efficiency, online efficiency, and communication efficiency. We utilize commit-and-prove zero-knowledge proofs to achieve public verifiable cheater identification, which incurs no extra costs in the optimistic case. We implemented ECHO and proved that our protocol achieves covert security and is highly efficient, outperforming previous maliciously secure protocols.

Acknowledgment

The authors used OpenAI's GPT-5 series to assist with the implementation of ECHO protocols, with debugging parts of the accompanying software, and polishing the writing. All AI-assisted code was subsequently reviewed, tested, and verified by the authors before being used in the paper. All reported experiments were executed and recorded by the authors, and no experimental data or results were generated by an AI system. The authors take full responsibility for the implementation and the reported results.

References

[1] A. C. Yao, "Protocols for secure computations," in *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 1982, pp. 160–164.

[2] P. Mohassel and P. Rindal, "Aby^3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 35–52. [Online]. Available: https://doi.org/10.1145/3243734.3243760

[3] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "Astra: high throughput 3pc over rings with application to secure prediction," in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2019, pp. 81–92.

[4] A. Patra and A. Suresh, "BLAZE: blazing fast privacy-preserving machine learning," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/blaze-blazing-fast-privacy-preserving-machine-learning/

[5] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "Swift: Super-fast and robust privacy-preserving machine learning," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2651–2668.

[6] R. Wails, A. Johnson, D. Starin, A. Yerukhimovich, and S. D. Gordon, "Stormy: Statistics in tor by measuring securely," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 615–632. [Online]. Available: https://doi.org/10.1145/3319535.3345650

[7] A. B. Pedin and N. Siasi, "Secure and decentralized anonymous e-voting scheme," in *Proceedings of the 2023 ACM Southeast Conference*, ser. ACMSE '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 172–176. [Online]. Available: https://doi.org/10.1145/3564746.3587107

[8] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "Muse: Secure inference resilient to malicious clients," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2201–2218.

[9] M. Jagielski, D. Escudero, R. Rachuri, and P. Scholl, "Covert attacks on machine learning training in passively secure mpc," *arXiv preprint arXiv:2505.17092*, 2025.

[10] A. P. K. Dalskov, D. Escudero, and M. Keller, "Fantastic four: Honest-majority four-party secure computation with malicious security," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 2183–2200. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/dalskov

[11] Y. Li, D. Escudero, Y. Duan, Z. Huang, C. Hong, C. Zhang, and Y. Song, "Sublinear distributed product checks on replicated secret-shared data over z2k without ring extensions," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 825–839. [Online]. Available: https://doi.org/10.1145/3658644.3690260

[12] Y. Li, Y. Duan, Z. Huang, C. Hong, C. Zhang, and Y. Song, "Efficient 3PC for binary circuits with application to Maliciously-Secure DNN inference," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 5377–5394. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/li-yun

[13] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Annual International Cryptology Conference*. Springer, 1991, pp. 420–432.

[14] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries," in *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, ser. Lecture Notes in Computer Science, vol. 4392. Springer, 2007, pp. 137–156. [Online]. Available: https://iacr.org/archive/tcc2007/43920137/43920137.pdf

[15] G. Asharov and C. Orlandi, "Calling out cheaters: Covert security with public verifiability," in *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, ser. Lecture Notes in Computer Science, X. Wang and K. Sako, Eds., vol. 7658. Springer, 2012, pp. 681–698. [Online]. Available: https://doi.org/10.1007/978-3-642-34961-4_41

[16] V. Kolesnikov and A. J. Malozemoff, "Public verifiability in the covert model (almost) for free," in *ASIACRYPT (2)*. Springer, 2015, pp. 210–235. [Online]. Available: https://www.iacr.org/archive/asiacrypt2015/94520319/94520319.pdf

[17] C. Hong, J. Katz, V. Kolesnikov, W. Lu, and X. Wang, "Covert security with public verifiability: Faster, leaner, and simpler," in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, ser. Lecture Notes in Computer Science, Y. Ishai and V. Rijmen, Eds., vol. 11478. Springer, 2019, pp. 97–121. [Online]. Available: https://doi.org/10.1007/978-3-030-17659-4_4

[18] S. Faust, C. Hazay, D. Kretzler, and B. Schlosser, "Generic compiler for publicly verifiable covert multi-party computation," in *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, ser. Lecture Notes in Computer Science, A. Canteaut and F. Standaert, Eds., vol. 12697. Springer, 2021, pp. 782–811. [Online]. Available: https://doi.org/10.1007/978-3-030-77886-6_27

[19] P. Scholl, M. Simkin, and L. Siniscalchi, "Multiparty computation with covert security and public verifiability," in *3rd Conference on Information-Theoretic Cryptography, ITC 2022, July 5-7, 2022, Cambridge, MA, USA*, ser. LIPIcs, D. Dachman-Soled, Ed., vol. 230. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 8:1–8:13. [Online]. Available: https://doi.org/10.4230/LIPIcs.ITC.2022.8

[20] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 805–817. [Online]. Available: https://doi.org/10.1145/2976749.2978331

[21] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein, "High-throughput secure three-party computation for malicious adversaries and an honest majority," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2017, pp. 225–255.

[22] T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, A. Nof, K. Ohara, A. Watzman, and O. Weinstein, "Optimized honest-majority mpc for malicious adversaries — breaking the 1 billion-gate per second barrier," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 843–862.

[23] E. Boyle, N. Gilboa, Y. Ishai, and A. Nof, "Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 869–886.

[24] C. Baum, E. Orsini, and P. Scholl, "Efficient secure multiparty computation with identifiable abort," in *Theory of Cryptography*. Springer, 2016, pp. 461–490.

[25] M. Rivinius, P. Reisert, D. Rausch, and R. Küsters, "Publicly accountable robust multi-party computation," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2022, pp. 2430–2449.

[26] R. Cohen, J. Doerner, Y. Kondi, and A. Shelat, "Secure multiparty computation with identifiable abort via vindicating release," in *Advances in Cryptology – CRYPTO 2024*, ser. Lecture Notes in Computer Science, vol. 14927. Springer, 2024, pp. 36–73.

[27] M. Rivinius, "MPC with publicly identifiable abort from pseudorandomness and homomorphic encryption," in *Advances in Cryptology – EUROCRYPT 2025*. Springer, 2025, pp. 270–300.

[28] M. Abspoel, A. Dalskov, D. Escudero, and A. Nof, "An efficient passive-to-active compiler for honest-majority mpc over rings," in *International Conference on Applied Cryptography and Network Security*. Springer, 2021, pp. 122–152.

[29] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM, 2020, pp. 1575–1590. [Online]. Available: https://doi.org/10.1145/3372297.3417872

[30] R. Cramer, I. Damgård, D. E. Escudero, P. Scholl, and C. Xing, "SPDZ2k: Efficient MPC mod 2^k for dishonest majority," in *IACR Cryptology ePrint Archive*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:64580200

[31] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2007, pp. 52–78.

[32] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Zero-knowledge proofs on secret-shared data via fully linear pcps," in *Annual International Cryptology Conference*. Springer, 2019, pp. 67–97.

[33] Z. Chen, "Efficient three-party computation: An information-theoretic approach from cut-and-choose," 2019. [Online]. Available: https://arxiv.org/abs/1908.03718

[34] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof, "Fast large-scale honest-majority MPC for malicious adversaries," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, ser. Lecture Notes in Computer Science, H. Shacham and A. Boldyreva, Eds., vol. 10993. Springer, 2018, pp. 34–64. [Online]. Available: https://doi.org/10.1007/978-3-319-96878-0_2

[35] E. Boyle, N. Gilboa, Y. Ishai, and A. Nof, "Efficient fully secure computation via distributed zero-knowledge proofs," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 244–276.

[36] ——, "Sublinear gmw-style compiler for MPC with preprocessing," in *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, ser. Lecture Notes in Computer Science, T. Malkin and C. Peikert, Eds., vol. 12826. Springer, 2021, pp. 457–485. [Online]. Available: https://doi.org/10.1007/978-3-030-84245-1_16

[37] M. Hamilis, E. Boyle, N. Gilboa, Y. Ishai, and A. Nof, "Preprocessing for life: Dishonest-majority mpc with a trusted or untrusted dealer," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 41–41.

[38] S. G. Choi, J. Katz, A. J. Malozemoff, and V. Zikas, "Efficient three-party computation from cut-and-choose," in *Advances in Cryptology – CRYPTO 2014*, ser. Lecture Notes in Computer Science, J. A. Garay and R. Gennaro, Eds., vol. 8617. Berlin, Heidelberg: Springer, 2014, pp. 513–530. [Online]. Available: https://doi.org/10.1007/978-3-662-44381-1_29

[39] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, "A new approach to practical active-secure two-party computation," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, 2012, pp. 681–700. [Online]. Available: https://doi.org/10.1007/978-3-642-32009-5_40

[40] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti,

Eds., vol. 7417. Springer, 2012, pp. 643–662. [Online]. Available: https://doi.org/10.1007/978-3-642-32009-5_38

[41] S. S. Burra, E. Larraia, J. B. Nielsen, P. S. Nordholt, C. Orlandi, E. Orsini, P. Scholl, and N. P. Smart, "High-performance multi-party computation for binary circuits based on oblivious transfer," *J. Cryptol.*, vol. 34, no. 3, p. 34, 2021. [Online]. Available: https://doi.org/10.1007/s00145-021-09403-1

[42] I. Damgård, C. Orlandi, and M. Simkin, "Black-box transformations from passive to covert security with public verifiability." Springer-Verlag, 2020.

[43] D. Chaum, "Blind signature system," in *Advances in cryptology*. Springer, 1984, pp. 153–153.

[44] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," USA, Tech. Rep., 1996.

[45] Y. Ishai, M. Prabhakaran, and A. Sahai, "Founding cryptography on oblivious transfer - efficiently," in *CRYPTO*, 2008, pp. 572–591.

[46] Y. Lindell, E. Oxman, and B. Pinkas, "The ips compiler: Optimizations, variants and concrete efficiency," in *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference*, ser. Lecture Notes in Computer Science, vol. 6841. Springer, 2011, p. 255. [Online]. Available: https://www.iacr.org/archive/crypto2011/68410255/68410255.pdf

[47] T. Attema, V. Dunning, M. Everts, and P. Langenkamp, "Efficient compiler to covert security with public verifiability for honest majority mpc," in *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Proceedings*, ser. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), G. Ateniese and D. Venturi, Eds. Germany: Springer, Jun. 2022, pp. 663–683, funding Information: Acknowledgements. The research activities that led to this result were funded by ABN AMRO, CWI, De Volksbank, Rabobank, TMNL, PPS-surcharge for Research and Innovation of the Dutch Ministry of Economic Affairs and Climate Policy, TNO's Appl. AI programme and the Vraaggestuurd Programma Cyber Security & Resilience, part of the Dutch Top Sector High Tech Systems and Materials program. Publisher Copyright: © 2022, Springer Nature Switzerland AG.; 20th International Conference on Applied Cryptography and Network Security, ACNS 2022, ACNS ; Conference date: 20-06-2022 Through 23-06-2022.

[48] D. Genkin, Y. Ishai, M. Prabhakaran, A. Sahai, and E. Tromer, "Circuits resilient to additive attacks with applications to secure computation," in *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, D. B. Shmoys, Ed. ACM, 2014, pp. 495–504. [Online]. Available: https://doi.org/10.1145/2591796.2591861

[49] D. Genkin, Y. Ishai, and A. Polychroniadou, "Efficient multi-party computation: From passive to active security via secure SIMD circuits," in *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, ser. Lecture Notes in Computer Science, R. Gennaro and M. Robshaw, Eds., vol. 9216. Springer, 2015, pp. 721–741. [Online]. Available: https://doi.org/10.1007/978-3-662-48000-7_35

[50] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[51] M. Ito, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 72, no. 9, pp. 56–64, 1989.

[52] B. Bünz, B. Fisch, and A. Szepieniec, "Transparent snarks from dark compilers," in *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*. Springer, 2020, pp. 677–706.

[53] M. Campanelli, D. Fiore, and A. Querol, "Legosnark: Modular design and composition of succinct zero-knowledge proofs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2075–2092.

