# MaSTer: Maliciously Secure Truncation for Replicated Secret Sharing without Pre-Processing

Martin Zbudila, Erik Pohle, Aysajan Abidin, and Bart Preneel

COSIC, KU Leuven, Belgium
firstname.lastname@esat.kuleuven.be

**Abstract.** Secure multi-party computation (MPC) in a three-party, honest majority scenario is currently the state-of-the-art for running machine learning algorithms in a privacy-preserving manner. For efficiency reasons, fixed-point arithmetic is widely used to approximate computation over decimal numbers. After multiplication in fixed-point arithmetic, truncation is required to keep the result's precision. In this paper, we present an efficient three-party truncation protocol secure in the presence of an active adversary without pre-processing and improve on the current state-of-the-art in MPC over rings using replicated secret sharing (RSS). By adding an efficient consistency check, we lift the efficient but only passively secure three-party truncation protocol from the ABY3 framework by Mohassel and Rindal into the malicious setting without pre-processed data. Our benchmark indicates performance improvements of an order of magnitude in the offline phase for a single batch training. Finally, we apply our protocol to a real-world application for diagnostic prediction based on publicly available ECG heartbeat data. We achieve an improvement by a factor of two in the total throughput for both LAN and WAN settings.

**Keywords:** truncation · fixed-point arithmetic · replicated secret sharing · privacy-preserving machine learning

## 1 Introduction

The advancements in artificial intelligence (AI) and machine learning (ML) have brought about significant changes, ranging from predictive analytics to personalised recommendations, reshaping our interaction with the digital world. Because these technologies rely on substantial amounts of potentially sensitive data, ensuring data privacy has become a critical concern during the training of AI/ML models and the execution of AI/ML model inference. This has prompted the need for efficient privacy-enhancing tools to address these privacy challenges. Secure multi-party computation (MPC), a cryptographic technique that enables multiple distrustful parties to collaborate on computations without exposing their input data, stands out as a promising privacy-enhancing solution in the fast evolving field of ML. As long as the adversary corresponds to the threat model for the MPC protocol, data privacy can be ensured. Indeed,

privacy-preserving ML (or PPML) based on MPC has been an active area of research in recent years, see for example [29,28,35,11,23,31] and the references therein. Note that in MPC-based PPML, the data owners do not necessarily participate in the computation. There are scenarios where the computing parties are independent of the data owners, allowing the incorporation of data from an arbitrary number of data owners. The state-of-the-art in MPC-based PPML in the three-party setting considers security in an honest majority setting.

One of the key components for high-performance evaluation of PPML algorithms over MPC is the efficient computation on secret-shared decimal numbers. Since accurate floating point arithmetic is expensive, an approximation, namely fixed-point arithmetic, is employed. Here, one encodes a decimal number with fixed precision as an integer of size large enough to accommodate the expected range. Multiplication of two fixed-point integers doubles the length of the fractional part and thus requires efficient truncation afterwards to retain the original precision. Since computation over the ring $\mathbb{Z}_{2^\ell}$, $\ell \geq 1$ allows for efficient utilisation of native instructions in modern processors, secret sharing over $\mathbb{Z}_{2^\ell}$ enables efficient MPC where one considers the fixed-point integers as ring elements.

Truncation is an essential part when working with fixed-point arithmetic. Since it allows retaining the original precision by discarding the least significant bits, one does not need to choose a large ring to accommodate the doubling of the length of the fractional part, allowing us to limit the ring size. SecureML [29] first proposed a technique allowing *probabilistic* truncation by local operations on the shares in a two-party setting in semi-honest security. Here, "probabilistic" means that the value reconstructed from the truncated shares is offset by at most 1 from the correct value with high probability.

The current state-of-the-art three-party maliciously secure truncation is the ABY3 protocol by Mohassel and Rindal [28], adopted by follow-up works such as the Falcon protocol [35]. In the online phase, this truncation requires only one round of communication and one ring element to be sent per party. The pre-processing phase for truncation involves a costly evaluation of subtraction circuits, resulting in lower combined throughput. However, in the context of real-time applications such as diagnostic disease prediction based on sensitive data (cf. Sect. 7), the bottleneck caused by the pre-processing phase can be more pronounced. Real-time users would require immediate responses and queries with batches of small sizes. Such real-time application scenarios underscore the inefficiency of the offline phase and call for the elimination of the pre-processing.

In this paper, we design and implement a replicated secret sharing (RSS) based protocol that computes the truncation of fixed-point numbers without pre-processing securely in the presence of one malicious adversary in the three-party setting. Our protocol achieves an overall low cost of one amortised round of communication and one ring element sent per party. Our protocol augments the truncation method from SecureML [29] and is inspired by its extension to a three-party RSS environment in the semi-honest setting from the ABY3 framework [28]. We benchmark our proposal in MP-SPDZ [22] and compare its performance with that of ABY3 [28]. We also report on the PPML training of MNIST

data in the SecureML (3-layer dense) and LeNet-5 network architectures and the training of the CIFAR-10 dataset on the AlexNet architecture.

**Contributions.** We make the following contributions:

- We propose a novel maliciously secure truncation protocol designed specifically for replicated secret-sharing (RSS) for three parties. Our secure truncation protocol builds upon the SecureML [29] probabilistic truncation method and its subsequent extension to a three-party RSS environment in the semi-honest setting using the ABY3 framework [28]. Unlike other truncation protocols for malicious security, our truncation protocol does not require pre-processing.
- We provide detailed error analysis on the errors occurring in the SecureML truncation, expanding on the correctness analysis provided by [29, Theorem 1]. We further provide a thorough security analysis of our proposed protocol and prove its security against a static malicious adversary with abort. Our protocol gives the adversary a bit more power which is appropriately captured in the functionality. We show that a model evaluated with sufficiently high fixed-point precision is robust to this adversarial influence.
- To assess the practical performance of our protocol, we conduct benchmarking experiments using the MP-SPDZ [22] framework and compare the efficiency of our truncation protocol against the state-of-the-art ABY3 malicious truncation. We make our implementation public on github[1]. By eliminating the pre-processing phase for truncation due to our protocol, the execution time of the offline phase for training one epoch is improved by up to an order of magnitude. Similarly, by removing the pre-processing phase of the truncation protocol, the total communication of the offline phase for training is lowered by an order of magnitude.
- To illustrate the practical relevance of our truncation protocol, we present an application in the context of privacy-preserving neural network inference for the classification of ECG heartbeat signals. Our neural network identifies the occurrence of arrhythmia in a heartbeat signal, and classifies the signal into one of five classes, in accordance to the AAMI EC57 standard, as described by Kachuee et al. [21]. We achieve an accuracy of 95.9%. We improve the total running time, and thus the overall throughput by up to a factor of three in both LAN and WAN settings.

The remainder of the paper is organised as follows. After an overview of the related works in Sect. 2, we introduce the preliminary background necessary to understand the paper in Sect. 3. Section 4 presents our maliciously secure three-party truncation protocol. We prove the security of the protocol in the UC framework in Sect. 5. We benchmark our protocol in the MP-SPDZ [22] framework and compare it to ABY3 malicious truncation in Sect. 6. We discuss a concrete use-case for PPML in Sect. 7 and conclude the paper in Sect. 8.

[1] `https://github.com/KULeuven-COSIC/master-truncation`

## 2 Related Work

Machine learning operations involve arithmetic on decimal numbers expressed in a fixed-point representation to achieve maximal efficiency in MPC. Therefore, truncation is a crucial component when designing an MPC framework for machine learning. Currently, the most efficient protocols for evaluating ML models are based on three-party replicated secret sharing in the semi-honest security setting. For practical deployments, semi-honest security might not be sufficient as a security guarantee, however, only a few works focus on malicious security.

Regarding truncation in MPC over rings, we distinguish between relatively efficient approaches achieving probabilistic truncation and approaches focusing on a faithful truncation result that are much more heavy in communication. A faithful, sometimes called precise truncation, denotes a truncation protocol that outputs a secret-sharing of an arithmetic right shift of the value with probability 1. Some prior works achieving faithful truncation allow an error of $\pm 1$ on the least significant bit (LSB), which we will refer to as "small". On the other hand, probabilistic truncation fails to output a correct truncation result (accounting for the small error) with a non-zero probability, we will call such error "big".

The probabilistic approaches follow the line of work proposed by Catrina et al. [9], with ABY3 [28] formalising the approach for three-party RSS admitting one malicious adversary. Multiple follow-up works adapted these truncation protocols, including Astra [10] and Falcon [35]. Trident [11] builds on the approach of ABY3 by proposing optimisations to the pre-processing phase. Blaze [31] and SWIFT [23] similarly adopt the optimisations proposed by Trident [11] elevating the security to guaranteed output delivery (GOD) in both the three and four-party settings. Taking a different approach to the pre-processing phase, Escudero et al. [15] propose a protocol inspired by the online phase of ABY3 that removes the big error by handling overflow using edaBits (extended daBits [33]). These pre-processed shared bits help with arithmetic-to-Boolean conversions between sharings and are directly applicable to comparison and truncation protocols, compatible with any corruption setting.

Our protocol is also designed for three-party RSS in a malicious setting. However, we take a different approach to achieve malicious security compared to the above-mentioned works. Instead, our protocol utilises the semi-honest three-party truncation that was proposed by the ABY3 framework.

Following the line of work of edaBits [15], Baccarini et al. [3] propose an improved generation of these pre-processed bits, designing multiple protocols for $n$-party RSS. However, while their pre-processing phase for the truncation triples improves on the approach of Escudero et al., they achieve probabilistic truncation with a constant number of rounds in the semi-honest setting with high communication cost (see Table 1). Finally, Piranha [36] and Fantastic Four [14] build a truncation protocol on a combination of SWIFT [23] and the work of Dalskov et al. [13]. Similar to SWIFT, Fantastic Four [14] also achieves malicious security with GOD in the four-party setting.

In the two-party scenario, SecureML [29] proposed a local probabilistic truncation, requiring no communication, with similar error probabilities as in the

three-party case. Subsequently, a line of works focusing on faithful truncation emerged, with the most recent ones including Cryptflow2 [32] or the work of Zou et al. [38]. Furthermore, Cheetah [20] reduces the communication cost compared to the previous works at the cost of admitting the small error. Bicoptor 2.0 [37] modifies the SecureML truncation protocol to obtain a faithful truncation. However, their resulting shares are in a smaller ring and thus would require an additional modulo switch protocol for compatibility with subsequent protocols in the circuit. Our protocol directly uses the SecureML truncation, in combination with the three-party semi-honest protocol from ABY3 [28], achieving security with abort.

A different approach for truncation uses function secret sharing (FSS) [19,18] in a 2-online-party computation model admitting a semi-honest adversary. This approach relies on a trusted third party (dealer) that pre-computes and distributes large function shares (keys), in order to avoid an expensive offline phase. For a single truncation, $\mathcal{O}(\ell)$ AES calls are required in the online phase, while the communication cost is similar to the RSS-based schemes, with $\ell$ bits communicated in one round. Here $\ell$ denotes the size of the ring $\mathbb{Z}_{2^\ell}$, and $k$ denotes the fixed-point precision.

Hence, to the best of our knowledge, the current state-of-the-art in truncation protocols for three-party RSS with malicious security are protocols based on the proposal of ABY3. While many follow-up works adapt the ABY3 truncation and its optimisations for pre-processing, the online phase remains identical. Nevertheless, creating the required pre-processed truncation triples remains a major bottleneck. The line of works of [11,31,23] try to minimise the communication cost in the offline phase by incorporating protocols with lower offline cost but increasing the communication cost in the online phase. We give an overview of the communication cost, round complexity, and security model of the most relevant works for truncation and our protocol in Table 1.

## 3 Preliminaries

In this section, we explain our security model, then introduce fixed-point arithmetic and the notation used throughout the paper. Later, we describe three-party replicated secret sharing for which our truncation is designed, and revisit the two main building blocks of our protocol, the two-party probabilistic truncation of SecureML [29], and the three-party truncation of ABY3 [28].

### 3.1 Security Setting

We consider a setting consisting of three servers, $P = \{P_1, P_2, P_3\}$, with pairwise secure, authenticated channels. We work in the honest majority setting, i.e., tolerating up to one corrupted party. In this paper, we consider security with abort against one malicious static adversary and prove the security of our protocol in the UC framework [7].

Table 1: Semi-honest and maliciously secure truncation protocols in the two and three-party setting over a ring $\mathbb{Z}_{2^\ell}$. The communication is in bits per party, $\kappa$ is a security parameter and $k$ denotes the fixed-point precision.

| | Parties | Comm. | Rounds | Result | Security |
|---|---|---|---|---|---|
| SecureML [29] | 2 | - | - | Probabilistic | Semi-honest |
| Cryptflow2 [32] | 2 | $\kappa\ell + \kappa k + \kappa + \ell$ | $\lceil log(\ell) \rceil + 1$ | Faithful | Semi-honest |
| Cheetah [20] | 2 | $13\ell$ | $\lceil log(\ell) \rceil$ | Faithful | Semi-honest |
| Zou et al. [38] | 2 | $\kappa\ell + \kappa + \ell + m^*$ | $\lceil log(\ell) \rceil + 1$ | Faithful | Semi-honest |
| Llama [19] | 2 | $\ell + k(\kappa + \ell + 2) + \kappa + \ell$ | 2 | Faithful | Semi-honest |
| Sigma [18] | 2 | $\mathcal{O}(\kappa\ell)$ | 3 | Faithful | Semi-honest |
| Baccarini et al. [3] | $n$ | $2\ell^2 - 3\ell - 2$ | 2 | Probabilistic | Semi-honest |
| Dalskov et al. [13] | 2 | $\ell$ | 1 | Probabilistic | Semi-honest |
| | 3 | $2\ell$ | 2 | Probabilistic | Semi-honest |
| edaBits [15] | $n$ | $\mathcal{O}(\ell \cdot log(\ell))$ | $\mathcal{O}(log(\ell))$ | Probabilistic | Malicious |
| SWIFT [23] | 3 | $13\ell$ | 2 | Probabilistic | Malicious |
| ABY3 [28] | 3 | $\ell$ | 1 | Probabilistic | Semi-honest |
| | $n$ | $20(2\ell - k) + 2\ell$ | $\mathcal{O}(\ell)$ | Probabilistic | Malicious |
| **MaSTer** | 3 | $\ell$ | 1 | Probabilistic | Malicious |

$^* m \in [0, 2\kappa\lceil log\lceil \frac{k-1}{4} \rceil \rceil)$

### 3.2 Fixed-Point Arithmetic

Many of the currently most efficient MPC frameworks for privacy-preserving machine learning are based on integer arithmetic in a ring $\mathbb{Z}_{2^\ell}$, e.g., [35,23,36], or a field $\mathbb{F}_p$ [2]. For efficiency, fixed-point arithmetic is adopted to run ML models. A real number $\tilde{x} \in \mathbb{R}$ is transformed to a fixed-point approximation $x$ by setting $x = \lfloor \tilde{x} \cdot 2^k \rfloor \in \mathbb{Z}$, where $k$ denotes the fixed precision and $\lfloor \cdot \rfloor$ denotes the floor function. After multiplication, the precision is doubled, i.e., let $x, y \in \mathbb{Z}$ be fixed-point encodings with precision $k$, then $w = x \cdot y$ has a precision of $2k$, and hence we need to truncate to preserve the fixed-point arithmetic. Truncation denotes a division by $2^k$, or equivalently a logical right shift by $k$ bits.

### 3.3 Notation

Given a fixed-point encoded secret $x \in (-2^{\ell_x}, 2^{\ell_x})$ with precision $k$, where $\ell_x$ denotes the length of the input $x$ and $k < \ell_x < \ell - 1$, we define $z$ to be the encoding in the ring $\mathbb{Z}_{2^\ell}$.

**Definition 1 (Encoding of $x$).** *Let $x$ be an integer, the encoding $z$ of $x$ in $\mathbb{Z}_{2^\ell}$ is $z = x$, if $x \geq 0$, and $z = 2^\ell - |x|$, if $x < 0$.*

We use the notation $[\![z]\!]$ to denote a 2-out-of-3 replicated secret sharing (RSS), where $[\![z]\!] = (s_1, s_2, s_3)$, such that $s_1 + s_2 + s_3 = z$. We use $[\![z]\!]_i = (s_i, s_{i+1})$ for

$1 \leq i \leq 3$, where $P_0 = P_3$, $P_1 = P_4$ etc., to denote the shares of individual parties. Further, we use $\langle z \rangle$ to denote a 2-out-of-2 additive sharing of $z$, i.e., $\langle z \rangle_1 + \langle z \rangle_2 = z$ and $\langle z \rangle_i$ denotes the share of party $i$. Similarly, we use $[z]$ to denote a 3-out-of-3 additive sharing of $z$. Moreover, we use the notation $[z]^B$ to denote a binary sharing of $z$, i.e., $z = \bigoplus_{i=1}^{3} [z]_i^B$. Further, $\mathsf{rshift}(z, k)$ denotes logical right shift of $z$ by $k$ bits, and $\lfloor z \rfloor$ denotes probabilistic truncation of SecureML [29] (see Sect. 3.5). We use the notation $\leftarrow\!\!\$\ A$ to denote uniformly random sampling from a finite set $A$. More specifically, $r \leftarrow\!\!\$_{i,j}\ \mathbb{Z}_{2^\ell}$ denotes a uniformly random sampling of $r \in \mathbb{Z}_{2^\ell}$ from shared randomness between parties $i$ and $j$ (see Appendix A). In the context of pseudo-random functions (PRFs), we use $\kappa$ to denote the security parameter. We write $\leftarrow$ to denote assignment from probabilistic algorithms and $:=$ for deterministic assignment. Finally, we use the notation $\mathbb{E}[X]$ to denote the expected value of a random variable $X$.

### 3.4 Three-Party Replicated Secret Sharing

In this work, we use a 3-party replicated secret sharing scheme, meaning an additive secret sharing, so any two parties can reconstruct the secret as explained in Sect. 3.3. Below, we briefly describe arithmetic operations using RSS. For more details, we refer the reader to Araki et al. [1] for the semi-honest setting and to Furukawa et al. [16] for the malicious setting.

Let $a, b \in \mathbb{Z}_{2^\ell}$ be public constants. Let $[\![x]\!] = (x_1, x_2, x_3)$ and $[\![y]\!] = (y_1, y_2, y_3)$.

**Linear operations.** $a \cdot [\![x]\!] + b = (ax_1 + b, ax_2, ax_3)$.
**Addition of two secret values.** $[\![x + y]\!] = (x_1 + y_1, x_2 + y_2, x_3 + y_3)$.
**Multiplication of two secret values.** $[\![z]\!] = [\![x]\!] \cdot [\![y]\!]$ is obtained by

$$s_i := x_i y_i + x_i y_{i+1} + x_{i+1} y_i + \alpha_i, \qquad \text{for } i \in \{1, 2, 3\},$$

where $\alpha_1 + \alpha_2 + \alpha_3 = 0$, i.e., $\alpha_i$ is a random share of zero that can be obtained locally by a call to a pseudo-random function (PRF) with pre-shared keys (see Fig. 8). Note that party $i$ can obtain $s_i$ locally. To transfer the 3-out-of-3 sharing of $xy$ to RSS, party $i$ sends $s_i$ to $P_{i-1}$.

In the presence of a malicious adversary, the multiplication result is checked for correctness in a post-processing phase as described by Furukawa et al. [16]. The protocol for generating randomness $\alpha_i$ is depicted in Fig. 8 in Appendix A and is based on [16, Protocol 2.5] elevated to a ring setting as in [28].

### 3.5 Two-Party Probabilistic Truncation

The main building block of our protocol relies on the two-party semi-honest probabilistic truncation proposed by SecureML [29]. The authors observe that the following local operations on a two-party additive sharing of $z$ amount to a truncation by $k$ bits with an error limited to the least significant bit (LSB) with

a large probability. If party $P_1$ computes $\langle \lfloor z \rfloor \rangle_1 := \mathsf{rshift}(\langle z \rangle_1, k)$ and party $P_2$ computes $\langle \lfloor z \rfloor \rangle_2 := 2^\ell - \mathsf{rshift}(2^\ell - \langle z \rangle_2, k)$ locally, then the result is a sharing of $\lfloor x \rfloor = \mathsf{trc}(x) \pm 1$ with probability $1 - 2^{\ell_x + 1 - \ell}$ (see [29, Theorem 1]), where $\mathsf{trc}(x)$ is defined in Def. 2. As previously mentioned, we will refer to the error of $\pm 1$ as a small error. However, with probability of $2^{\ell_x+1-\ell}$ an error of magnitude $2^{\ell-k}$ can occur. Here $k$ denotes the fixed-point precision. We will refer to this error as large/big. We provide a thorough analysis of the errors in Theorem 1, expanding on the results presented in SecureML [29]. We provide a proof of the theorem in Appendix B.

**Definition 2 (Truncation).** *Let $x \in \mathbb{Z}$ be a fixed point encoding of $\tilde{x} \in \mathbb{R}$, with fixed-point precision of $k$ bits, as described in Sect. 3.2. Then, we can write $x = x_1 2^k + x_2$, with $x_2 \in [0, 2^k)$. We define the truncation of $x$ as $\mathsf{trc}(x) = x_1$.*

**Theorem 1 (SecureML truncation error).** *Let $R \in [0, 2^\ell)$, and let $z$ be the encoding of $x$ (cf. Def. 1). We decompose $x = x_1 2^k + x_2$ with $0 \leq x_1 < 2^{\ell-k}$, $0 \leq x_2 < 2^k$, and $R = R_1 2^k + R_2$ with $0 \leq R_1 < 2^{\ell-k}$, $0 \leq R_2 < 2^k$. Then, $\mathit{rshift}(z + R, k) + 2^\ell - \mathit{rshift}(R, k) = \mathsf{trc}(x) + c_1 + c_2 2^{\ell-k}$, where*

$$c_1 = \begin{cases} 1 & \text{if } x \geq 0 \wedge x_2 + R_2 \geq 2^k \\ -1 & \text{if } x < 0 \wedge |x_2| > R_2 \\ 0 & \text{else} \end{cases}, c_2 = \begin{cases} 1 & \text{if } x < 0 \wedge x_1 + R_1 \geq 2^{\ell-k} \\ -1 & \text{if } x \geq 0 \wedge |x_1| > R_1 \\ 0 & \text{else} \end{cases}.$$

*Furthermore, note that when $R \in [2^{\ell_x}, 2^\ell - 2^{\ell_x})$, $c_2 = 0$ for all such $R$. The probability of a uniformly random $R$ being in this range is $1 - 2^{\ell_x+1-\ell}$. Thus, with probability $1 - 2^{\ell_x+1-\ell}$, $\mathit{rshift}(z + R, k) + 2^\ell - \mathit{rshift}(R, k) = \mathsf{trc}(x) + c_1$.*

The truncation error $c_1$ in Theorem 1 is therefore formally given by the following function $e_k(z, k)$ in Def. 3.

**Definition 3 (SecureML small truncation error).** *Let $z$ be the encoding of $x$ (see Def. 1) and $0 \leq R < 2^\ell$. The truncation error $e_k(z, k)$ when truncating $z$ by $k$ bits having randomness $R$ in the shares (i.e. $\langle z \rangle = (z + R, -R)$), is defined as*

$$e_k(z, R) = \begin{cases} 1 & \text{if } z < 2^{\ell-1} \wedge (z \mod 2^k) + (R \mod 2^k) \geq 2^k \\ -1 & \text{if } z > 2^{\ell-1} \wedge (R \mod 2^k) < (2^\ell - z \mod 2^k) \\ 0 & \text{else} \end{cases}.$$

### 3.6 Three-Party Probabilistic Truncation

ABY3 [28] introduces two truncation methods for three-party RSS schemes. The authors note that the two-party local truncation of SecureML [29] fails when naively extended to three parties. Nevertheless, assuming semi-honest security and at most one corrupted party, two parties can transform a RSS sharing $[\![z]\!]$ to a 2-out-of-2 sharing $\langle z \rangle$, perform the SecureML truncation with local operations and re-share the result to obtain shares of $[\![\lfloor z \rfloor]\!]$. This extended truncation $\Pi_{\mathsf{trunc1}}$

is formally described in [28, Fig. 2] and is another building block of our protocol. We use this protocol combined with the SecureML truncation to achieve a secure truncation protocol against a malicious adversary, as described in Sect. 4.