[54] Y. Li, C. Ye, Y. Hu, I. Morpheus, Y. Guo, C. Zhang, Y. Zhang, Z. Sun, Y. Lu, and H. Wang, "Zkcplus: Optimized fair-exchange protocol supporting practical and flexible data exchange," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 3002–3021. [Online]. Available: https://doi.org/10.1145/3460120.3484558

[55] H. Lipmaa, "Prover-efficient commit-and-prove zero-knowledge snarks," *Int. J. Appl. Cryptogr.*, vol. 3, no. 4, pp. 344–362, 2017. [Online]. Available: https://doi.org/10.1504/IJACT.2017.10010328

[56] T. Attema, R. Cramer, and L. Kohl, "A compressed $\varsigma $-protocol theory for lattices," in *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, ser. Lecture Notes in Computer Science, T. Malkin and C. Peikert, Eds., vol. 12826. Springer, 2021, pp. 549–579. [Online]. Available: https://doi.org/10.1007/978-3-030-84245-1_19

[57] M. Campanelli, A. Faonio, D. Fiore, A. Querol, and H. Rodríguez, "Lunar: A toolbox for more efficient universal and updatable zksnarks and commit-and-prove extensions," in *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III*, ser. Lecture Notes in Computer Science, M. Tibouchi and H. Wang, Eds., vol. 13092. Springer, 2021, pp. 3–33. [Online]. Available: https://doi.org/10.1007/978-3-030-92078-4_1

[58] D. F. Aranha, E. M. Bennedsen, M. Campanelli, C. Ganesh, C. Orlandi, and A. Takahashi, "ECLIPSE: enhanced compiling method for pedersen-committed zksnark engines," in *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I*, ser. Lecture Notes in Computer Science, G. Hanaoka, J. Shikata, and Y. Watanabe, Eds., vol. 13177. Springer, 2022, pp. 584–614. [Online]. Available: https://doi.org/10.1007/978-3-030-97121-2_21

[59] H. Lipmaa, "On black-box knowledge-sound commit-and-prove snarks," in *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. Guo and R. Steinfeld, Eds., vol. 14439. Springer, 2023, pp. 41–76. [Online]. Available: https://doi.org/10.1007/978-981-99-8724-5_2

[60] C. Ganesh, V. Nair, and A. Sharma, "Dual polynomial commitment schemes and applications to commit-and-prove snarks," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds. ACM, 2024, pp. 884–898. [Online]. Available: https://doi.org/10.1145/3658644.3690219

[61] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," *IACR Cryptol. ePrint Arch.*, p. 1002, 2020. [Online]. Available: https://eprint.iacr.org/2020/1002

[62] M. Keller and K. Sun, "Secure quantized training for deep learning," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 2022, pp. 10 912–10 938. [Online]. Available: https://proceedings.mlr.press/v162/keller22a.html

[63] Z. Huang, W. jie Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure Two-Party deep neural network inference," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 809–826. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong

[64] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl, "Improved primitives for MPC over mixed arithmetic-binary circuits," in *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds., vol. 12171. Springer, 2020, pp. 823–852. [Online]. Available: https://doi.org/10.1007/978-3-030-56880-1_29

[65] Y. Duan, Y. Li, Z. Huang, C. Hong, T. Wei, and C. Zhang, "ECHO: Efficient Covertly-Secure Three-party Computation with Applications to Private Machine Learning," Cryptology ePrint Archive, Paper 2026/216, 2026. [Online]. Available: https://eprint.iacr.org/2026/216

[66] A. P. K. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 4, pp. 355–375, 2020. [Online]. Available: https://doi.org/10.2478/popets-2020-0077

[67] M. B. Santos, D. Mouris, M. Ugurbil, S. Jarecki, J. Reis, S. Sengupta, and M. de Vega, "Curl: Private LLMs through Wavelet-Encoded Look-Up Tables," in *Proceedings of the Conference on Applied Machine Learning in Information Security*, Arlington, Virginia, USA, October 24–25, 2024, pp. 1–31.

[68] D. Rotaru and T. Wood, "Marbled circuits: Mixing arithmetic and boolean circuits with active security," in *Progress in Cryptology – INDOCRYPT 2019*, F. Hao, S. Ruj, and S. Sen Gupta, Eds. Cham: Springer International Publishing, 2019, pp. 227–249.

[69] C. Baum, L. Braun, C. D. de Saint Guilhem, M. Klooß, E. Orsini, L. Roy, and P. Scholl, "Publicly verifiable zero-knowledge and post-quantum signatures from vole-in-the-head," in *Annual international cryptology conference*. Springer, 2023, pp. 581–615.

[70] C. Baum, W. Beullens, S. Mukherjee, E. Orsini, S. Ramacher, C. Rechberger, L. Roy, and P. Scholl, "One tree to rule them all: Optimizing ggm trees and owfs for post-quantum signatures," in *Advances in Cryptology – ASIACRYPT 2024*, K.-M. Chung and Y. Sasaki, Eds. Singapore: Springer Nature Singapore, 2025, pp. 463–493.

[71] E. Boyle, N. Gilboa, and Y. Ishai, "Function Secret Sharing," in *Advances in Cryptology - EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, vol. 9057, pp. 337–367, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-662-46803-6_12

[72] ——, "Function Secret Sharing: Improvements and Extensions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Vienna Austria: ACM, Oct. 2016, pp. 1292–1303. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978429

[73] N. Gilboa and Y. Ishai, "Distributed Point Functions and Their Applications," in *Advances in Cryptology – EUROCRYPT 2014*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, A. Kobsa, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, D. Terzopoulos, D. Tygar, G. Weikum, P. Q. Nguyen, and E. Oswald, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, vol. 8441, pp. 640–658, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-642-55220-5_35

[74] K. Cheng, Y. Xia, A. Song, J. Fu, W. Qu, Y. Shen, and J. Zhang, "Mosformer: Maliciously Secure Three-Party Inference Framework for Large Transformers," 2025, publication info: Published elsewhere. ACM CCS 2025. [Online]. Available: https://eprint.iacr.org/2025/1510

[75] L. de Castro and A. Polychroniadou, "Lightweight, maliciously secure verifiable function secret sharing," in *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 150–179. [Online]. Available: https://doi.org/10.1007/978-3-031-06944-4_6

APPENDIX A
COVERT SECURITY

Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-input function, and $\mathcal{F}_{\text{SECF}}$ is an ideal functionality that computes f with strong explicit cheat formulation. Let Corr be the set of corrupted parties controlled by $\mathcal{S}$. We now elaborate on the ideal execution in the following.

Inputs: Each honest party P_i sends its input x_i to $\mathcal{F}_{\text{SECF}}$. The ideal adversary $\mathcal{S}$ gets an auxiliary input z, and sends inputs on behalf of all corrupted parties to $\mathcal{F}_{\text{SECF}}$.

Abort Options: If a corrupted party P_i sends (abort, i) or $(\mathsf{corrupted}, i)$ as its input to $\mathcal{F}_{\text{SECF}}$, $\mathcal{F}_{\text{SECF}}$ sends to (abort, i) or $(\mathsf{corrupted}, i)$ respectively to all honest parties and halts. If multiple parties send the abort (or corrupted) message, $\mathcal{F}_{\text{SECF}}$ only keeps one of them. If both the two types of messages are received, $\mathcal{F}_{\text{SECF}}$ ignores the corrupted messages.

Attempted Cheat: If $\mathcal{S}$ sends (cheat, i) as the input of a corrupted party P_i, then $\mathcal{F}_{\text{SECF}}$ works as follows:

- Detected: With probability ϵ, $\mathcal{F}_{\text{SECF}}$ sends $(\mathsf{corrupted}, i)$ to $\mathcal{S}$ and all the honest parties.
- Undetected: With probability $1-\epsilon$, $\mathcal{F}_{\text{SECF}}$ sends undetected and the honest parties' inputs to $\mathcal{S}$. Then $\mathcal{S}$ specifies an output value y_i for each honest party P_i, and $\mathcal{F}_{\text{SECF}}$ hands each output y_i to P_i.

The ideal execution ends at this point. Otherwise, if no (abort, i), $(\mathsf{corrupted}, i)$ or (cheat, i) has been sent, the ideal execution continues as follows.

Functionality Answers Adversary: Let $\vec{x} := (x_1, \cdots, x_n)$ be the received inputs by $\mathcal{F}_{\text{SECF}}$. The functionality computes the outputs $(y_1, \cdots, y_n) := (x_1, \cdots, x_n)$, and sends y_i to $\mathcal{S}$ for all $i \in \mathsf{Corr}$.

Functionality Answers Honest Parties: The adversary $\mathcal{S}$ either sends back continue or (abort, i) w.r.t. a corrupted party P_i to $\mathcal{F}_{\text{SECF}}$. If it sends continue, $\mathcal{F}_{\text{SECF}}$ returns the output value y_i to each honest party P_i. Otherwise if it sends (abort, i), then $\mathcal{F}_{\text{SECF}}$ hands (abort, i) to all honest parties.

Outputs: An honest party always outputs the message it received from $\mathcal{F}_{\text{SECF}}$. The corrupted parties output nothing. The adversary outputs an arbitrary probabilistic polynomial-time computable function of the initial inputs of the corrupted parties, the auxiliary input z, and the messages received from $\mathcal{F}_{\text{SECF}}$.

Let $\mathsf{IDEAL}^{\epsilon}_{f,\mathcal{S}(z),\mathsf{Corr}}(\vec{x}, 1^\kappa)$ be the outputs of the honest parties and $\mathcal{S}$ in an ideal execution. Let Π be a protocol that realizes the function f, which internally may use an auxiliary ideal functionality $\mathcal{G}$. Let $\mathsf{REAL}^{\epsilon}_{\Pi,\mathcal{A}(z),\mathsf{Corr}}(\vec{x}, 1^\kappa)$ be the outputs of the honest parties and $\mathcal{A}$ in a real execution. We now define the security of the protocol against covert adversaries with ϵ-deterrent.

Definition A.1: **(Covert security with ϵ-deterrent.)** Let f, Π, ϵ be as above. We say the protocol Π *securely computes f in the presence of a covert adversary with ϵ-deterrent* in the $\mathcal{G}$-hybrid model if for any non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real world, there exists a probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal world s.t. for every $\mathsf{Corr} \subseteq [n]$, every balanced vector $\vec{x} \in (\{0,1\}^*)^n$, and every auxiliary input $z \in \{0,1\}^*$:

$$\{\mathsf{IDEAL}^{\epsilon}_{f,\mathcal{S}(z),\mathsf{Corr}}(\vec{x}, 1^\kappa)\}_{\kappa\in\mathbb{N}} \equiv_c \{\mathsf{REAL}^{\epsilon}_{\Pi,\mathcal{A}(z),\mathsf{Corr}}(\vec{x}, 1^\kappa)\}_{\kappa\in\mathbb{N}}.$$

Let $\mathsf{OUTPUT}_j(\mathsf{REAL}_{\Pi,\mathcal{A}(z),\mathsf{Corr}}(\vec{x}, 1^\kappa))$ be the output of an honest party P_j in a real execution described above, and id_i be the identity of the party P_i. Let $\mathsf{Blame}, \mathsf{Judge}$ be as described in Section III-B. We now introduce the formal definition of covert security with public verifiability below.

Definition A.2: **(Covert security with ϵ-deterrent and public verifiability.)** Let $f, \Pi, \epsilon, \mathsf{Blame}, \mathsf{Judge}$ be as above. We say $(\Pi, \mathsf{Blame}, \mathsf{Judge})$ *securely computes f in the presence of a covert adversary with ϵ-deterrent and public verifiability* in the $\mathcal{G}$-hybrid model if the following conditions hold:

1) **Simulatability with ϵ-deterrence:** The protocol Π (where the honest parties broadcast cert if it detects cheating) is secure against a covert adversary with ϵ-deterrence factor in the $\mathcal{G}$-hybrid model according to the strong explicit cheat formulation, but with a difference in the abort options: if a corrupted party sends (abort, i) to $\mathcal{F}_{\text{SECF}}$, the functionality sends abort to the honest parties and halts.
2) **Public Verifiability:** For any probabilistic polynomial-time adversary $\mathcal{A}$ corrupting parties P_i for $i \in \mathsf{Corr} \subseteq [n]$, there exists a negligible function $\mathsf{negl}(\kappa)$ s.t. for all

$(\vec{x}, z) \in (\{0,1\}^*)^{n+1}$ and $j \in [n] \backslash \mathsf{Corr}$, the following holds: if

$$\mathsf{OUTPUT}_j(\mathsf{REAL}_{\Pi,\mathcal{A}(z),\mathsf{Corr}}(\vec{x}, 1^\kappa)) = (\mathsf{corrupted}, i),$$

then $\Pr[\mathsf{Judge}(\mathsf{cert}_j) = \mathsf{id}_i] \geq 1 - \mathsf{negl}(\kappa)$.