The approach described above on its own is not secure in the presence of a malicious adversary. ABY3, therefore, proposes another technique to achieve a maliciously secure truncation based on the line of works started by [9]. The protocol requires a pre-processed pair of shares of a random $r \in \mathbb{Z}_{2^\ell}$, $[\![r]\!]$ and $[\![\lfloor r \rfloor]\!]$, from which the parties can compute $[\![\lfloor z \rfloor]\!] = [\![\lfloor r \rfloor]\!] + \lfloor (z-r) \rfloor$. The online phase consists of a single round where $\ell$ bits are sent to reveal $z-r$. In the offline phase, the parties generate random Boolean shares $[r]^B \in \mathbb{Z}_2^\ell$ in replicated secret sharing form and locally compute the $k$-bit truncation $[r']^B \in \mathbb{Z}_2^\ell$ on the Boolean shares. To convert the shares into arithmetic shares in $\mathbb{Z}_{2^\ell}$, ABY3 proposes that the parties compute two subtraction circuits, one to obtain shares of $r$ and one for $r'$, to emulate $\mathbb{Z}_{2^\ell}$ arithmetic on the Boolean shares using a protocol of Furukawa et al. [16]. Note that this entails evaluating two subtraction circuits per truncation triple. The line of works started by Trident [11] optimises this approach by applying conversion of a binary to arithmetic sharing and then trivially obtaining the difference. The latest work SWIFT [23] following this approach achieves the cost of twelve ring elements sent over one round in the offline phase with the same online cost as ABY3.

## 4 MaSTer

When we investigate the suitability of the semi-honest ABY3 truncation ($\Pi_{\mathsf{trunc1}}$, see Sect. 3.6) in the malicious setting, the only leverage the adversary has is corrupting $P_1$ and modifying the result share of $\lfloor z \rfloor$ when re-sharing. All other operations are local. Consequently, in order to employ $\Pi_{\mathsf{trunc1}}$, we only need to check if the re-sharing was done correctly. Furthermore, parties $P_2$ and $P_3$ have sufficient information about $z$ due to the RSS. In an abstract sense, we run $\Pi_{\mathsf{trunc1}}$ among all three parties and the two-party SecureML truncation protocol among $P_2$ and $P_3$ in parallel. We, therefore, obtain *unchecked* RSS shares $[\![\lfloor z \rfloor]\!]$ of the result for all parties. We then use the shares $\langle \lfloor z \rfloor \rangle$ of $P_2$ and $P_3$ to verify the correctness of the RSS shares that were dealt by $P_1$. This is illustrated in Fig. 1. A corrupt $P_1$ cannot modify $[\![\lfloor z \rfloor]\!]$ without being detected by $P_2$ and $P_3$ while corrupt $P_2/P_3$ cannot influence the creation of $[\![\lfloor z \rfloor]\!]$. Note that an adversary controlling $P_2/P_3$ can force an abort by wrongfully claiming that $P_1$ modified the result shares.

We can check the consistency of the RSS shares by checking the difference of $[\![\lfloor z \rfloor]\!]$ and $\langle \lfloor z \rfloor \rangle$. However, due to the probabilistic error in the LSB of the truncation protocols, the difference may not be zero even with correctly created shares. We analyse the error and show that for an honest execution, the difference is $\pm 1$ with high probability (see Sect. 4.3).

Note that the correctness of the truncation (disregarding malicious influence) relies on the correctness of $\Pi_{\mathsf{trunc1}}$, which is an extension of SecureML, and thus requires the input $z$ to be a valid encoding of $x \in (-2^{\ell_x}, 2^{\ell_x})$ (see Def. 1) to

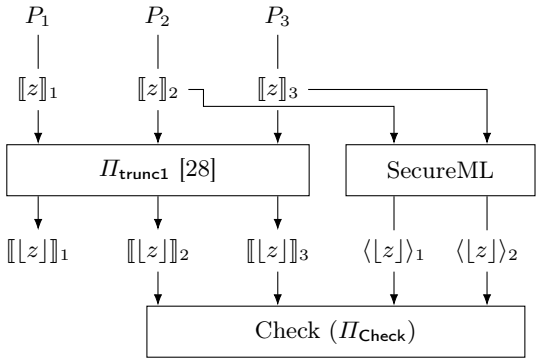


Fig. 1: MaSTer truncation protocol visualised.

produce a correct truncation $\lfloor z \rfloor$. Therefore, MaSTer cannot readily be used to generate pre-processed random truncations $(r, \lfloor r \rfloor)$ where $r \leftarrow\!\!\$\ \mathbb{Z}_{2^\ell}$ for the use in other, maliciously secure truncation protocols such as [9,28,11,23]. As in $\Pi_{\mathsf{trunc1}}$, we require $\ell_x < \ell - 1$.

### 4.1 Protocol

We employ a similar approach as Furukawa et al. [16] does for multiplication, with an optimistic execution of a semi-honest protocol in the online phase and a check in the post-processing phase (cf. Sect. 3.4). The protocol (see Fig. 2) hence consists of two phases. The first phase is analogous to the semi-honest truncation with re-sharing afterwards and results in *unchecked* shares of the truncated multiplication result in RSS. A cheating $P_1$ must send $s'_2 + \Delta$ in the first round, where $\Delta$ is an additive error. In order to detect this potential malicious behaviour in the first phase, we introduce a second phase where parties $P_2$ and $P_3$ compute the two-party truncation where $P_1$ does not contribute. Therefore, $P_2$ and $P_3$ now hold two independently created sharings of the truncated value, allowing them to compute the difference and hence verifying the correctness of the value sent by $P_1$, up to an error of $\pm 1$. Concretely, $P_2$ and $P_3$ compute $\gamma_i := s'_i - s''_i$, $i \in \{1, 2\}$ and $\gamma_3 := s'_3$, where $s'_j$, $j \in \{1, 2, 3\}$ are components of the RSS share dealt by $P_1$ and $s''_i$ is the sharing of the two-party truncation. $P_2$ stores $\gamma_{2,m}$ and $\gamma_{3,m}$ of the $m$-th truncation and $P_3$ stores $\gamma_{1,m}$ and $\gamma_{3,m}$. After $M$ truncations (or at the output phase), $P_2$ and $P_3$ use one communication round to send $\gamma_{2,1}, \ldots, \gamma_{2,M}$ and $\gamma_{1,1}, \ldots, \gamma_{1,M}$, respectively, and then check $\forall 1 \leq j \leq M \,|\, \sum_{i=1}^{3} \gamma_{i,j} | \leq 1$. The second phase thus consists of a batched check of the optimistically executed truncations. The batched consistency check (Fig. 3 requires one round of communication among $P_2$ and $P_3$ to check $M$ truncations in parallel. Our protocol therefore achieves an amortised cost of one round of communication. We outline optimisations of the consistency check in Sect. 4.2.

### 4.2 Further communication cost reduction

We now aim to reduce the communication cost in the parallel consistency check. By computing $a_1 := \gamma_1$ ($P_3$) and $a_2 := \gamma_2 + \gamma_3$ ($P_2$), the two parties now hold a two-party additive sharing and wish to check whether $a_1 + a_2 \in \{-1, 0, 1\}$. First note that if the check were $a_1 + a_2 = 0$, $P_2$ could hash all of its shares $H(a_2^{(1)} || \dots || a_2^{(m)})$ and $P_3$ hashes the negation of its shares $H(2^\ell - a_1^{(1)} || \dots || 2^\ell - a_1^{(m)})$. The parties would only need to share the hash and compare. However, this is no longer feasible since, for example, $P_3$ needs to check $1 + 2^\ell - a_1^{(i)}$, $2^\ell - a_1^{(i)}$ or $2^\ell - a_1^{(i)} - 1$ without knowing which of the three options yields the same hash. Instead, each party adds their share of $a$ to a Bloom filter [5,34] and sends the Bloom filter to the other party. Then, each party performs set membership tests of $1 + 2^\ell - a_j^{(i)}$, $2^\ell - a_1^{(i)}$ and $2^\ell - a_1^{(i)} - 1$, and aborts the consistency check if none of the three elements are in the filter. Since Bloom filters are probabilistic, there is a probability of $\epsilon$ of a false positive, i.e., in our context, a value $a^{(j)}$ does not reconstruct to $\{-1, 0, 1\}$. Dimensioned such that $\epsilon \leq \min(2^{-\sigma}, 2^{\ell_x + 2 - \ell})$ where $\sigma$ is the statistical security parameter and $2^{\ell_x + 2 - \ell}$ is the failure probability of the consistency check due to large errors happening (see Theorem 2), the size of the Bloom filter is about $c \log_2(1/\epsilon) = c \cdot \min(40, \ell - \ell_x - 2)$ bits per element where $1 \leq c \leq 1.44$ depends on the exact type of filter. For our parameters where $\ell = 96, \ell_x = 16$, employing Bloom filters saves roughly 58% communication in that phase. Further, one may choose a larger filter to reduce $\epsilon$ and compress the filter during transit [27]. However, filling the filter and querying for membership requires additional local computation.

**Truncation protocol $\Pi_{\mathsf{Trunc}}$**

| $P_1$<br>$[\![z]\!]_1 = (s_1, s_2)$ | $P_2$<br>$[\![z]\!]_2 = (s_2, s_3)$ | $P_3$<br>$[\![z]\!]_3 = (s_3, s_1)$ |
|---|---|---|
| $s_1' := r' \leftarrow\!\$_{1,3}\ \mathbb{Z}_{2^\ell}$ | | $s_1' := r' \leftarrow\!\$_{1,3}\ \mathbb{Z}_{2^\ell}$ |
| $s_2' := \mathsf{rshift}(s_1 + s_2, k) - r' \xrightarrow{s_2'}$ | $s_3' := 2^\ell - \mathsf{rshift}(2^\ell - s_3, k)$<br>$s_2'' := 2^\ell - \mathsf{rshift}(2^\ell - s_2, k)$<br>$\gamma_2 := s_2' - s_2''$<br>$\gamma_3 := s_3'$ | $s_3' := 2^\ell - \mathsf{rshift}(2^\ell - s_3, k)$<br>$s_1'' := \mathsf{rshift}(s_3 + s_1, k)$<br>$\gamma_1 := s_1' - s_1''$<br>$\gamma_3 := s_3'$ |
| Output $[\![\lfloor z \rfloor]\!]_1 := (s_1', s_2')$ | Output $[\![\lfloor z \rfloor]\!]_2 := (s_2', s_3')$<br>Store $\gamma_2, \gamma_3$ | Output $[\![\lfloor z \rfloor]\!]_3 := (s_3', s_1')$<br>Store $\gamma_1, \gamma_3$ |
| *Post-processing* | $\vdots$<br>$\Pi_{\mathsf{Check}}$ | $\vdots$<br>$\Pi_{\mathsf{Check}}$ |

Fig. 2: The truncation protocol $\Pi_{\mathsf{Trunc}}$ to truncate $z$ by $k$ bits.

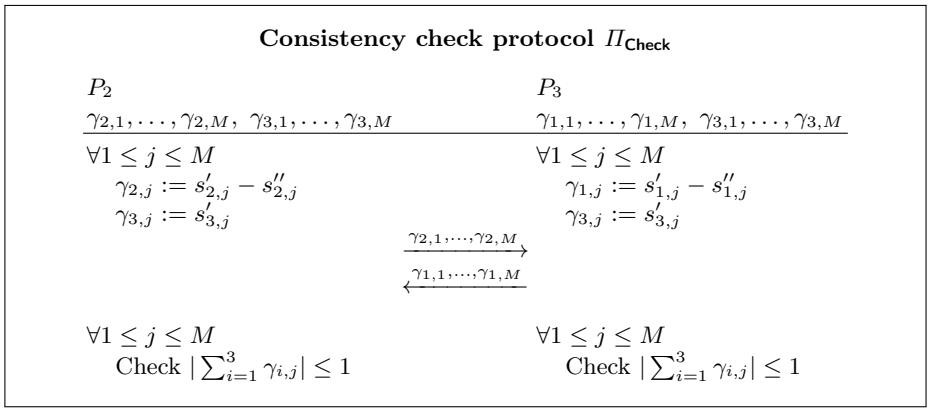


Fig. 3: The consistency check protocol to check a batch of $M$ unchecked truncations.

### 4.3 Error Analysis

In this section, we detail the truncation errors and their role in the consistency check (see Theorem 2). Later, we expand on the probability of detected malicious behaviour and provide more details on the relation between the protocol parameters and the multiplicative depth of the computed circuit.

In the following theorem, we show the correctness of the consistency check. We provide a proof of the theorem in Appendix C.

**Theorem 2 (Correctness of consistency check).** *Let $x_1$ be the result of a truncated value $x \in X$. Let $\mathbb{E}[x_1]$ denote the expected value of $x_1$. In an honest execution of $\Pi_{\mathsf{Trunc}}$ (see Fig. 2), the consistency check holds with probability* $\Pr\left[\sum_{i=1}^{3} \gamma_i \in \{0, \pm 1\}\right] \geq 1 - 2^{k+2-\ell} \cdot \mathbb{E}[x_1] \geq 1 - 2^{\ell_x+2-\ell}$.

A variety of works focusing on precise or faithful truncation [38,20,32] motivate their work by arguing that the probability of a large error occurring in a large neural network is rather high. In our protocol this large error would lead to an abort even if no malicious error was introduced. We therefore stress the necessity of a mindful selection of the input size $\ell_x$ in relation to the ring size $\ell$, based on the expected distribution of values to be truncated $\mathbb{E}[x]$. We note that for large networks with a number of multiplications exceeding the order of $10^{12}$, the usual ring size of $\ell = 64$ bits would not be sufficient, assuming a uniformly distributed $x$. However, in ML applications $x$ is expected to be rather small, resulting in $\mathbb{E}[x_1] \ll 2^{\ell_x - k}$. Therefore a concrete model and ML network analysis is required for a secure choice of parameters. To achieve statistical security with parameter $\sigma$, the parameters would be required to satisfy $n \cdot 2^{k+2-\ell} \cdot \mathbb{E}[x_1] \leq 2^{-\sigma}$, where $n$ denotes the total number of multiplications in the computed circuit and $\mathbb{E}[x_1] \leq 2^{\ell_x - k}$. Recall $x_1$ here denotes the result of a truncation.

We note that authors of Bicoptor 2.0 [37] made an observation about the large truncation error. They noted the large error is always $\pm 2^{\ell-k}$ and they proposed mitigation of such error by interpreting the output shares modulo $2^{\ell-k}$. However, this approach is not directly applicable to our protocol. If it was to be applied to the consistency check, such operation would allow an adversary to add an error $\Delta > 2$ when $\Delta \bmod 2^{\ell-k} = 0$ and the check would not be able to detect this. Alternatively, the truncation result could be interpreted as sharing in $\mathbb{Z}_{2^{\ell-k}}$. This in turn would require a modulo switch protocol which would introduce further overhead. We leave the exploration of different techniques of mitigation as a future work.

## 5 Security

We now turn to proving security of our proposed probabilistic truncation protocol. Importantly, and as many other PPML protocols, we require secure, i.e., confidential and mutually authenticated, channels between the parties. We assume these are set up in advance before the protocol runs. This can be achieved with, e.g., client- and server-side authentication in TLS.

### 5.1 Faithful and Probabilistic Truncation

At USENIX Security '23, Li et al. [26] advocated for precise truncation protocols as opposed to the line of work on probabilistic truncation protocols that introduce errors on the result. They showed that truncation with a truncation tuple $[\![r]\!], [\![\lfloor r \rfloor]\!]$ with $r \leftarrow\$ \mathbb{Z}_{2^\ell}$ in the style of [9,28] does not securely realise a truncation functionality that adds the probabilistic truncation error in the ideal world solely based on the plaintext value to be truncated. In the real world, the error distribution also depends on $r$. In fact, the authors show that in the real protocol the adversary learns some function of $r$ and can thus distinguish the worlds based on the reconstructed result of the truncation. This result highlights the importance of a properly defined functionality and rigorous proofs even for small sub-protocols in a PPML framework. Therefore, we give a formal definition of our desired functionality $\mathcal{F}_{\mathsf{Trunc}}$ (see Fig. 4) and provide a detailed proof. To prevent these issues, our functionality has to extract the randomness in the share and therefore takes all randomness into account when sampling the truncation error. Unfortunately, this results in a less simple description of the ideal behaviour of truncation.

### 5.2 Soundness

First, we investigate the soundness of the consistency check, i.e., the probability that it detects cheating when cheating occurred. For this setting, we assume $P_1$ is corrupted by the adversary and sends $s'_{2,\mathcal{A}} = s'_2 + \Delta$ for $\Delta \neq 0$. Modifying the expressions of Theorem 1, we get $\sum_{i=1}^{3} \gamma_i + \Delta = c'_1 - c''_1 + \Delta$ with probability $\geq 1 - 2^{k+2-\ell} \cdot \mathbb{E}[x_1] \geq 1 - 2^{\ell_x+2-\ell}$, since $\mathbb{E}[x_1] \leq 2^{\ell_x-k}$. The adversary can therefore

**Functionality $\mathcal{F}_{\mathsf{Trunc}}$**

The functionality receives $[\![z]\!]_j, k$ from all parties $P_j$ and receives the adversaries output share $(s'_i, s'_{i+1})$. If the adversary corrupts $P_1$, it also receives an additive error $\Delta$.

- $\mathcal{F}_{\mathsf{Trunc}}$ unpacks the RSS sharing as $(s_1, s_2, s_3)$.
- $\mathcal{F}_{\mathsf{Trunc}}$ reconstructs $z := s_1 + s_2 + s_3$ and sets $r_1 := 2^\ell - s_3$ and $r_2 := 2^\ell - s_2$.
- $\mathcal{F}_{\mathsf{Trunc}}$ computes $b := e_k(z, r_1)$ and $b' := e_k(z, r_1) - e_k(z, r_2)$ (see Def. 3).
- $\mathcal{F}_{\mathsf{Trunc}}$ truncates $z$ by $k$ bits, $\lfloor z \rfloor_k$ and sets $s'_{i-1} := \lfloor z \rfloor_k + \Delta + b - s'_i - s'_{i+1}$.
- If $|b' + \Delta| \leq 1$, the functionality outputs $(s'_j, s'_{j+1})$ to party $P_j$, else the functionality sends abort to the adversary, waits for continue and then sends abort to all honest parties.

Fig. 4: MaSTer truncation functionality $\mathcal{F}_{\mathsf{Trunc}}$.

set the truncation error to $c' + \Delta$ under the constraint that $|c' - c'' + \Delta| \leq 1$. Since $|c' - c''| \leq 1$ with probability $\geq 1 - 2^{\ell_x + 2 - \ell}$, any additive error $|c' + \Delta| > 2$ added by a malicious $P_1$ will be detected with the same probability $\geq 1 - 2^{\ell_x + 2 - \ell}$. We note that an adversary can therefore add an additive error $|\Delta| \leq 2$ without getting detected. This adversarial influence opens up a new attack surface for the adversary. We expand on the attacker possibilities in Sect. 5.4. Further, we analyse the error probability of $\Delta = \pm 2$ in Appendix D.

### 5.3 Security of MaSTer

We prove security of the truncation protocol in the UC framework [7]. For this, we define an ideal functionality $\mathcal{F}_{\mathsf{Trunc}}$ (see Fig. 4) and a simulator $\mathcal{S}$. We give further details about the UC security definition including static security with abort in Appendix E. A protocol $\Pi$ securely realises the ideal functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$, there is a simulator $\mathcal{S}$ such that all environments $\mathcal{Z}$ cannot distinguish between the ideal world and the real world. We use a pairwise shared PRF between the three parties, formalised as $\mathcal{F}_{\mathsf{cr}}$ in Fig. 6, and give a protocol $\Pi_{\mathsf{cr}}$ implementing the functionality for completeness in Appendix A. We use the simplified UC framework [8] that takes care of authenticated channels, message scheduling, etc. The main security theorem for the MaSTer protocol is as follows.

**Theorem 3.** *The protocol $\Pi_{\mathsf{Trunc}}$ (see Fig. 2) securely realises $\mathcal{F}_{\mathsf{Trunc}}$ (see Fig. 4) in the $\mathcal{F}_{\mathsf{cr}}$-hybrid model against a static malicious adversary with abort who corrupts a single party.*

*Proof.* We setup a simulator $\mathcal{S}$ that internally runs $\mathcal{A}$ and passes along all communication between $\mathcal{A}$ and $\mathcal{Z}$. Therefore, we can assume $\mathcal{A}$ is a dummy adversary

that acts and reports back as told by $\mathcal{Z}$. Thus, in the following, $\mathcal{Z}$ and $\mathcal{A}$ are used interchangeably. The goal of the proof is to show that the simulated view of $\mathcal{A}$ (that is run internally by $\mathcal{S}$) of the protocol in the ideal world is indistinguishable for $\mathcal{Z}$ from the view of $\mathcal{A}$ in the real world. Note that in the $\mathcal{F}$-hybrid model, both in the real and in the ideal world, the protocol parties interact with the $\mathcal{F}$ hybrids when needed. In the real world, these functionalities are proper ideal functionalities. In the ideal model, the simulator also plays the role of the hybrid functionalities towards the adversary.

In the following, we describe a simulator $\mathcal{S}$ that simulates the honest parties towards the adversary $\mathcal{A}$ in an ideal execution with access to $\mathcal{F}_{\mathsf{Trunc}}$. Notably, $\mathcal{S}$ does not know the inputs of the honest parties it simulates. However, $\mathcal{S}$ knows the inputs of the corrupted parties, i.e., in this case the input share of the value to be truncated. This allows $\mathcal{S}$ to replicate and track all computation results of $\mathcal{A}$ if it follows the protocol. Let $i$ be the index of the corrupted party, i.e., $\mathcal{A}$ controls $P_i$. The index $j$ denotes the honest parties.

In the first step, the simulator $\mathcal{S}$ obtains access to the PRF $F_k$ since it simulates $\mathcal{F}_{\mathsf{cr}}$ towards $\mathcal{A}$, and knows $\mathcal{A}$'s share $(s_i, s_{i+1})$.

**If** $P_i = P_1$**,** $\mathcal{S}$ receives $s'_{2,\mathcal{A}}$ from $\mathcal{A}$. Since the simulator knows $s_1, s_2$ and $r'$ (from $F_k$), it computes $s'_{2,\mathcal{S}} := \mathsf{rshift}(s_1 + s_2, k) - r'$, and sets $\Delta := s'_{2,\mathcal{A}} - s'_{2,\mathcal{S}}$. The simulator calls $\mathcal{F}_{\mathsf{Trunc}}(P_i, (s_1, s_2), \Delta)$ and sends $(r', s'_{2,\mathcal{A}})$ as the output share. If the functionality sends abort, the simulator sends abort to the adversary and sends continue to the functionality, else it outputs what $\mathcal{A}$ outputs.

**If** $P_i = P_2$**,** the simulator samples $r \leftarrow\!\!\$\ \mathbb{Z}_{2^\ell}$ uniformly at random, and sends $r$ as $s'_2$ to the adversary on behalf of $P_1$. The simulator computes $s'_3 := 2^\ell - \mathsf{rshift}(2^\ell - s_3, k)$. The simulator calls $\mathcal{F}_{\mathsf{Trunc}}(P_i, (s_2, s_3), *)$ and sends $(r, s'_3)$ as the adversary's output share. If the functionality sends abort, the simulator sends abort to the adversary, waits for continue and then sends continue to the functionality. Else, $\mathcal{S}$ obtains $b'$ from $\mathcal{F}_{\mathsf{Trunc}}$ and computes $\gamma_3$, $s''_2$ and sends $\gamma_1 := 2^\ell - \gamma_3 + s''_2 - r + b'$ to the adversary on behalf of $P_3$. The simulator receives $\gamma'_2$ from the adversary. Note that if the adversary computes the $\gamma$-check, it obtains $\sum \gamma_i = \left(2^\ell - \gamma_3 + s''_2 - r + b'\right) + (r - s''_2) + \gamma_3 = b'$. The simulator checks $|b' + \gamma'_2 - r + s''_2| \leq 1$. If this holds, the simulator continues and outputs what $\mathcal{A}$ outputs. If not, the simulator sends abort to the adversary and then aborts.

**If** $P_i = P_3$**,** the simulator computes $s'_3, s'_1, \gamma_1$ and $\gamma_3$. The simulator calls $\mathcal{F}_{\mathsf{Trunc}}(P_i, (s_3, s_1), *)$ and sends $(s'_3, s'_1)$ as output shares. If the functionality sends abort, the simulator sends abort to the adversary, waits for continue and then sends continue to the functionality. Else, $\mathcal{S}$ obtains $b'$ and sends $2^\ell - \gamma_1 - \gamma_3 + b'$ as $\gamma_2$ to the adversary on behalf of $P_2$. The simulator receives $\gamma'_1$ from the adversary and checks if $|\gamma'_1 - \gamma_1 + b'| \leq 1$. If this holds, the simulator continues and outputs what $\mathcal{A}$ outputs. If not, the simulator sends abort to the adversary and then aborts.

We now show that for each case, $\mathcal{S}$ creates a view for $\mathcal{A}$ that is indistinguishable to the real world execution. Note that the (reconstructed) output is

identical in the real and ideal world since $\mathcal{F}_{\mathsf{Trunc}}$ adds the same truncation error on the least significant bit. Further, the adversary's view of the input and output shares fully determines the input and output shares of the honest parties if the adversary also knows the shared value (cf. [16, Def. 2.3]). Therefore, the truncation error and abort conditions in the ideal world must exactly match the side-effects in the real protocol. Since the randomness in the input shares is known by the adversary, the functionality extracts it and bases the abort condition (for a $\Delta = 0$) and truncation error on the extracted randomness. This way, the abort behaviour due to the adversarially added $\Delta$, and the sign and value of the truncation error are identical to the real world execution.