3) **Defamation Freeness:** For any probabilistic polynomial-time adversary $\mathcal{A}$ corrupting parties P_i for $i \in \mathsf{Corr} \subseteq [n]$, there exists a negligible function $\mathsf{negl}(\kappa)$ s.t. for all $(\vec{x}, z) \in (\{0,1\}^*)^{n+1}$ and $j \in [n] \backslash \mathsf{Corr}$,

$$\Pr[\mathsf{Judge}(\mathsf{cert}^*) = \mathsf{id}_j \mid \mathsf{cert}^* \leftarrow \mathcal{A}] \leq \mathsf{negl}(\kappa).$$

APPENDIX B
SIGNATURES, COMMITMENTS, AND CP-NIZK PROOFS

A. Signature Schemes

A signature scheme consists of the following algorithms.

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)$: takes as input a security parameter, and generates a secret signing key sk and a public verification key pk.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}; m)$: takes as input a message x, signs x with the signing key sk, and outputs a signature σ.
- $b \leftarrow \mathsf{Vrfy}(\mathsf{pk}; \sigma)$: verifies the validity of the signature with the verification key pk, and outputs $b \in \{0,1\}$ indicating accept $(b = 1)$ or reject $(b = 0)$.

B. Commitments and Zero-knowledge Proofs

A commitment scheme [52] is a tuple of three algorithms $\mathsf{COM} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open})$ where

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa)$: takes as input a security parameter, and generates public parameters pp.
- $(\mathsf{Com}, r) \leftarrow \mathsf{Commit}(\mathsf{pp}; x)$: takes as input x, produces a commitment Com and an opening hint r.
- $b \leftarrow \mathsf{Open}(\mathsf{pp}; \mathsf{Com}, x, r)$: verifies the opening of commitment Com to x with the opening hint r, and outputs $b \in \{0,1\}$ indicating accept $(b = 1)$ or reject $(b = 0)$.

We say a commitment scheme is (computationally) **binding** if for any probabilistic polynomial-time adversary$\mathcal{A}$,

$$\Pr\left[\begin{array}{c|c} b_0 = b_1 \neq 0 \\ \wedge \\ x_0 \neq x_1 \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa) \\ (\mathsf{Com}, x_0, x_1, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b_0 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{Com}, x_0, \pi_0) \\ b_1 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{Com}, x_1, \pi_1) \end{array}\right] \leq \mathsf{negl}(\kappa).$$

We say a commitment is (perfectly) **hiding** if for any adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} \mathcal{A}(\mathsf{pp}, c) = b \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa), \\ (x_0, x_1) \leftarrow \mathcal{A}(\mathsf{pp}), \\ b \xleftarrow{\$} \{0,1\}, \\ (c, r) \leftarrow \mathsf{Commit}(\mathsf{pp}; x_b) \end{array}\right] = \frac{1}{2}.$$

A Non-Interactive Zero-Knowledge (NIZK) proof of knowledge is a tuple of algorithms $\mathsf{ZK} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Vrfy})$:

- $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa)$: takes as input a security parameter, generates a common reference string crs.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}; s, w)$: takes as input a witness w and statement s, and outputs a proof π for $\mathsf{R}(s, w) = 1$.
- $b \leftarrow \mathsf{Vrfy}(\mathsf{crs}; s, \pi)$: verifies the proof π on the statement s, and outputs $b \in \{0,1\}$ indicating accept $(b = 1)$ or reject $(b = 0)$.

We say a zero-knowledge proof of knowledge has (perfect) **completeness**, if for any adversary $\mathcal{A}$ and for every $(s, w) \in \mathsf{R}$,

$$\Pr\left[\begin{array}{c} \mathsf{Vrfy}(\mathsf{crs}; s, \pi) = 1 \end{array} \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa), \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}; s, w) \end{array}\right] = 1.$$

We say a zero-knowledge argument of knowledge has (computational) **knowledge soundness** if for any polynomial-time adversary $\mathcal{A}$, there exists a polynomial-time extractor $\mathcal{E}$ s.t.

$$\Pr\left[\begin{array}{c} (s, w) \notin \mathsf{R} \wedge \\ \mathsf{Vrfy}(\mathsf{crs}; s, \pi) = 1 \end{array} \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa), \\ (s, \pi; w) \leftarrow (\mathcal{A}||\mathcal{E})(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\kappa).$$

We say a zero-knowledge argument of knowledge has (perfect) **zero-knowledge** if for every pair of $(s, w) \in \mathsf{R}$ and every adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a simulator $\mathcal{S}$ s.t.

$$\Pr\left[\begin{array}{c} (s, w) \in \mathsf{R} \wedge \\ \mathcal{A}_1(\mathsf{crs}; s, \pi) = 1 \end{array} \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa), \\ (s, w) \leftarrow \mathcal{A}_2(\mathsf{crs}), \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}; s, w) \end{array}\right]$$
$$= \Pr\left[\begin{array}{c} (s, w) \in \mathsf{R} \wedge \\ \mathcal{A}_1(\mathsf{crs}; s, \pi) = 1 \end{array} \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa), \\ (s, w) \leftarrow \mathcal{A}_2(\mathsf{crs}), \\ \pi \leftarrow \mathcal{S}(\mathsf{crs}; s) \end{array}\right].$$

APPENDIX C
ADDITIONAL RELATED WORKS

A. Function Secret Sharing(FSS)-based Approach

Function Secret Sharing (FSS) [71], [72] consists of two separate phases: key generation (Gen) and function evaluation (Eval). In the Gen phase, parties generate shared keys for the FSS scheme; in the Eval phase, they evaluate the function on these keys with their respective inputs. The distributed point function (DPF) [73] is a prominent instance of FSS, widely used to construct 2-server private information retrieval (PIR) and related cryptographic protocols.

Recent work has explored FSS-based approaches for malicious security in multi-party computation. Notably, Mosformer [74] proposes a maliciously secure three-party inference framework for secure comparison in large transformers, leveraging **verifiable** distributed point functions (VDPF) [75]. VDPF extends DPF with an additional verification procedure that uses zero-knowledge proofs to validate evaluation correctness. However, Mosformer does not report offline computation costs. Our evaluation reveals that VDPF key generation incurs significant computational overhead, making it impractical for certain applications. In contrast, our approach achieves substantially lower offline costs through covert security mechanisms.

B. G.O.D. in Maliciously Secure 3PC

Guaranteed Output Delivery (G.O.D.) is widely regarded as the strongest security guarantee achievable in maliciously secure computation. Previous works [23], [10], [5] achieve G.O.D. by identifying a conflicting pair of two parties (so that one of them must be corrupted), and treating the remaining party as a trusted party, who then receives inputs from all the other parties and computes the functionality in clear. In the general n-party setting, a similar approach is adopted in [35] by identifying conflicting pairs and then eliminating them from the computation, until the left parties obtain the correct outputs.

C. Covertly Secure Protocols

a) The Cut-and-Choose Paradigm.: The cut-and-choose paradigm is widely adopted in previous covertly secure protocols, where the parties need to run multiple, say, t instances of the semi-honest protocol in parallel, then randomly pick $t-1$ instances and check if they are faithfully executed by opening all the inputs and random tapes of the parties. Note that this only applies to preprocessing protocols (where the parties have no private inputs, like the garbling phase or Beaver triple generation [13]), to avoid input privacy leakage arising from opening. This cut-and-choose paradigm achieves a deterrence factor of $1-\frac{1}{t}$. By setting $t=2$, it can obtain a deterrence factor of 50%. However, to achieve a larger deterrence factor such as 80% or 90%, t will increase to 5 and 9 respectively, leading to notable communication and computational overhead of the semi-honest protocol.

b) Achieving Public Verifiability.: To achieve public verifiability based on the above cut-and-choose approach, previous works [19], [18], [42], [17], [16] use signatures to identify cheaters when checking fails. However, a cheating party may refuse to open its inputs and randomness by aborting the protocol after the random $t-1$ instances are chosen. This hinders the honest parties from generating a valid certificate of accusation. In the n-party setting, the state-of-the-art works [19], [18] introduce time-lock puzzles [44] to solve this issue, which allows a party to lock some data until a certain time has passed. In their compilers, the parties first compute puzzles to lock the shared coin used for sampling the $t-1$ instances and the random tapes used in those $t-1$ instances (in [18], the puzzle generation needs to be executed using a maliciously secure computation protocol, as the shared coin needs to be kept secret), and then sign the puzzles and broadcast the signatures. This ensures that even if the corrupted party aborts the protocol after the coin is made public, the honest parties are still able to obtain the random tapes of the corrupted party by solving its puzzles, and thus yield a valid public certificate of accusing the party. However, this approach incurs an unavoidable cost for computing the puzzles in a maliciously secure way, even in the optimistic case where no cheating is detected.

c) Beyond Preprocessing Protocols.: As aforementioned, the above cut-and-choose-based paradigm is only applicable to preprocessing protocols. Previous work [42], [18], [19] have shown that a general covertly secure computation protocol with public verifiability can be obtained by compiling the preprocessing protocol into a covertly secure one and then combining it with a maliciously secure online protocol. Additionally, previous works [46], [42], [18] have demonstrated another way to compile general input-dependent secure computation protocols based on the player virtualization paradigm [45]. Specifically, each real party now shares its private input to t virtual parties using a t-out-of-t secret sharing scheme, and simulates their executions of the semi-honest protocol. Rather than running t instances of a semi-honest n-party protocol, those virtual parties now run one instance of a semi-honest $(t\cdot n)$-party protocol. To detect cheating, each party is required to randomly pick $t-1$ virtual parties and open their input shares as well as random tapes to check their behaviors. Note that privacy is now preserved due to the property of the secret sharing scheme. However, the scaled number of parties may blow up both the communication and computational costs when a large deterrence factor is required. For RSS-based protocol, this issue is more severe as the communication cost increases exponentially with the number of parties.

APPENDIX D
INSTANTIATIONS OF FUNCTIONALITIES

A. Instantiating $\mathcal{F}_{input}$

a) Input Sharing.: We have introduced the sharing semantics of ASTRA [3] in Section III-C. In Protocol D.1 we describe the protocol of ASTRA for instantiating $\mathcal{F}_{\text{input}}$, where a party shares a secret $x \in R$ to the other parties.

Functionality C.1: $\mathcal{F}_{\text{VrfyIP}}$

Let $\mathcal{S}$ be the ideal world adversary.

1) $\mathcal{F}_{\text{VrfyIP}}$ receives the prover's identity j, a parameter m, and replicated secret shares of $\{(\llbracket\vec{\mu}^{(i)}\rrbracket, \llbracket\vec{\nu}^{(i)}\rrbracket, \llbracket w^{(i)}\rrbracket)\}_{i=1}^{m}$ from the honest parties, and checks whether the shares are consistent. If not, $\mathcal{F}_{\text{CheckZero}}$ sends the identity j and honest parties' shares to $\mathcal{S}$, then receives an output $\mathsf{out} \in \{\mathsf{accept}, \mathsf{abort}\}$ from $\mathcal{S}$, and passes the output to the honest parties. Otherwise, it reconstructs v the whole sharings $(\llbracket\vec{\mu}^{(i)}\rrbracket, \llbracket\vec{\nu}^{(i)}\rrbracket, \llbracket w^{(i)}\rrbracket)$ for all $i \in [m]$, and sends the identity j and the shares of corrupted parties to $\mathcal{S}$. In addition, if P_j is corrupted, $\mathcal{F}_{\text{VrfyIP}}$ also sends the whole sharings $\{(\llbracket\vec{\mu}^{(i)}\rrbracket, \llbracket\vec{\nu}^{(i)}\rrbracket, \llbracket w^{(i)}\rrbracket)\}_{i=1}^{m}$ to $\mathcal{S}$.
2) $\mathcal{F}_{\text{VrfyIP}}$ checks if the equation $w^{(i)} = \vec{\mu}^{(i)} \cdot \vec{\nu}^{(i)}$ holds for all $i \in [m]$.
   - If it doesn't hold for some $i \in [m]$, $\mathcal{F}_{\text{VrfyIP}}$ sends abort to all honest parties and $\mathcal{S}$.
   - Otherwise, $\mathcal{F}_{\text{VrfyIP}}$ receives a command $\mathsf{out} \in \{\mathsf{accept}, \mathsf{abort}\}$ from $\mathcal{S}$ and sends out to the honest parties.