Note that the protocol also aborts with probability $2^{\ell_x+2-\ell}$ independent of the adversarial influence. This bound stems from the correctness probability of SecureML (cf. Sect. 3.5 and [29, Theorem 1]). As done in previous works, e.g., SecureML and ABY3, the ring parameters can be adapted to be large enough so that this event becomes negligible, so we will not consider this case further on.

**If** $P_i = P_1$**,** since $\mathcal{A}$ does not receive data in this step, the only possibility to distinguish the two worlds is by sending a non-zero additive error $\Delta$ in $s_2'$, by observing the reconstructed output or the abort behaviour. Since in this case, $\mathcal{S}$ does not abort, output and abort behaviour with $\Delta = 0$ is already handled as described above. For $\Delta \neq 0$, the abort probability is the same in both worlds since $c' - c'' = b'$ by construction and thus the checks in the real world and in $\mathcal{F}_{\mathsf{Trunc}}$ evaluate to the same value.

**If** $P_i = P_2$**,** $\mathcal{A}$ receives $s_2'$ and $\gamma_1$. Further by sending different $\gamma_2$, it can influence abort behaviour. Let $\alpha_1 = \mathsf{rshift}(s_1 + s_2, k)$, $\alpha_2 = 2^\ell - \mathsf{rshift}(2^\ell - s_3, k)$, $\beta_1 = \mathsf{rshift}(s_3 + s_1, k)$ and $\beta_2 = 2^\ell - \mathsf{rshift}(2^\ell - s_2, k)$ be shorthand notation for the terms, then $\mathcal{A}$ sees $(\alpha_1 - r', r' - \beta_1)$ with $r' := F_k(cnt)$ in the real world while the ideal view is $(r, -\alpha_2 + \beta_2 - r + b')$ with $r \leftarrow\!\$\ \mathbb{Z}_{2^\ell}$. In the ideal world, we now sample $r$ as $\alpha_1 - r'$ where $r' := F_k(cnt)$. Clearly, $r$ and $\alpha_1 - r'$ are identically distributed (under the PRF assumption). Now the ideal world view is

$$\begin{aligned}&\big(\alpha_1 - r', -\alpha_2 + \beta_2 - (\alpha_1 - r') + b'\big)\\ &= \big(\alpha_1 - r', -\alpha_2 + \beta_2 - \alpha_1 + r' + (\alpha_1 + \alpha_2 - \beta_1 - \beta_2)\big) = (\alpha_1 - r', r' - \beta_1)\,,\end{aligned}$$

which is identical to the real world view. Note that $\sum \gamma_i = \alpha_1 + \alpha_2 - \beta_1 - \beta_2 + (r' - r') = b'$. $\mathcal{A}$ can influence the abort behaviour only when sending a different $\gamma_2' = \gamma_2 + \Delta$, $\Delta \neq 0$. In the real world, the protocol aborts if $|\gamma_1 + \gamma_2 + \Delta + \gamma_3| > 1$, more precisely if $|b' + \Delta| > 1$. In the ideal world, the simulator aborts if $|b' + \gamma_2' - r + s_2''| > 1$, which is $|b' + (\gamma_2 + \Delta) - r + s_2''| = |b' + (r - s_2'' + \Delta) - r + s_2''| = |b' + \Delta|$. The abort behaviour in both worlds is therefore the identical.

**If** $P_i = P_3$**,** $\mathcal{A}$ receives $\gamma_2$ and sends $\gamma_1' = \gamma_1 + \Delta$. In the real world, $\mathcal{A}$ receives $\gamma_2$ and in the ideal world it receives $2^\ell - \gamma_1 - \gamma_3 + b' = \gamma_2$ since $\sum \gamma_i = b'$. Thus both worlds yield identical distributions. In the real world, the protocol aborts if $|\gamma_1' + \gamma_2 + \gamma_3| > 1$, and in the ideal world the simulator aborts if

$|\gamma_1' - \gamma_1 + b'| > 1$, clearly $|\gamma_1' - \gamma_1 + b'| = |b' + \Delta|$ and thus exhibits the same abort behaviour.

Note that a corrupted $P_1$ can send an arbitrary value instead of $s_2'$ to $P_2$. In this case the protocol would continue with the computation using the corrupted shares. However, note that $s_3'$ remains uniformly random in the view of $P_1$, hence $P_1$ cannot infer any further information about the secret $z$. Moreover, before revealing any shares, the parties would run the check to verify correctness (cf. Fig. 3). Thus a corrupted value would be detected and honest parties abort.

### 5.4 Implications of $\Delta$ Introduced by an Adversary

From the soundness results in Sect. 5.2 and $\mathcal{F}_{\mathsf{Trunc}}$ (Fig. 4), it follows that the consistency check detects malicious behaviour up to a small error $\Delta$ that can be introduced by an adversary with high probability. It is therefore crucial to demonstrate the effect of the error $\Delta$. We perform an analysis of the error impact by running a forward pass of a dense neural network (DNN) described in Sect. 7 without the error as a baseline and with the error introduced in each truncation. Figure 5 depicts the absolute value of the difference between two cases. We find that the effect of the adversary influencing the result with $\Delta = \pm 1$ and $\Delta = \pm 2$ is in the order of $10^{-3}$ for precision of 16 bits, and $10^{-2}$ for 13-bit precision, thus has no impact for this particular network. While the impact for $\Delta = \pm 2$ is larger, the probability of detection grows exponentially (cf. Sect. 5.2). Still, the new adversarial capability requires more study in future work. The attack can be viewed as an instance of an adversarial example attack, where, in addition to the query sample, the client also sets noise to the model weights. As such, this scenario can be studied separately outside the scope of this paper. Nevertheless, we believe that a careful choice of fixed-point precision lowers the impact any attacker might have.

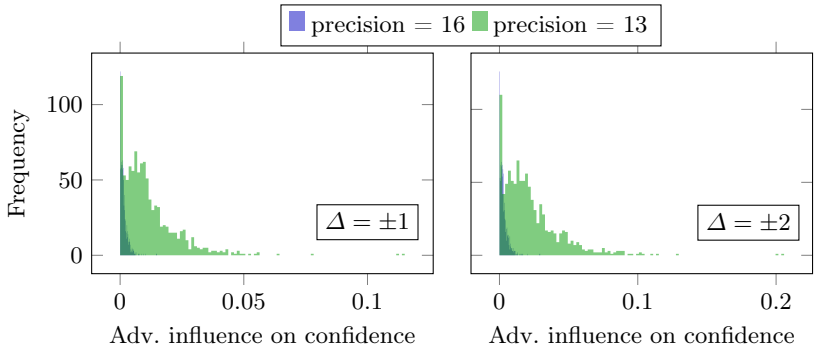


Fig. 5: Illustration of the impact on the predicted confidence levels of a 5-layer network inference when adding truncation errors of $\pm 1$ and $\pm 2$.

Table 2: Performance data of the pre-processing for ABY3 truncation tuples in the LAN and WAN setting. In our setup, the performance starts to scale linearly from $10^6$ tuples onwards.

| | Mult. | $10^3$ | $10^4$ | $10^5$ | $10^6$ | $10^7$ | $10^8$ |
|---|---|---|---|---|---|---|---|
| Time (s) | LAN | 0.03 | 0.07 | 0.24 | 2.07 | 22.92 | 237.73 |
| | WAN | 2.36 | 3.77 | 14.42 | 87.73 | 834.77 | 8309.08 |
| Data sent (MB) | | 2.66 | 11.65 | 106.51 | 1015.25 | 10092.70 | 100926.90 |

## 6 Benchmarks

We benchmark our protocol $\Pi_{\mathsf{Trunc}}$ (see Fig. 2) against the generic maliciously secure truncation protocol of ABY3 [28] in replicated secret sharing. We omit a direct comparison to optimised versions of ABY3 pre-processing due to a significant implementation effort required as the source code of neither [11,23] is available. We do however implement the online and offline phases separately, to demonstrate performance of both parts individually. The optimised truncation protocols such as the one of SWIFT [23] are based on the same online phase as ABY3, hence our online phase is directly comparable to these. ABY3 uses the same offline phase for truncation triple generation as the maliciously secure protocol of Furukawa et al. [16], which is implemented in MP-SPDZ. Our experimental setup is detailed in Sect. 6.1.

The pre-processing phase of the ABY3 truncation is run separately to accommodate the specified protocol requirements by [28]. Due to its high cost we only benchmark the creation of fewer triples. The numbers reported for the ABY3 truncation pre-processing are then scaled accordingly. Table 2 demonstrates that for the creation of one million pre-processed triples onwards, the execution time and the data sent scale linearly. To be conservative, all of the data reported in the experiments is scaled according to pre-processing for $10^8$ multiplications. This is to mitigate any inaccuracy caused by scaling from smaller values.

### 6.1 Experimental Setup

We implement our and the ABY3 truncation protocols in the MP-SPDZ [22] framework and benchmark on three servers with a 4-core 2.40GHz Intel Xeon CPU and 16GB of RAM. We simulate a LAN setting with 0.1ms latency and 10Gbps bandwidth, and a WAN setting with 40ms round trip time and 100Mbps bandwidth. We modify the implementation of the MPC protocol for three-party malicious security in the ring $\mathbb{Z}_{2^\ell}$ in MP-SPDZ by inserting both our protocol and the ABY3 truncation in the online phase. The protocol implemented in MP-SDPZ is based on the pre-processing of Cramer et al. [12] and re-sharing of Araki et al. [1]. The pre-processing of the ABY3 truncation triples is run separately in an MPC protocol for Boolean three-party RSS based on Furukawa

et al. [16] as specified in [28], however with Beaver multiplication [4] instead of their post-sacrifice method.

Similarly to previous works in PPML [29,35,28], we run experiments in three commonly benchmarked networks. For training on the MNIST dataset (one epoch of training: forward and backward pass with batch size 128), we run a three-layer dense neural network introduced by SecureML [29] with ReLU and Softmax activation functions, and also run the widely known 7-layer CNN LeNet-5 [25]. Further, we run the same training setup with a smaller batch size of 16 due to RAM constraints with the CIFAR-10 dataset on an 11-layer CNN AlexNet [24]. We omit benchmarks of training in the WAN setting due to significant time required for their completion. For inference results in the WAN setting we refer the reader to Sect. 7. We run the ABY3 benchmark in a 64-bit ring, with $k = 7$, $\ell_x = 17$. In order to satisfy parameter analysis outlined in Sect. 4.3, we benchmark MaSTer in a 96-bit ring. We set $\sigma = 40$ and we note that the number of truncations for all experiments is $n < 2^{36}$. Therefore, it is satisfied that $2^{36} \cdot 2^{17+2-96} \leq 2^{-40}$. We further note the choice of $k$ is subjected to the MP-SPDZ framework implementation of probabilistic truncations, which requires $k + 9 \leq \ell_x$ and $\ell_x + k \leq \ell - \sigma$. We stress that the choice of $k$ is for benchmarking only, and while it has no effect on the performance, as mentioned in Sect. 5.4 the choice of $k$ possibly determines an impact of a malicious attacker.

### 6.2 Results

We showcase the performance results in Table 3. We report the results of the offline and online phases separately. The pre-processing of ABY3 truncation triples is run separately using the protocol of Furukawa et al. [16] and the performance numbers are then added to the offline phase of ABY3 benchmark. The data sent represents the total data sent among all the parties.

For network training using our truncation, the online phase takes marginally more time, but we improve significantly in the offline phase. The increased run-time in the online phase occurs mainly due to computation in a larger ring, which is also reflected in the sent data. In the offline phase, by removing the need to compute expensive subtraction circuits, we improve the offline run-time by up to an order of magnitude and we lower the communication overhead by two orders of magnitude. We note that our implementation is for demonstration purposes only and has not been optimised for the best performance.

## 7 Application: Privacy-Preserving ECG Diagnostic

Consider a scenario concerning diagnostic prediction using sensitive medical data. In this case, MPC would be a desirable tool to perform ML inference to preserve the privacy of both the data and the model owners. We therefore consider inference of small batch sizes, as this resembles queries of individual users in a real-world system. Further, we note that a lighter pre-processing phase increases

Table 3: Training of MNIST and CIFAR-10 datasets with different networks. We run a forward and a backward pass in each network, with 1 batch and full dataset (60000 samples), respectively.

| | Network | Protocol | Offline | | Online | |
|---|---|---|---|---|---|---|
| | | | Time(s) | Comm.(MB) | Time(s) | Comm.(MB) |
| 1 batch | MNIST (SecureML) | ABY3 | $3.29 \cdot 10^2$ | $1.13 \cdot 10^5$ | $2.50 \cdot 10^1$ | $2.25 \cdot 10^3$ |
| | | **MaSTer(96)** | $7.62 \cdot 10^1$ | $5.95 \cdot 10^3$ | $3.19 \cdot 10^1$ | $3.37 \cdot 10^3$ |
| | MNIST (LeNet-5) | ABY3 | $6.17 \cdot 10^3$ | $2.32 \cdot 10^6$ | $2.54 \cdot 10^2$ | $2.85 \cdot 10^4$ |
| | | **MaSTer(96)** | $9.75 \cdot 10^2$ | $8.95 \cdot 10^4$ | $3.52 \cdot 10^2$ | $4.28 \cdot 10^4$ |
| | CIFAR-10 (AlexNet) | ABY3 | $5.17 \cdot 10^4$ | $1.84 \cdot 10^7$ | $2.91 \cdot 10^3$ | $2.32 \cdot 10^5$ |
| | | **MaSTer(96)** | $9.97 \cdot 10^3$ | $7.58 \cdot 10^5$ | $3.53 \cdot 10^3$ | $3.44 \cdot 10^5$ |
| Full | MNIST (SecureML) | ABY3 | $1.48 \cdot 10^5$ | $5.37 \cdot 10^7$ | $9.19 \cdot 10^3$ | $8.43 \cdot 10^5$ |
| | | **MaSTer(96)** | $3.07 \cdot 10^4$ | $2.71 \cdot 10^6$ | $1.19 \cdot 10^4$ | $1.26 \cdot 10^6$ |

the overall throughput for small batch sizes, which is desirable for applications with real-time response requirement.

Most frameworks designed for privacy-preserving ML consider the offline-online paradigm [4] where a substantial part of the computational load can be moved to the input-independent pre-processing phase. For this approach to be efficient, the clients are assumed to request computations occasionally (e.g., an organisation running inference at the end of every day). This then creates the time for the MPC servers to perform pre-processing when they don't engage in input-dependent computations. This results in major improvements of the online phase. However, in the scenario described above, the servers are constantly active, responding to queries of both individual users and large organisations. This reduces the time of low load and pre-processing, thus increases the latency of a prediction. Consequently, it is important to maximise the overall throughput. Given the example of medical data, we run classification of ECG heartbeat data over a dense neural network. We run experiments on the same setup as described in Sect. 6, evaluating the DNN model trained on the MIT-BIH Arrhythmia dataset [30] together with PTB Diagnostic ECG database [6], both available at [17]. The trained model comprises five dense layers with a ReLU activation function on the first four and the Softmax function on the last layer.

We demonstrate the overall performance in the LAN setting in Table 4 and WAN setting in Table 5. As before, we run ABY3 in the 64-bit ring with $k = 8$, $\ell_x = 16$. Given the lower number of total truncations in the forward pass of the network, we run MaSTer in both the 64 and 96-bit rings for comparison. We note that the abort condition has not been triggered due to the large error in any of our experiments.

We report the execution time of the online and offline phase separately. The data sent denotes the data sent by all parties collectively. Finally, we showcase the total throughput of classifications per minute. We compare the performance

when running inference with a different number of samples per inference query, ranging from a single sample, through a batch size of 128 samples to the whole dataset. We observe an improvement between a factor of two and three for both LAN and WAN settings, depending on the query size. When running the inference on a trained model, we achieve a classification accuracy of 95.9%, same as ABY3 and plaintext.

Table 4: Performance of high-throughput use-case inference with different number of samples per query in the LAN setting. Throughput (TP) is denoted in samples per minute.

| Number of samples | Number of truncations | Protocol | Offline | Online | Total | Data Sent | TP |
|---|---|---|---|---|---|---|---|
| | | | Time (s) | | | (MB) | |
| 1 | $2.36 \cdot 10^4$ | ABY3 | 0.21 | 0.08 | 0.29 | 33.79 | 206.90 |
| | | **MaSTer(64)** | 0.06 | 0.08 | 0.14 | 6.99 | 428.57 |
| | | **MaSTer(96)** | 0.07 | 0.07 | 0.14 | 9.55 | 428.57 |
| 128 | $3.02 \cdot 10^6$ | ABY3 | 9.73 | 0.86 | 10.60 | 3621.68 | 724.53 |
| | | **MaSTer(64)** | 3.18 | 0.95 | 4.13 | 576.75 | 1859.56 |
| | | **MaSTer(96)** | 4.36 | 1.10 | 5.46 | 780.22 | 1406.59 |
| 21892 | $5.16 \cdot 10^8$ | ABY3 | 1811 | 142 | 1953 | 716386 | 672.46 |
| | | **MaSTer(64)** | 514 | 152 | 666 | 98316 | 1972.25 |
| | | **MaSTer(96)** | 729 | 167 | 896 | 132892 | 1465.98 |

Table 5: Performance of high-throughput use-case inference with different number of samples per query in the WAN setting. Throughput (TP) is denoted in samples per minute.

| Number of samples | Number of truncations | Protocol | Offline | Online | Total | Data Sent | TP |
|---|---|---|---|---|---|---|---|
| | | | Time (s) | | | (MB) | |
| 1 | $2.36 \cdot 10^4$ | ABY3 | 10.21 | 3.55 | 13.76 | 33.79 | 4.36 |
| | | **MaSTer(64)** | 3.70 | 5.28 | 8.98 | 6.99 | 6.68 |
| | | **MaSTer(96)** | 7.98 | 6.30 | 14.28 | 9.55 | 4.21 |
| 128 | $3.02 \cdot 10^6$ | ABY3 | 326.17 | 58.11 | 384.28 | 3621.68 | 19.99 |
| | | **MaSTer(64)** | 55.75 | 82.32 | 138.07 | 576.75 | 55.62 |
| | | **MaSTer(96)** | 103.67 | 80.19 | 183.86 | 780.22 | 41.78 |
| 21892 | $5.16 \cdot 10^8$ | ABY3 | 53267 | 9889 | 63157 | 716386 | 20.80 |
| | | **MaSTer(64)** | 9250 | 14015 | 23265 | 98316 | 56.46 |
| | | **MaSTer(96)** | 17091 | 13567 | 30658 | 132892 | 42.84 |

## 8 Conclusion

We presented a maliciously secure truncation protocol combining the two-party semi-honest truncation protocol of SecureML with a consistency check. Unlike previous works for malicious security, we do not require pre-processing. We prove our protocol is secure with abort against one malicious corrupted party in the UC framework. In the end, we demonstrate the improvement of our truncation experimentally by comparing training on the commonly used benchmarking networks, namely SecureML, LeNet-5, and AlexNet. We improve the performance of the offline phase by an order of magnitude, achieving higher total throughput. We also show a potential application where overall throughput needs to be maximised, and hence advocate for an efficient protocol without a heavy pre-processing phase.

**Acknowledgments.** This work was supported by the Flemish Government through FWO SBO project MOZAIK S003321N and by CyberSecurity Research Flanders with reference number VR20192203.

## References

1. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 805–817. ACM, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978331
2. Attrapadung, N., Hamada, K., Ikarashi, D., Kikuchi, R., Matsuda, T., Mishina, I., Morita, H., Schuldt, J.C.N.: Adam in private: Secure and fast training of deep neural networks with adaptive moment estimation. Proc. Priv. Enhancing Technol. **2022**(4), 746–767 (2022). https://doi.org/10.56553/popets-2022-0131
3. Baccarini, A.N., Blanton, M., Yuan, C.: Multi-party replicated secret sharing over a ring with applications to privacy-preserving machine learning. Proc. Priv. Enhancing Technol. **2023**(1), 608–626 (2023). https://doi.org/10.56553/popets-2023-0035
4. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, August 11-15, 1991, Proceedings. Lecture Notes in Computer Science, vol. 576, pp. 420–432. Springer (1991). https://doi.org/10.1007/3-540-46766-1_34
5. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (jul 1970). https://doi.org/10.1145/362686.362692
6. Bousseljot, R., Kreiseler, D., Schnabel, A.: Nutzung der EKG-Signaldatenbank CARDIODAT der PTB über das Internet. Biomedical Engineering / Biomedizinische Technik **40**(s1), 317–318 (1995)
7. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001. pp. 136–145. IEEE Computer Society (2001). https://doi.org/10.1109/SFCS.2001.959888

8. Canetti, R., Cohen, A., Lindell, Y.: A simpler variant of universally composable security for standard multiparty computation. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9216, pp. 3–22. Springer (2015). https://doi.org/10.1007/978-3-662-48000-7_1
9. Catrina, O., de Hoogh, S.: Improved primitives for secure multiparty integer computation. In: Garay, J.A., Prisco, R.D. (eds.) Security and Cryptography for Networks, 7th International Conference, SCN 2010, September 13-15, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6280, pp. 182–199. Springer (2010). https://doi.org/10.1007/978-3-642-15317-4_13
10. Chaudhari, H., Choudhury, A., Patra, A., Suresh, A.: Astra: High throughput 3pc over rings with application to secure prediction. In: Sion, R., Papamanthou, C. (eds.) Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, CCSW@CCS 2019, November 11, 2019. pp. 81–92. ACM (2019). https://doi.org/10.1145/3338466.3358922
11. Chaudhari, H., Rachuri, R., Suresh, A.: Trident: Efficient 4pc framework for 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), https://www.ndss-symposium.org/ndss-paper/trident-efficient-4pc-framework-for-privacy-preserving-machine-learning/
12. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: Spdz2k: Efficient MPC mod $2^k$ for dishonest majority. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 769–798. Springer (2018). https://doi.org/10.1007/978-3-319-96881-0_26
13. Dalskov, A.P.K., Escudero, D., Keller, M.: Secure evaluation of quantized neural networks. Proc. Priv. Enhancing Technol. **2020**(4), 355–375 (2020). https://doi.org/10.2478/popets-2020-0077
14. Dalskov, A.P.K., Escudero, D., Keller, M.: Fantastic four: Honest-majority four-party secure computation with malicious security. In: Bailey, M., Greenstadt, R. (eds.) 30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021. pp. 2183–2200. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/dalskov
15. Escudero, D., Ghosh, S., Keller, M., Rachuri, R., Scholl, P.: Improved primitives for MPC over mixed arithmetic-binary circuits. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12171, pp. 823–852. Springer (2020). https://doi.org/10.1007/978-3-030-56880-1_29
16. Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10211, pp. 225–255 (2017). https://doi.org/10.1007/978-3-319-56614-6_8
17. Goldberger, A.L., Amaral, L.A., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E.: Physiobank, phys-

iotoolkit, and physionet: Components of a new research resource for complex physiologic signals. Circulation **101**(23), E215–20 (Jun 2000)
18. Gupta, K., Jawalkar, N., Mukherjee, A., Chandran, N., Gupta, D., Panwar, A., Sharma, R.: SIGMA: secure GPT inference with function secret sharing. IACR Cryptol. ePrint Arch. p. 1269 (2023), https://eprint.iacr.org/2023/1269
19. Gupta, K., Kumaraswamy, D., Chandran, N., Gupta, D.: Llama: A low latency math library for secure inference. Proc. Priv. Enhancing Technol. **2022**(4), 274–294 (2022). https://doi.org/10.56553/popets-2022-0109
20. Huang, Z., Lu, W., Hong, C., Ding, J.: Cheetah: Lean and fast secure two-party deep neural network inference. In: Butler, K.R.B., Thomas, K. (eds.) 31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022. pp. 809–826. USENIX Association (2022), https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong
21. Kachuee, M., Fazeli, S., Sarrafzadeh, M.: Ecg heartbeat classification: A deep transferable representation. CoRR **abs/1805.00794** (2018), http://arxiv.org/abs/1805.00794
22. Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020. pp. 1575–1590. ACM (2020). https://doi.org/10.1145/3372297.3417872
23. Koti, N., Pancholi, M., Patra, A., Suresh, A.: Swift: Super-fast and robust privacy-preserving machine learning. In: Bailey, M., Greenstadt, R. (eds.) 30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021. pp. 2651–2668. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/koti
24. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Bartlett, P.L., Pereira, F.C.N., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012. pp. 1106–1114 (2012), https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
25. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
26. Li, Y., Duan, Y., Huang, Z., Hong, C., Zhang, C., Song, Y.: Efficient 3pc for binary circuits with application to maliciously-secure dnn inference. In: Calandrino, J.A., Troncoso, C. (eds.) 32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023. USENIX Association (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/li-yun
27. Mitzenmacher, M.: Compressed bloom filters. In: Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing. p. 144–150. PODC '01, Association for Computing Machinery, New York, NY, USA (2001). https://doi.org/10.1145/383962.384004
28. Mohassel, P., Rindal, P.: Aby$^3$: A mixed protocol framework for machine learning. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 35–52. ACM (2018). https://doi.org/10.1145/3243734.3243760
29. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San