B. Instantiating $\mathcal{F}_{VrfyIP}$

We give the formal definition and instantiation of $\mathcal{F}_{\text{VrfyIP}}$ in Functionality C.1 and Protocol C.1. We apply the same optimizations from [11] to the protocol, and have the following Lemma from [11].

Lemma D.1: Let k, s, q be positive integers, and $T = 2\lceil \log_q m \rceil + 1$. The protocol Π_{VrfyIP} securely computes $\mathcal{F}_{VrfyIP}$ in the $\mathcal{F}_{coin}$-hybrid model in the presence of a malicious adversary controlling one party, with soundness error bounded by

$$2^{-\lambda} + (1-2^{-\lambda}) \sum_{i=0}^{s} \frac{1}{2^{i+1}} \left(\sum_{j=0}^{T-2} \binom{s-i+j}{s-i} \cdot \frac{1}{2^{s-i+1+j}} \right)^2 + \frac{1}{2^{s+1}}.$$

APPENDIX E
SECURITY PROOF OF Cvt-MAC$^{\mathsf{On}}$

We show the formal description of Cvt-MAC$^{\mathsf{On}}$ in Protocol D.2.

A. Soundness Analysis

We analyze arbitrary deviations in multiplication and MAC computation over $R = \mathbb{Z}_{2^{k+s}}$. Only P_1, P_2 receive online verification results and circuit outputs; P_0 receives no messages after submitting its inputs.

a) Corrupted P_0: P_0 could add nonzero error modulo 2^k e on computing $r'_{xy} = r_x r_y + e$, which is covered by $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$. For the authenticated randomness, let α be the key defined by the shares received by P_1, P_2, and write

$$\widehat{\widetilde{r}}_x^{(i)} = \alpha r_x^{(i)} + \epsilon_x^{(i)}, \qquad \widehat{\widetilde{r}}_y^{(i)} = \alpha r_y^{(i)} + \epsilon_y^{(i)},$$
$$\widehat{\widetilde{w}}_0 = \alpha \sum_i \eta^{(i)} r_{xy}^{(i)} + \epsilon_0, \qquad \widehat{\widetilde{r}}_w = \alpha r_w + \epsilon_w.$$

These errors are fixed by P_0's preprocessing messages. Substitution into the two MAC computations gives

$$\widehat{\widetilde{w}} = \alpha w + \sum_i \eta^{(i)} \left(\delta_x^{(i)} \epsilon_y^{(i)} + \delta_y^{(i)} \epsilon_x^{(i)} \right) + \epsilon_0,$$
$$\widehat{\widetilde{w}}' = \alpha w + \epsilon_w.$$

Protocol C.1: Π_{VrfyIP} **(Verifying Inner-Product Triples Known by a Single Party)**

Input: The parties hold $\{(\llbracket\vec{a}_i\rrbracket, \llbracket\vec{b}_i\rrbracket, \llbracket c_i\rrbracket)\}_{i=1}^m$ over $\mathbb{Z}_{2^k}$ as input.
Auxiliary Input: Prover index j (where P_j knows the whole input sharings). Additionally, the parties agree on the parameters s, q and λ.

1) **Random Linear Combination over $\mathbb{Z}_2$**
   a) The parties invoke $\mathcal{F}_{\text{Coin}}$ to generate a random seed of size σ, and locally expand the seed and obtain random binary coefficients $\vec{\gamma}_1, \ldots, \vec{\gamma}_\lambda \in \{0,1\}^m$.
   b) For all $i \in [\lambda]$, all parties set $\llbracket\vec{\mu}_i\rrbracket = (\gamma_{i,1} \cdot \llbracket\vec{a}_1\rrbracket, \ldots, \gamma_{i,m} \cdot \llbracket\vec{a}_m\rrbracket)$, $\llbracket\vec{\nu}_i\rrbracket = (\llbracket\vec{b}_1\rrbracket, \ldots, \llbracket\vec{b}_m\rrbracket)$, $\llbracket w_i\rrbracket = \sum_{j=1}^m \gamma_{i,j} \cdot \llbracket c_j\rrbracket$.
2) **Lifting Inner-Product Triples to $\mathbb{Z}_{2^{k+s}}$**
   For all $i \in [\lambda]$,
   a) The parties view $(\llbracket\vec{\mu}_i\rrbracket, \llbracket\vec{\nu}_i\rrbracket, \llbracket w_i\rrbracket)$ as replicated secret sharings over $\mathbb{Z}_{2^{k+s}}$. P_j computes $h_i :=_{k+s} \vec{\mu}'_i \cdot \vec{\nu}'_i - w'_i$, and additively shares $h_i/2^k$ over $\mathbb{Z}_{2^s}$.
   b) The parties locally multiply their shares of $\llbracket h_i/2^k\rrbracket$ by 2^k modulo 2^{k+s}, and obtain $\llbracket h_i\rrbracket$ over $\mathbb{Z}_{2^{k+s}}$.
   c) The parties set $\llbracket\tilde{w}'_i\rrbracket :=_{k+s} \llbracket w'_i\rrbracket + \llbracket h_i\rrbracket$.
3) **Merging into One Inner-Product Triple**
   a) The parties invoke $\mathcal{F}_{\text{Coin}}$ to sample random coefficients $\theta_1, \ldots, \theta_\lambda \in \mathbb{Z}_{2^{k+s}}$.
   b) The parties set $\llbracket\vec{x}\rrbracket :=_{k+s} (\theta_1 \cdot \llbracket\vec{\mu}'_1\rrbracket, \ldots, \theta_\lambda \cdot \llbracket\vec{\mu}'_\lambda\rrbracket)$, $\llbracket\vec{y}\rrbracket :=_{k+s} (\llbracket\vec{\nu}'_1\rrbracket, \ldots, \llbracket\vec{\nu}'_\lambda\rrbracket)$ and $\llbracket z\rrbracket :=_{k+s} \sum_{i=1}^\lambda \theta_i \cdot \llbracket\tilde{w}'_i\rrbracket$.
4) **Reducing the Dimension of the Inner-Product Triple**
   The parties repeat the following steps until the dimension of $(\llbracket\vec{x}\rrbracket, \llbracket\vec{y}\rrbracket, \llbracket z\rrbracket)$ is at most q.
   a) For all $i, i' \in [q]$ and $(i,i') \neq (1,1)$, the prover party P_j computes $z_{i,i'} :=_{k+s} \vec{x}_i \cdot \vec{y}_{i'}$ and additively shares $z_{i,i'}$ to P_1, P_2.
   b) The parties compute $\llbracket z_{1,1}\rrbracket$ by $\llbracket z_{1,1}\rrbracket :=_{k+s} \llbracket z\rrbracket - \sum_{i=2}^q z_{i,i}$.
   c) All parties invoke $\mathcal{F}_{\text{Coin}}$ to randomly sample $\{\alpha_i\}_{i=1}^q, \{\beta_i\}_{i=1}^q$ in $\mathbb{Z}_{2^{k+s}}$.
   d) The parties locally compute $\llbracket\vec{x}'\rrbracket :=_{k+s} \sum_{i=1}^q \alpha_i \cdot \llbracket\vec{x}_i\rrbracket$, $\llbracket\vec{y}'\rrbracket :=_{k+s} \sum_{i=1}^q \beta_i \cdot \llbracket\vec{y}_i\rrbracket$ and $\llbracket z'\rrbracket :=_{k+s} \sum_{i=1}^q \sum_{i'=1}^q \alpha_i\beta_{i'} \cdot \llbracket z_{i,i'}\rrbracket$.
   e) The parties set $(\llbracket\vec{x}\rrbracket, \llbracket\vec{y}\rrbracket, \llbracket z\rrbracket) :=_{k+s} (\llbracket\vec{x}'\rrbracket, \llbracket\vec{y}'\rrbracket, \llbracket z'\rrbracket)$.
5) **Checking the Final Multiplication Triple**
   The parties hold $(\llbracket\vec{x}\rrbracket, \llbracket\vec{y}\rrbracket, \llbracket z\rrbracket)$ of dimension at most q.
   a) The prover party P_j randomly samples $x_0, y_0 \in \mathbb{Z}_{2^{k+s}}$. Then P_j additively shares x_0, y_0 to P_1, P_2 over $\mathbb{Z}_{2^{k+s}}$.
   b) For all $i, i' \in \{0, 1, \ldots, q\}$ and $(i,i') \neq (1,1)$, P_j computes $z_{i,i'} :=_{k+s} x_i \cdot y_{i'}$ and additively shares it to P_1, P_2 over $\mathbb{Z}_{2^k k+s}$.
   c) The parties compute $\llbracket z_{1,1}\rrbracket$ by $\llbracket z_{1,1}\rrbracket :=_{k+s} \llbracket z\rrbracket - \sum_{i=2}^q z_{i,i}$. Note that P_j shares $z_{0,0} :=_{k+s} x_0 \cdot y_0$ to all parties in this step.
   d) The parties invoke $\mathcal{F}_{\text{Coin}}$ to randomly sample $\{\alpha_i\}_{i=1}^q, \{\beta_i\}_{i=1}^q$ in $\mathbb{Z}_{2^{k+s}}$. The parties set $\alpha_0 = \beta_0 = 1$.
   e) The parties locally compute $\llbracket x'\rrbracket :=_{k+s} \sum_{i=0}^q \alpha_i \cdot \llbracket x_i\rrbracket$, $\llbracket y'\rrbracket :=_{k+s} \sum_{i=0}^q \beta_i \cdot \llbracket y_i\rrbracket$ and $\llbracket z'\rrbracket :=_{k+s} \sum_{i=0}^q \sum_{i'=0}^q \alpha_i\beta_{i'} \cdot \llbracket z_{i,i'}\rrbracket$.
   f) The parties reconstruct $\llbracket x'\rrbracket, \llbracket y'\rrbracket, \llbracket z'\rrbracket$ and check whether $z' =_{k+s} x' \cdot y'$. If not, all parties abort. Otherwise, they output accept.

Hence the first zero check tests

$$o = \sum_i \eta^{(i)} \left(\delta_x^{(i)} \epsilon_y^{(i)} + \delta_y^{(i)} \epsilon_x^{(i)}\right) + \epsilon_0 - \epsilon_w. \tag{3}$$

Even if these terms cancel, the second zero check tests the errors directly using fresh random coefficients:

$$\begin{aligned} o' = & -\sum_i \xi^{(i)} \epsilon_x^{(i)} - \sum_i \xi^{(m+i)} \epsilon_y^{(i)} \\ & - \xi^{(2m+1)} \epsilon_0 - \xi^{(2m+2)} \epsilon_w. \end{aligned} \tag{4}$$

For a nonzero error vector, let ν be the minimum 2-adic valuation of its nonzero components. Conditional on passing the first check, o' is uniform over $2^\nu R$, since the ξ's are sampled afterwards. Thus

$$\Pr[o' = 0 \mid o = 0] = 2^{-(k+s-\nu)}.$$

If some error is nonzero modulo 2^k, this probability is at most $2^{-(s+1)}$. For those nonzero errors that with all zeros in lower k bits, they do not change circuit outputs modulo 2^k. Although these errors could introduce selective-failure attacks, since no output (including abort) is provided to P_0, a corrupted P_0 will receive nothing about honest parties' secret.

Protocol D.1: $\Pi_{\textbf{Share}}$ **(Input Sharing).**
Input: x.
Auxiliary Input: Party index j where P_j holds x.