Jose, CA, USA, May 22-26, 2017. pp. 19–38. IEEE Computer Society (2017). https://doi.org/10.1109/SP.2017.12
30. Moody, G.B., Mark, R.G.: The impact of the mit-bih arrhythmia database. IEEE engineering in medicine and biology magazine **20**(3), 45–50 (2001)
31. Patra, A., Suresh, A.: 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), https://www.ndss-symposium.org/ndss-paper/blaze-blazing-fast-privacy-preserving-machine-learning/
32. Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow2: Practical 2-party secure inference. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, November 9-13, 2020. pp. 325–342. ACM (2020). https://doi.org/10.1145/3372297.3417274
33. Rotaru, D., Wood, T.: Marbled circuits: Mixing arithmetic and boolean circuits with active security. In: Hao, F., Ruj, S., Gupta, S.S. (eds.) Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11898, pp. 227–249. Springer (2019). https://doi.org/10.1007/978-3-030-35423-7_12
34. Tarkoma, S., Rothenberg, C.E., Lagerspetz, E.: Theory and practice of bloom filters for distributed systems. IEEE Communications Surveys & Tutorials **14**(1), 131–155 (2012). https://doi.org/10.1109/SURV.2011.031611.00024
35. Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: Falcon: Honest-majority maliciously secure framework for private deep learning. Proc. Priv. Enhancing Technol. **2021**(1), 188–208 (2021). https://doi.org/10.2478/popets-2021-0011
36. Watson, J., Wagh, S., Popa, R.A.: Piranha: A gpu platform for secure computation. In: Butler, K.R.B., Thomas, K. (eds.) 31st USENIX Security Symposium, USENIX Security 2022, August 10-12, 2022. pp. 827–844. USENIX Association (2022), https://www.usenix.org/conference/usenixsecurity22/presentation/watson
37. Zhou, L., Song, Q., Zhang, S., Wang, Z., Wang, X., Li, Y.: Bicoptor 2.0: Addressing challenges in probabilistic truncation for enhanced privacy-preserving machine learning. CoRR **abs/2309.04909** (2023). https://doi.org/10.48550/ARXIV.2309.04909
38. Zou, H., Xiao, Y., Zhang, R.: Semi-honest 2-party faithful truncation from two-bit extraction. IACR Cryptol. ePrint Arch. p. 1159 (2023), https://eprint.iacr.org/2023/1159

## A Protocol $\Pi_{\mathsf{cr}}$

In Fig. 7, we give a possible protocol to securely realise $\mathcal{F}_{\mathsf{cr}}$ (Fig. 6) in the presence of a static malicious adversary. We use the protocol for generating correlated randomness from [16, Protocol 2.5] to generate a uniform random shared value among all three parties for the setup and then open this share to $P_1$ and $P_3$. The opened value is used as key $k$ to the PRF. Subsequent calls to sample simply evaluate the PRF keyed with $k$ on a unique counter. The protocol $\Pi_{\mathsf{cr}}$ is a specialised variant where only two parties learn the opened value.

In the random oracle model, the setup of a random key $k$ between party $P_1$ and $P_3$ can also be implemented by having each party draw a random value $r$, commit to it via the random oracle and send the commitment. Then each party decommits and uses the xor of the two random values as $k$.

In Fig. 8 we provide a protocol for correlated randomness of 0 among all three parties.

**Functionality $\mathcal{F}_{\mathsf{cr}}$**

Let $F : \{0,1\}^\kappa \times \{0,1\}^\kappa \mapsto \mathbb{Z}_{2^\ell}$ be a keyed pseudo-random function. The functionality generates correlated randomness between a pair $(P_i, P_j)$

**Setup** The adversary sends a key $k_{\mathcal{A}}$ and the functionality samples $k \leftarrow\!\!\$\ \{0,1\}^\kappa$ uniformly at random and sends $F_{k_{\mathcal{A}}}(0) \oplus k$ to $P_i$ and $P_j$.

**Sample** On input $cnt$ of $P_i$ and $P_j$, the functionality computes $r := F_k(cnt)$ and sends $r$ to $P_i$ and $P_j$.

Fig. 6: Functionality $\mathcal{F}_{\mathsf{cr}}$.

## B Proof of Theorem 1

*Proof.* We consider two cases. First, let $0 \leq z < 2^{\ell_x}$, i.e., $x \geq 0$ and $z = x$. Then, $z + R = (x_1 + R_1)2^k + x_2 + R_2$. Since $x_2 + R_2$ may be larger or equal $2^k$, we have $z + R = (x_1 + R_1 + c_1)2^k + x_2 + R_2 - c_1 2^k$. Further, $x_1 + R_1$ may overflow over $2^{\ell-k}$, hence we get $z + R = c_2 2^\ell + (x_1 + R_1 + c_1 - c_2 2^{\ell-k})2^k + x_2 + R_2 - c_1 2^k$. Consequently, $\mathsf{rshift}(z+R)+2^\ell-\mathsf{rshift}(R) = 2^\ell+x_1+R_1+c_1-c_2 2^{\ell-k}+2^\ell-R_1 = \lfloor x \rfloor + c_1 - c_2 2^{\ell-k}$ where $c_1 = 1$ if $z_2 + R_2 \geq 2^k$ and $c_1 = 0$ otherwise. Similarly, $c_2 = 1$ if $x_1 + R_1 \geq 2^{\ell-k}$ and $c_2 = 0$ otherwise. Note that $x_1 = \lfloor x \rfloor$ by definition and $2^\ell = 0$ in the ring.

Second, let $2^\ell - 2^{\ell_x} \leq z < 2^\ell$, i.e., $x < 0$ and $z = 2^\ell - |x|$. So $z + R = 2^\ell - (R_1 - |x_1|)2^k + R_2 - |x_2|$. Since $R_2 - |x_2|$ may be negative, we have $z + R = (R_1 - z_1 - c_1)2^k + R_2 - z_2 + c_1 2^k$ where $c_1 = 1$ if $R_2 < |x_2|$ and $c_1 = 0$ otherwise. Similarly, $R_1 - |x_1|$ might underflow, hence we have $z + R = -c_2 2^\ell + (R_1 - |x_1| - c_1 + c_2 2^{\ell-k})2^k + R_2 - z_2 + c_1 2^k$ where $c_2 = 1$ if $|x_1| > R_1$ and $c_2 = 0$ otherwise. Thus $\mathsf{rshift}(z + R) + 2^\ell - \mathsf{rshift}(R) = -c_2 2^\ell + R_1 - |x_1| - c_1 + c_2 2^{\ell-k} - R_1 = \lfloor x \rfloor - c_1 + c_2 2^{\ell-k}$.

## C Proof of Theorem 2

We start by introducing Lemma 1.

**Protocol $\Pi_{\mathsf{cr}}$**

Let $F : \{0,1\}^\kappa \times \{0,1\}^\kappa \mapsto \mathbb{Z}_{2^\ell}$ be a keyed pseudo-random function. The protocol generates correlated randomness between a pair $(P_i, P_j)$.

**Setup** Each party $P_t$ samples a key $k_t \leftarrow\!\!\$\ \{0,1\}^n$ uniformly at random.
- $P_t$ sends $k_t$ to $P_{t-1}$ and thus obtains $k_{t+1}$.
- $P_t$ computes $r_t := F_{k_t}(0), r_{t+1} := F_{k_{t+1}}(0)$. This creates a RSS share of a random value.
- The parties send $P_i$'s missing share to $P_i$ and $P_j$'s missing share to $P_j$.
- Now $P_i$ and $P_j$ hold $(r_i, r_{i+1})$ and $(r_j, r_{j+1})$, respectively, and received $r_{i-1}$ and $r_{j-1}$, respectively, from two independent parties. $P_i/P_j$ abort if the received values don't agree. Note that the other party only knows two of the three shares.
- $P_i$ and $P_j$ set $k := \bigoplus_{t=1}^{3} r_t$.

**Sample** $P_i$ and $P_j$ locally compute $F_k(cnt)$ and use the result of the PRF as random element in $\mathbb{Z}_{2^\ell}$.

Fig. 7: The protocol $\Pi_{\mathsf{cr}}$.

**Protocol $\Pi_{\mathsf{cr0}}$**

Let $F : \{0,1\}^\kappa \times \{0,1\}^\kappa \mapsto \mathbb{Z}_{2^\ell}$ be a keyed pseudo-random function.

**Setup** Each party $P_i$ samples a key $k_i \leftarrow\!\!\$\ \{0,1\}^n$ uniformly at random. Then, $P_i$ sends $k_i$ to $P_{i+1}$ and thus obtains $k_{i-1}$.

**Sample** Upon input $cnt$, $P_i$ locally computes $F_{k_i}(cnt) - F_{k_{i-1}}(cnt)$ and outputs the result of the PRF as random element in $\mathbb{Z}_{2^\ell}$.

Fig. 8: The protocol $\Pi_{\mathsf{cr0}}$.

**Lemma 1.** *Let $m \leq \ell$. Let $R$ be a uniformly random variable defined on $[0, 2^m)$ and let $X$ be a random variable defined on $[0, 2^m)$. Then,* $\Pr[X+R \geq 2^m] = \frac{\mathbb{E}[X]}{2^m}$.

*Proof.*

$$\begin{aligned}\Pr[X + R \geq 2^m] = \Pr[R \geq 2^m - X] &= \sum_{x=1}^{2^m-1} \Pr[X = x] \sum_{r=2^m-x}^{2^m-1} \Pr[R = r] \\ &= \sum_{x=1}^{2^m-1} \Pr[X = x] \sum_{r=2^m-x}^{2^m-1} \frac{1}{2^m} = \sum_{x=1}^{2^m-1} \Pr[X = x] \frac{x}{2^m} \\ &= \frac{1}{2^m} \sum_{x=1}^{2^m-1} x \Pr[X = x] = \frac{1}{2^m} \sum_{x=0}^{2^m-1} x \Pr[X = x] = \frac{\mathbb{E}[X]}{2^m}\,.\end{aligned}$$

We now proceed to the proof of Theorem 2.

*Proof.* Let $z \in \mathbb{Z}_{2^\ell}$ be the encoding of $-2^{\ell_x} < x < 2^{\ell_x}$. Let $s_1, s_2, s_3$ be the components of the RSS sharing of $z$. We set $s_1 = z + R' + R''$, $s_2 = 2^\ell - R'$, $s_3 = 2^\ell - R''$ where $R'$ and $R''$ are random elements in $\mathbb{Z}_{2^\ell}$. Whenever required, we decompose a ring element $x$ into $x_1, x_2$ s.t. $x = x_1 2^k + x_2$ where $0 \leq x_1 < 2^{\ell-k}$ and $0 \leq x_2 < 2^k$. Note that since $x \leq 2^{\ell_x}$, $x_1 \leq 2^{\ell_x - k}$. Consequently, $\mathsf{rshift}(x_1 2^k + x_2) = x_1$. Firstly note that $\sum_{i=1}^3 \gamma_i = \sum_{i=1}^3 s'_i - \sum_{i=1}^2 s''_i$. Then,

$$\begin{aligned}\sum_{i=1}^{3} s'_i &= r' + \mathsf{rshift}(s_1 + s_2, k) - r' + 2^\ell - \mathsf{rshift}(2^\ell - s_3, k) \\ &= \mathsf{rshift}(z + R'', k) + 2^\ell - \mathsf{rshift}(R'', k)\,.\end{aligned}$$

We apply Theorem 1 and obtain $\sum_{i=1}^3 s'_i = \lfloor x \rfloor + c'_1 + c'_2 2^{\ell-k}$ where $c'_1 = 1$ if $x_2 + R''_2 \geq 2^k$, $c'_1 = -1$ if $|x_2| \geq R''_2$, or $c'_1 = 0$ otherwise. Similarly, $c'_2 = 1$ if $|x_1| > R''_1$, $c'_2 = -1$ if $x_1 + R''_1 \geq 2^{\ell-k}$ and $c'_2 = 0$ otherwise.

For $\sum_i s''_i$, we obtain

$$\begin{aligned}\sum_{i=1}^{2} s''_i &= \mathsf{rshift}(s_1 + s_3, k) + 2^\ell - \mathsf{rshift}(2^\ell - s_2, k) \\ &= \mathsf{rshift}(z + R', k) + 2^\ell - \mathsf{rshift}(R', k)\,.\end{aligned}$$

Again, we apply Theorem 1 and get $\sum_{i=1}^2 s''_i = \lfloor x \rfloor + c''_1 + c''_2 2^{\ell-k}$ where $c''_1 = 1$ if $x_2 + R'_2 \geq 2^k$, $c''_1 = -1$ if $|x_2| \geq R'_2$, or $c''_1 = 0$ otherwise. Similarly, $c''_2 = 1$ if $|x_1| > R'_1$, $c''_2 = -1$ if $x_1 + R'_1 \geq 2^{\ell-k}$ and $c''_2 = 0$ otherwise.

Now, we have that $\sum_{i=1}^3 \gamma_i = \lfloor x \rfloor + c'_1 + c'_2 2^{\ell-k} - \lfloor x \rfloor - c''_1 - c''_2 2^{\ell-k} = (c'_1 - c''_1) + (c'_2 - c''_2) 2^{\ell-k}$. We want to show that $\sum_{i=1}^3 \gamma_i \in \{0, \pm 1\}$. Thus, we need to show $|c'_1 - c''_1| \leq 1$ and $c'_2 - c''_2 = 0$. Let us first derive the probability for $c'_2 - c''_2 = 0$.

Note that $c'_2, c''_2 = \pm 1 \iff R'', R' \in [0, 2^{\ell_x}) \cup [2^\ell - 2^{\ell_x}, 2^\ell)$. Since $R', R''$ are uniformly random by definition, we have that $\Pr[|x_1| > R''_1 \mid R'' \in [0, 2^{\ell_x})] =$

$\Pr[|x_1| > R'_1 \mid R' \in [0, 2^{\ell_x})] = \frac{\mathbb{E}[x_1]}{2^{\ell_x - k}}$. This follows from Lemma 1, observing that $|x_1|, R'_1, R''_1 \in [0, 2^{\ell_x - k})$. Similarly, $\Pr[x_1 + R''_1 \geq 2^{\ell-k} \mid R'' \in [2^\ell - 2^{\ell_x}, 2^\ell)] = \Pr[x_1 + R'_1 \geq 2^{\ell-k} \mid R' \in [2^\ell - 2^{\ell_x}, 2^\ell)] = \frac{\mathbb{E}[x_1]}{2^{\ell_x - k}}$. This follows again from Lemma 1, noting that $x \in [0, 2^{\ell_x - k})$ and $R'_1, R''_1 \in [2^{\ell-k} - 2^{\ell_x - k}, 2^{\ell-k})$.

Let condition $A(\Gamma) = \Gamma \in [0, 2^{\ell_x}) \cup [2^\ell - 2^{\ell_x}, 2^\ell)$ for $\Gamma \in \{R', R''\}$. Therefore, we have

$$\begin{aligned}
&\Pr[c'_2 - c''_2 = 0] \geq \Pr[c'_2 = 0 \wedge c''_2 = 0] \\
&\quad = 1 - \Pr[A(R'')] \cdot \Pr[|x_1| > R''_1 \vee x_1 + R''_1 \geq 2^{\ell-k} \mid A(R'')] \cdot \\
&\quad 1 - Pr[A(R')] \cdot \Pr[(|x_1| > R'_1 \vee x_1 + R'_1 \geq 2^{\ell-k}) \mid A(R')] \\
&\quad = \left(1 - 2^{\ell_x + 1 - \ell} \cdot \frac{\mathbb{E}[x_1]}{2^{\ell_x - k}}\right)^2 \geq 1 - 2^{k+2-\ell} \cdot \mathbb{E}[x_1] \,.
\end{aligned}$$

For $x_1$ uniformly random, this would become $\Pr[c'_2 - c''_2 = 0] \geq 1 - 2^{\ell_x + 1 - \ell}$.

It remains to show that $|c'_1 - c''_1| \leq 1$. This is violated only if $c'_1 = 1$ and $c''_1 = -1$, or if $c'_1 = -1$ and $c''_1 = 1$. In both cases, the condition s.t. $c'_1$ has the fixed value contradicts the precondition of $x$ for $c''_1$, e.g., $x \geq 0 \wedge x < 0$. Thus, the probability that $R'$ and $R''$ are within $[2^{\ell_x}, 2^\ell - 2^{\ell_x})$ and no overflow or underflow occurs is $\left(1 - 2^{\ell_x + 1 - \ell} \cdot \frac{\mathbb{E}[x_1]}{2^{\ell_x - k}}\right)^2 \geq 1 - 2^{k+2-\ell} \cdot \mathbb{E}[x_1]$.

## D Analysis of Adversarial Influence with $\Delta = \pm 2$

**Definition 4.** *We define the sign function as*

$$sgn(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases} .$$

We will now show $\Pr[|c' - c'' + \Delta| = 1 \mid \text{sgn}(x) \wedge \Delta = \pm 2] \leq \frac{\mathbb{E}[x_2]}{2^k} - \frac{(\mathbb{E}[x_2])^2}{2^{2k}} \leq \frac{1}{4}$. Firstly, note that

$$\begin{aligned}
&\Pr[|c' - c'' + \Delta| = 1 \mid \text{sgn}(x) \wedge \Delta = \pm 2] \\
&= \Pr[c' = 1 \wedge c'' = 0 \mid \text{sgn}(x) = 1 \wedge \Delta = -2] \cdot \Pr[\text{sgn}(x) = 1] \cdot \Pr[\Delta = -2] + \\
&\Pr[c' = 0 \wedge c'' = 1 \mid \text{sgn}(x) = 1 \wedge \Delta = 2] \cdot \Pr[\text{sgn}(x) = 1] \cdot \Pr[\Delta = 2] + \\
&\Pr[c' = -1 \wedge c'' = 0 \mid \text{sgn}(x) = -1 \wedge \Delta = 2] \cdot \Pr[\text{sgn}(x) = -1] \cdot \Pr[\Delta = 2] + \\
&\Pr[c' = 0 \wedge c'' = -1 \mid \text{sgn}(x) = -1 \wedge \Delta = -2] \cdot \Pr[\text{sgn}(x) = -1] \cdot \Pr[\Delta = -2] \,.
\end{aligned}$$

We will show $\Pr[c' = 1 \wedge c'' = 0 \mid \text{sgn}(x) = 1 \wedge \Delta = -2] \cdot \Pr[\text{sgn}(x) = 1] \cdot \Pr[\Delta = -2]$. We note that the other cases are analogous.

**Case** $c' = 1 \wedge c'' = 0$**.** From Theorem 1 we get that $c' = 1$ if $x_2 + R''_2 \geq 2^k$ and $c'' = 0$ when $x_2 + R'_2 < 2^k$ for $x \geq 0$. Therefore,

$$\begin{aligned}
&\Pr[c' = 1 \wedge c'' = 0 \mid \text{sgn}(x) = 1 \wedge \Delta = -2] \cdot \Pr[\text{sgn}(x) = 1] \cdot \Pr[\Delta = -2] \\
&= \Pr[x_2 + R''_2 \geq 2^k \wedge x_2 + R'_2 < 2^k] \cdot \Pr[\text{sgn}(x) = 1] \cdot \Pr[\Delta = -2] \,.
\end{aligned}$$

Since $x_2, R'_2, R''_2 \in [0, 2^k)$ with $R', R''$ uniformly random, we can now apply Lemma 1. Thus, we get

$$\begin{aligned}&\Pr[c' = 1 \wedge c'' = 0 \mid \mathrm{sgn}(x) = 1 \wedge \Delta = -2] \cdot \Pr[\mathrm{sgn}(x) = 1] \cdot \Pr[\Delta = -2] \\ &= \left( \frac{\mathbb{E}[x_2]}{2^k} - \frac{(\mathbb{E}[x_2])^2}{2^{2k}} \right) \cdot \Pr[\mathrm{sgn}(x) = 1] \cdot \Pr[\Delta = -2] \,.\end{aligned}$$

Now, we distinguish two types of adversarial strategies. Firstly, consider the adversary that adds a constant error of either $\Delta = 2$ or $\Delta = -2$. Wlog., let the adversary pick $\Delta = 2$. Hence, $\Pr[\Delta = 2] = 1$ and $\Pr[\Delta = -2] = 0$. Therefore, we get

$$\begin{aligned}&\Pr[|c' - c'' + \Delta| = 1 \mid \mathrm{sgn}(x) \wedge \Delta = \pm 2] \\ &= \left( \frac{\mathbb{E}[x_2]}{2^k} - \frac{(\mathbb{E}[x_2])^2}{2^{2k}} \right) \cdot (\Pr[\mathrm{sgn}(x) = -1] + \Pr[\mathrm{sgn}(x) = 1]) \leq \frac{\mathbb{E}[x_2]}{2^k} - \frac{(\mathbb{E}[x_2])^2}{2^{2k}} \,.\end{aligned}$$

Secondly, consider the case when adversary picks $\Delta = \pm 2$ at random. Then we get $\Pr[\Delta = 2] = \Pr[\Delta = -2] = \frac{1}{2}$. Thus,

$$\begin{aligned}&\Pr[|c' - c'' + \Delta| = 1 \mid \mathrm{sgn}(x) \wedge \Delta = \pm 2] \\ &= 2 \cdot \left( \frac{\mathbb{E}[x_2]}{2^k} - \frac{(\mathbb{E}[x_2])^2}{2^{2k}} \right) \cdot (\Pr[\mathrm{sgn}(x) = -1] + \Pr[\mathrm{sgn}(x) = 1]) \cdot \frac{1}{2} \\ &\leq \frac{\mathbb{E}[x_2]}{2^k} - \frac{(\mathbb{E}[x_2])^2}{2^{2k}} \,.\end{aligned}$$

Now, let us find a local maxima of $f(x) = \left( \frac{x}{2^k} - \frac{x^2}{2^{2k}} \right)$, for $x \in [0, 2^k)$. We have

$$\frac{\mathrm{d}f}{\mathrm{d}x} = 2^{-k} - x \cdot 2^{-2k} = 0 \iff x = 2^{k-1} \,,$$

and

$$f(0) = f(2^k) = 0 < f(2^{k-1}) = \frac{1}{4} \,.$$

Therefore, we can conclude $\left( \frac{\mathbb{E}[x_2]}{2^k} - \frac{(\mathbb{E}[x_2])^2}{2^{2k}} \right) \leq \frac{1}{4}$, with global maximum when $\mathbb{E}[x_2] = 2^{k-1}$, e.g., when $x_2$ is uniformly random.

## E The UC Security Model

In the following, we expand on the universal composability framework (UC) as proposed by Canetti et al. [7,8]. We say that if a protocol $\Pi$ securely realises a given functionality $\mathcal{F}$ in the UC framework, the leakage of the protocol through inputs, outputs, intermediary values or abort conditions is equivalent to the leakage that occurs when an adversary and honest parties interact with the incorruptible third-party $\mathcal{F}$ *even* under arbitrary composition with other protocols

or (more useful for PPML) as sub-protocol for larger protocols that supply input and receive output from $\Pi$.

We target static malicious security with abort. This means, the adversary $\mathcal{A}$ corrupts a subset of parties at the start of the protocol and can make them behave arbitrarily, including aborting the protocol execution prematurely. In this setting, we assume a rushing adversary that determines the order of arrival for each message that is sent over the network. In particular, this means that in each communication round, the adversary may see all messages directed at a corrupted party before it responds. Thus, the adversary always receives the output of the protocol first and may decide to abort, preventing the honest parties from learning the output.

In the UC framework, there are two additional actors to $\Pi$. The adversary $\mathcal{A}$ that attacks the protocol $\Pi$ by corrupting a subset of parties and the environment $\mathcal{Z}$ that attacks the security simulation which is defined as a distinguishing game between two executions.

- $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}$ runs an attack on the protocol $\Pi$ by $\mathcal{A}$ as follows. $\mathcal{Z}$ chooses and sends input values for the honest parties to the honest MPC parties. Then, $\mathcal{A}$, controlling the corrupted parties, executes $\Pi$ with the honest parties. At the end of the protocol, the honest parties send their output to $\mathcal{Z}$. $\mathcal{Z}$ can communicate with $\mathcal{A}$ arbitrarily throughout the whole process.
- $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ runs an attack on the simulator $\mathcal{S}$. $\mathcal{Z}$ chooses inputs for the honest parties and sends them directly to $\mathcal{F}$. $\mathcal{Z}$ receives the output of the honest parties directly from $\mathcal{F}$. $\mathcal{S}$ plays the role of the adversary from the real execution and interacts arbitrarily with $\mathcal{Z}$.

If $\Pi$ securely realises $\mathcal{F}$, then for each adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ that for each environment $\mathcal{Z}$, $\mathcal{Z}$ cannot distinguish between the real execution $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}$ or the idealised one $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.