Preprocessing:

1) If $P_j = P_0$, P_0 and P_t for each $t \in \{1,2\}$ run $r_{x,t} := \mathsf{PRG}_{\mathsf{K}_t}(\cdot)$.
2) If $P_j = P_t$ where $t \in \{1,2\}$, P_0 and P_t run $r_{x,t} := \mathsf{PRG}_{\mathsf{K}_t}(\cdot)$, and then all the parties run $r_{x,3-t} := \mathsf{PRG}_{\mathsf{K}}(\cdot)$.
3) P_0 sets its share as $(r_{x,1}, r_{x,2})$.

Online:

1) P_j computes $\delta_x := x - (r_{x,1} + r_{x,2})$, and sends it to P_t for $t \in \{1,2\}$ and $t \neq j$.
2) P_t for each $t \in \{1,2\}$ sets its share of x as $(\delta_x, r_{x,t})$.

b) Corrupted P_1 or P_2: Assume P_1 is corrupted; the other case is symmetric. Since P_0 is honest, preprocessing is correct. For gate i, let X_i, Y_i be the input values defined by the honest parties' current shares, including earlier errors. If P_1 sends $\widehat{\delta}^{(i)}_{z,1}$ instead of its prescribed message $\delta^{(i)}_{z,1}$, define

$$\varphi^{(i)} = \widehat{\delta}^{(i)}_{z,1} - \delta^{(i)}_{z,1}.$$

Then the gate output is $Z_i = X_iY_i + \varphi^{(i)}$. This covers arbitrary message replacements; for example, an affine replacement $a\delta^{(i)}_{z,1} + b$ gives $\varphi^{(i)} = (a-1)\delta^{(i)}_{z,1} + b$.

The prescribed MAC computations authenticate $\sum_i \eta^{(i)} X_i Y_i$ and $\sum_i \eta^{(i)} Z_i$, respectively. Changes u, v to their local MAC shares and a further change d to the submitted zero-check share combine into $\Delta = u - v + d$. The first check therefore tests

$$o = -\alpha\Lambda + \Delta, \qquad \Lambda = \sum_i \eta^{(i)} \varphi^{(i)}. \tag{5}$$

The gate errors are fixed before the committed seed reveals the coefficients $\eta^{(i)}$, and Δ is submitted while α remains hidden. Acceptance requires $\alpha\Lambda = \Delta$, the same MAC-forgery equation analyzed in [28, Lemma 2]. If some $\varphi^{(i)}$ is nonzero modulo 2^k, its acceptance probability is at most $(s+1)2^{-s}$, as detailed in Lemma E.1.

B. Proof of Security

We have the following theorem.

Theorem E.1: Let f be a three-party functionality over $\mathbb{Z}_{2^k}$ with outputs only to P_1, P_2, under the output convention of Protocol D.2. Let $s \in \mathbb{N}$ be an integer. Suppose PRG *is a secure (pseudo-)random generator. Then Protocol D.2 securely computes f in the $(\mathcal{F}_{coin}, \mathcal{F}^{\epsilon}_{VrfyIP}, \mathcal{F}_{checkzero})$-hybrid model in the presence of a covert adversary controlling one party with deterrence factor ϵ and abort where $\epsilon = 1 - \tau$ and τ is the soundness error of Π^{Abort}_{Cov}.*

Proof E.1:

We first give the following lemma.

Lemma E.1: For any adversary $\mathcal{A}$ who corrupts P_t for $t \in \{1,2\}$ and introduces a multiplication error nonzero modulo 2^k in the online phase of Protocol D.2, the honest parties will abort with soundness error bounded by $2^{-s+\log(s+1)}$.

Proof E.2: By Equation (5), acceptance requires

$$\alpha\Lambda =_{k+s} \Delta. \tag{6}$$

Let $n = k + s$ and $\nu = \min_i v_2(\varphi^{(i)}) < k$, with $v_2(0) = n$. Replacing the PRG output by uniform coefficients incurs only a negligible difference. Then Λ is uniform over $2^\nu R$. For $j < n$, conditioned on $v_2(\Lambda) = j$, Equation (6) has either no solution or 2^j solutions for the hidden uniform key α. Its acceptance probability is therefore at most $2^{-(n-j)}$, even when Δ depends on the coefficients.

For $j \leq k$, this bound is at most 2^{-s}. For $k < j \leq n$, we have $\Pr[v_2(\Lambda) = j] \leq 2^{-(j-k)}$, including $\Lambda = 0$ when $j = n$. Thus, in the uniform-coefficient experiment,

$$\Pr[\alpha\Lambda = \Delta] \leq 2^{-s} + \sum_{j=k+1}^{n} 2^{-(j-k)} 2^{-(n-j)} = (s+1)2^{-s}.$$

□

Let $\mathcal{S}$ be an ideal world adversary and $\mathcal{A}$ a real-world adversary controlling one corrupted party. In the beginning, $\mathcal{S}$ holds the corrupted party's input. We consider two different cases.

Case 1: P_0 **is corrupted:** $\mathcal{S}$ works as follows.

1) $\mathcal{S}$ first simulates the preprocessing step. $\mathcal{S}$ invokes $\mathcal{A}$ internally to get all the randomness used by P_0, and receives shares of $r^{(i)}_{xy}$ from P_0 on behalf of the honest parties. Then $\mathcal{S}$ simulates $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$. $\mathcal{S}$ first recovers $r^{(i)}_x, r^{(i)}_y$ using these shares received from P_0. Besides, it also recovers $r^{(i)}_{xy}$, denoted by $r'^{(i)}_{xy}$. Then it checks if $r'^{(i)}_{xy} =_k r^{(i)}_x \cdot r^{(i)}_y$ holds for all $i \in [m]$. If for some $i \in [m]$ the equation doesn't hold, then with probability ϵ, $\mathcal{S}$ sends $(\mathsf{corrupted}, 0)$ to P_0, and outputs whatever $\mathcal{A}$ outputs and terminates. Otherwise, $\mathcal{S}$ sends $\mathsf{undetected}$ to P_0, and also sends the received shares of $r^{(i)}_x, r^{(i)}_y, r^{(i)}_{xy}$ to P_0 as the honest parties' inputs to $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$, and waits for P_0 to return a message as each honest party's output. Then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and terminates. If all the checks are passed, $\mathcal{S}$ simulates $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$ sending accept to the honest parties, and continues to receive shares of α, $\widetilde{r}^{(i)}_x, \widetilde{r}^{(i)}_y$ as well as $\widetilde{r}_w, \widetilde{w}_0$ on behalf of the honest parties from $\mathcal{A}$, then proceeding to the next step.
2) $\mathcal{S}$ simulates the input sharing step. For each input wire, $\mathcal{S}$ emulates $\mathcal{F}_{\text{input}}$ and receives shares of this input held by P_0.
3) $\mathcal{S}$ recovers the authentication errors from P_0's preprocessing messages as in Section E-A. It sends no online verification messages to P_0. Simulating the honest parties' verification outcomes additionally requires matching their abort distribution, as discussed in Hyb_3 below.
4) No output reconstruction messages or online abort notifications are sent to P_0. The simulator outputs whatever $\mathcal{A}$ outputs and terminates.

Hybrid Arguments. The following hybrids describe the simulation of P_0's view and identify the remaining condition for joint-output simulation.

Hyb_1: In this hybrid, $\mathcal{S}$ uses the honest parties' inputs and honestly emulates the honest parties. This corresponds to the real world.

Hyb_2: In this hybrid, $\mathcal{S}$ runs Step (1) to simulate the preprocessing phase. The only difference between Hyb_2 and Hyb_1 is that now $\mathcal{S}$ simulates $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$ rather than the parties invoking it. Since $\mathcal{S}$ knows exactly the shares of honest parties received from P_0, it can tell whether P_0 has cheated or not. If P_0 behaved honestly, then $\mathcal{S}$ simulate $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$ returning accept to the honest parties, which is the same as in Hyb_1. Otherwise, P_0 returns $\mathsf{detected}$ with probability ϵ. This behaves the same as $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$ in Hyb_1. Additionally, if the functionality returns $\mathsf{undetected}$, since $\mathcal{S}$ knows the inputs of the honest parties (i.e., the received shares of the correlated randomness from P_0), it can return them to P_0 as $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$ does in Hyb_1. Thus, we have $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_1$.

Hyb_3: Online verification produces no messages to P_0, so replacing this phase preserves its view. This alone does not establish equivalence of the joint distributions: high-bit authentication errors can make the honest parties' abort probability depend on their inputs through Equation (3). Matching this distribution in the ideal world

Protocol D.2: $\Pi_{\mathsf{Cvt\text{-}MAC^{On}}}$ **(Securely Computing an Arithmetic Circuit over Ring $\mathbb{Z}_{2^k}$ with Abort)**
Input: Each party P_j where $j \in \{0,1,2\}$ holds inputs $v_j^{(i)} \in \mathbb{Z}_{2^k}$ where $i \in [d]$.
Auxiliary Input: The parties hold a description of an arithmetic circuit C over $\mathbb{Z}_{2^k}$ on inputs of length d. Let m be the number of multiplication gates in C. The parties agree on a parameter $s \in \mathbb{N}$. ***Output:*** Only P_1, P_2 receive circuit outputs or online abort notifications. After submitting its inputs, P_0 receives no online messages or functionality outputs.

Preprocessing:

1) ***Circuit Preprocessing***
   a) For each input wire w.r.t. the input $v_j^{(i)}$, the parties run the preprocessing step of Π_{Share} on $v_j^{(i)}$ over $\mathbb{Z}_{2^{k+s}}$.
   b) For each gate g in a predetermined topological order, if g is a multiplication (or addition) gate, the parties run the preprocessing step of Π_{Mul} on the two input sharings (or P_0 locally add its shares, respectively) over $\mathbb{Z}_{2^{k+s}}$.
   c) For each output wire v, P_0 sends $\mathsf{H}(r_{v,2})$ to P_1 and $\mathsf{H}(r_{v,1})$ to P_2 for the consistency checks in Rec.
2) ***Verification Preprocessing***
   a) Denote by $\{[r_x^{(i)}], [r_y^{(i)}], [r_{xy}^{(i)}]\}_{i=1}^m$ the m triples produced in the circuit preprocessing step. The parties call $\mathcal{F}_{\text{VrfyIP}}$ on these triple to check if $r_x^{(i)} \cdot r_y^{(i)} =_k r_{xy}^{(i)}$ for each $i \in [m]$. If an abort is received, the parties abort and then halt, otherwise they proceed to the next step.
   b) P_0 uniformly picks a randomness α from $\mathbb{Z}_{2^{k+s}}$, and computes $\widetilde{r}_x^{(i)} :=_{k+s} \alpha \cdot r_x^{(i)}$, $\widetilde{r}_y^{(i)} :=_{k+s} \alpha \cdot r_y^{(i)}$. Then P_0 additively shares α and $\{\widetilde{r}_x^{(i)}\}_{i=1}^m, \{\widetilde{r}_y^{(i)}\}_{i=1}^m$ to P_1, P_2.
   c) P_0 random picks a seed $\mathsf{sd} \in \{0,1\}^\kappa$, and additively shares it to P_1, P_2 along with the seed's commitment. P_0 run PRG with sd to derive m randomness $\{\eta^{(i)}\}_{i=1}^m$, and computes $\widetilde{w}_0 := \sum_{i=1}^m \eta^{(i)} \widetilde{r}_{xy}^{(i)}$. Let $\{r_z^{(i)}\}_{i=1}^m$ be the randomnesses prepared for the outputs of the m multiplication gates. P_0 computes $\widetilde{r}_w := \alpha \cdot \sum_{i=1}^m \eta^{(i)} \cdot r_z^{(i)}$, and additively shares $\widetilde{w}_0, \widetilde{r}_w$ to P_1, P_2.

Online: All communication and functionality outputs after input sharing are restricted to P_1, P_2.

1) ***Input Sharing***
   For each input wire w.r.t. the input $v_j^{(i)}$, the parties run the online step of Π_{Share} on $v_j^{(i)}$ over $\mathbb{Z}_{2^{k+s}}$.
2) ***Circuit Evaluation***
   For each gate g in the predetermined topological order, if g is a multiplication (or addition) gate, the parties run the online step of Π_{Mul} on the two input sharings to g (or P_1, P_2 locally add their shares, respectively) over $\mathbb{Z}_{2^{k+s}}$.
3) ***Verification***
   Let $\{[\![z^{(i)}]\!]\}_{i=1}^m$ be the outputs of the m multiplication gates generated in the previous step.
   a) P_1, P_2 compute $[\widetilde{w}] := \sum_{i=1}^m \eta^{(i)}(\delta_x^{(i)}\delta_y^{(i)}[\alpha] + \delta_x^{(i)}[\widetilde{r}_y^{(i)}] + \delta_y^{(i)}[\widetilde{r}_x^{(i)}]) + [\widetilde{w}_0]$.
   b) P_1, P_2 reconstruct sd, check the commitment, and derive m randomness $\{\eta^{(i)}\}_{i=1}^m$ using PRG with sd. Then P_1, P_2 compute $\delta_w := \sum_{i=1}^m \eta^{(i)}\delta_z^{(i)}$, and $[\widetilde{w'}] := \delta_w[\alpha] + [\widetilde{r}_w]$.
   c) P_1, P_2 compute $[o] :=_{k+s} [\widetilde{w}] - [\widetilde{w'}]$, and then call $\mathcal{F}_{\text{checkzero}}$ on $[o]$ to check if $o =_{k+s} 0$. If an reject is received, P_1, P_2 sends abort to each other and then halt.
   d) P_1, P_2 reconstruct α, and call $\mathcal{F}_{\text{coin}}$ to obtain $2m+2$ randomness $\{\xi^{(i)}\}_{i=1}^{2m+2}$. P_1, P_2 compute $[o'] := \sum_{i=1}^m \xi^{(i)}(\alpha[r_x^{(i)}] - [\widetilde{r}_x^{(i)}]) + \sum_{i=1}^m \xi^{(m+i)}(\alpha[r_y^{(i)}] - [\widetilde{r}_y^{(i)}]) + \xi^{(2m+1)}(\alpha \sum_{i=1}^m \eta^{(i)}[r_{xy}^{(i)}] - [\widetilde{w}_0]) + \xi^{(2m+2)}(\alpha[r_w] - [\widetilde{r}_w])$, and then call $\mathcal{F}_{\text{checkzero}}$ on $[o']$. If any reject is received, P_1, P_2 sends abort to each other and then halt.
4) ***Output Reconstruction***
   If no abort command is received in all the previous steps, for every output gate corresponding to P_j with $j \in \{1,2\}$, P_1, P_2 run Rec to reconstruct the output value to P_j, using the hash supplied by P_0 during preprocessing. Otherwise, P_1, P_2 sends abort to each other and halt.

requires a separate simulation argument; the soundness bound for errors nonzero modulo 2^k does not cover this case.

Hyb_4: Output reconstruction takes place only between P_1, P_2. Step (4) therefore preserves P_0's empty reconstruction transcript.

Hyb_5: In this hybrid, $\mathcal{S}$ runs Step (2) to simulate the input sharing phase. Compared to Hyb_4 where the parties invoke $\mathcal{F}_{\text{input}}$ to share their inputs as in the real world, now $\mathcal{S}$ emulates $\mathcal{F}_{\text{input}}$ by sending random values as the honest parties' shares to P_0. Due to the secrecy property of the secret sharing scheme and the use of PRG, the two distributions are (computationally) indistinguishable, and we have $\mathsf{Hyb}_5 \equiv_c \mathsf{Hyb}_4$.

These steps simulate P_0's view without honest inputs. Completing the joint-output simulation still requires resolving the abort distribution in Hyb_3.

Case 2: P_1 or P_2 is corrupted: W.L.O.G., we assume P_1 is the corrupted party. The ideal world adversary $\mathcal{S}$ works as follows.

1) $\mathcal{S}$ simulates the preprocessing step. $\mathcal{S}$ invokes $\mathcal{A}$ internally to get all the random shares of $r_x^{(i)}, r_y^{(i)}, r_z^{(i)}$ held by P_1. $\mathcal{S}$ plays the role of P_0 to share $r_{xy}^{(i)}$ by sending random values to P_1. Then $\mathcal{S}$ simulates $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$. As it knows all the additive shares of $r_x^{(i)}, r_y^{(i)}$ and $r_{xy}^{(i)}$ held by P_1, $\mathcal{S}$ checks the correctness of the messages sent from P_1 to $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$. If they are incorrect, $\mathcal{S}$ knows that P_1 has cheated; it then returns $\mathsf{detected}$ to P_1 with probability ϵ, outputs whatever P_1 outputs and then terminates. Otherwise, $\mathcal{S}$ returns $\mathsf{undetected}$ as well as valid shares of some random triples $(r_x^{(i)}, r_y^{(i)}, r_{xy}^{(i)})$ (w.r.t. P_1's shares) as the honest parties' inputs to P_1, and then receives a message from P_1 as each honest party's output. Then $\mathcal{S}$ outputs whatever P_1 outputs and then terminates. Otherwise, if P_1 is not cheating, $\mathcal{S}$ sends random values to P_1 as its shares of α $\widetilde{r}_x^{(i)}, \widetilde{r}_y^{(i)}$ (or invokes $\mathcal{A}$ internally to get its random shares of $\widetilde{r}_x^{(i)}, \widetilde{r}_y^{(i)}$).
2) $\mathcal{S}$ simulates the input sharing step. For each input wire, $\mathcal{S}$ emulates $\mathcal{F}_{\text{input}}$ and receives shares of this input held by P_1.
3) $\mathcal{S}$ simulates the circuit evaluation step. For each multiplication gate, $\mathcal{S}$ sends a random value to P_1 as $\delta_{z,2}^{(i)}$ on behalf of P_2.
4) $\mathcal{S}$ simulates the online verification step. As $\mathcal{S}$ knows all the inputs and randomness of P_1, as well as all the messages sent to P_1 by itself during the previous step, it can tell if P_1 has cheated by checking each $\delta_{z,1}^{(i)}$. If P_1 is not cheating, $\mathcal{S}$ sends a random value as sd_2 on behalf of P_2, and then simulate $\mathcal{F}_{\text{checkzero}}$ returning accept to P_1. Otherwise, $\mathcal{S}$ sends $(\mathsf{cheat}, 1)$ to the ideal functionality. If it receives $\mathsf{detected}$ from the ideal functionality, $\mathcal{S}$ picks sd_2 s.t. the derived randomness from the recovered sd satisfy that $\sum_{i=1}^{m} \eta^{(i)}\varphi^{(i)} \neq_{k+s} 0$, and sends sd_2 to P_1 on behalf of P_2. Then it simulates P_2 outputting abort, then outputs whatever $\mathcal{A}$ outputs and terminates. Otherwise, if $\mathcal{S}$ receives $\mathsf{undetected}$ and the honest parties' inputs from the ideal functionality, it picks sd_2 s.t. the derived randomness from the recovered sd satisfy that $\sum_{i=1}^{m} \eta^{(i)}\varphi^{(i)}$ has an inverse over $\mathbb{Z}_{2^{k+s}}$. Then it receives the share of o from P_1 as the input of $\mathcal{F}_{\text{checkzero}}$, then computes the potential additive error Δ (that is the difference between the actually received share from P_1 and the one that should be sent from P_1). Next $\mathcal{S}$ returns accept to P_1 on behalf of $\mathcal{F}_{\text{checkzero}}$, then computes $\alpha :=_{k+s} \Delta \cdot (\sum_{i=1}^{m} \eta^{(i)}\varphi^{(i)})^{-1}$, and obtains α_2 from the computed α and the share of α held by P_1. $\mathcal{S}$ sends α_2 to P_1 on behalf of P_2. Then it simulates $\mathcal{F}_{\text{coin}}$ sending public coefficients to P_1, and checks whether the share of o' sent by P_1 when calling $\mathcal{F}_{\text{checkzero}}$ is correct or not. If not, $\mathcal{S}$ emulates $\mathcal{F}_{\text{checkzero}}$ sending reject to P_1, otherwise it sends accept.
5) $\mathcal{S}$ simulates the output reconstruction step. $\mathcal{S}$ receives the outputs of the corrupted party P_1 from the ideal functionality. As $\mathcal{S}$ knows the inputs and randomness as well as all the received messages of P_1, for each output v, $\mathcal{S}$ calculates the shares $\delta_v, r_{v,t}$ held by P_1 of v. Then it further computes $r_{v,2} :=_{k+s} v - \delta_v - r_{v,t}$, and sends it to P_1 on behalf of P_2.
6) $\mathcal{S}$ simulates the cheater identification step as in Case 1.

Hybrid Arguments. We use the following hybrid arguments to show that the joint distribution of the views of the corrupted parties and the output of the honest parties are indistinguishable in the ideal world and the real world.

Hyb_1: In this hybrid, $\mathcal{S}$ uses the honest parties' inputs and honestly emulates the honest parties. This corresponds to the real world.

Hyb_2: In this hybrid, $\mathcal{S}$ runs Step (1) to simulate the preprocessing phase. The only difference between Hyb_2 and Hyb_1 is that now $\mathcal{S}$ simulates $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$ rather than the parties invoking it. Since $\mathcal{S}$ knows exactly the shares held by P_1, it can tell whether P_1 has cheated or not. If P_1 behaved honestly, then $\mathcal{S}$ sends accept to P_1, which is the same as in the real world. Otherwise, it returns $\mathsf{detected}$ with probability ϵ. This behaves the same as $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$ in the real world. Additionally, if $\mathcal{S}$ returns $\mathsf{undetected}$, then it can return valid randomness and the corresponding shares as honest P_0 and P_2's inputs as $\mathcal{F}^{\epsilon}_{\text{VrfyIP}}$ does in the real world. Thus, this makes no difference to $\mathcal{A}$ both in the ideal world and the real world, and we have $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_1$.

Hyb_3: In this hybrid, $\mathcal{S}$ runs Step (3) to simulate the circuit evaluation phase. The only difference between Hyb_3 and Hyb_2 is that $\mathcal{S}$ sends a random value rather than an honestly computed share $\delta_{z,2}$ to P_1. Since $r_{z,2}$ is a (pseudo-)random value generated by PRG and in Hyb_2, $\delta_{z,2}$ is masked by $r_{z,2}$, the views of $\mathcal{A}$ in two hybrids are (computationally) indistinguishable. Thus we have $\mathsf{Hyb}_3 \equiv_c \mathsf{Hyb}_2$.

Hyb_4: In this hybrid, $\mathcal{S}$ runs Step (4) to simulate the online verification phase. In Hyb_4, $\mathcal{S}$ simulates $\mathcal{F}_{\text{checkzero}}$ rather than the parties invoking it in Hyb_3. As $\mathcal{S}$ can exactly compute the additive error introduced by P_1, it can tell whether P_1 has cheated or not. If it detects cheating, it queries the ideal functionality and returns $\mathsf{detected}$ with probability ϵ. If the cheating is detected by the ideal functionality, then $\mathcal{S}$ chooses the seed accordingly s.t. the accumulated error doesn't vanish over $\mathbb{Z}_{2^{k+s}}$. Otherwise, $\mathcal{S}$ chooses the seed as well as α accordingly s.t. the accumulated error vanishes over $\mathbb{Z}_{2^{k+s}}$. This corresponds to the real-world execution. Thus we have $\mathsf{Hyb}_4 \equiv \mathsf{Hyb}_3$.

Hyb_5: In this hybrid, $\mathcal{S}$ runs Step 5 to simulate the output reconstruction phase. For each output of P_1, $\mathcal{S}$ can first compute the two additive shares of this output held by P_1 using its inputs and views. Then using the output value that $\mathcal{S}$ receives from the ideal functionality, $\mathcal{S}$ can calculate the additive share that should be sent from the honest parties to P_1. which are exactly the same as in Hyb_4. Thus, the two distributions are identical, and $\mathsf{Hyb}_5 \equiv \mathsf{Hyb}_4$.

Hyb_6: In this hybrid, $\mathcal{S}$ runs Step 2 to simulate the input sharing step as in Case 1. Analogously, we have $\mathsf{Hyb}_6 \equiv_c \mathsf{Hyb}_5$.

Hyb_7: In this hybrid, the honest parties use the outputs received from the functionality as in Case 1. Analogously, we have $\mathsf{Hyb}_7 \equiv \mathsf{Hyb}_6$.

Note that Hyb_7 corresponds to the ideal world scenario, and in all the above hybrids, the joint distribution of the views of the adversary and the output of the honest parties remains unchanged. This shows that the two distributions in the ideal world and the real world are (computationally) indistinguishable. This concludes our proof.

APPENDIX F
SOUNDNESS ANALYSIS OF PROTOCOL E.1

For concreteness, we instantiate the protocol using the RSS scheme. A single-bad-triple sampling argument is insufficient because two incorrect triples can occupy the same bucket and pass its check. The following argument therefore establishes the worst-case algebraic soundness of the preprocessing phase: a soundness failure means that verification accepts while at least one incorrect triple is retained. We consider a static adversary that corrupts at most one party and work in the additive-attack model stated in the preliminaries. We analyze the ideal-shuffle and ideal-consistency hybrid and use the ordering implicit in Protocol E.1: all shares defining the $M+1$ triples are fixed before the shuffle. The shuffle is then sampled freshly and uniformly,

Protocol E.1: $\Pi_{\mathsf{Cvt\text{-}Triple}^{\mathsf{On}}}$ **(Securely Computing Binary Circuits with Abort)**

Input: Each party P_j where $j \in \{0,1,2\}$ holds inputs $v_j^{(i)} \in \mathbb{Z}_2$ where $i \in [d]$.

Auxiliary Input: The parties hold a description of an binary circuit C over $\mathbb{Z}_2$ on inputs of length d. Let N be the number of AND gates in C. The parties agree on parameters M and T s.t. $M - T = N$.

Preprocessing:

1) ***Generate Random Triples***
   a) The parties locally generate $M+1$ pairs $(\llbracket a^{(i)} \rrbracket, \llbracket b^{(i)} \rrbracket)$ using PRG with shared keys for $i \in [M+1]$, and run the multiplication protocol to obtain $\llbracket c^{(i)} \rrbracket := \llbracket a^{(i)} \rrbracket \cdot \llbracket b^{(i)} \rrbracket$.
2) ***Cut-and-Bucket***
   a) The parties randomly shuffle the random $M+1$ triples, and open the first triple to check its correctness.
   b) If the opened triple is correct, the parties divide the first $2T$ unopened triples into T buckets, each of size 2, and verify the first triple with the second one in each triple.
   c) If all the checks are passed, the parties outputs the left unchecked $M - 2T$ triples and all the first triples in these T buckets.

Online:

1) ***Input Sharing***
   The parties call $\mathcal{F}_{\text{input}}$ to share their inputs.
2) ***Circuit Evaluation***
   Let $\{\llbracket a^{(i)} \rrbracket, \llbracket b^{(i)} \rrbracket, \llbracket c^{(i)} \rrbracket\}_{i=1}^{N}$ be the prepared AND triples. For each gate g in the predetermined topological order:
   a) If g is an addition gate, the parties locally add the two input sharings.
   b) If g is an AND gate (w.r.t. index i) with input sharings $\llbracket x^{(i)} \rrbracket$ and $\llbracket y^{(i)} \rrbracket$, the parties compute $\rho^{(i)} := x^{(i)} \oplus a^{(i)}$ and $\sigma^{(i)} := y^{(i)} \oplus b^{(i)}$. Then they run Rec to open $\rho^{(i)}, \sigma^{(i)}$, and locally compute $\llbracket z^{(i)} \rrbracket := \rho^{(i)} \cdot \sigma^{(i)} \oplus \rho^{(i)} \cdot \llbracket b^{(i)} \rrbracket \oplus \sigma \cdot \llbracket a^{(i)} \rrbracket \oplus \llbracket c^{(i)} \rrbracket$.
3) ***Output Reconstruction:***
   If no abort command is received in all the previous steps, for every output gate corresponding to some party P_j, the parties run Rec to reconstruct the output value to P_j. Otherwise, they broadcast abort to the other parties and halt.

independently of all messages and randomness generated so far. Consistency or reconstruction failures cause rejection. A computational implementation of these idealized steps adds its distinguishing or failure probability to the bound below.

For a triple $\tau = (x, y, z) \in \mathbb{Z}_2^3$, define its error bit by

$$e(\tau) := z \oplus xy.$$

Thus, τ is correct if and only if $e(\tau) = 0$. Let $\tau' = (a, b, c)$ be the second triple in the same bucket, and set $\rho = x \oplus a$ and $\sigma = y \oplus b$. The value opened in the bucket check is

$$\begin{aligned} z \oplus \big(\rho\sigma \oplus \rho b \oplus \sigma a \oplus c\big) &= (z \oplus xy) \oplus (c \oplus ab) \\ &= e(\tau) \oplus e(\tau'). \end{aligned}$$

Consequently, over $\mathbb{Z}_2$ a bucket passes exactly when its two triples have the same error bit: a mixed correct–incorrect bucket is rejected, whereas a correct–correct or incorrect–incorrect bucket passes.

Let M, T be integers satisfying $1 \le T \le \lfloor M/2 \rfloor$, and let

$$U := M - 2T$$

be the number of unchecked positions. In addition to these U positions, the protocol retains the first triple in each of the T buckets, for a total of $U + T = M - T = N$ output triples. We prove that the exact worst-case soundness error of the verification procedure is

$$\epsilon_{\mathsf{snd}}(M, T) = \frac{\max\{M - 2T, 1\}}{M + 1}. \tag{7}$$

Proof F.1: Fix all messages and randomness generated up to, but not including, the fresh shuffle coins, and condition on the resulting pre-shuffle state. This fixes every error bit, including any errors chosen adaptively from the preceding transcript, while the shuffle remains uniform and independent. Let t be the number of incorrect triples. The case $t = 0$ cannot cause a soundness failure, so suppose $1 \le t \le M + 1$.

The random shuffle is equivalent to assigning the triples uniformly to the following labeled positions: one opening position, two ordered positions in each of T buckets, and U unchecked positions. Hence, the positions of the t incorrect triples form a uniformly random t-subset of the $M + 1$ positions. Let E_t be the layout event that the opening position is correct and every bucket contains either zero or two incorrect triples. This is exactly the event that the fixed nonzero error pattern evades all algebraic checks. Any nonempty position set satisfying these conditions leaves an incorrect output: an incorrect triple in an unchecked position is retained directly, and an incorrect–incorrect bucket retains its first triple. A voluntary abort can only turn such an event into an abort, not into an accepted incorrect output.

Suppose that exactly r buckets contain two incorrect triples. The remaining $t - 2r$ incorrect triples must occupy unchecked positions. Therefore, with the convention $\binom{n}{k} = 0$ for $k < 0$ or $k > n$, the exact probability of this favorable layout, conditioned on t, is

$$\Pr[E_t] = \frac{A_t}{\binom{M+1}{t}}, \qquad A_t := \sum_{r=0}^{T} \binom{T}{r} \binom{U}{t - 2r}. \tag{8}$$

It remains to bound A_t. Let $\mathcal{P}$ be the set of the M non-opening positions, let $\mathcal{U} \subseteq \mathcal{P}$ be the set of the U unchecked positions, and

let $\mathcal{F}_t$ be the family of t-subsets of $\mathcal{P}$ that intersect every bucket in either zero or two positions. Equation (8) gives $|\mathcal{F}_t| = A_t$. Define

$$V := \max\{U, 1\}$$

and choose a label set $\mathcal{Q}$ of size V as follows. If $U > 0$, let $\mathcal{Q} := \mathcal{U}$ and fix $q_\star \in \mathcal{Q}$; if $U = 0$, let $\mathcal{Q} := \{\star\}$ and set $q_\star := \star$.

Consider all compatible sets with one marked position,

$$\mathcal{D}_t := \{(F, z) \mid F \in \mathcal{F}_t,\ z \in F\},$$

so that $|\mathcal{D}_t| = tA_t$. Define

$$\Phi : \mathcal{D}_t \longrightarrow \mathcal{Q} \times \{S \subseteq \mathcal{P} \mid |S| = t - 1\}$$

by

$$\Phi(F, z) := \begin{cases} (z, F \setminus \{z\}), & \text{if } U > 0 \text{ and } z \in \mathcal{U}, \\ (q_\star, F \setminus \{z\}), & \text{if } z \text{ is a bucket position.} \end{cases}$$

This map is injective. In the first case, its second component intersects every bucket in zero or two positions; the first component identifies z. In the second case, its second component has exactly one bucket containing a single position; this bucket identifies the missing mate z. The two cases cannot collide because their second components have different bucket patterns. When $U = 0$, only the second case occurs and the same recovery argument applies. It follows that

$$tA_t \le V \binom{M}{t-1} = \frac{tV}{M+1} \binom{M+1}{t}, \tag{9}$$

and Equations (8) and (9) yield

$$\Pr[E_t] \le \frac{V}{M+1} = \frac{\max\{M - 2T, 1\}}{M+1}. \tag{10}$$

The bound is tight. If $U > 0$, an adversary that introduces one incorrect triple succeeds exactly when that triple occupies one of the U unchecked positions, with probability $U/(M+1)$. If $U = 0$, an adversary that introduces two incorrect triples succeeds exactly when they occupy the two positions of one bucket, with probability

$$\frac{T}{\binom{2T+1}{2}} = \frac{1}{M+1}.$$

Thus, Equation (7) is the exact worst-case error of the idealized verification procedure for every allowed M, T. Since the bound holds for every fixed pre-shuffle state, averaging over that state also proves it for a randomized adversary. The probability here is unconditional; choosing to abort cannot increase the probability of an accepted incorrect output.

In the ideal-shuffle and ideal-consistency hybrid, this preprocessing verification achieves algebraic deterrent factor ϵ exactly when

$$\max\{M - 2T, 1\} \le (1 - \epsilon)(M + 1). \tag{11}$$

In the usual case $2T < M$, this is equivalent to

$$T \ge \left\lceil \frac{\epsilon(M+1) - 1}{2} \right\rceil.$$

Still assuming $2T < M$ and using $M - T = N$, the same condition can be written as

$$T \ge \left\lceil \frac{\epsilon(N+1) - 1}{2 - \epsilon} \right\rceil.$$

If the concrete shuffle and consistency mechanisms have total distinguishing or failure probability η, the sufficient real-implementation condition is

$$\frac{\max\{M - 2T, 1\}}{M+1} + \eta \le 1 - \epsilon.$$

In particular, when M is divisible by four, $T = M/4$ already gives a soundness error strictly below $1/2$; no additional bucket is required.

Algorithm G.1: Blame
Input: The view of party P_j.
Auxiliary Input: A signature scheme $\mathsf{SG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$.

Algorithm:

1) Parse the last message in the cheater identification procedure as $(\mathsf{corrupted}, \mathsf{id}_i)$.
2) Run $\sigma_j := \mathsf{SG.Sign}(\mathsf{sk}_j; \mathsf{id}_i)$, and sends σ_j to P_{3-i-j}.
3) After receiving σ_{3-i-j}, output $\mathsf{cert}_j := (\mathsf{id}_i, \sigma_j, \sigma_{3-i-j})$.

Algorithm G.2: Judge
Input: A certificate cert_j.
Auxiliary Input: A signature scheme $\mathsf{SG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$.

Algorithm:

1) Parse $\mathsf{cert}_j := (\mathsf{id}_i, \sigma_j, \sigma_{3-i-j})$.
2) For each $t \in \{j, 3-i-j\}$, run $\mathsf{SG.Vrfy}(\mathsf{pk}_t; \mathsf{id}_i, \sigma_t)$. If both checks are passed, output id_i, otherwise output $\perp$.

APPENDIX G
DETAILED CHEATER IDENTIFICATION MECHANISMS

A. General Cheater Identification

Note that the above cheater identification approach cannot apply to the normal (symmetric) MAC-based and triple-based approaches, as it's hard to identify an honest party without introducing other primitives. We thus design a general cheater identification mechanism using CP-NIZK proofs. The high-level idea is that, once an abortion occurs, we require each party to generate a zero-knowledge proof showing that it faithfully executed the protocol w.r.t. all the commitments (to its input and random tape) and signed messages.

To be more specific, after each party P_j picks its random tape tp_j, we require P_j to commit to its input $\vec{v}_j$ and random tape tp_j, and then broadcast the commitment (denoted by $\mathsf{Com}_{\vec{v}_j}, \mathsf{Com}_{\mathsf{tp}_j}$) to the other parties. The hiding property of the commitment scheme ensures the privacy of the input and random tape, and the binding property guarantees that the corrupted party cannot forge other inputs or random tape after they are chosen. When an abortion occurs during the execution of the protocol, the party P_j first commits to all the signatures of its sent messages, and broadcasts the commitments, denoted by $\mathsf{Com}_{\vec{\sigma}_{j,i}}$ for $i \in \{j-1, j+1\}$ to the other parties, and meanwhile receives a commitment $\mathsf{Com}_{\vec{\sigma}_{i,j}}$ from each P_i. Then P_j is required to generate a zero-knowledge proof π_j showing that: all its sent messages with signatures (w.r.t. $\mathsf{Com}_{\vec{\sigma}_{j,i}}$) are computed faithfully according to the protocol specification using its inputs (w.r.t. $\mathsf{Com}_{\vec{v}_j}$), random tape ($\mathsf{Com}_{\mathsf{tp}_j}$), and all the previously received messages with signatures (w.r.t. $\mathsf{Com}_{\vec{\sigma}_{i,j}}$).

Note that once the corrupted party ever cheated and led to the abortion, it cannot generate a valid proof showing its "honest" behaviors given the knowledge soundness property of the proof scheme. Thus, the party who cannot provide a valid proof will be identified as a cheater. Meanwhile, the zero-knowledge property ensures that no extra information is leaked about the honest parties' private inputs.

We give a formal description of the above procedure in Protocol G.1. We claim this approach applies to any input-dependent

Protocol G.1: $\Pi_{\mathbf{Cov}}^{\mathbf{PV}}$ **(General Cheater Identification)**

Input: Each party P_j where $j \in \{0,1,2\}$ holds inputs $v_j^{(i)} \in \mathbb{Z}_{2^k}$ where $i \in [d]$.

Auxiliary Input: A covertly secure protocol $\Pi_{\text{Cov}}^{\text{Abort}}$ with abort. The parties agree on a signature scheme $\mathsf{SG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$, a commitment scheme $\mathsf{COM} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open})$, and a zero-knowledge proof scheme $\mathsf{ZK} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Vrfy})$. Let $\mathsf{NMF}_j^{(\ell)}$ be the next message function of $\Pi_{\text{Cov}}^{\text{Abort}}$ that is executed by P_j in round ℓ, which takes as input the current view $\mathsf{view}_j^{(\ell)}$, and outputs the messages that need to be sent.

1. Setup

1) Each party P_j runs $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow \mathsf{SG.Gen}(1^\kappa)$ to generate a signing key sk_j and a verification key pk_j, and broadcasts pk_j to the other two parties. The parties run $\mathsf{pp} \leftarrow \mathsf{COM.Gen}(1^\kappa)$ and $\mathsf{crs} \leftarrow \mathsf{ZK.Gen}(1^\kappa)$ to obtain the parameters pp and crs.

2. Commitment

1) Each party P_j samples a random tape denoted by tp_j. Let $\vec{v}_j$ be the vector of P_j's inputs. P_j runs $\mathsf{COM.Commit}(\mathsf{pp}; \vec{v}_j)$ and $\mathsf{COM.Commit}(\mathsf{pp}; \mathsf{tp}_j)$ to obtain commitments $\mathsf{Com}_{v_j}, \mathsf{Com}_{\mathsf{tp}_j}$, and broadcast the two commitments to the other two parties.

3. Protocol Execution

The parties run protocol $\Pi_{\text{Cov}}^{\text{Abort}}$ with their individual inputs and random tape. For each round ℓ of protocol $\Pi_{\text{Cov}}^{\text{Abort}}$:

1) For each party P_j, let $\mathsf{view}_j^{(\ell)}$ be the current view of it. P_j runs $(\mathsf{msg}_{j,j-1}^{(\ell)}, \mathsf{msg}_{j,j+1}^{(\ell)}) = \mathsf{NMF}_j^{(\ell)}(\mathsf{view}_j^{(\ell)})$ to compute the messages that should be sent to the other parties. If any message is abort, it broadcasts abort and goes to Step 4. Otherwise, it proceeds to the next step.
2) Each party P_j runs $\sigma_{j,i}^{(\ell)} := \mathsf{SG.Sign}(\mathsf{sk}_j; \ell||j||i||\mathsf{H}(\mathsf{msg}_{j,i}^{(\ell)}))$ to sign the messages, and sends the signature $\sigma_{j,i}^{(\ell)}$ along with the message to P_i. For each message $\mathsf{msg}_{i,j}^{(\ell)}$ received by P_j, P_j verifies the signature $\sigma_{i,j}^{(\ell)}$ by running $\mathsf{SG.Vrfy}(\mathsf{pk}_i; \mathsf{msg}_{i,j}^{(\ell)}, \sigma_{i,j}^{(\ell)})$.
3) If the signature $\sigma_{i,j}^{(\ell)}$ is invalid, P_j broadcasts (accuse, i) to the other parties. P_i then broadcasts another message $\hat{\mathsf{msg}}_{i,j}^{(\ell)}$ with a signature $\hat{\sigma}_{i,j}^{(\ell)}$ to the other two parties, who then check if the new signature is valid w.r.t the new message. If it is still invalid, the other two parties broadcast $(\mathsf{corrupted}, i)$ and go to Step 4. Otherwise, P_j uses the new message in the following execution.

4. Cheating Identification

Let ρ be the current number of rounds in the execution of the protocol $\Pi_{\text{Cov}}^{\text{Abort}}$.

1) Each party P_j runs $\pi_j := \mathsf{ZK.Prove}(\mathsf{crs}; s, w)$ to prove:
   a) for each $\ell \in [\rho]$, there exist some secret messages $\mathsf{msg}_{j,j-1}^{\ell}, \mathsf{msg}_{j,j+1}^{\ell}$ that are correctly computed w.r.t. the function $\mathsf{NMF}_j^{(\ell)}$ and the current view, i.e., $(\mathsf{msg}_{j,j-1}^{\ell}, \mathsf{msg}_{j,j+1}^{\ell}) = \mathsf{NMF}_j^{(\ell)}(\mathsf{view}_j^{(\ell)})$, and
   b) for all $\ell \in [\rho]$ and $i \in \{j-1, j+1\}$, the signatures $\sigma_{j,i}, \sigma_{i,j}$ are correctly computed w.r.t. the signature scheme and the messages, i.e., $\sigma_{j,i} = \mathsf{Sign}(\mathsf{sk}_j; \ell||j||i||\mathsf{H}(\mathsf{msg}_{j,i}^{(\ell)}))$.
2) Each party P_j signs its proof π_j and gets a signature σ_{π_j}. If the last message sent by P_j before Step 4 is $(\mathsf{corrupted}, i)$, P_j sends its proof and signature to the other party P_{3-i-j}, then outputs $(\mathsf{corrupted}, i)$ and halts. Otherwise, it broadcasts the proof and the signature to the other parties. and proceeds to the next step.
3) If P_j doesn't receive a proof and signature from P_i, it outputs $(\mathsf{corrupted}, i)$ and halts. Otherwise, it verifies the received proofs by running $\mathsf{ZK.Vrfy}(\mathsf{crs}; \pi_i)$ and $\mathsf{SG.Vrfy}(\mathsf{pk}_i; \pi_i, \sigma_{\pi_i})$. If any check on π_i or σ_{π_i} fails, it outputs $(\mathsf{corrupted}, i)$ as well.

5. Output

1) If no $(\mathsf{corrupted}, i)$ is output in previous steps, the parties output what the protocol $\Pi_{\text{Cov}}^{\text{Abort}}$ outputs.

online protocols, with no extra costs in the optimistic case (apart from those coming from commitments and signatures). Besides, almost all modern ZKP schemes are (or can be simply adapted to) CP-NIZK proof systems, and they typically feature different merits (such as $O(N)$ proving time, or $O(1)$ proof size and verifier time). Therefore, one can choose a CP-NIZK scheme according to the practical needs.

B. Blame *and* Judge *Algorithm*

We show the Blame and Judge algorithms in Algorithm G.1 and G.2. We give the following theorem.

Theorem G.1: Let f be a three-party functionality over $\mathbb{Z}_{2^k}$. For any protocol Π_{Cov}^{Abort} securely computes f with covert security with abort and deterrence factor ϵ, $(\Pi_{Cov}^{PV}, \mathsf{Blame}, \mathsf{Judge})$ securely computes f in the presence of a covert adversary controlling one party, with public verifiability and deterrence factor ϵ.

Proof G.1: We prove the following three properties.

Simulatability. Simulatability follows from $\Pi_{\text{Cov}}^{\text{Abort}}$ as it securely computes f with covert security with abort.

Public Verifiability. We now show that if in a real execution the honest parties output $(\mathsf{corrupted}, i)$, then the probability that the algorithm Judge outputs $\perp$ rather than P_i is negligible. First note that in $\Pi_{\text{Cov}}^{\text{PV}}$, the parties output $(\mathsf{corrupted}, i)$ in the following three cases:

1) the party P_i fails to broadcast a message with a valid signature, or the commitment to its signatures is invalid,
2) the party P_i fails to broadcast a proof and a signature in the cheating identification phase, or
3) the proof or the signature of the proof from P_i is invalid.

In the first case, the parties proceed to generate zero-knowledge proofs to show their honest behaviors, and send their proofs to each other as certificates, then outputting $(\mathsf{corrupted}, i)$. In the second and third cases, the parties directly output $(\mathsf{corrupted}, i)$. Note that in the first two cases, the certificate of each honest party consists of two groups of proofs and signatures, which all come from the honest parties, while in the third case, the certificate consists of the proofs coming from all the parties.

The event that the algorithm Judge outputs $\perp$ indicates that

- In the first two cases, the check $\mathsf{ZK.Vrfy}(\mathsf{crs}; \pi_t)$ or $\mathsf{SG.Vrfy}(\mathsf{pk}_t; \pi_t, \sigma_{\pi_t})$ fails for some $t \in \{i-1, i+1\}$, or
- In the third case, the checks $\mathsf{ZK.Vrfy}(\mathsf{crs}; \pi_i)$ and $\mathsf{SG.Vrfy}(\mathsf{pk}_i; \pi_i, \sigma_{\pi_i})$ succeed, or the check on $\mathsf{ZK.Vrfy}(\mathsf{crs}; \pi_t)$ or $\mathsf{SG.Vrfy}(\mathsf{pk}_t; \pi_t, \sigma_{\pi_t})$ fails for some $t \in \{i-1, i+1\}$.

Due to the completeness of the zero-knowledge proof scheme and the correctness of the signature scheme, the proofs and signatures generated by the honest parties will always pass the checks. Due to the knowledge soundness of the zero-knowledge proof scheme and the unforgeability of the signature scheme, a malicious party cannot forge a proof or signature to pass the check except with negligible probability. Thus, we can conclude that in all three cases, the probability that the algorithm Judge outputs $\perp$ is negligible, and $\Pr[\mathsf{Judge}(\mathsf{cert}_j) = \mathsf{id}_i] \geq 1 - \mathsf{negl}(\kappa)$.

Defamation Freeness. If a corrupted party P_i controlled by $\mathcal{A}$ can successfully generate a certificate s.t. $\mathsf{Judge}(\mathsf{cert}_i) = \mathsf{id}_j$ for $j \in \{i-1, j+1\}$, then it must hold that

- P_i successfully generated a valid proof π_i with a signature that passes the checks, and
- P_i forges an invalid proof or signature of P_j.

Given the knowledge soundness property of the zero-knowledge proof scheme, if any signed message sent by a corrupted party is inconsistent with the protocol specification, the probability that it can fake a proof to show the message is honestly computed is negligible. Thus, although a corrupted P_i can generate an invalid proof or signature and claim that they come from a "cheated" P_j, it cannot make Judge output P_j as it cannot generate another valid proof π_i to pass the check. Thus, we conclude that $\Pr[\mathsf{Judge}(\mathsf{cert}^*) = \mathsf{id}_j \mid \mathsf{cert}^* \leftarrow \mathcal{A}] \leq \mathsf{negl}(\kappa)$. This concludes our proof. □